Advancements in Safe Deep Reinforcement Learning for Real-Time Strategy Games and Industry Applications

Per-Arne Andersen

Advancements in Safe Deep Reinforcement Learning for Real-Time Strategy Games and Industry Applications

Doctoral Dissertation for the Degree *Philosophiae Doctor (Ph.D)* at
the Faculty of Engineering and Science,
Department of Information and Communication Technology,
Specialization in Artificial Intelligence

University of Agder
Faculty of Engineering and Science
2022

Doctoral dissertations at the University of Agder: 361
ISSN: 1504-9272
ISBN: 978-82-8427-070-8

Print: 07 Media
Kristiansand

Dedicated to my family

Tina, Imre, and Joachim

Preface

This dissertation is a result of research work carried out at the Department of Information and Communication Technology (ICT), Center for Artificial Intelligence (CAIR) at the University of Agder (UiA) in Grimstad, Norway. The work started as part of an integrated Ph.D. program from January 2017 until January 2018 and was full-time until December 2021. Professor Morten Goodwin, UiA, is the primary supervisor for this Ph.D. work. Professor Ole-Christoffer Granmo, UiA, is the co-supervisor. The Department of ICT, UiA, funds this research under Associate Professor Folke Haugland, head of the department.

Production note: LaTeX is the primary software for writing this dissertation and the papers produced during the Ph.D. study. We acquire results from algorithms and simulation using C++ and PYTHON with various libraries and tooling. Most notable is the use of TENSORFLOW for building the models.

Acknowledgments

I want to thank my primary supervisor, Professor Morten Goodwin, for his guidance throughout my studies and for constantly pushing me to be a better researcher. Thanks to my co-supervisor, Professor Ole-Christoffer Granmo, for all the support and help when needed. Thanks to Ass. Prof. Folke Haugland for following up on my work and allowing me to get more involved in academia. Also, my warmest thanks to Tina Mortensen Kjær for being my better half and keeping up with my all-dayers, all-nighters, and even all-*monthers* in the last couple of years. Tina, none of this work would have been possible without the love, support, and organization you have provided throughout my Ph.D. studies. Finally, I would like to thank Rashmi Gupta at UiA, Kristian Hovde Liland, and Oliver Tomic as NMBU for their immense help in reviewing this thesis.

Abstract

Deep reinforcement learning has attracted considerable attention from industry and academia, among others, because of its success in solving intricate video games and industrial applications. Recent advancements in hardware and computing exponentially increase computational power availability, facilitating deep neural networks training. These networks can learn the RL behavior policy from high-dimensional data and perform significantly better than exact tabular solutions, albeit requiring considerably more computer resources.

Games are among the most used applications to assess reinforcement learning (RL) algorithms' behavioral properties and planning efficiency. They can provide the data structure and volume required to train deep learning models. Specially crafted games can express real-world industry applications to reduce setup costs while drastically increasing reproducibility. RL can improve efficiency in industrial applications where expert systems dominate the scene, reduce manual and potentially dangerous labor. The problem with applied industrial reinforcement learning is that traditional methods learn by trial and error. Because of this, RL agents risk encountering catastrophic events during learning, which can cause damage to humans or equipment. Therefore, using games to train and study safe RL agents is appealing.

Real-Time Strategy (RTS) games are especially captivating because of their high dimensional state and action spaces. Furthermore, RTS games share many attributes with industrial and real-world applications, such as simultaneous actions, imperfect information, and system stochasticity. Recent advancements show that model-free RL algorithms can learn superhuman performance in games such as StarCraft II, again using substantial computational power. Therefore, the downside is that these algorithms are expensive and hard to train, making it challenging to use the same methods for industrial applications. There are also substantial state-space complexity gaps in open-source environments. This restricts algorithm evaluation to only a subset of tasks required for operating sufficiently in industry applications.

Game environments: This thesis addresses the environment gap by proposing six new game environments to evaluate RL algorithms in several tasks. Deep Line Wars and Deep RTS are two novel RTS environments for testing algorithms in long-term planning under imperfect information. Deep Maze is a flexible labyrinth environment for learning RL agents to navigate mazes from memory. Deep Warehouse is a specially crafted environment for evaluating the safety of RL algorithms in automated storage and retrieval systems (ASRS), which is the exclusive focus of this thesis. An ASRS has autonomous vehicles that seek to maximize item throughput in a three-dimensional grid. The design goal of the proposed environments is to facilitate a plethora of additional problems for RL algorithm evaluation. Therefore, all environments provide parameters that adjust the problem complexity and a flexible scenario engine that can challenge algorithms in various problems, such as memory and control. We empirically show that our environments are significantly more computationally efficient than environments of similar complexity. The diversity of proposed environments can help fill the complexity gap in the literature. We finally introduce the Center for Artificial Intelligence and Reinforcement Learning (CaiRL) toolkit for high-performance RL research, which collects all proposed environments in a single runtime.

Model-based RL: This thesis also introduces new energy-efficient, high-performance RL algorithms for RTS games and industry-near simulations using the introduced environments. Model-free reinforcement learning shows promising results in simulated environments but is inadequate for industrial applications. They need to collect millions of samples and learn through trial and error. Conversely, model-based reinforcement learning (MBRL) exploits a known or learned dynamics model, which can substantially increase sample efficiency. Therefore, model-based RL is a more robust study choice for industrial applications than model-free RL methods. The current model-based RL literature shows that deep learning-based models perform best but have several shortcomings. Deep learning models are often sensitive to hyper-parameters, and slight changes to the real environment significantly affect the model accuracy. Furthermore, existing models do not account for safety or risk when deriving behavioral policies, making such methods problematic for industrial applications.

This thesis addresses some of these challenges and proposes novel model-based reinforcement learning methods that focus on decision safety and sample efficiency. Our algorithms, the Dreaming Variational Autoencoder (DVAE), Deep Variational Q-Networks (DVQN), and Observation Reward Action Cost Learning Ensemble (ORACLE), combine model-based RL and variational Bayesian methods to train dynamics models on existing and proposed environments. The DVAE algorithm uses recurrent neural networks and variational autoencoders to learn the dynamics model and show effectiveness in primitive environments. DVQN uses variational autoencoders and deep Q-Networks for interpretable and separable latent spaces and contributes to automatic options discovery in hierarchical reinforcement learning. Finally, ORACLE combines state-space, recurrent neural, and stochastic neural networks. The algorithm shows state-of-the-art prediction capabilities while using auxiliary safety objectives for safer learning.

We then take advantage of the dynamics model to train model-free algorithms offline. Furthermore, we use risk-directed exploration and curiosity to build risk-sensitive agents for improving decision safety in games and industrial applications. We empirically show that our methods in most cases perform better than state-of-the-art model-free and model-based algorithms in traditional RL benchmarks, RTS games, and simulated industrial applications.

In a nutshell, we believe that the game environments, RL methods, and studies presented in this dissertation will advance the state-of-the-art research within the studied topics and contribute positively towards solutions for enabling model-based RL in industry applications.

Summary

Denne avhandlingen fremmer banebrytende algoritmer innen kunstig intelligens-baserte beslutningssystemer. Disse algoritmer er testet i sanntids strategispill som StarCraft II og i oppdragskritiske industrielle systemer. Forskningsområdet er dyp forsterkende læring (deep reinforcement learning), en kombinasjon av dyp læring og forsterkende læring. Det overordnede målet med dette forskningsarbeidet er å gjøre datasystemer i stand til å nå optimale beslutningssekvenser uten å gjøre feil.

Spill brukes ofte til å teste effektiviteten til forsterkningslæringssystemer. For eksempel kan spill gi simuleringer av virkelige industriapplikasjoner som reduserer eksperimentkostnadene og forbedrer reproduserbarheten. Forsterkende læring kan eliminere manuelt eller risikabelt arbeid i industrielle omgivelser. Ekspert systemer dominerer automasjon av industrielle miljøer i dag, hvor det kan være vanskelig å definerer optimale regelsett for komplekse problem. Tradisjonelle forsterkende læringssystemer lærer ved prøving og feiling. Som et resultat risikerer forsterkende læringsagenter å skade mennesker eller utstyr mens de lærer. Derfor kan bruk av spill for å lære forsterkende læringsmidler å operere trygt eliminere disse risikoene. Nøkkeleffekten av å løse disse bekymringene er å muliggjøre svært effektive og sikre autonome systemer som eksisterer i ulike former i samfunnets daglige rutine.

Kompleksiteten til sanntidsstrategispill er interessant for forskning på kunstig intelligens. Oppgaver som krever samtidige operasjoner, ufullkommen informasjon og systemtilfeldighet er elementer i sanntids strategispill. Med den siste utviklingen lærer forsterkningslæringsalgoritmer å oppnå overmenneskelig ytelse i spill som StarCraft II. Ulempen er at disse algoritmene er dyre og vanskelige å trene, noe som gjør dem vanskelige å bruke i industrielle applikasjoner.

Forskningsgapet: Reinforcement learning er en prosess der maskinen søker å maksimere et tilbakemeldingssignal gjennom prøving og feiling. Nåværende banebrytende forsterkende læringsalgoritmer har vesentlige begrensninger fordi de krever mye utforskning for å lære gode beslutningssekvenser. Denne utforskende tilnærmingen kan føre til uønskede utfall i virkelige systemer.

Generelt følger forsterkende læring en risikonøytral læringsstrategi, der fatale beslutninger står sentralt i læringsmålet. Slike feil kan ikke tolereres i oppdragskritiske systemer og krever sikkerhet for å forhindre skade på menneskelig og virkelig utstyr. Som et resultat er det behov for å utvikle nye opplæringsalgoritmer for å bevare sikkerheten under læring.

Til slutt bruker banebrytende forskning datakraftskrevende spill, som StarCraft II. Dette krever dyre datasystemer som ikke er allment tilgjengelige for alle forskningsinstitusjoner. Det finnes andre alternativer, men de mangler fleksibiliteten til å justere ønsket vanskelighetsgrad og datakraftskrevende kompleksitet.

Det er betydelige utfordringer å ta tak i i denne oppgaven. Oppsummert har forsterkende læring lav effektivitet, fokuserer mest på risikonøytral trening, og har begrenset tilgang til variable kontekster og testmiljø for eksperimentering. Dette etterlater ulike hull der det er betydelig rom for forbedring. For å tette disse hullene mot bedre beslutningstaking i industrilignende miljøer deler vi forskningen inn i tre separate emner:

Emne 1: Spillmiljøer for forsterkende læringsforskning med fleksible oppgaver

Emne 2: Modellbasert forsterkningslæring for mer effektiv forsterkningslæring i sanntidsstrategispill

Emne 3: Sikker forsterkningslæring for industrilignende systemer

Emne 1: Spillmiljøer: Dette forskningsarbeidet tar for seg hullet i eksperimentelle miljøer ved å foreslå seks nye spillmiljøer for å evaluere forsterkningslæringsalgoritmer. Deep Line Wars og Deep RTS er to nye sanntidsstrategispill for å teste algoritmer i planlegging og læring ved mangelfull informasjon. Deep Maze er et fleksibelt labyrintspill for å lære forsterkende læringsalgoritmer å navigere i labyrinter fra hukommelse. Deep Warehouse er et spesiallaget spill for å evaluere sikkerheten til forsterkningslæringsalgoritmer i Automated Storage and Retrieval Systems (ASRS), som er det eksklusive fokuset i dette forskningsarbeidet for industrilignende miljø. Et ASRS har autonome kjøretøy som søker å maksimere varegjennomstrømning i et tredimensjonalt rutenett. Alle spill gir parametere som justerer problemkompleksiteten og en fleksibel scenariomotor som kan utfordre algoritmer i ulike problemer, som minne og kontroll. Vi viser empirisk at disse spillene er betydelig mer datakraftseffektive enn spill med lignende kompleksitet. Mangfoldet av foreslåtte spill bidrar til å fylle kompleksitetsgapet i den vitenskapelige litteraturen. Avsluttningsvis introduserer vi Center for Artificial Intelligence and Reinforcement Learning (CaiRL) for forskning på høyytelses forsterkende læring, som samler alle miljøbidrag i et enkelt forskningsverktøy.

Emne 2: Modellbasert forsterkningslæring: Dette forskningsarbeidet foreslår modellbaserte forsterkningslæringsteknikker som fokuserer på effektivitet og sikkerhet i beslutningsprøver. Avhandlingen presenterer Dreaming Variational Autoencoder (DVAE) som lærer å etterligne hvordan spillmotordynamikken fungerer. Læringen skjer gjennom læring ved demonstrasjoner. Etter at læringsfasen er ferdig, kan tradisjonelle, ineffektive forsterkningslæringsalgoritmer trygt trene ved å bruke spilltilnærmingen ved akselererte hastigheter.

Videre presenterer denne avhandlingen Observation Reward Action Cost Learning Ensemble (ORACLE) som på samme måte lærer hvordan spillmotoren fungerer, men som kan lære mer kompleks spilldynamikk. Derfor er ORACLE mer egnet for spill med avansert grafikk som StarCraft II, men trenger å balansere treningstid og nøyaktighet.

Emne 3: Sikker forsterkningslæring: Det finnes flere metoder for å trene forsterkningslæringsalgoritmer på en sikrere måte i virkelige miljøer som krever sikkerhet. Dette eksperimentelle arbeidet viser at det er mulig å redusere feilraten under trening uten å legge urealistiske begrensninger eller forutsetninger på læringsmålene. Konkret presenterer arbeidet et rammeverk for å lære en atferdsmodell av et system. Denne modellen brukes deretter til å utføre forsterkende læringsutforskning i et fullstendig isolert læringsmiljø.

Hovedresultat: Avhandlingen bidrar med fire åpen kildekode-spill for å berike mangfoldet av tilgjengelige spill for forsterkende læringsforskning. Følgelig er det nå mer tilgjengelig for utdanningsinstitusjoner å justere problemkompleksiteten basert på tilgjengelig finansiering og beregningsressurser. Alle bidragene er samlet inn i CaiRL forskningsverktøysettet som fokuserer på å redusere kostnadene på eksperimenter, og går i retning mer effektive spill for forskning.

Avhandlingen presenterer fire algoritmer for modellbasert forsterkningslæring, hvor to av algoritmene fokuserer på å forbedre sikkerheten under læring i oppdragskritiske systemer. Avhandlingen går mot å løse noen av kjerneutfordringene ved forsterkende læring, nemlig sikkerhet og lærings-effektivitet.

Vi tror at spillmiljøene, forsterkningslæringsmetodene og studiene som presenteres i denne avhandlingen bidrar til å flytte fremmover forskningen innenfor de studerte temaene og bidrar positivt til løsninger for å muliggjøre generell bruk av forsterkende læring i spill og kritiske industrielle applikasjoner.

Per-Arne Andersen
April 8, 2022
Grimstad, Norway

List of Publications

The author of this dissertation is the first author and the principal contributor of all the included papers listed below. Papers A-M in the first set of the following list represents the main research achievements and are reproduced as Part II of this dissertation. The other papers which are listed in the second set are complementary to the primary focus.

Papers Included in the Dissertation

Paper A Andersen, P.-A., Goodwin, M., & Granmo, O.-C. (2017a). FlashRL: A Reinforcement Learning Platform for Flash Games. *Norwegian Informatics Conference*

Paper B Andersen, P.-A., Goodwin, M., & Granmo, O.-C. (2017b). Towards a Deep Reinforcement Learning Approach for Tower Line Wars. In M. Bramer & M. Petridis (Eds.), *Artificial intelligence xxxiv* (pp. 101–114). Springer International Publishing. https://doi.org/10.1007/978-3-319-71078-5_8

Paper C Andersen, P.-A., Goodwin, M., & Granmo, O.-C. (2018a). Deep RTS: A Game Environment for Deep Reinforcement Learning in Real-Time Strategy Games. *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, 1–8. https://doi.org/10.1109/CIG.2018.8490409

Paper D Andersen, P.-A., Goodwin, M., & Granmo, O.-C. (2018b). The Dreaming Variational Autoencoder for Reinforcement Learning Environments. In Max Bramer & M. Petridis (Eds.), *Artificial intelligence xxxv* (XXXV, pp. 143–155). Springer, Cham. https://doi.org/10.1007/978-3-030-04191-5_11 , **Best student paper in the technical stream.**

Paper E Andersen, P.-A., Goodwin, M., & Granmo, O.-C. (2019). Towards Model-Based Reinforcement Learning for Industry-near Environments. In M. Bramer & M. Petridis (Eds.), *Artificial intelligence xxxvi* (pp. 36–49). Springer International Publishing. https://doi.org/10.1007/978-3-030-34885-4_3

Paper F Andersen, P.-A., Goodwin, M., & Granmo, O.-C. (2020b). Increasing Sample Efficiency in Deep Reinforcement Learning using Generative Environment Modelling. *Expert Systems*. https://doi.org/10.1111/exsy.12537

Paper G Andersen, P.-A., Goodwin, M., & Granmo, O.-C. (2020e). Towards Safe Reinforcement-learning in Industrial Grid-warehousing. *Information Sciences*, *537*, 467–484. https://doi.org/10.1016/j.ins.2020.06.010

Paper H Andersen, P.-A., Goodwin, M., & Granmo, O.-C. (2020a). CostNet: An End-to-End Framework for Goal-Directed Reinforcement Learning. In *Artificial intelligence xxxvii* (pp. 94–107). Springer, Cham. https://doi.org/10.1007/978-3-030-63799-6_7

Paper I Andersen, P.-A., Goodwin, M., & Granmo, O.-C. (2020d). Safer Reinforcement Learning for Agents in Industrial Grid-Warehousing. *6th International Conference on Machine Learning, Optimization, and Data Science (LOD)*, *12566 LNCS*, 169–180. https://doi.org/10.1007/978-3-030-64580-9_14

Paper J Andersen, P.-A., Goodwin, M., & Granmo, O.-C. (2020c). Interpretable Option Discovery using Deep Q-Learning and Variational Autoencoders. *3rd International Conference on Intelligent Technologies and Applications*, *1382*, 127–138. https://doi.org/10.1007/978-3-030-71711-7_11

Paper K Andersen, P.-A., Goodwin, M., & Granmo, O.-C. (2021b). ORACLE: End-to-End Model Based Reinforcement Learning. In *Artificial intelligence xxxviii* (pp. 1–14). Springer, Cham. https://doi.org/10.1007/978-3-030-91100-3_4

Paper L Andersen, P.-A., Goodwin, M., & Granmo, O.-C. (2021a). CaiRL: A High-Performance Reinforcement Learning Environment Toolkit. *IEEE Conference on Games*, 10 , In Review.

Paper M Andersen, P.-A., Goodwin, M., & Granmo, O.-C. (2021c). Towards Safe and Sustainable Reinforcement Learning for Real-Time Strategy Games. *Artificial Intelligence* , In Review.

Other Publications Not Included in the Dissertation

Paper 14 Andersen, P.-A., Kråkevik, C., Goodwin, M., & Yazidi, A. (2016). Adaptive task assignment in online learning environments. *ACM International Conference Proceeding Series*, *13-15-June*. https://doi.org/10.1145/2912845.2912854

Paper 15 Sharma, J., Andersen, P.-A., Granmo, O.-C., & Goodwin, M. (2020). Deep Q-Learning With Q-Matrix Transfer Learning for Novel Fire Evacuation Environment. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*. https://doi.org/10.1109/tsmc.2020.2967936

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

A2C	Advantage Actor-Critic.
A3C	Asynchronous Advantage Actor-Critic.
AGC	Adaptive Gradient Clipping.
AI	Artificial Intelligence.
ALE	Atari Learning Environment.
ASRS	Automated Storage and Retrieval Systems.
AVI	Amortized Variational Inference.
BMDP	Belief Markov Decision Process.
CEM	Cross-Entropy Method.
CISR	Curriculum Induction for Safe Reinforcement Learning.
CMDP	Constrained Markov Decision Process.
CNN	Convolutional Neural Networks.
CPU	Central Processing Unit.
DARLA	DisentAngled Representation Learning Agent.
DDQN	Double Deep Q-Networks.
DL	Deep Learning.
DNN	Deep Neural Networks.
DP	Dynamic Programming.
DQN	Deep Q-Networks.
DRL	Deep Reinforcement Learning.
DT	Decision Trees.
DVAE	Dreaming Variational Autoencoder.
DVQN	Deep Variational Q-Networks.
EC	Episodic Control.
ELBO	Evidence Lower Bound.
ELF	Extensive, Lightweight, and Flexible Research Platform.
ELU	Exponential Linear Units.

ER	Experience Replay.
GAN	Generative Adversarial Networks.
GDE	Goal-Directed Exploration.
GDRL	Goal-Directed Reinforcement Learning.
GPU	Graphics Processing Unit.
GRU	Gated Recurrent Unit.
HMM	Hidden Markov Model.
HPC	High-Performance Computing.
HRL	Hierarchical Reinforcement Learning.
IAI	Industrial Artificial Intelligence.
KL	Kullback Leibler Divergence.
LGSSM	Linear Gaussian State Space Model.
LL	Latent Leap.
LQR-FLM	Fitted Linear Models.
LSTM	Long short-term memory.
LTCRNN	Liquid-Time Constant.
MBPO	Model-Based Policy Optimization.
MBRL	Model-Based Reinforcement Learning.
MC	Markov Chain.
MCM	Monte Carlo Methods.
MCTS	Monte Carlo Tree Search.
MDNRNN	Mixture Density Network Recurrent Neural Network.
MDP	Markov Decision Process.
ME-TRPO	Model-Ensemble Trust-Region Policy Optimization.
MSE	Mean Squared Error.
NDIGO	Neural Differential Information Gain Optimisation.
ORACLE	Observation Reward Action Cost Learning Ensemble.

PBO	Pixel Buffer Object.
PCA	Principal Component Analysis.
PETS	Probabilistic Ensembles with Trajectory Sampling.
PG	Policy Gradient.
PGM	Probabilistic Graphical Model.
PlaNet	Deep Planning Network.
POMDP	Partial Observable Markov Decision Process.
PPO	Proximal Policy Optimization.
ReLU	Rectified Linear Unit.
RL	Reinforcement Learning.
RNN	Recurrent Neural Networks.
RTS	Real-Time Strategy.
S-DVAE	Safe Dreaming Variational Autoencoder.
S-ORACLE	Safe ORACLE.
SAC	Soft Actor-Critic.
SARSA	State Action Reward State Action.
SC2LE	StarCraft II Learning Environment.
SIMD	Single instruction, multiple data.
SLBO	Stochastic Lower Bound Optimization.
SMDP	Semi Markov Decision Process.
SOLAR	Stochastic Optimal Control with Latent Representations.
SRSSM	Stochastic Recurrent State Space Models.
SSM	State Space Models.
SWA	Stochastic Weight Averaging.
SWAGAN	Stochastic Weight Averaged Generative Adversarial Network.
t-sne	t-distributed stochastic neighbor embedding.
TD	Temporal Difference.
TRPO	Trust Region Policy Optimization.
VAE	Variational Autoencoder.
VQ	Vector-Quantized.
VQ-VAE	Vector-quantized Variational Autoencoder.

PART I

Chapter 1

Introduction

The research on Real-Time Strategy (RTS) games has advancements over the past decade, especially for RTS games and in the industrial setting. Much research is still missing, which this thesis aims to address. There is a need for new environments towards solving superhuman performance in RTS games, and few algorithms have proven applicable for mission-critical industry applications. Reinforcement Learning (RL) algorithms need improved sample efficiency, decision safety, and interpretability for industry applications. The hope is that insights from RTS games can prove valuable for industry applications.

This chapter introduces the fundamental research questions guiding our work, thoroughly discusses the motivation, and presents intended approaches for addressing the problems. The overall goal of this thesis is to improve decision safety and sample efficiency in Model-Based Reinforcement Learning (MBRL) towards solving RTS games and industrial applications. We assert that RTS games are helpful for safety and performance evaluation in RL algorithms' and can directly map to real-world industrial applications problems. A structured summary of the work carried out during the Ph.D., and a presentation of the thesis organization concludes this chapter.

1.1 Reinforcement Learning

Our human lives comprise constant decision-making, small and large, in a world we observe as sequential. We decide to move our feet based on observations from our environment, eat when we feel hungry, and choose a life partner or career. Some decisions yield instant feedback, such as eating, while others provide reward signals after a long time, such as education. Therefore, we live in a world of sequential data, driven by constant decision-making. The universe comprises simple functions. For example, the laws of physics result in complex phenomena, such as the physical planet we inhabit, the forests we dwell in, or the natural resources we consume. In this world environment, humankind and other biological creatures strive to achieve a prolonged and healthy life by altering the future with decision-making based on input from the physical feedback functions. This unique ability to make decisions towards desired goals is essential for survival in nature and society. Consequently, our intelligence is measurable based on how accurately we can predict the causality of past decisions (Legg and Hutter, 2007). We experience the resulting cause-and-effect chain at discrete-time intervals using our sensory system and adjust future decision-making toward our projected goal through processing and reasoning (Wolpert et al., 2019). How to optimize our internal thought process more effectively towards reaching those goals is at the core of sequential decision-making and is the fundamental problem studied in reinforcement learning.

Reinforcement Learning (RL) is the branch of Artificial Intelligence (AI) that studies solutions for goal-directed sequential decision-making problems only using numerical reward signals. Agents learn through trial and error, which connect closely with the exploration-exploitation trade-off. The trade-off between obtaining new knowledge and the need to use that knowledge to improve performance is one of the most basic trade-offs, and optimal performance requires some balance between exploratory and exploitative behaviors. Work from the last decade progressively extends classical RL with Deep Learning (DL) techniques.

The combination of RL and Deep Neural Networks (DNN), Deep Reinforcement Learning (DRL), removes the limitation of RL algorithms from mainly solving simple optimization problems and enables solving a multitude of complex problems. These problems range from playing computer games from pixels, mastering the game of Go, or learning complex humanoid motion (Silver et al., 2016; Starke et al., 2021; Vinyals et al., 2019). Reinforcement Learning algorithms are either model-based or model-free. The model-based method uses a learned or known model to fuel the training of a behavior policy. A policy is the combination of parameters that define the observed behavior of an RL algorithm. Model-based RL is often more sample-efficient than model-free methods, at the cost of higher demands for computational power. This dissertation only seeks to address model-based DRL, so we use the terms RL interchangeably. DNNs comprise highly expressive non-linear models that parameterize arbitrary functions from observed data and significantly reduce the need for manual feature and policy engineering. These functions map observations of high-dimensional inputs, e.g., images, to decisions or actions. The benefits of DNN models introduce several practical problems to RL policy learning. DRL algorithms are sensitive to hyperparameters and often require comprehensive tuning to find good policies beyond local optimality.

With the increased DNN model complexity, DRL algorithms learn slower and thus have a high sample complexity. Therefore, DNN's are impractical for problems with limited data or strict limitations on climate emissions (Sutton and Barto, 2018, Chapter 11). Algorithms with low efficiency are economically costly and carry increased climate emissions which are not sustainable long term. Furthermore, traditional DRL algorithms are risk-neutral, meaning they balance the exploration-exploitation trade-off. Such trade-off does not account for uncertainty about the future, making deploying them in safety-critical real-world systems challenging. There is also a shortage of adequate, open-source environments for testing algorithms in an industrial-grade context, making it hard to evaluate algorithms affordably.

1.2 Industrial Artificial Intelligence

Industrial Artificial Intelligence (IAI) is a systematic discipline focusing on developing, validating, deploying, and maintaining AI control systems for industrial applications with sustainable performance, recently proposed by Lee, 2020; Peres et al., 2020. IAI is challenging because the infrastructure, e.g., the hardware and software, requires industrial-grade reliability. There is a very low tolerance for error, making it crucial to handle uncertainty in the best possible manner. Industry applications such as AI control of robotic arms or motorized vehicles have high economic costs, and the equipment is often heavyweight. If controlled inadequately, such AI can damage humans or equipment and cause substantial economic losses. Therefore, IAI systems should operate sustainably in terms of performance.

There are notable efforts towards improving DRL for industrial applications (Levine et al., 2016a). However, the mainstream research is far from achieving industrial-grade performance in sample-efficiency, adversarial-robustness, safety, or sustainability (Polydoros and Nalpantidis, 2017). For example, learning an RL agent to operate well in Atari 2600 games takes several days of trial and error (Mnih et al., 2015). Conversely, StarCraft II requires months of training using expensive hardware (Vinyals et al., 2019). DNN's differs from traditional machine learning in that their capacity can scale from thousands to billions of parameters. Such scaling quickly becomes a challenge for complex environments because it becomes harder to determine if agents learn the essence of a task or if it overfits the task. In both cases, an algorithm might succeed in the short term. However, the latter might fail if the environment dynamics change. The long training time is because of large state-spaces, ranging from 10^{10} states for Atari 2600 games to 10^{36000} states for StarCraft II. State-space is the space that holds all possible state compositions for a particular problem. DRL fundamentally relies on balancing the exploration-exploitation trade-off often requiring millions of timesteps to learn near-optimal behaviors.

DRL is difficult to combine with real-time real-world IAI systems because of low sample rates and the danger that equipment harms humans or the environment during trial and error. On the other hand, simulated environments can provide a near-identical environment for training, do not risk damaging equipment, and enable training faster than in the real world. Simulated environments are, for these reasons, widely used in research and pre-production configurations.[1]

[1]One such simulated environment is the Autstore Simulator, which we demonstrate in Figure 4.7.

1.3 Real-Time Strategy Games

Real-Time Strategy (RTS) games are interesting because they share many attributes with real-world scenarios and high-precision industry problems.[2] An RTS game is a multiplayer game where participants position strategic structures and control units (e.g., soldier units) to gain control of land areas on the map. A player can construct additional units and structures at the expense of resources accumulated throughout the game. Resources are spread evenly throughout the map, with possible contested resources that encourage competitive play. Therefore, RTS games provide an excellent platform for evaluating resource management, low-latency planning, and risk evaluation for RL algorithms. The challenge is that few alternatives are available for research-oriented RTS games. State-spaces are either too simplistic to resemble industry-like systems (e.g., Micro RTS (Ontanon, 2013)) or too complex to yield meaningful evaluation without extensive computational power (e.g., StarCraft II (Vinyals et al., 2019)).

Solving a complex game environment is an exciting feat because it illustrates that RL gradually matures towards the ultimate challenge of solving advanced and complex real-world problems safely (Ding and Dong, 2020). RL is still in its infancy. While many fundamental problems are solved, there is still room for improvement in sample efficiency, safety, and generalization (Arulkumaran et al., 2017). In the interim, RTS games are an appealing platform to benchmark algorithms because they share similar characteristics to real-world problems (Sethy et al., 2015). We present a list of central challenges in RL and the shared characteristics with real-world problems:

Challenge: Reward Sparsity

Real World: Decisions made towards reducing climate emissions are rewarded several years later. (Henderson et al., 2020a)

RTS Games: Immediate actions are usually rewarded when a game ends and not after each consecutive action. (Silver et al., 2021)

Challenge: Partial Observability

Real World: Autonomous driving has an unknown number of traffic participants with the uncertainty of which actions other cars will make. (Kobayashi, 2019)

RTS Games: The game is only partially observable, hidden in "*Fog of War*", meaning participants can only observe the game state in a radius of friendly units and buildings. The participants can change the environment without revealing that knowledge to the opponent. (Ontanon, 2013)

[2]We use the term industry-like, industrial applications, and industry applications interchangeably.

Challenge: High Dimensionality

Real World: Data from patients with thousands of dimensions (features) or high-resolution images from cameras. (Mahmud et al., 2018)

RTS Games: RTS games have large maps (e.g., 256x256 actionable map tiles) with the ability to have 500 units with 100 different actions at each timestep. (Vinyals et al., 2019)

Challenge: Planning over long time horizons

Real World: Autonomous driving requires planning from the start to the destination. Actions must come in strict order to prevent catastrophic outcomes. (Kusy and Zajdel, 2014)

RTS Games: To succeed in RTS games, the player must plan hundreds of steps to beat the opponent long-term, but incorrect actions are often recoverable. (Vinyals et al., 2019)

Challenge: Learning Safely

Real World: Learning to drive a car happens under the supervision of an experienced (expert) driver, and no accident must occur during the learning phase or after learning to drive the car.

RTS Games: Many RTS games have environmental damage that regresses the agent's progress. The agent must learn to avoid this damage, preferably without damage ever occurring. The agent must learn to avoid this damage, preferably without damage ever occurring. Learning to adapt to the environment can also come to the agent's advantage. For example, learning to use the environment as part of the strategy to defeat the opponent.

Following these comparisons, it falls naturally to use RTS games as a platform for researching task solvability and finding techniques applicable to safety-critical real-world applications (Levine et al., 2016b). The hope is that RL can play a substantial role in solving safety-critical real-world applications through learning problems virtually. These problems range from minor optimizations such as vehicle routing to key problems such as driving autonomously in traffic and optimizing the climate emissions of the engine exhaust system (X. Hu et al., 2019).

1.4 Motivation and Research Questions

The focus of this thesis is three-fold:

1. We aim to study the use of variational autoencoders and state-space models for learning highly expressive dynamics models in MBRL.
2. We aim to use the learned dynamics models for learning model-free RL techniques more efficiently using RTS games as the primary test bench.
3. We aim to improve the decision safety of our model-based RL approach towards better applicability in industry-like applications.

We present each research question with motivations and our scientific approach to addressing the question. We intend that the research questions overlap partially to better combine the work throughout the dissertation. We investigate the following research questions:

Question 1: *To what extent can we reduce the complexity gap for game environments in reinforcement learning research?*

Motivation: Much of current state-of-the-art research is conducted in economic or computational expensive software such as Mujoco and StarCraft II. While other alternatives exist, they have state-spaces that are either too simplistic or complex, leaving a gap in the diversity of experimental environments. This gap motivates us to create novel environments that are computationally efficient, following community standards that are particularly suited for AI research.

Approach: We approach the state-space complexity gap by proposing novel environments for RTS game research. First, we aim to study the current state-of-the-art environments and identify how to address the state-space complexity gap. Finally, we aim to collect all the environmental contributions on a highly efficient experiment execution platform, where the aim is to decrease the computational overhead of game environments. We believe that answering this question allows complex environments to execute significantly faster, reducing the climate footprint and reducing costs economically and computationally.

Question 2: *How can the sample efficiency of model-free reinforcement learning algorithms be increased to acquire good behavioral policies faster in complex RTS games?*

Motivation: Current state-of-the-art reinforcement learning often requires millions of samples and days of training before reaching a sufficient level of expertise in simple environments. For complex environments, the training time, cost, and hardware requirements increase dramatically. Due to the low sample efficiency in current algorithms, reaching adequate expertise levels becomes difficult.

Approach: Model-Based Reinforcement Learning (MBRL) algorithms are usually more sample efficient than model-free alternatives. This is because having a model of the environment allows learning behaviors using a model instead of balancing the exploration-exploitation trade-off. We propose a model-based RL scheme that learns a dynamics model from observations and can use knowledge from external actors, such as expert systems. This enables learning good behavioral policies in model-free algorithms more efficiently than traditional model-free learning methods.

Question 3: *How can RTS game environments support reinforcement learning towards the goal of real-world industrial applications?*

Motivation: RTS games share many characteristics with real-world industrial applications, such as simultaneous actions, imperfect information, stochasticity, and state-space complexity. This motivates us to study reinforcement learning for industrial AI control through the lens of RTS games because it is far simpler and more efficient to experiment in simulated environments. Second, RTS games are among the most popular game genres in recent times, which means that scientific contributions to AI control can significantly benefit the game industry.

Approach: We approach this problem using the proposed RTS game environments from research question 1 to build specialized mini-games that resemble parts of an industry-like environment. We propose a model-based approach to increase sample efficiency to learn good strategies within an acceptable timeframe on commodity hardware. Following this research approach, we supplement new challenges to the RTS games. Finally, we propose a virtual automated storage and retrieval system to evaluate the performance in industry-like applications.

Question 4: *To what extent are deep variational autoencoders effective as a mechanism to learn the dynamics of virtual environments?*

Motivation: The variational autoencoder is a powerful generative model that is now extensively used to represent high-dimensional complex data via a low-dimensional latent space. The original variational autoencoder models process input data vectors independently and learn parameters unsupervised. RTS games and industry applications often have high-dimensional complex state spaces. This motivates a combination of variational autoencoders and reinforcement learning approaches to compress the state-space to a compact representation to train RL algorithms in a model-based manner.

Approach: We apply variational autoencoders with deep reinforcement learning to propose a novel technique for model-based reinforcement learning. We demonstrate that the algorithms are effective in relevant virtual environments. We then aim to find improved methods to model time dependencies between states, specifically using recurrent neural networks. Furthermore, we test novel techniques such as stochastic weight averaging to train the proposed algorithm better.

Question 5: *How can state-space modeling combined with recurrent neural networks for planning be used to learn the dynamics of a composite game environment?*

Motivation: State-space modeling is a mathematical framework for modeling a physical system as a set of inputs, outputs, and state variables. Like variational autoencoders, the state-space model learns a set of unknown state variables representing the environment dynamics. Our preliminary work finds that state-space models exceed the expressive capabilities of variational autoencoders because they account for temporal dependencies. In combination with recurrent neural networks, it is possible that state-space modeling can support long-horizon predictions for dynamics models. This motivates us to continue research in state-space modeling towards an end-to-end solution for model-based reinforcement learning capable of encoding games with high state-space complexity.

Approach: We approach this question by reviewing prior work combining deep learning and state-space models. Following our study, we take inspiration from prior work and design a state-space model for predictions, combined with recurrent neural networks for learning temporal dependencies, using amortized variational inference to train the model.

Question 6: *How can we improve the decision safety of risk-neutral model-free reinforcement learning algorithms?*

Motivation: Recent work proposes novel innovations in risk-neutral model-free reinforcement learning algorithms. They demonstrate the ability to learn high-dimensional data combined with deep learning techniques by only following a single scalar reward value. A challenge with these algorithms is that they are not concerned with safety during learning, making them less applicable for mission-critical industry applications. This motivates us to investigate whether changing the reward signal can encourage standard model-free reinforcement learning agents to travel more safe and conservative trajectories. Furthermore, this motivates us to study if agents can act safely by learning from a dynamics model (e.g., learned model of the environment). We wish to study if model-free techniques can be easily used with dynamics modeling and learn fully offline by learning from agents that have demonstrated safe behavior. In effect, we hope to create novel model-based algorithms using existing model-free algorithms.

Approach: We address the problem by augmenting the learning techniques in our proposed model-based methods. Specifically, we investigate if it is possible to limit the optimization space using universal constraints that fit any environment. . Reinforcement learning is driven primarily by a scalar reward signal. We investigate if it is possible to shape rewards to automatically follow more safe trajectories while leading to near-global optimal policies. Specifically, we study risk-directed exploration and risk-sensitive action selection methods towards solving safe reinforcement learning.

Question 7: *How can we apply deep reinforcement learning algorithms to industry-like applications without the risk of damaging humans and real-world equipment?*

Motivation: AI systems require safety to prevent damage to humans and real-world equipment. Our opinion is that this is the most dominant challenge that separates the field of reinforcement learning and mass adoption for practical use in industrial applications. Because industrial systems tend to be expensive, it is not acceptable to deploy risk-neutral algorithms because they can cause irreversible damage to the environment. This motivates us to build an algorithm that better accounts for risk and increases decision safety at training and inference time.

Approach: We approach this question by creating training schemes that minimize the need for training in real-world systems. We use our proposed model-based reinforcement learning approach to train agents following this scheme and evaluate the failure rate of tested algorithms in industry-like environments.

Limitations: This thesis aims to advance the field of model-based reinforcement learning for RTS games and apply discovered techniques to industrial applications. However, presenting a fully functional model that is deployable in a real-world industrial environment is not within the scope of this Ph.D. work. Instead, we limit the scope to include simulated Automated Storage and Retrieval Systems (ASRS). We create several new environments, including RTS and ASRS game environments, to fill the

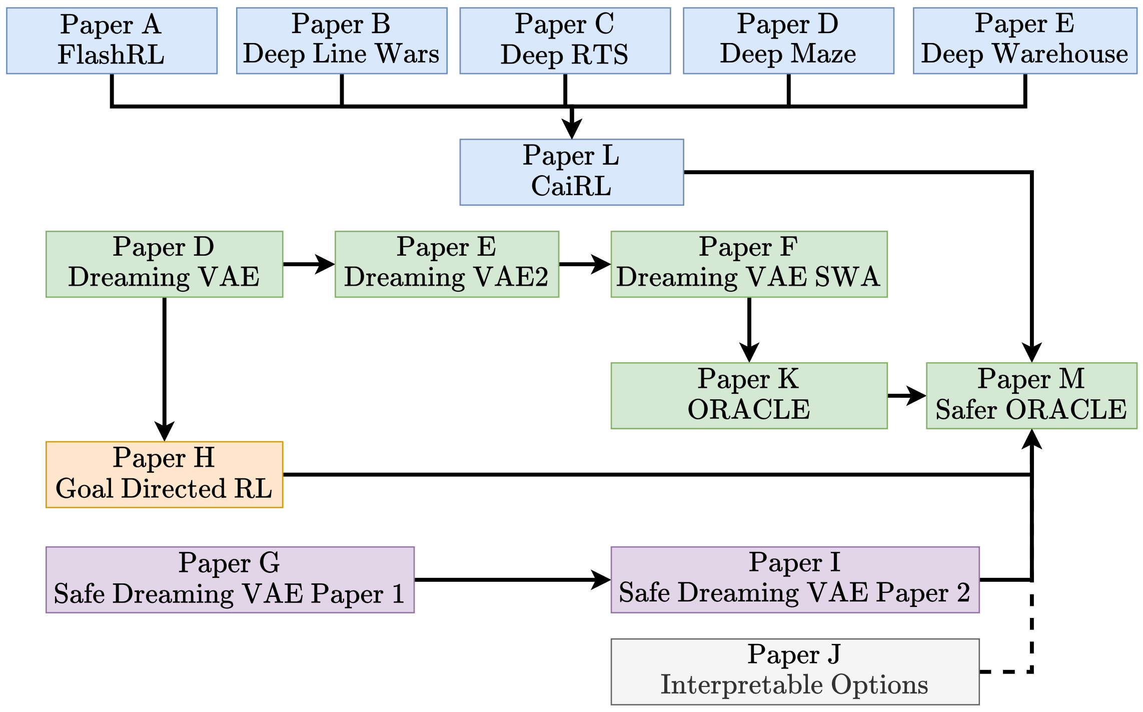


Figure 1.1: Organization of contributions. The color codes illustrate the following topics. 🟦 **Blue** illustrate contributions of novel research environments, ⬜ **Gray** is for interpretability and hierarchical RL (options). 🟪 **Purple** denotes safe reinforcement learning, and 🟧 **Yellow** represents work in goal-directed reinforcement learning. Finally, 🟩 **Green** is our main contribution in model-based and safe RL.

state-space complexity gap in the literature. Furthermore, we confine the scope to only include mini-game versions of complex environments such as StarCraft II for RTS game research because of limited hardware availability. We limit the study's central part to using the proposed RTS game environment contributions because computational efficiency allows substantially faster algorithm training. In the final version of our contributed algorithm, we present a modular algorithm that can integrate well with other domains than games. One such example is integrating the techniques in larger industrial problems or game genres.

1.5 Publications

Our contributions are in published peer-reviewed conferences and journals or submitted for peer review but waiting for feedback. Part I of the dissertation includes 15 contributions during the Ph.D. work covering four main topics. The four topics are environments, hierarchical RL, model-based RL, safe RL, and goal-directed RL. Figure 1.1 presents a structured overview of all publications and shows the progression towards our final solution. Blue color represents contributions that primarily focus on studies in understanding and designing environments suited to reinforcement learning research. The yellow color represents work on goal-directed reinforcement learning. Purple color denotes our work on safe reinforcement learning. Gray color describes our work on the interpretability of options, part of hierarchical RL research. The green color represents the work on model-based reinforcement learning and is the main focus of our research. A summary of peer-reviewed work carried out during this Ph.D. work follows.

Paper A: FlashRL: A Reinforcement Learning Platform for Flash Games

Relates to: Question 1.

This paper introduces the Flash Reinforcement Learning (FlashRL) platform, which expands the availability of novel environments for reinforcement learning research. FlashRL aims to support Flash games beyond the end-of-life of Flash in web-browsers. The lash runtime was arguably the de facto standard for graphics and animations in the first decade of 2000. The FlashRL execution platform provides a standardized API for accessing thousands of Flash games using the OpenAI Gym toolkit. This contribution enables researchers to expand the testing of reinforcement learning algorithms to novel Flash game environments, which has previously been challenging because of the gap in tools available. FlashRL demonstrates high-performance using only 5% of the total utilized CPU during training of deep reinforcement learning agents. Finally, we demonstrate promising results using reinforcement learning algorithms such as Deep Q-Networks.

Paper B: Towards a Deep Reinforcement Learning Approach for Tower Line Wars

Relates to: Question 1.

This paper proposes Deep Line Wars, a game environment between Atari 2600 and Starcraft II, mainly targeting deep reinforcement learning algorithm research. The environment is a variant of Tower Line Wars from Warcraft III, Blizzard Entertainment. As proof that the environment can harbor deep reinforcement algorithms, we propose and apply a new Deep Q-Networks architecture. The proposed architecture aims to simplify the game observation state-complexity to reduce the computational requirement compared to traditional DQN. Our experiments show that the proposed architecture can learn to play the environment well and score 33% better than standard DQN, proving the game environment's usefulness.

Paper C: Deep RTS: A Game Environment for Deep Reinforcement Learning in Real-Time Strategy Games

Relates to: Question 1.

This paper introduces the Deep RTS game environment for testing cutting-edge artificial intelligence algorithms for RTS games. Deep RTS is a high-performance RTS game made specifically for artificial intelligence research. It supports accelerated learning, meaning that it can learn at a magnitude of 50 000 times faster than existing RTS games. Deep RTS has a flexible configuration, enabling research in several RTS scenarios, including partially observable state-spaces and map complexity. We demonstrate that Deep RTS lives up to our promises by comparing its performance with Micro RTS, ELF, and StarCraft II on high-end consumer hardware. We show that Deep Q-Networks beat random-play agents. This indicates that the state representation carries meaningful information that algorithms can learn. The primary focus of this contribution is to present Deep RTS, which is publicly available at https://github.com/cair/DeepRTS.

Paper D: The Dreaming Variational Autoencoder for Reinforcement Learning Environments

Relates to: Question 1, Question 2, and Question 4.

This paper presents the Dreaming Variational Autoencoder (DVAE), a neural network-based generative modeling architecture for exploration in environments with sparse feedback. We further present Deep Maze, a novel and flexible maze engine that challenges DVAE in partial and fully-observable state-spaces, long-horizon tasks, and deterministic and stochastic problems. We show initial findings and encourage more research in reinforcement learning driven by generative exploration.

Paper E: Towards Model-Based Reinforcement Learning for Industry-near Environments

Relates to: Question 2, Question 3, and Question 4.

This paper presents the Dreaming Variational Autoencoder v2 (DVAE-2), a model-based reinforcement learning algorithm that increases sample efficiency, enabling algorithms with low sample efficiency to function better in real-world environments. We introduce the Deep Warehouse environment for industry-near testing of autonomous agents in logistic warehouses. We illustrate that the DVAE-2 algorithm improves the sample efficiency for the Deep Warehouse compared to model-free methods.

Paper F: Increasing Sample Efficiency in Deep Reinforcement Learning using Generative Environment Modelling

Relates to: Question 2, and Question 4. This article presents the Dreaming Variational Autoencoder (DVAE) with Stochastic Weight Averaging (SWA) and Generative Generative Adversarial Networks (GAN) (DVAE-SWAGAN), a neural network-based generative modeling architecture for exploration in environments with sparse feedback. We present comprehensive results using the Deep Maze environment using the DVAE-SWAGAN algorithm in partial and fully observable state-spaces, long-horizon tasks, and deterministic and stochastic problems. We compare our approach against other model-free algorithms and encourage future studies in reinforcement learning driven by generative exploration.

Paper G: Towards Safe Reinforcement-learning in Industrial Grid-warehousing

Relates to: Question 2, Question 4, Question 6, and Question 7.

This paper modifies the Dreaming Variational Autoencoder (DVAE) for safely learning good policies with a significantly lower risk of catastrophes occurring during training. The algorithm combines variational autoencoders, risk-directed exploration, and curiosity to train deep-Q Networks inside "dream" states. We introduce a novel environment, Deep Warehouse (ASRS-Lab), for research in the safe learning of autonomous vehicles in grid-based warehousing. The work shows that the proposed algorithm has better sample efficiency with similar performance to novel model-free deep reinforcement learning algorithms while maintaining safety during training.

Paper H: CostNet: An End-to-End Framework for Goal-Directed Reinforcement Learning

Relates to: Question 2, and Question 6.

This paper introduces a novel reinforcement learning algorithm for predicting the distance between two states in a Markov Decision Process. The learned distance function works as an intrinsic reward that fuels the agent's learning. Using the distance-metric as a reward, we show that the algorithm performs comparably to model-free RL while having significantly better sample-efficiently in several test environments.

Paper I: Safer Reinforcement Learning for Agents in Industrial Grid-Warehousing

Relates to: Question 4, Question 6, and Question 7.

This paper introduces advances in the novel Safer Dreaming Variational Autoencoder, combining policy constraints, external knowledge, and risk-directed exploration for learning good policies. We show that the proposed method performs comparably to model-free algorithms without safety constraints using model-based reinforcement learning. Furthermore, we empirically verify that our algorithm has a substantially lower risk of entering catastrophic states.

Paper J: Interpretable Option Discovery using Deep Q-Learning and Variational Autoencoders

Relates to: Question 4.

This paper presents the Deep Variational Q-Networks (DVQN), which combines deep generative- and reinforcement learning. The algorithm finds good policies from a Gaussian distributed latent space, particularly useful for defining options. The DVQN algorithm uses MSE and KL-divergence regularization, combined with traditional Q-Learning updates. The algorithm learns a latent space representing good policies with state clusters for options. We show that the DVQN algorithm is a promising approach for identifying initiation and termination conditions for option-based reinforcement learning. Experiments show that the DVQN algorithm, with automatic initiation and termination, has comparable performance to DQN (Rainbow) and can maintain stability when trained for extended periods after convergence.

Paper K: ORACLE: End-to-End Model Based Reinforcement Learning

Relates to: Question 2, and Question 5.

This paper proposes an end-to-end model-based reinforcement learning algorithm, the Observation Reward Action Cost Learning Ensemble (ORACLE), for learning model-free algorithms to act in environments without trial and error in the real environment. This method is beneficial for existing installations that employ existing decision-making systems, such as an expert system.

The proposed algorithm has the same fundamental learning principles as the Dreaming Variational Autoencoder but differs substantially architecturally. The model uses principles from state-space modeling, recurrent neural networks, and Bayesian Deep Learning to create a highly expressive model for learning industry environments. We show that the algorithm is more sample efficient and performs comparably with existing model-free approaches. We also demonstrate how the algorithm is actor agnostic, enabling existing model-free algorithms to operate in a model-based context.

Paper L: CaiRL: A High-Performance Reinforcement Learning Environment Toolkit

Relates to: Question 1.

This paper addresses the need for a platform that efficiently provides a framework for running RL experiments. We propose CaiRL Environment Toolkit as an efficient, compatible, and more sustainable alternative for training learning agents and detail recommendations on developing efficient simulations.

There is an increasing focus on developing sustainable artificial intelligence. However, there is little effort in current literature to improve the environmental efficiency for running simulation environments. The most popular development toolkit for reinforcement learning, OpenAI Gym, is built using Python, a powerful but slow programming language. To overcome the slowness of Python, we propose a platform on C++ that gives the same flexibility but at magnitudes faster speeds.

CaiRL also presents the first reinforcement learning Toolkit with a built-in Adobe Flash emulator for running legacy Flash games for reinforcement learning research. We empirically demonstrate that CaiRL performs significantly better through a thorough comparison of the classic control domains. Furthermore, we illustrate that CaiRL is fully compatible with OpenAI Gym for running reinforcement learning experiments.

Paper M: Towards Safe and Sustainable Reinforcement Learning for Real-Time Strategy Games

Relates to: Question 2, Question 3, Question 5, Question 6, and Question 7.

This article presents a novel model-based DRL approach for tackling complex environments to reduce the need for failures during training. Specifically, our approach demonstrates successful learning while still considering robust safety awareness, minimizing risk, and reducing computational costs compared to model-free RL methods.

We empirically verify the Safe ORACLE (S-ORACLE) in multiple complex and stochastic game environments such as Deep RTS, ELF: MiniRTS, Micro RTS, Deep Warehouse, and StarCraft II. We show that our algorithm performs better than state-of-the-art model-free and model-based approaches in the tested environments.

1.6 Thesis Outline

This dissertation composes of two parts. Part I provides an overview of the work carried out throughout the Ph.D. study. Part II includes the publications and in-review articles representing this thesis's main contribution, seen in the list of contributions. The remainder of this dissertation is structured as follows.

Chapter 2: Background presents the background literature of the techniques used in this thesis. This includes Markov Decision Processes, Reinforcement Learning, Safe Reinforcement Learning, and various Deep Learning modeling techniques.

Chapter 3: Literature Review describes a thorough literature review of scientific advances in reinforcement learning that motivates and inspires our contributions. The keywords for our study are model-based, safety, environments, goal-directed RL, interpretable RL, and, hierarchical RL.

Chapter 4: Software Contributions and Evaluation describes our scientific software contributions to novel reinforcement learning environments. We present new environments to fill the state complexity gap in current state-of-the-art and discuss our motivations, design specifications and provide baseline results and evaluations.

Chapter 5: Algorithm Contributions presents our main contributions of novel techniques for safe model-based reinforcement learning in RTS games towards a functional industry-grade reinforcement learning solution. Specifically, we present the motivation for carrying out the work, and describe the details of our algorithms. We provide the hyperparameters that led to the best results in our experiments, and summarize the algorithm contributions.

Chapter 6: Contribution Evaluation empirically evaluates our algorithm contributions using the proposed software contributions, including state-of-the-art environments in the reinforcement learning literature. Each section presents a hypothesis that we aim to address in the experiments and evaluations

Chapter 7: Conclusion and Future Work concludes Part I of this thesis and discusses the final achievements of having carried out the Ph.D. work. Finally, we outline future research directions that could potentially improve the work presented in this thesis.

Part II presents the publications produced during the Ph.D. work in their entirety. There are thirteen publications labeled from Paper A to Paper M. The papers are listed in chronological order, roughly representing the flow of this thesis. See Figure 1.1 for a detailed illustration of the research progression.

Chapter 2

Background

The fundamentals of reinforcement learning include Markov decision processes, model-free reinforcement learning algorithms such as tabular methods, deep learning variants, and model-based reinforcement learning. Extensions towards better safety include uncertainty identification, constrained Markov decision processes, and risk-based objectives. Furthermore, Bayesian and variational Bayesian methods can learn expressive models for model-based reinforcement learning and interpretable RL.

This chapter thoroughly describes the background and preliminary information required to understand the contributions of this thesis. We introduce the fundamentals of reinforcement learning, including Markov decision processes, model-free reinforcement learning algorithms such as tabular methods, deep learning variants, and model-based reinforcement learning. We then dive into the techniques of safe reinforcement learning and describe core modeling techniques used in this thesis. These techniques include state-space models, recurrent neural networks, and autoencoders. Finally, we describe topics this work has not completed but has built momentum for future work.

2.1 Markov Decision Processes

The essence of reinforcement learning is teaching machine algorithms to make a sequence of decisions in a dynamic system. Markov Decision Process (MDP) is a mathematical framework that defines a class of stochastic sequential decision processes and is RL algorithms' fundamental building block. This thesis focus on games with a finite number of states and actions. Therefore, we consider finite MDP's. Figure 2.1 visualizes the MDP framework, a discrete-time sequential process of making decisions then observing o_t with its corresponding reward r_t. The observation is either fully observable such that $o_t = s_t$ or partially observable such that $o_t \neq s_t$. (Monahan, 1982)

Definition 1 *An MDP model is expressed as a tuple* $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ *where* $\mathcal{S}$ *is the state-space,* $\mathcal{A}$ *is the action-space available to the agent at every timestep,* $\mathcal{P} : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ *is the transition function,* $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ *is the reward function, and* $\gamma \to [0, 1]$ *is the discount-factor (Puterman, 1990).*

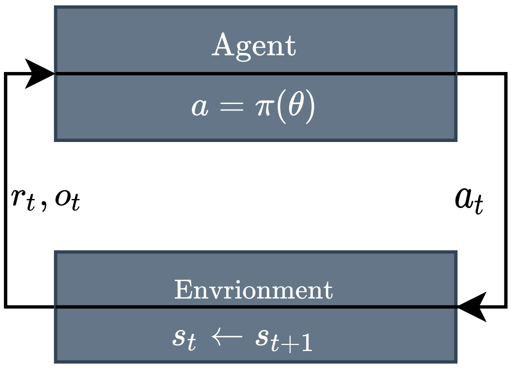


Figure 2.1: The agent-environment synergy in an MDP (2018). The agent makes actions in the environment, which triggers a transition to the next state. The agent observes the new state with the corresponding reward signal.

The transition function $\mathcal{P}$ describes the probability of transitioning from a particular state s_t to s_{t+1} with the corresponding reward r after taking action a. The transition function contains all information about the MDP,

$$\begin{aligned} \mathcal{P}(s_{t+1}, r|s_t, a_t) &= Pr\left[\mathcal{S}_{t+1} = s_{t+1}, \mathcal{R}_{t+1}|\mathcal{S}_t = s_t, \mathcal{A}_t = a_t\right] \\ &= \sum_{s_{t+1}\in\mathcal{S}} \sum_{r_t\in\mathcal{R}} p(s_{t+1}, r_t|s_t, a_t) = 1, \end{aligned} \tag{2.1}$$

where $\mathcal{P}$ is defined for the next state s_{t+1}, $\forall s \in \mathcal{S}$, and $\forall a \in \mathcal{A}(s)$. From the four-argument transition function in Equation 2.1, we can derive a state-transition function $\mathcal{T} : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ and a reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ for all state-action pairs. The state-transition function is defined,

$$\begin{aligned} \mathcal{T}(s_{t+1}|s_t, a_t) &= Pr\left[\mathcal{S}_t = s_{t+1}|\mathcal{S}_{t-1} = s, \mathcal{A}_{t-1} = a\right] \\ &= \sum_{r\in\mathcal{R}} \mathcal{P}(s_{t+1}, r|s_t, a_t), \end{aligned} \tag{2.2}$$

where the probability of entering the next state s_{t+1} is dependent on the current state s_t and the taken action a_t. The reward function is defined,

$$\begin{aligned} \mathcal{R}(s, a) &= \mathbb{E}\left[\mathcal{R}_{t+1}|\mathcal{S}_t = s, \mathcal{A}_t = a_t\right] \\ &= \sum_{r_t\in\mathcal{R}} r \sum_{s_{t+1}\in\mathcal{S}} \mathcal{P}(s_{t+1}, r_t|s_t, a_t), \end{aligned} \tag{2.3}$$

where $\mathcal{R}$ is the expected reward that the agent receives after making action a_t, triggering transition to state s_{t+1}. An MDP has the Markov property if the next state and expected next reward only depends on the current state and action. The Markov property exists for a MDP only if,

$$\begin{aligned} &Pr\left[\mathcal{S}_{t+1} = s_{t+1}, \mathcal{R}_{t+1} = r_t|s_t, a_t, \dots s_0, a_0\right] \\ &\quad = Pr\left[\mathcal{S}_{t+1} = s_{t+1}, \mathcal{R}_{t+1} = r_t|s_t, a_t\right] = 1. \end{aligned}$$

For complex game environments or real-world applications, the Markov property rarely holds because fully observable data of the dynamics system are hidden ($o_t \neq s_t$). It is challenging to characterize the next state without the Markov property unless it is possible to observe the current state in its entirety. (Hausknecht and Stone, 2015).

2.1.1 Partially Observable Markov Decision Processes

In complex games such as Starcraft II and Dota 2, only partial state observations are accessible (Berner et al., 2019; Vinyals et al., 2019). The agent must exploit partial observations to construct beliefs of the true state. Partial Observable Markov Decision Process (POMDP) is a generalization of MDPs that accounts for partial observability of states. Therefore, it is widely used in literature to formalize optimization problems such as games and industry applications. POMDP is defined as a tuple $\mathcal{M}_{\mathcal{POMDP}} = \langle \mathcal{S}, \mathcal{A}, \Omega, \mathcal{P}, \mathcal{R}, \mathcal{O}, \gamma \rangle$ of three sets, three functions, and one constant. The definition is similar to regular MDP's. However, it includes a set of observations $\Omega = \{o_1, o_2, \dots, o_n\}$ and the agent perception model $O : \mathcal{S} \times \mathcal{A} \rightarrow \Pi(\Omega)$ where $\Pi(\Omega)$ represent the probability distribution on Ω (Littman et al., 1995). Figure 2.2 illustrates a POMDP where the agent can only take actions a_t dependent on observations o_t. The problem with observations is that they may not capture the necessary information to succeed in the environment. Let us consider some examples.

Example 1 A game of pong where the observation is a singular pixel image.

Example 2 A game of pong where the x-axis at position 20%-80% is masked (black area without vision).

In the first example, the agent cannot determine the direction or velocity of the ball. For the second example, the agent cannot fully determine the ball position at the 20%-80% range of the x-axis, making it nearly impossible to learn how the ball moves. To alleviate these problems, the agent can make decisions based on a history of observations. At state s_{t+n}, one can think of this as having prior knowledge. The problem is that storing prior observations in a buffer becomes infeasible for games with near-infinite states, making this approach inadequate for complex games.

2.1.2 Belief Markov Decision Processes

Another approach to capture information from history is to encode observations into a belief state using belief MDPs. A belief state b is a summarization of previous observations into a probability distribution over all states $s \in \mathcal{S}$, where $b(s_t) = Pr(s_t | o_{1 \dots t})$, represents the probability that the environment is in state s_t (Rodriguez et al., 2000). Given that we

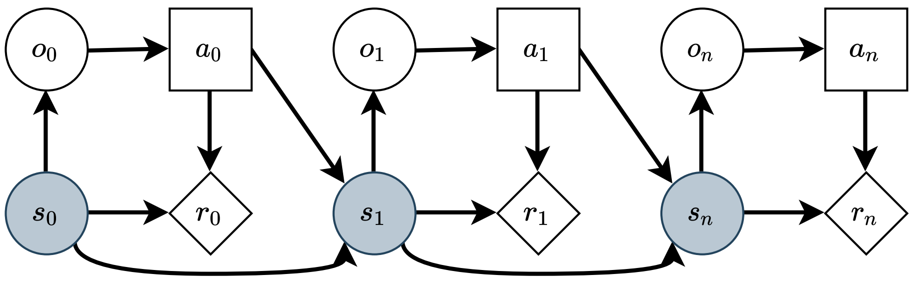


Figure 2.2: A POMDP system. o, a, s, r denotes observations, actions, states, and rewards, respectively. The agent can only observe $o_0 \ldots o_n$ after making an action where the underlying state $s_t \ldots s_n$ is hidden from the agent.

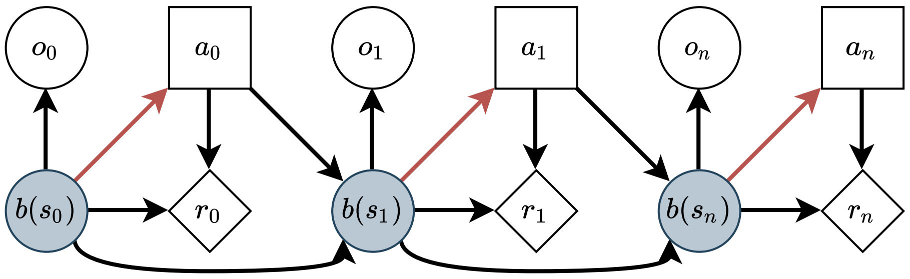


Figure 2.3: A Belief MDP. b, a, s, r denotes belief-state distributions, actions, states, and rewards, respectively. In contrast to POMDP's, the underlying state s_t is replaced with a belief state distribution. States sampled from the belief state distribution are used to make decisions in the MDP, which has similar traits to fully observable MDP's.

have an initial belief state b_0, we can compute belief states,

$$\begin{aligned}
&b_{t+1}(s_{t+1}) = \mathcal{P}(s_{t+1}|o_t, a_t, b_t) \\
&\propto \mathcal{P}(o_t|s_{t+1}, a_t, b_t)P(s_{t+1}|a_t, b_t) \\
&\propto O(o_t|s_{t+1}, a_t)\mathcal{P}(s_{t+1}|a_t, b_t) \\
&\propto O(o_t|s_{t+1}, a_t)\sum_{s_t \in \mathcal{S}} \mathcal{P}(s_{t+1}|a_t, b_t, s_t)\mathcal{P}(s_t|a_t, b_t) \\
&\propto \underbrace{O(o_t|s_{t+1}, a_t)}_{\text{Observation Model}}\sum_{s_t \in \mathcal{S}} \underbrace{\mathcal{T}(s_{t+1}|s_t, a_t)}_{\text{State-Transition}}\underbrace{b_t(s_t)}_{\text{belief}},
\end{aligned} \tag{2.4}$$

which is a sufficient statistic for $b_t \equiv o_{1 \ldots t}$. The reward function $\mathcal{R}(s_t, a_t)$ requires hidden-state information, but we can introduce belief states such that $\mathcal{R}(b_t, a_t) = \sum_{s \in \mathcal{S}} b_t(s_t)\mathcal{R}(s_t, a_t)$. The belief state distribution b_t can memorize historical observations. We assume that this is enough information to represent the hidden-state s_t to treat the POMDP similarly to fully-observable MDP's, as seen in Figure 2.3. (Littman et al., 1995; Xiang and Foo, 2021)

2.2 Reinforcement Learning

Analogous to human intelligence, a behavior policy is the brains of an algorithm and expresses parameters for a function that aims to behave optimally in a given problem. This section explains traditional RL methods such as tabular and function approximation methods. These methods work well in many games or real-world applications but require substantial data sampling to learn good behaviors. As the state-space complexity increases in environments, training time increases exponentially, one of the major constraints of state-of-the-art reinforcement learning. (Sutton and Barto, 2018)

The ultimate goal of a reinforcement learning agent is to find the optimal policy π^*. A policy represents a mapping from state observations to action probabilities $\pi(a_t|s_t)$. RL algorithms categorize into three learning types; value-based, policy-based, and a combination called actor-critic-based algorithms. Our work utilizes all three categories, with an additional focus on value-based methods. Value-based methods aim to learn a scalar quantification of goodness between states and actions. We can indirectly infer an optimal policy by learning an optimal value function. Policy-based or direct policy search learn parameters for a policy function. Conversely to value-based methods, policy-based methods learn the policy direcly, fueled using a feedback function. Actor-critic algorithms combine the merits of both approaches by alternating between steps to estimate the value function and policy gradient updates.

There are also variations of on-policy and off-policy where off-policy algorithms can learn using historical data and data samples from other policies. Lastly, there are model-based and model-free algorithms where model-based algorithms learn using a predicted model (unknown, but learned from samples) or a known environment model. In contrast, model-free algorithms learn solely by trial and error in an unknown environment. RL algorithms primarily frame the learning problem as an MDP and usually have an update procedure as follows:

1. Read current observation s_t
2. Make a decision based on observation $\pi(a_t|s_t)$
3. Receive reward r_t
4. Update policy estimates with a learning algorithm. Go back to step 1.

The optimal policy π^* is the policy in policy-space Π that maximize the return,

$$\pi^* = \arg\max_{\pi \in \Pi} \mathbb{E}\left[G|\pi\right] \tag{2.5}$$

where,

$$\begin{aligned} G_t &= \mathcal{R}_{t+1} + \gamma \mathcal{R}_{t+2} + \gamma^2 \mathcal{R}_{t+3} \\ &= \sum_{k=0}^{\infty} \gamma^k \mathcal{R}_{t+k+1}, \end{aligned} \tag{2.6}$$

is the cumulative discounted return. The discount factor $0 \leq \gamma \leq 1$ quantifies the importance between immediate and distant rewards, where $\gamma = 0$ considers only immediate rewards. $\gamma = 1$ weight immediate and distant rewards equally (2018). Perhaps the most central equation in RL is the Bellman Expectation Equations,

$$\begin{aligned} v_\pi(s_t) &= \mathbb{E}_\pi \left[G_t | \mathcal{S}_t = s \right] \\ &= \mathbb{E}_\pi \left[\mathcal{R}_{t+1} + \gamma v_\pi(\mathcal{S}_{t+1}) | \mathcal{S}_t = s) \right] \end{aligned} \tag{2.7}$$

where $v_\pi(s_t)$ quantifies how good it is for the agent to be in state s_t following policy π (Kiumarsi et al., 2018). The Bellman Equation, famously from dynamic programming, defines a recursive function that expresses a relationship between the value of a state and the successor state (Bellman, 1952; Sutton and Barto, 2018). However, the state-value function is not practical when quantifying how good actions are in state s_t. This is because the state-value function is a sum of all action values. The state-action value function $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ quantifies how good action a_t is in state s_t. The state-action value function,

$$\begin{aligned} &Q_\pi(s_t, a_t) \\ &= \mathbb{E}_\pi \left[\mathcal{R}_{t+1} + \gamma v(\mathcal{S}_{t+1}) | \mathcal{S}_t = s, \mathcal{A}_t = a \right] \\ &= \mathbb{E}[\mathcal{R}_{t+1} + \gamma \mathbb{E}_{a \sim \pi} Q(\mathcal{S}_{t+1}, a) \mid \mathcal{S}_t = s, \mathcal{A}_t = a], \end{aligned} \tag{2.8}$$

is similar to the state-value function $Q_\pi(s_t, a_t)$ but decomposes the value-function into values for individual actions. (Kaelbling et al., 1996). Consequently, we find the optimal policy π^* indirectly through $Q^*(s_t, a_t)$ or $V^*(s_t)$ functions, or in other words, solving the Bellman Equation[1] (Arulkumaran et al., 2017),

$$Q^*(s, a) = \mathbb{E}[\mathcal{R}_{t+1} + \gamma \max_{a'} Q^*(\mathcal{S}_{t+1}, a_t)].$$

2.2.1 Q-Learning

The most central algorithm for fundamental RL is the Q-Learning algorithm (Watkins and Dayan, 1992). Q-Learning is a value-based algorithm and uses the Q-function from Equation 2.8 as the basis for deriving the policy. Q-Learning is a model-free algorithm. Such algorithms work independently of the underlying MDP dynamics $\mathcal{T}$ and are off-policy. Off-policy describes the learning methodology, in this case, that it can learn from

[1] RL theory has significantly more ground to cover, but we have left out non-essential parts for this dissertation. We recommend (Van Otterlo and Wiering, 2012) for further reading.

samples collected by other policies, such as a random policy. The Q-Learning algorithm follows the Bellman equations (Equation 2.8), where

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\delta_t, \tag{2.9}$$

where $0 \leq \alpha \leq 1.0$ is the learning rate, and δ_t is the Bellman residual,

$$\delta_t = \mathcal{R}_{t+1} + \gamma \underbrace{\max_{a\in\mathcal{A}} Q(s_{t+1}, a_t)}_{\text{off-policy}} - Q(s_t, a_t). \tag{2.10}$$

The Bellman residual δ_t denotes the Temporal Difference (TD) error between current and subsequent state estimates. The name TD derives from its use of changes, or differences, in predictions over successive time steps to drive the learning process. This procedure is named bootstrapping, e.g., we update our estimates of $Q(s_t, a_t)$ with another estimation $\max_{a\in\mathcal{A}} Q(s_{t+1}, a)$. Generally, Q-Learning assumes taking the best action greedily (max-operator) given current knowledge (Jaakkola et al., 1994). The max operator assumes that all future actions are optimal, allowing other policies to make decisions off-policy. It is also possible to derive another popular algorithm, State Action Reward State Action (SARSA), by omitting the max operator such that the Bellman residual,

$$\delta_t^{SARSA} = \mathcal{R}_{t+1} + \gamma \underbrace{Q(s_{t+1}, a_{t+1})}_{\text{on-policy}} - Q(s_t, a_t),$$

is an on-policy TD update instead. The Q-Values are usually stored in a table. From a computer perspective, such tables consume memory, usually between 32 and 64 bits per table cell. Therefore, it becomes infeasible to use traditional RL in larger problems because the table would exhaust the available system memory. For this reason, we use function approximators to learn the latent representation (function), that best matches an exact solution for the state-action value table.

2.2.2 Deep Q-Networks

Since traditional RL relies on tables to store parameters, it becomes infeasable to assume an exact function to solve our policy optimization problem (Sutton and Barto, 2018). Instead, function approximation and specifically using neural networks is an appealing approach. This is because they demonstrate the capability to learn high-dimensional functions (Mnih et al., 2015). DQN tries to estimate the Q-Table so that $Q(s_t, a_t; \theta) \sim Q^*(s_t, a_t)$. Learning of the parameters θ is done through minimizing the following loss-objective,

$$\mathcal{L}(\theta_i) = \mathbb{E}_{s_t,a_t,r_t,s_{t+1}\sim\mathcal{P}(.)}\left[(\delta_i|\theta_i)^2\right], \tag{2.11}$$

where δ_i is the Bellman residual from Equation 2.10. The first term is a reward, the second term is the greedy estimation from the target Q-network, and the third term is the inference Q-network. The reason why two Q-networks are used is because of the max operation combined with bootstrapping. Since Q-Learning assumes that all actions from s_t and onwards are optimal, bootstrapping overestimates the Q-values and will as $s_t \to \infty$ become significantly overestimated. This problem is called the maximization bias. Therefore, using independent estimators, we can unbiased Q-value estimates of the actions selected using the opposite estimator. We can thus avoid maximization bias by disentangling our updates from biased estimates.

2.2.3 Policy Gradient Algorithms

In contrast to value-based methods, Policy Gradient (PG) algorithms aim to find an optimal behavior policy through direct policy search. The policy is defined as a parametrized function w.r.t θ and computes gradients based on an objective function,

$$\begin{aligned} J(\theta) &= \sum_{s_t \in \mathcal{S}} d_{\pi_\theta}(s_t) v_{\pi_\theta}(s_t) \\ &= \sum_{s_t \in \mathcal{S}} d_{\pi_\theta}(s_t) \sum_{a_t \in \mathcal{A}} \pi_\theta(a_t|s_t) Q_{\pi_\theta}(s_t, a_t) \\ &\propto \mathbb{E}_{s_t,a_t,r_t,s_{t+1} \sim \mathcal{P}(.)} \left[\ln \pi_\theta(a_t|s_t) Q_{\pi_\theta}(s_t, a_t) \right], \end{aligned} \tag{2.12}$$

where $d_{\pi_\theta}(s_t)$ denotes the stationary distribution for π_θ. (Sutton, 2019). Gradient updates are performed using *gradient ascent* as we wish to find parameters θ for π_θ that yields the highest return, written as $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$. The gradient theorem (Sutton and Barto, 2018) finds that the gradient w.r.t θ of the objective function $J(\theta)$ is,

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[Q_{\pi_\theta}(s_t, a_t) \nabla_\theta \ln \pi_\theta(a_t|s_t)].$$

$Q_{\pi_\theta}(s_t, a_t)$ is interchangeable with any return-based function (e.g., advantage function). $\mathbb{E}_{\pi_\theta}$ indicates the empirical average over a finite set of samples, sampled using the algorithm (Schulman et al., 2016). Strictly speaking, this means that policy gradient algorithms are traditionally on-policy. Many algorithms build on the policy gradient framework. Perhaps most notable is the Proximal Policy Optimization (PPO) algorithm for its substantial empirical performance and simplicity (Schulman et al., 2017). PG algorithms are notoriously difficult to train because estimates have high variance and no bias. Vanilla PG is perhaps most known for this behavior, making the algorithm more susceptible to local optima and performing poorly across larger state spaces. We define the ratio $ro_t(\theta)$ between current policy $\pi_\theta(a_t|s_t)$ and previous policy $\pi_{\theta_{old}}(a_t|s_t)$ such that $ro_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}}$. If we substitute $\ln \pi_\theta(a_t|s_t)$ in Equation 2.12, the objective becomes,

$$J^{CPI}(\theta) = \mathbb{E}_{s_t,a_t,r_t,s_{t+1} \sim \mathcal{P}(.)} \left[ro(\theta) Q_{\pi_\theta}(s_t, a_t) \right],$$

which is the Trust Region Policy Optimization (TRPO) objective in Schulman et al., 2015. The problem with J^{CPI} is that maximization leads to excessively large policy updates making learning unstable. The work on PPO proposes a clipping scheme to reduce the size of policy updates,

$$J^{CLIP}(\theta) = \mathbb{E}_{s_t,a_t,r_t,s_{t+1}\sim\mathcal{P}(.)}\left[\min(ro(\theta)Q_{\pi_\theta}(s_t,a_t), \text{clip}(ro(\theta), 1-\epsilon, 1+\epsilon)Q_{\pi_\theta}(s_t,a_t))\right],$$

where ϵ is a hyperparameter $0 \le \epsilon \le 1$, typically set in the range of $\epsilon \sim 0.2$.

2.2.4 Model-Based Reinforcement Learning

Model-Based Reinforcement Learning (MBRL) follows the standard MDP derivation. It involves learning the transition-function $\mathcal{T}$ from observed data (Moerland et al., 2020). The goal is to find some parameters θ_m so that the estimated transition function $\hat{\mathcal{P}}(s_{t+1}, r_t|s_t, a_t; \theta_m) \cong \mathcal{P}(s_{t+1}, r_t|s_t, a_t)$. In this work, we learn to derive the reward-function $\hat{\mathcal{R}} : \mathcal{S} \times \mathcal{A}$ and state-transition function $\hat{\mathcal{T}} : \mathcal{S} \times \mathcal{A}$ because we aim to quantify uncertainties in reward function estimates for risk-sensitive RL. This is further detailed in Section 2.3.1. While it is common to incorporate decision-making into the model with algorithms such as Cross-Entropy Method (CEM), our methods are model agnostic, enabling the training of model-free algorithms (e.g., Q-Learning) using an estimated environment model. Model-free approaches are more studied and have significantly better reward performance than model-based approaches (Vinyals et al., 2019). Unlike model-free algorithms, model-based algorithms are usually more sample efficient (Sutton, 1990). In combination with a learned model of the environment, the aim is to combine the best of three worlds: sample efficiency, reward performance, and risk-awareness incorporated.

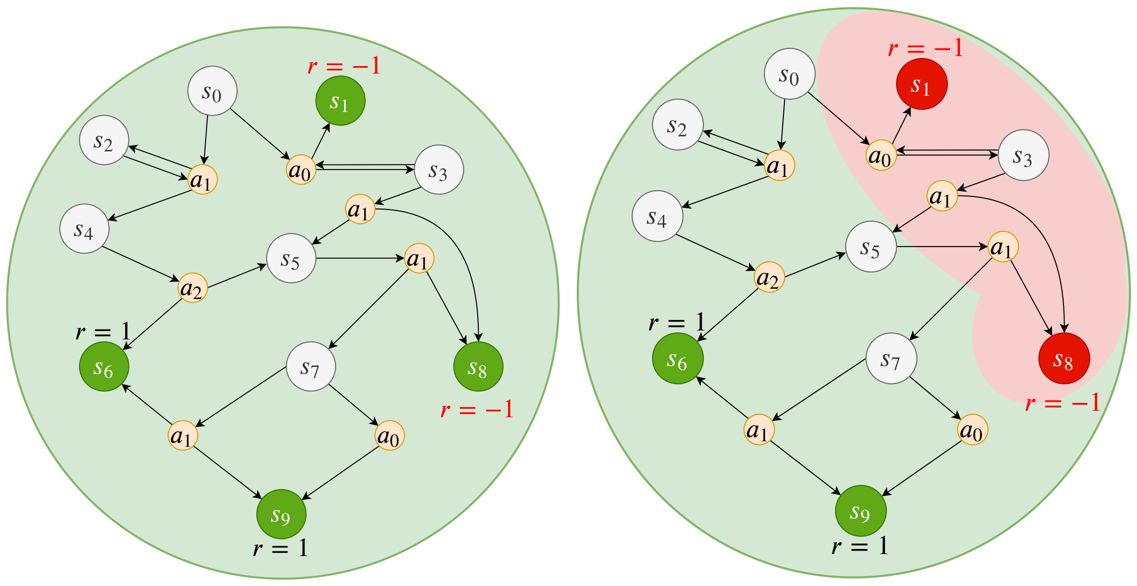
The goal of an estimated environment model $\hat{\mathcal{T}} : \mathcal{S} \times \mathcal{A}$ is to learn parameters θ_m that best can reflect the behavior and characteristics of the unknown dynamics $\mathcal{T}$ from Equation 2.2. The estimated dynamics model is with this referred to as a dynamics model and is defined,

$$\hat{\mathcal{T}}(\hat{s}_{t+1}|\hat{s}_t, a_t; \theta_m) \cong \mathcal{T}(s_{t+1}|s_t, a_t), \tag{2.13}$$

where we assume that the estimated model is in some way captures information of the unknown MDP similar to Hidden Markov Model (HMM)'s (Eddy, 2004). There are many approaches to learning such models. However, we focus on Variational Autoencoder (VAE)'s (Kingma and Welling, 2013), state-space models (Kalman, 1960), and recurrent neural networks (Hochreiter and Schmidhuber, 1997).

2.3 Safe Reinforcement Learning

Traditional RL learns the optimal policy according to an *optimization criterion*. This optimization criterion varies with algorithms but is commonly implemented to minimize time



(a) Illustration of unsafe MDP state-space

(b) Illustration of safety-constrained MDP state-space.

Figure 2.4: Illustration of an MDP where actions are made according to a policy π. The green nodes demonstrate safe terminal states in the state-space, gray is safe, neutral states, red nodes are catastrophic state, and orange nodes are the transition given an action. The left figure follows traditional RL optimization where trial and error occurs to recognize bad states. The right figure illustrates a policy that has constraints and knowledge for actions leading to states with negative feedback (red area).

or maximize reward. The *return maximization criterion* is frequently used in Q-Learning, seen in Equation 2.9. This objective back propagates Q-estimates of the following state to the former state.

It becomes evident that there is no safety guarantee in the traditional view of reinforcement learning (Berkenkamp et al., 2017). The primary focus is for the agent to find the policy that maximizes some feedback signal. Through Dynamic Programming (DP), Monte Carlo Methods (MCM), or Temporal Difference (TD), find a way to learn by trial and error. Trial and error are insufficient for mission-critical systems, because there are usually no room for error. Therefore, it is natural to seek an alternative path towards learning good policies while reducing the number of visited catastrophic states.

Figure 2.4 illustrates a deterministic MDP in the view of a traditional RL agent (Figure 2.4a) and an agent that is safety-aware (Figure 2.4b). The MDP considers state-space $\mathcal{S} = \{s_0 \dots s_9\}$ and an action-space $\mathcal{A} = \{a_0 \dots a_2\}$ controlled using policy $\pi(a_t|s_t)$, with the probability of transitioning to the next state $\mathcal{P}(s_{t+1}|s_t, a_t)$. The traditional model-free RL agent must explore by trial and error to learn a policy that would keep a distance from catastrophic states. This implies that the agent would with high probability, take action a_0 in state s_0 and enter state s_1 leading to a catastrophic terminal outcome. Such outcome in a real-world system could potentially damage humans or equipment. The motivation for a safer learning system becomes evident. The idea is to find a method to map knowledge of good (green) and bad (red) state-space regions prior to exploration.

2.3.1 Uncertainty

Most algorithms are considered risk-neutral from a traditional RL view because they rely on balancing exploration and exploitation until finding a good strategy. As clearly seen in the update equation for Q-Learning (Equation 2.8) and the deep learning version for DQN (Equation 2.11). Traditional RL algorithms learn by maximizing rewards (Equation 2.5) that are not necessarily designed to guide agents safely to goal states. The problem with risk-neutrality is that safety-critical systems such as the Deep Warehouse simulator (Described in Section 4.4), traditional algorithms fail to learn without relying on experience from catastrophic states (Paper G).

One approach towards safe RL is to quantify the *epistemic* or *aleatoric* uncertainty and define it as a *risk* signal. *Aleatoric uncertainty* stems from the observations done of the environment. An environment might have some inherent noise and stochasticity that is not controllable by the agent, which further amplifies belief MDP (however, the belief distribution is epistemic uncertainty) and POMDP's because of added uncertainty beyond the observable state-space. Aleatoric uncertainty can be thought of as not changeable but quantifiable. Conversely, *epistemic uncertainty* is the model uncertainty, or in other words, the uncertainty of whether model predictions are correct with current knowledge. A model can quantify (learn) epistemic model uncertainty, which correlates tightly with the model's overall prediction performance. (Kendall and Gal, 2017). In a stochastic MDP, several measurable uncertainties exist, such as model, reward, and value uncertainty (Bagnell et al., 2001).

Model uncertainty is the uncertainty of the transition function $\mathcal{P}$ in an MDP. It is successfully used in literature to guide agents safely during learning (Kearns and Singh, 2002) in known MDP models. This work considers MDP's where the transition function (environment model) is unknown. For this reason, we must estimate the model, which adds additional model uncertainty.

Reward uncertainty is the uncertainty of receiving a specific reward r_t at state s_t given the action made. The intrinsic motivation from Schmidhuber, 2010 describes an auxiliary reward signal from the extrinsic reward function $\mathcal{R}$. Work from Pathak et al., 2017 similarly proposes a dynamics-based prediction error used as a secondary reward signal.

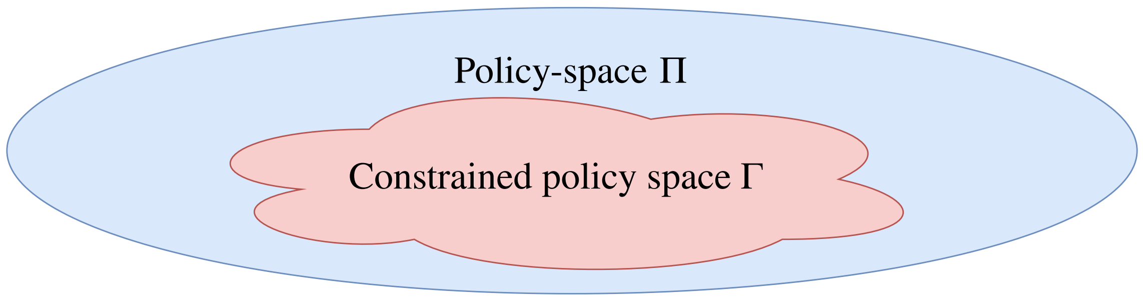


Figure 2.5: The policy-space (blue) Π and the subset of policies (red) $\Gamma \subseteq \Pi$, where each policy $\pi \in \Gamma$ must satisfy the constraints $i_i \in I$.

Value uncertainty considers the value function $v_\pi(s)$ as a source of risk. Due to the recursive nature of the Bellman equations (Equation 2.7), the value-function estimates increasingly accumulate errors over time $t \to \infty$.

2.3.2 Constrained Markov Decision Processes

Constrained Markov Decision Process (CMDP) extends the original MDP framework from Altman, 1999 that replaces the conventional risk-neutral conditional expectation of cumulative reward objectives with a notion of risk measure.

Risk is a function that quantifies the danger or uncertainty of making an action following policy $\pi(a_t|s_t)$ (Heger, 1994). It is founded on the uncertainty associated with future events. It is inevitable since the consequences of actions are unknown when an action is made in systems with unknown dynamics (Shen et al., 2013). Numerous methods quantify risks, such as *Risk-Sensitive Criterion* (Geibel and Wysotzki, 2005), *Worst Case Criterion* (Gaskett, 2003), and *Constrained Criterion* (Moldovan and Abbeel, 2012) where this work focus on the latter. A policy that disregards risk evaluation is *risk-neutral*, and the learning objective is to maximize the expectation of returns,

$$\max_{\pi\in\Pi} \mathbb{E}_\pi(G) = \max_{\pi\in\Pi} \mathbb{E}_\pi(\sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}),$$

which express the same objective as in Equation 2.6. For algorithms that follow this objective, it is clear that the algorithm is risk-neutral by design because safe trajectories are seldom the same trajectory as maximizing rewards. This motivates adjustments to the objective function to make the policy more risk-aware when maximizing the return.

The Constrained Criterion is an appealing extension to the standard MDP framework, described as the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma, I)$. I is a set of constraints applied to the MDP. Such constraints guarantee that learning policies are constrained to a subset of the full state-space, typically used to reduce or eliminate unsafe policies, as illustrated in Figure 2.4. The most promising work in CMDP's and safe RL is the work of Berkenkamp et al.,

2017. The general form of the constrained criterion is defined,

$$\max_{\pi \in \Pi} \mathbb{E}_\pi(G) \;\; \textit{subject to} \;\; i_i \in I, i_i = \{h_i \gtreqless \alpha_i\} \tag{2.14}$$

where i_i is the n_{th} constraint in the set I that must be satisfied in the MDP. Additionally, h_i is a function related to the return G, an upper or lower bound to the threshold value α_i. Consider all constraints satisfied, then the policy-space is reduced to a subset $\Gamma \subseteq \Pi$, and the policy exists only within this subset $\pi \in \Gamma$. The idea is that the constraints lead to a smaller policy-space, where it is more likely that a safe solution is found, seen in Figure 2.5. Given that the algorithm selects policies only from the *safe* subset Γ, the objective function can be written as

$$\max_{\pi \in \Gamma} E_\pi(G) \tag{2.15}$$

which is the standard definition of expected return from Equation 2.6, but w.r.t to a subset of safe policies Γ.

Constraint selection is a delicate user-defined process of designing signals using data from the MDP, which largely depends on the specific problem (2017; Geibel and Wysotzki, 2005). Mainstream literature uses Lyapunov and Barrier functions which guarantees safety analytically within a subset of the policy search space (Chow et al., 2018). The problem with these methods is that they are highly dependent on model assumptions which are usually infeasible to achieve for complex real-world systems. Section 3.2.1 details state-of-the-art literature on Lyapunov and barrier functions.

2.3.3 Risk-directed Exploration

Following Edith et al., 2005, risk-directed exploration is an auxiliary risk-driven feedback signal that guides action selection in RL algorithms. The exploration-risk function $\Psi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$,

$$\Psi(s_t, a_t) = w\mathrm{H} - (1 - w)\frac{\mathbb{E}[\mathcal{R}(s_t, a_t)]}{\max_{a_t \in A} |\mathbb{E}[\mathcal{R}(s_t, a_t)]|}, \tag{2.16}$$

where H is the entropy function,

$$\mathrm{H}(s_t, a_t) = -\hat{\mathcal{T}}(s_{t+1}|s_t, a_t) \log \hat{\mathcal{T}}(s_{t+1}|s_t, a_t). \tag{2.17}$$

The entropy H of a stochastic process is a measured uncertainty that suits well for quantifying risk. The risk is denoted Ψ and, as seen in Equation 2.16, is used as a trade-off between normalized expected return and system entropy (Yang and Qiu, 2005a). The risk is weighted $0 \leq w \leq 1$ where $w \geq .5$ appreciate the stochastic nature of the system more. Furthermore, we define a utility function $\mathcal{U} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$,

$$\mathcal{U}(s_t, a_t) = \rho(1 - \Psi(s_t, a_t)) + (1 - \rho)\pi(a_t|s_t), \tag{2.18}$$

where $0 \leq \rho \leq 1$ controls risk-awareness of the agent. For $\rho = 0$, the algorithm behaves risk-neutral. As $\rho \to 1$, the agent becomes increasingly aware of risks in the decision-making process. Note that the utility function $\mathcal{U}$ is policy agnostic and works with policy and value based algorithms (e.g., PG and DQN). Moreover, value-based methods (e.g., Q-Learning), are usable with sampling techniques such as ϵ-greedy or Boltzmann distributions (softmax). (Edith et al., 2005; Tijsma et al., 2017)

2.3.4 Risk-sensitive Reinforcement Learning

Risk-sensitive RL refers to the branch of safe RL, which expresses the balance of weighted risk metric and the return function,

$$\max_{\pi \in \Pi}(\mathbb{E}_{\pi}(\mathcal{R}(s_t, a_t)) - \beta\omega). \tag{2.19}$$

The first term in Equation 2.19 is the expectation of returns (Equation 2.3). The second term is the weighted $0 \leq \beta \leq 1$ risk-function $\omega : \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ (note that this is omega ω not to be confused with w in Equation 2.16) (Geibel and Wysotzki, 2005). Literature has studied different definitions of risk, such as using uncertainty from the TD-Error (Campos and Langlois, 2003; Mihatsch and Neuneier, 2002), reward uncertainty (Gosavi, 2009), and using a set of error-states (Geibel and Wysotzki, 2005). This work uses two sources of uncertainty in the system, particularly the dynamics model entropy and the variance in a set of predicted rewards, similar to Gosavi, 2009. Like Romoff et al., 2018, we consider a function approximator to learn a reward signal without inducing additional noise because model predictions are already noisy. Unique to recent literature, we combine risk-sensitive RL and risk-directed exploration (Equation 2.16), adjusting for explorative safety long term, inspired by Edith et al., 2005, and short term through risk-using weighting of the return function.

2.3.5 Goal-directed Reinforcement Learning

Goal-Directed Reinforcement Learning (GDRL) is not directly a technique for reducing risk. However, it has appealing properties that reduce the probability of entering catastrophic states. Therefore, works in the literature adopt GDRL as a method towards risk reduction (del R. Millán, 1995). GDRL separates the learning into two phases. Phase one aims to solve the Goal-Directed Exploration (GDE) problem (Smirnov et al., 1996). To solve the GDE problem, agents must determine at least one viable path from the initial state to a goal state. In phase two, agents use the learned path to find a near-optimal path. The two phases iterate until the agent policy converges (Debnath et al., 2018). The modeling task is to compute costs using neural network approximators that follow an *episodic* training scheme. During exploration, the algorithm records a buffer of visited states. When the agent enters a terminal state, states in the buffer are labeled with

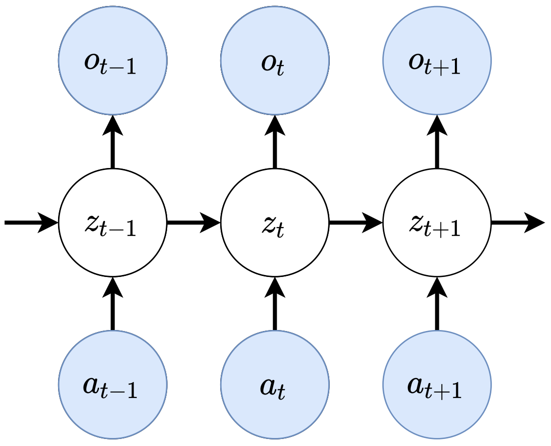

$$\begin{tikzcd}
o_{t-1} & o_t & o_{t+1} & \\
z_{t-1} \arrow[u] \arrow[r] & z_t \arrow[u] \arrow[r] & z_{t+1} \arrow[u] \arrow[r] & {} \\
a_{t-1} \arrow[u] & a_t \arrow[u] & a_{t+1} \arrow[u] &
\end{tikzcd}$$

Figure 2.6: A graphical representation (PGM) of a state-space model. o_t is observations, z_t is latent variables (e.g., hidden system dynamics), and a_t denotes a function or variable that influences the system's transition.

the corresponding Euclidean distance from the goal. The training is a supervised learning problem but requires a sufficient set of historical data for the estimator to accurately predict distances between the current state and the terminal state.

2.4 Machine Learning Models

This section presents machine learning models central to reinforcement learning and prominent to the success of our contributions. Variational mnference and Probabilistic models have shown outstanding performance in computer vision. Algorithms based on these principles offer incredible capabilities to learn expressive models. Therefore, they are also an appealing choice for model-based RL, used to learn a dynamics model from collecting data during exploration. Reinforcement learning heavily relies on techniques from other branches of machine learning research. For example, the most significant discovery in recent times is using deep learning models in place of exact tabular solutions, which immensely increase expressive capabilities. These techniques span several disciplines, such as State Space Models (SSM) from control theory, Recurrent Neural Networks (RNN), Variational Autoencoder (VAE), and Vector-quantized Variational Autoencoder (VQ-VAE) which are deep variational Bayesian methods. [2]

2.4.1 State Space Models

State Space Models (SSM) is particularly interesting because of its remarkable history of predicting system dynamics and future events. One such example is trajectory estima-

[2]We refer the reader to Kingma and Welling, 2019 for a detailed introduction to variational inference and probabilistic models.

tion in the aerospace industry. At the Apollo 11 project, SSM's contributed to land Neil Armstrong and Buzz Aldrin on the moon in the late '60s (McGee and Schmidt, 1985) using the work of Kalman, 1960. Since then, state space methods have been applied in many subjects, including economics, finance, political science, environmental science, road safety, and medicine (Commandeur et al., 2011). Figure 2.6 illustrates a graphical representation of an SSM.

SSM's are defined for a given sequence of observations $o_{1:T} = [o_1, \dots x_T]$ that depends on input variables. This dissertation inputs actions from RL agents $a_{1:T} = [a_1, \dots, a_T]$. At each time step, the latent-state variable z_t encapsulates all past information up to the present. The states in an SSM form a Markov Chain (MC). They are highly similar to the assumptions in MDPs, such as having the Markov property. SSMs demonstrate excellent capabilities for expressive non-linear systems and system identification (Schön et al., 2011). There are three types of SSMs. *Filtering* for noise reduction of present state using previous states, *smoothing* for estimating previous states using future states, and *prediction* to estimate future states using previous states.

Smoothing computes the posterior $p_\theta(z_t|o_{1:T}, a_{1:T})$ using past, present, and future information to predict the latent variable. Because smoothing uses future information, it is limited to offline learning but performs better due to additional data. It is possible to redefine smoothing for a fixed-lag interval k such that $p_\theta(z_t|o_{1:t+k}, a_{1:t+k})$. It is applicable in an online setting when the system allows for a k timesteps delay.

Filtering computes the posterior using past information $p_\theta(z_t|o_{1:t}, a_{1:t})$ to retrieve the present latent-state. Filtering is widely used in literature because of its ability to work online. Perhaps the most widely known model is the Linear Gaussian State Space Model (LGSSM) (Kalman Filter) from (Kalman, 1960).

Prediction is for computing the future posterior latent-state variable $p_\theta(z_{t+k}|o_{1:t}, a_{1:t+k})$. Prediction is the primary focus of this thesis. It is particularly interesting because it allows predicting future events based on historic state and action information.

Using a neural network function approximator, we model the SSM problem as a non-linear stochastic process. We take inspiration from the seminal work of Kingma and Welling, 2013 using amortized variational inference techniques, similar to the efforts of Fraccaro, 2018. The goal is to design the function $p_\theta(z_t|b_{t-1}, a_{t-1})$ to predict future latent vectors where b_{t-1} is the belief state function and a_{t-1} is the RL agent's action at state s_t. Generally, one can almost think of SSM's as a VAE, but significantly better capabilities to learn temporal state depedencies. In practice, the temporal dependencies are learned using recurrent neural networks, and therefore, the SSM method functions similarly to VAEs.

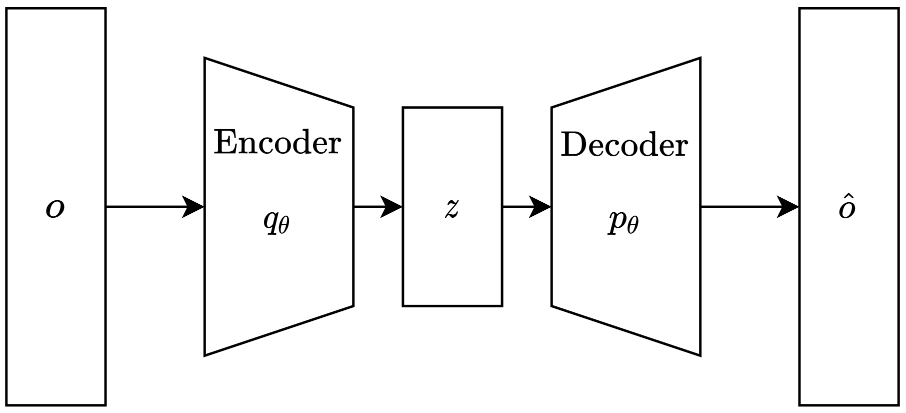


Figure 2.7: A traditional autoencoder. The encoder q_θ takes an observation o as input and computes it to a compact latent space variable (latent representation). Consequently, the decoder can decode the latent space variables where the goal is to decompress the data so that the original input is retrieved $\hat{o}_t$. Several training methods exist for deep autoencoders, but the most common method is using mean-squared error between the observed datapoint o_t and the decompressed output $\hat{o}_t$.

2.4.2 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are interesting because they aim to learn data dependencies stretched through time. This work tests several RNN methods such as Liquid-Time Constant (LTCRNN) networks from Hasani et al., 2021 and Gated Recurrent Unit (GRU) from Cho et al., 2014. However, we found Long short-term memory (LSTM) networks from Hochreiter and Schmidhuber, 1997 to perform better in most tested environments. Following Belief Markov Decision Process (BMDP)'s theory, it is possible to learn a belief distribution with sufficient observations statistics $b_t \equiv o_{1...t}$. The LSTM layer is part of the generative network to predict present belief states b_t based on the previous belief state b_{t-1} and action a_{t-1}. However, they only use the notation z_t after adding stochasticity to the LSTM prediction.

2.4.3 Deep Autoencoders

Autoencoders are commonly used in supervised learning to encode arbitrary input to a compact representation and use a decoder to reconstruct the original data from the encoding. The purpose of autoencoders is to store redundant data into a densely packed vector form. In its simplest form, an autoencoder consists of a feed-forward neural network where the input and output layer has equal neuron capacity and the hidden layer is smaller and used to compress the data, seen in Figure 2.7. This is also commonly called the "bottleneck". Such autoencoder can be defined as: $q_\theta : O \rightarrow Z$, $p_\theta : Z \rightarrow O$, where $q_\theta, p_\theta : \arg\min_{q_\theta, p_\theta} ||O - (q_\theta \times p_\theta)\hat{O}||^2$. In this notation, O defines the input data-space, Z,

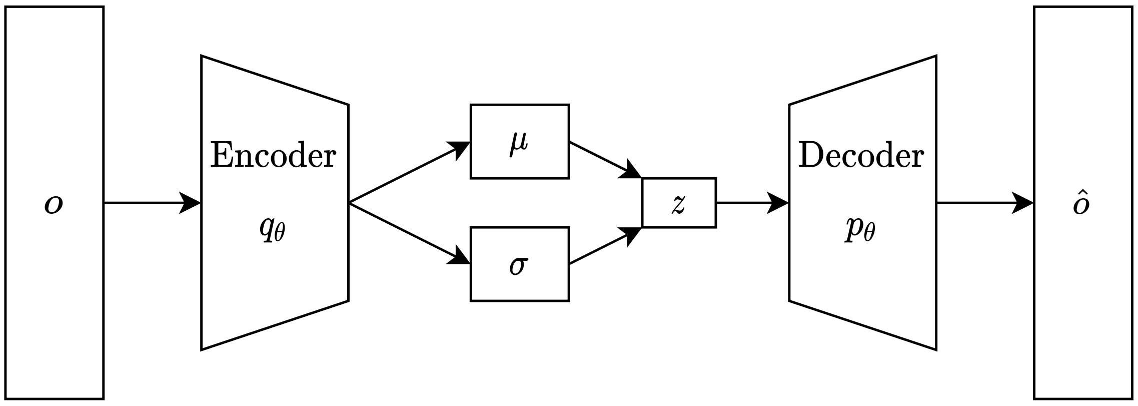


Figure 2.8: A convolutional variational autoencoder. Convolutions extract features from input pixels o_t and parametrize a Gaussian distribution by splitting the last layer of the encoder to mean μ and standard deviation σ. This allows using the reparameterization trick, which allows gradients to flow through the network for optimization. We sample the latent space variables z_t from the Gaussian such that $z_t \sim \mathcal{N}(\mu, \sigma; \theta)$ and decodes the latent variables using the decoder p_θ. The decoder output $\hat{o}_t$ is a generative sample variation of the input o_t. The network is trained with variational inference, specifically using a reconstruction loss (mean-squared error) and a Kullback Leibler Divergence (KL) as the regularization term).

the latent space, and $\hat{O}$ as the reconstructed data space. Traditional autoencoders are not generative, meaning that they cannot learn to interpolate between observed samples in a partial state-space. This reason motivates looking into alternative methods for generating novel samples from only a fraction of the data space. (Baldi, 2012)

2.4.4 Variational Autoencoders

The Variational Autoencoder (VAE) is a generative autoencoder that uses neural networks to parametrize probability distributions to define a probabilistic latent variable model efficiently. Figure 2.8 illustrates the VAE architecture where an observation o_t encodes to μ and σ that is used to sample Gaussians with the reparametrization trick. z_t is the sampled latent space variables that decode to the predicted observation $\hat{o}_t$. VAE's define a generative model and an inference network used to learn the parameters that best fit observed data. The generative model is the joint probability distribution $pr_\theta(x_t, z_t) = pr_\theta(x_t|z_t)pr_\theta(z_t)$ where $pr_\theta(z_t) = \mathcal{N}(z_t; 0, I)$ and the decoder is usually $pr_\theta(x_t|z_t) = \mathcal{N}(x_t; \mu, \sigma)$ with μ and σ being neural network estimators (Kingma and Welling, 2013). The inference network or the encoder allows computing a posterior approximation given a particular data point (e.g., an observation of a game). In particular, we use Amortized Variational Inference (AVI) that shares parameters over all observed data points (2013). The posterior approximation is defined as $p(z_t|x_t) = \mathcal{N}(z_t; \mu, \sigma)$ following the same principles as for the decoder. Parameter learning is performed using Evidence Lower Bound (ELBO), which consists of a reconstruction and regularization term (Equation 5.9 and Equation 5.12. The reconstruction term optimizes the style of the output (e.g., correct color and shapes), and the regularization term aims to model the data similarly to the prior (e.g., a Gaussian distribution). The network trains using ELBO because $pr_\theta(z_t|x_t) = \int_{z_t} \frac{pr_\theta(x_t|z_t)pr_\theta(z_t)}{pr_\theta(x_t)}dz$ is intractable (Zhang, Butepage, et al., 2019). Intuitively, the aim is to create a generative model that can reconstruct belief data over unseen data points given the best estimates of data seen thus far. This fits well with a model in RL but requires adjustments to account for temporal dimensions. Motivated by the challenges of posterior collapse in VAE, Van Den Oord et al., 2017 propose a categorical generative network using vector quantization and demonstrates superior encoding performance on several datasets.

2.4.5 Vector Quantized-Variational Autoencoder

Vector-quantized Variational Autoencoder (VQ-VAE)s is a type of variational autoencoder that uses vector quantization to obtain a discrete latent representation, in contrast to the continuous latent representation of traditional VAE's. VQ-VAE is different in that the encoder outputs are discrete. Additionally, the prior is learned in contrast to being a static distribution in VAE's. The algorithm incorporates ideas from vector quantization which forms its name VQ-VAE. The primary motivation behind this algorithm is that posterior collapse is circumvented, which is a significant problem in traditional VAE's. VQ-VAE uses a discrete codebook component, a list of vectors corresponding to an index. The codebook is used to quantize the bottleneck part (vector) of the autoencoder, and the encoder network output is used to lookup the codebook vector that is closest in Euclidean distance,

$$z_q(o_t) = \arg\min_i ||z_e(o_t) - e_i||_2 \tag{2.20}$$

where $z_e(o_t)$ is the encoder vector for an observation x_t, e_i, is the codebook vector at index i, and $z_q(o_t)$ is the quantized vector reconstructed by the decoder. (2017)

2.5 Interpretable and Option-based RL

This study envisions an algorithm that humans can interpret while simultaneously learn a multi-level policy for solving many different problems. For example, learning an algorithm to play StarCraft will hopefully allow other long-term planning problems to reuse the learned hierarchical behavioral policy.

Although interpretability and option-based RL are not this dissertation's primary focus, it is nevertheless central to model-based DRL, because (1) it is crucial to understand models in real-world environments (trust), and (2) it is potentially better to separate behavioral policies as options for multi-objective planning tasks. We present Deep Variational Q-Networks (DVQN) in Section 5.2, which aim to interpret latent spaces in reinforcement learning agents. As a result, we propose a new approach towards automatic options discovery in hierarchical reinforcement learning and the options framework.

Interpretable Reinforcement Learning

A longstanding problem of reinforcement learning (and machine learning) is explaining and interpreting the decision processes. With the broad adoption of deep learning techniques, explainability has become even more central, as it is nearly impossible to understand the influence of each parameter on the model. One may ask why it is vital to interpret these RL models. The most dominating response is the need to trust models for deployment in production environments. There are two prevalent methods in interpretable AI, specifically intrinsic explainability, where the model is inherently explainable. One such example is Decision Trees (DT). The second interpretability method is post-hoc, assuming an external algorithm that analyzes the decision-maker model and interprets the observed behavior or parameters. Interpretations are either local or global, where local interpretations interpret parts of the model. In contrast, a global interpretation can interpret the whole model.

Because the primary focus is to solve safe RL in RTS games, the work on interpretability is limited to post-hoc methods with local interpretability. Specifically, this thesis investigates the interpretation of game environment state spaces using dimension reduction techniques, such as VAE and VQ-VAE, similar to the work from Zahavy et al., 2016. More literature can be found in the detailed survey on interpretable RL from Puiutta and Veith, 2020.

Hierarchical Reinforcement Learning with Options

The options framework from Sutton et al., 1999 is a multi-policy scheme that functions through temporal abstractions in MDP's. In a regular MDP, time is perceived as a fixed sequence (e.g., every timestep is a fixed step of 0.5 seconds) where a policy operates over the MDP as a whole. The options framework formalizes the MDP to include a sequence of actions by an option-conditioned policy with a termination condition. An MDP has a set of options that induces a hierarchy that models the temporal abstraction (e.g., different time scales or timespans per option) and constitutes a Semi Markov Decision Process (SMDP). Therefore, SMDP's is the extension that allows the time between actions to be variable instead of fixed.

A typical SMDP defines actions at different levels of abstractions, and an excellent example of such MDP is robotic control problems. For example, the robot has to perform n_0 action every t_0 milliseconds for controlling legs to walk using policy π_0. However, for making hand gestures, the action-space may have n_1 actions requiring decision evaluation every t_1 milliseconds using π_1. This gives us two different policies (or options) that function over the same MDP but at different levels of abstractions.

This work does not create a hierarchical solution but creates momentum for such work in the future. Specifically using VAE's to interpret and cluster parts of the state-space, we show that it is possible to classify a different set of required behaviors, as seen in Section 6.3.

2.6 Summary and Overview

This chapter presents an overview of relevant topics for this Ph.D. dissertation in model-based reinforcement learning, model-free reinforcement learning, and safe-reinforcement learning as the major topics. We also present other less prevalent topics in our research, specifically interpretability and hierarchical reinforcement learning. We connect the present background to our contributions in Table 2.1.

Table 2.1: An detailed overview connecting the background to our proposed methods. The topic column describes the particular technique. The DVAE, ORACLE, and DVQN columns describe if the technique is used in the algorithm.

Topic	**DVAE**	**ORACLE**	**DVQN**
MDP’s	X	X	X
POMDP’s	X	X	X
Q-Learning	X	X	X
Policy Gradients	X	X	-
Model-Based RL	X	X	-
Uncertainty Learning	X	X	-
CMDP’s	X	-	-
Risk-Directed RL	X	X	-
Risk-Sensitive RL	X	X	-
Goal-Directed RL	X	X	-
State Space Models	-	X	-
Recurrent NN’s	X	X	-
Autoencoders	X	-	X
Variational Inference	X	X	X
Vector Quantization	-	X	-
Interpretable RL	-	-	X
Hierarchical RL	-	-	-

Chapter 3

Literature Review

A central part of research is learning from other researchers' discoveries and expanding on the body of knowledge. Scientific achievements are usually articles presenting reviews of related work that lead to novel discoveries, justification, and conclusions. This chapter presents the related literature to work carried out during this Ph.D. The chapter includes related work on model-based reinforcement learning, safe reinforcement learning, goal-directed reinforcement learning, interpretable reinforcement learning, hierarchical reinforcement learning, and a study of existing reinforcement learning environments.

3.1 Model-Based Reinforcement Learning

Recent literature shows that Model-based RL is becoming the frontier with several new algorithms that outperform model-free variants with a large margin. The most recent achievement is discrete world models with DreamerV2 from Hafner et al., 2021. DreamerV2 is the first reinforcement learning agent that achieves human-level performance on the Atari benchmark by learning behaviors fully offline in a world model. Prior work in Hafner et al., 2020 uses a similar architecture to Paper D by deriving latent dynamics that form estimations of future observations given an action. More recently, Ozair et al., 2021 proposed Vector-quantized Variational Autoencoder (VQ-VAE) for planning in RL. The authors use a stochastic variant of the Monte Carlo Tree Search (MCTS) algorithm to plan the agent's actions and the discrete latent variables representing the system's dynamics model. This approach shows state-of-the-art results in chess and illustrates that the approach scales to DeepMind Lab, a first-person 3D environment with complex visual state observations with only partial observability.

The Model-Ensemble Trust-Region Policy Optimization (ME-TRPO), formally proposed by Kurutach et al., 2018, is a Dyna-based algorithm for learning a dynamics model. The ME-TRPO method uses an ensemble of neural networks to form the dynamics model, which significantly reduces model bias, increasing its generalization abilities. The ensemble individually trains using single-step L2 loss in a supervised setting. After training ME-TRPO, the authors use Trust Region Policy Optimization (TRPO) from Schulman et al., 2015 as the model-free approach. The work shows significantly faster convergence in several continuous control tasks.

Luo et al., 2018 extend the ME-TRPO algorithm with Stochastic Lower Bound Optimization (SLBO). In comparison, it modifies the single-step L2 loss to multi-step L2-norm loss to the train ensemble dynamics model. The authors present a mathematical framework for the guaranteed monotonic improvement of the dynamics model. This framework is a meta-algorithm that iteratively builds a lower bound of the expected reward based on the learned dynamics model and sample trajectories. The authors then show that they can jointly maximize the lower bound over the dynamics model policy. The empirical results show that SLBO outperforms state-of-the-art algorithms in Swimmer, Half-Cheetah, Ant, Walker, and Humanoid environment in the Mujoco physics engine.

In Janner et al., 2019, the authors analyze previous methods and their capability to generalize well for longer time horizons. Their analysis suggests that reconstruction performance is sufficient for shorter time horizons but exponentially decreases for more extended rollouts due to uncertainty. The proposed algorithm is called Model-Based Policy Optimization (MBPO) and balances a trade-off between sample efficiency and performance. The authors suggest a prediction horizon between 1-15 states, up to 200 states. In conclusion, MBPO shows that model-based approaches can outperform state-of-the-art model-free reinforcement learning when tuned appropriately.

Deep Planning Network (PlaNet) from Hafner et al., 2019 is a model-based agent that interprets a state's pixels to learn the dynamics of an environment. The model learns a high-dimensional function that encodes observations and environment dynamics to latent variables. The agent samples actions based on the learned latent variables. The proposed algorithm showed significantly better sample efficiency than algorithms such as A3C and D4PG. The authors also show that their algorithm can learn multiple tasks using a single agent policy with a substantially faster convergence rate.

Chua et al., 2018 propose Probabilistic Ensembles with Trajectory Sampling (PETS). The algorithm uses an ensemble of bootstrap neural networks to learn a dynamics model of the environment over future states. The algorithm then uses this model to predict the best action for future states. The authors show that the algorithm significantly lowers sampling requirements for environments such as Half-Cheetah in Mujoco, compared to Soft Actor-Critic (SAC) and PPO.

Bangaru et al., 2016 proposed a method of deducing the MDP by introducing an adaptive exploration signal (pseudo-reward), obtained with a deep generative model using variational autoencoders. Their approach was to compute the Jacobian of each state and use it as the pseudo-reward when using deep neural networks to learn the state-generalization. They present two algorithm variations, MTRL-0, and MTRL-α, respectively, without and with the Jacobian pseudo-reward. The authors demonstrate that their method outperforms value-iteration. Furthermore, their dynamics model can accurately predict state observations in navigation tasks.

Xiao and Kesineni, 2016 proposed using Generative Adversarial Networks (GAN) for model-based RL. The goal was to utilize GAN for learning environment dynamics in a short-horizon timespan and combine this with the strength of far-horizon value iteration RL algorithms. The GAN architecture proposed illustrated near authentic generated images giving comparable results to Mnih et al., 2015. Azizzadenesheli et al., 2018 similarly verified the usefulness of GAN for model-based RL with theoretical analysis. Their method, Generative Adversarial Tree Search (GATS), combines GAN's and MCTS, demonstrating significant bias reduction in Q estimates. Furthermore, their method substantially reduces sample complexity by a factor of 200% for DQN.

Higgins, Pal, et al., 2017 proposed DisentAngled Representation Learning Agent (DARLA), an architecture for modeling the environment using β-VAE (Higgins, Matthey, et al., 2017). The trained model was used to learn the optimal policy of the environment using algorithms such as DQN (Mnih et al., 2015), A3C (Mnih et al., 2016), and Episodic Control (EC) (Blundell et al., 2016). DARLA is the first algorithm to introduce learning without access to the ground-truth environment during training to the best of our knowledge.

Buesing et al., 2018 evaluate the environment modeling capabilities in auto-regressive (AR) and state-space models (SSM). The models, deterministic SSM (dSSM-DET), stochastic SSM with VAE (dSSM-VAE), stochastic SSM (sSSM), and recurrent AR (RAR), is tested in four games using the Atari Learning Environment (ALE), namely BOWLING, CENTIPEDE, MS_PACMAN, and SURROUND. The results suggest that it is better to model the state-space than to utilize Monte-Carlo rollouts. Specifically, state-space models are significantly faster and produce more accurate results than auto-regressive methods.

Stochastic Optimal Control with Latent Representations (SOLAR) is an algorithm that learns the dynamics of an environment by exploiting the knowledge from a reinforcement learning policy. This enables the algorithm to learn local models used in policy learning for complex systems. SOLAR is built around a Probabilistic Graphical Model (PGM) structure that allows efficient learning of the environment model. The authors show that gradients give good direction for policy improvements during training by exploiting model locality. The algorithm was compared to model-free methods and showed significantly better performance and data-efficiency compared to algorithms such as Fitted Linear Models (LQR-FLM) (Zhang, Patras, et al., 2019).

Ha and Schmidhuber, 2018a proposed in *Recurrent World Models Facilitate Policy Evolution*, a novel architecture for training RL algorithms using variational autoencoders. The authors demonstrate that agents could successfully learn the environment dynamics and use this as an exploration technique requiring no interaction with the target domain. The architecture is mainly three components; vision, controller, and model. The vision model is a variational autoencoder that outputs a latent space variable of an observation. The latent space variable is processed in the model and feeds into the controller for action decisions. Their algorithms show state-of-the-art performance in self-supervised generative modeling for reinforcement learning agents.

The algorithm VMAV-C combines VAE and attention-based value function and Mixture Density Network Recurrent Neural Network (MDNRNN) from 2018a. This modification to the original World Models algorithm improved performance in the CartPole environment. They used the on-policy algorithm, PPO, to learn the optimal policy from the latent representation of the state-space (Liang et al., 2018).

Neural Differential Information Gain Optimisation (NDIGO) algorithm by Azar et al., 2019 is a self-supervised exploration model that learns a world model representation from noisy data. The primary characteristic of NDIGO is its robustness to noise because it weights negative feedback lower and provides more positive learning value. The authors show in their maze environment that the model successfully converges towards an optimal world model even when introducing noise. The author claims that the algorithm outperforms the previous state-of-the-art, the Recurrent World Model from Ha and Schmidhuber, 2018a.

Gregor, Jimenez Rezende, et al., 2019 proposed a scheme to train expressive generative models to learn belief-states of complex 3D environments with little prior knowledge. Their method effectively predicts multiple steps into the future (overshooting) and significantly improves sample efficiency. The authors illustrated model-free policy training in several environments in the experiments, including DeepMind Lab. However, the authors found it difficult to use their dynamics model in model-free agents directly.

3.2 Safe Reinforcement Learning

In most established systems in the industry, an expert system made from human reasoning acts as a controller in the environment (Lu, 2019). This is critical for safe and stable learning in real-world environments so that ongoing and new operations are not interrupted. There are several efforts towards better safety in RL. A common approach is formalizing the problem as a constrained MDP and using Lyapunov or Barrier functions to guarantee decision safety for a subset of the policy space under model assumptions. Another frequently used approach is to apply RL algorithms with human intervention when there is uncertainty about the outcome of actions. This section details related work where the primary goal is to improve or solve safe model-based RL.

3.2.1 Lyapunov Functions

Perhaps the most notable recent work in solving problems using safe model-based RL is the work of Berkenkamp et al., 2017. They propose a region of attraction-based method that is guaranteed safe, constrained to a set of safe states. The author presents a learning algorithm with two assumptions. First, the model must be is Lipschitz continuous. Second, requires a reliable and well-calibrated statistical model to allow accurate exploration, close to the ground truth model function (the environment). The author uses the region of attraction method from control theory and Lyapunov functions that constrain the problem to an invariant set of the policy space. Furthermore, the author finds that a Lyapunov function is derivable from the value function $v_\pi(s)$, given that all rewards are positive. The experiments and theoretical justifications demonstrate that the algorithm functions well for the inverted pendulum environment. Furthermore, their proposed algorithm improves performance while also holding safety constraints (2017). Chow et al., 2018 similarly use Lyapunov functions to solve Constrained Markov Decision Process (CMDP) problems with the assumption of a feasible baseline policy. The author proposes updating the Lyapunov function using bootstrapping and showing that the method integrates well with Q-Learning. The author does not provide convergence proof to the optimal policy but empirically demonstrates that their approach performs better than Lagrangian methods.

3.2.2 Barrier Functions

Like Lyapunov-based approaches, the use of barrier function aims to constrain the policy-space to a *safe set* of possible policies that are guaranteed to work safely. Cheng et al., 2019 propose a combination of model-based dynamics learning, shielded RL, and model-free algorithms as actors. Barrier functions are forward invariant, similar to Lyapunov functions. They often use a temperature hyperparameter to determine how constrained the policy space is. The method guarantees safety. However, it builds on the assumption that a determined set of safety policies are given before training. The authors demonstrate that their method outperforms novel model-free algorithms in sample efficiency and safety but at the cost of reward performance. In a similar direction, Yang and Qiu, 2005b propose a novel actor-critic barrier function structure for multi-agent safety-critical systems. Specifically, the authors propose a two-player game architecture for guaranteed safety during learning and solve a known model with safety guarantees.

3.2.3 Human Intervention

One of the fundamental questions to raise in a safety-critical environment is *when to trust your model* (Janner et al., 2019). In earlier work on Human Intervention techniques, Saunders et al., 2018 propose a simple framework for safely training an algorithm using human intervention for catastrophic states. At every timestep in the environment, a human participant evaluates the state and propose an action to the algorithm. For catastrophic actions, humans correct the agent with a negative feedback signal. The human intervention scheme is typically used with model-free algorithms. However, the human can be considered a corrected model of the policy space. The authors found their method to perform well, but it does not scale because humans must supervise the algorithm. Similarly, Turchetta et al., 2020 propose a curriculum-based approach, Curriculum Induction for Safe Reinforcement Learning (CISR), which assumes a teacher that intervenes in the event of catastrophic states. The teacher policy guides the student, intervenes, and puts the student in safe states if the CMDP criteria are not met. Authors describe the CISR as a meta-learning framework where the teacher policy is an optimizable hyperparameter. However, the algorithm makes assumptions that an intervention set exists before learning. The authors demonstrate that the student policy performs significantly better in the Frozen Lake environment when erroneous states are intervened compared to no intervention.

3.2.4 Summary

Many different approaches have been proposed throughout the last decade to improve safety in RL algorithms. We present work that uses Lyapunov or Barrier functions to guarantee decision safety for a subset of the policy space under model assumptions. These methods formalize the problem as a constrained MDP. The second approach is RL algorithms with human intervention when there is uncertainty about the outcome of actions. These methods perform sufficiently but require significant manual labor by humans to function. This work draws inspiration from several other works in safe RL, which the respective authors best describe in Bagnell et al., 2001; Berkenkamp et al., 2021; Campos and Langlois, 2003; del R. Millán, 1995; Edith et al., 2005; Geibel and Wysotzki, 2005; Mihatsch and Neuneier, 2002; Romoff et al., 2018, but is otherwise also referred to throughout this thesis. Furthermore, a comprehensive study in safe RL is described in García and Fernández, 2015.

3.3 Goal-directed Reinforcement Learning

Earlier studies have contributed significantly to improve the ability to solve reinforcement learning problems with a goal-directed approach. Perhaps the most well-known study of the Goal-Directed Reinforcement Learning (GDRL) problem begins with Koenig and Simmons, 1996. Their approach splits the problem into two phases; GDE and knowledge exploitation. The study finds that the convergence of GDRL-based $\hat{Q}$- and Q-learning closely relates to the state representation and volume of prior knowledge. Furthermore, their work shows that computationally intractable problems are tractable with minor modifications to the state- representation.

de S. Braga and Araújo, 1998 apply GDRL using temporal-difference learning to collect prior knowledge and create a reward and penalty surface explaining the environment dynamics. The map acts as an expert advisor for the TD algorithm and proves the policy performance. Their work shows that the concept of GDRL works well in grid-based environments and includes significantly better sample efficiency than Q-Learning.

Matignon et al., 2006 studied the importance of reward functions and initial Q-values for GDRL. The authors thoroughly studied the effect of different initial states of the Q-table and found it challenging to design a generic algorithm for initially setting optimal parameters. However, they found that initial values considerably impact the performance and sample efficiency. Furthermore, the author shows that adding a *goal bias leads to much faster learning* and recommends an *adjustable continuous reward function*.

Debnath et al., 2018 propose a hybrid approach, formalized as a GDRL problem. The first phase optimizes a dynamics model of the environment with samples from a model-free reinforcement learning policy. The second phase exploits the learned dynamics model to improve further the policy, similar to the work in Paper F. The authors show that GDRL-based algorithms accelerate learning and considerably improve sample efficiency.

3.4 Interpretable Reinforcement Learning

There are numerous attempts in the literature to improve interpretability with deep learning algorithms, but primarily in the supervised case. Zhang, Patras, et al., 2019 provide an in-depth survey of interpretability with Convolutional Neural Networks (CNN). Our approach is similar to Wang et al., 2019, where the authors propose an architecture for visual perception of the DQN algorithm. The difference, however, is primarily our focus on the interpretability of the latent space distribution via methods commonly found in variational autoencoders. Similar efforts to combine Q-Learning with VAE include Huang et al., 2020; Tang and Kucukelbir, 2017, which theoretically shows promising results but with limited focus on interpretability. Annasamy and Sycara, 2018 did notable work on interpretability using KL-distance for optimization but did not find convincing evidence for a deeper understanding of the model. The focus of our contribution deviates here and finds significant value in a shallow and organized latent space.

3.5 Hierarchical Reinforcement Learning

The learned latent space is valuable for selecting options in Hierarchical Reinforcement Learning (HRL). There has been increasing engagement in HRL research because of several appealing benefits such as sample efficiency and model simplicity (Barto et al., 2003).

Despite its growing attention, there are few advancements within this field compared to model-free RL. The options framework (Sutton et al., 1999) is perhaps the most promising approach for HRL in terms of intuitive and convergence guarantees. The options framework extends MDP's to Semi Markov Decision Process (SMDP) (Younes and Simmons, 2004). SMDP features temporal abstractions where multiple discrete time steps are generalized to a single step. These abstract steps define an option, where the option is a subset of the state-space. In the proposed algorithm, the structure of the latent space forms such temporal abstractions for options to form.

3.6 Environments

Several exciting game environments in the literature focus on state-of-the-art research in AI algorithms. Few game environments target the RTS genre. One reason may be that these environments are challenging to solve, and there are few ways to fit results with preprocessing tricks. It is, however, essential to include RTS games as part of the active research of deep reinforcement learning algorithms as they feature long-term planning. This section reviews existing game platforms and environments actively used in RL research. For this dissertation, we present results that use environments from Section 3.6.2, Section 3.6.7, Section 3.6.8, Section 3.6.9, and Section 3.6.10.

3.6.1 Stratagus

Stratagus is an open-source game engine that can create RTS-themed games. Wargus, a clone of Warcraft II, and Stargus, a clone of StarCraft I, are examples of games implemented in the Stratagus game engine. Stratagus is not an engine that targets machine learning explicitly, but several researchers have performed experiments in case-based reasoning (Fathy et al., 2010; Ontanon et al., 2008) and Q-Learning (Jaidee and Muñoz-Avila, 2012) using Wargus. Stratagus is still actively updated by contributions from the community.

3.6.2 Arcade Learning Environment

Bellemare et al., 2013 present the Atari Learning Environment (ALE) that enabled researchers to conduct cutting-edge research in general deep learning (2013). The package provides hundreds of Atari 2600 environments that help demonstrate the capabilities of DQN and the A3C algorithm in seminal work from Mnih et al., 2015. The platform has been a critical component in RL research advancements. (Mnih et al., 2016; Mnih et al., 2015)

3.6.3 TorchCraft

Synnaeve et al., 2016 present TorchCraft, a bridge between the scientific computing framework Torch and StarCraft I. TorchCraft is widely used in literature and is used for deep learning research (Churchill et al., 2017; Peng et al., 2017). Additionally, TorchCraft has replay data of over 65 000 StarCraft games from Lin et al., 2017.

3.6.4 Malmo Platform

The Malmo project is a platform built atop the popular game *Minecraft* proposed by Johnson et al., 2016. Minecraft is a 3D environment where the objective is to survive in a world of dangers, construct shelter, prevent starvation, and navigate through complicated environments. The authors claim that the platform has all characteristics that qualify it as a platform for general AI research.

3.6.5 ViZDoom

Kempka et al., 2016 propose ViZDoom, a platform for RL research using pixel-based inputs. Doom is a first-person shooter game in a semi-realistic 3D world where the overall goal is to survive or defeat enemies. The authors modify the game to support scenarios (tasks) such as move-and-shoot and more complex maze-navigation problems. Using a customized reward signal to fuel learning, the authors demonstrate that their implementation of DQN can learn good strategies with just a few hours of training using consumer hardware. The overall goal of their contribution is to enrich the availability of RL environments.

3.6.6 DeepMind Lab

Beattie et al., 2016 present a platform for 3D navigation and puzzle-solving tasks. The primary purpose of DeepMind Lab is to act as a platform for DRL research. DeepMind Lab provides rewards, pixel-based observations, and velocity information of the agents for every timestep. The authors propose a scenario-based environment that entertains problems in different topics such as memory and planning. The DeepMind Lab engine initially provides fruit-gathering levels in a static map, static navigation levels, procedurally-generated navigation levels, and laser-tag levels (similar to first-person shooter games).

3.6.7 OpenAI Gym

OpenAI Gym is "*a toolkit for developing and comparing reinforcement learning algorithms*", first proposed by Brockman et al., 2016. OpenAI Gym provides various environ-

Table 3.1: Micro RTS maps with the corresponding map size.

Map	**Size**
basesWorkers8x8A	8×8
basesWorkers16x16A	16×16
BWDistantResources32x32	32×32
(4)BloodBath.scmB	64×64
FourBasesWorkers8x8	8×8
TwoBasesBarracks16x16	16×16
NoWhereToRun9x8	9×8
DoubleGame24x24	24×24

ments from the following domains: Algorithmic tasks, Atari 2600, Board games, Box2d physics engine, MuJoCo physics engine, and Text-based environments. OpenAI Gym also hosts a website where researchers can submit their reward performance to compare algorithm efficiency. OpenAI Gym is open-source and encourages researchers to add support for their environments.

3.6.8 Micro RTS

Micro RTS is a simple RTS game designed to conduct AI research. The idea behind Micro RTS is to strip away the computational heavy game logic and graphics to increase the performance and enable researchers to test theoretical concepts quickly (Ontanon, 2013). The Micro RTS game logic is deterministic and includes fully and partially observable state-spaces. The primary field of research in Micro RTS is game-tree search techniques such as variations of MCTS and MiniMax (Barriga et al., 2017; Ontanon, 2013; Shleyfman et al., 2014). MicroRTS is also used for DL research, especially in the larger maps, seen in Table 3.1.

3.6.9 ELF: Mini-RTS

The goal in Mini-RTS is for the agent to destroy the opponent's base with its troops. Players have units, resources, and a base and must balance defensive and offensive planning economics. It is possible to expand the base with worker units to build barracks or expand offensive powers. The ELF game engine is tick-driven, meaning that the agent must make actions for every game tick. Mini-RTS is partially observable due to the fog-of-war, and thus the agent is only presented with imperfect information. However, Mini-RTS is significantly less complex logic-wise and graphically than StarCraft II. The Mini-RTS environment features two built-in strategies; AI-Simple and AI-Hit-and-run where both are

Table 3.2: List of mini-game maps in SC2LE.

Map
MoveToBeacon
DefeatRoaches
BuildMarines
CollectMineralShards
CollectMineralAndGas
FindAndDefeatZerglings
DefeatBanelingsAndZerglings

used in the experiments (Tian et al., 2017).

3.6.10 StarCraft II

StarCraft II Learning Environment (SC2LE) bridges the StarCraft II observation and action-space game interfaces to programming languages such as Python. SC2LE is an initiative from Blizzard Entertainment and DeepMind to demonstrate the capabilities of AI in RTS games. StarCraft II is a complex environment that requires short and long-term planning. It is difficult to observe a correlation between actions and rewards due to the imperfect state information and delayed rewards, making StarCraft II one of the hardest challenges for RL algorithms to solve (Vinyals et al., 2017). In addition to the full-game case, SC2LE features mini-games suitable for RTS research, as seen in Table 3.2.

Chapter 4

Software Contributions and Evaluation

The availability of game-playing environments is a prerequisite for RL research in RTS games. Despite many game environments in the literature, environments are too simplistic or require significant computational power. The lack of appropriately leveled game complexity makes RL research challenging because the tasks are either fully solved or expensive to use. Game complexity measures the difficulty and indicates the solvability of a task using current technology. For example, solving tasks that takes minutes in 2021, was nearly impossible to solve in the early 2000's. While state complexity does not account for uncertainty or stochasticity in an environment, it is often used in literature to measure difficulty. For example, Chess (8x8) has a state-complexity of $\sim 10^{50}$, Go (19x19) $\sim 10^{360}$, and for StarCraft II, game complexity estimations range from 10^{1685} to 10^{36000} (Pang et al., 2019) (Figure 4.11). In the extreme pace that RL research is moving, it is important to use benchmarks that provide replicable environments and experiment conditions. Therefore, it is essential to have flexible benchmarks and grow alongside research progress.

It is safe to say that there is a clear gap in the literature of experimental environments in RL, which motivates a scientific view on implementing new game environments for RL research. This chapter presents the resulting software contributions throughout the Ph.D. work and mainly tries to educate on enriching the availability of RL research platforms and how to design these platforms for working in an industry-near manner.

Chapter organization. We present contributions in-order to demonstrate the chain of motivations better. Each contribution presents a motivational introduction, design specifications, and baseline experiments using reinforcement learning algorithms. These experiments function as a baseline for parts of the contributions, but naturally, it is expected that future work should trivially outperform the presented results. Finally, we summarize the contribution.

4.1 Deep Line Wars

Deep Line Wars is a simplistic and highly configurable real-time strategy game simulator that narrows the state-space complexity gap between Atari 2600 and Starcraft II. Deep Line Wars aims to present a minimalistic interface to research planning in RL agents and typically captivates the strategic placement challenge in full-fledged RTS games.[1]

4.1.1 Motivation

Motivation 1 This contribution is motivated by the need for new RTS game environments for RL research. At the time of this contribution, few RTS game environments existed, and to the best of our knowledge, none were easily accessible with state-of-the-art machine learning tools in Python. [2]

Motivation 2 Multiplayer games are not broadly available in the literature, and planning is a demanding challenge for agents in RTS games. The motivation is for Deep Line Wars to cover the gap in available research environments.

4.1.2 Design Specifications

Deep Line Wars is a two-player version of the popular Line Tower Wars modification from the Warcraft III game by Blizzard Entertainment. Line Tower Wars is not directly an RTS game. However, Deep Line Wars features a majority of elements in the RTS genre, such as time-delayed objectives, resource management, offensive and defensive strategy planning.

Game Objective

The objective of Deep Line Wars is, as seen in Figure 4.1, to invade the opposing participant with units until the health reaches zero. For each unit that enters the black area on the map, the opposing participant's health reduces by one. The participants purchase units for gold, which spawns at random locations on black tiles, and automatically walks towards the other participant's black line. It is possible to defend the base by building towers that shoot projectiles at the opposing participant's units. When units die, the opposing player receives a configurable amount of the unit's gold cost. Purchasing units increase the participant's income, distributed at a fixed time interval, typically every 10 seconds throughout the game. All variables such as income ratio, income frequency, damage, and

[1]Deep Line Wars is open-source and freely available at https://github.com/cair/deep-line-wars and https://github.com/cair/deep-line-wars-2

[2]Note that since the publication of our work, several others have proposed RTS environment such as SC2LE (Vinyals et al., 2017)

unit health pools are configurable.

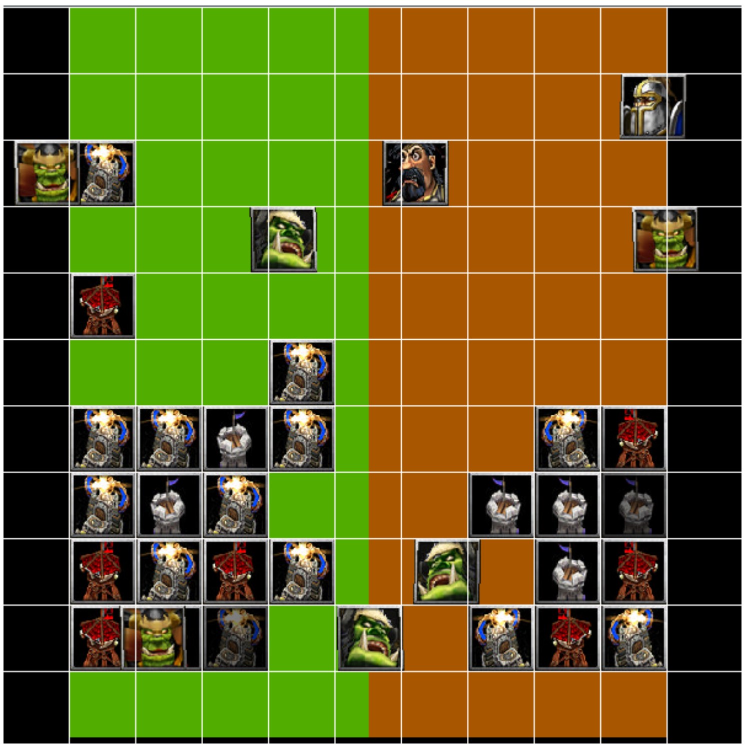


Figure 4.1: The Deep Line Wars environment. Paper B

The primary skillset required to master the Deep Line Wars environment is to:

1. create a unit purchase strategy (planning),
2. place defensive buildings strategically on the map, and
3. maintain a healthy balance between offensive and defensive behavior to maximize defense, offense, and economy,

and combined, guarantees victory for the best performing participant.

State Space

Deep Line Wars models the game-state as raw game frames as seen in Figure 4.1, or a 3-D matrix where dimensions of the matrix are height $\times$ width $\times$ features. It is possible to create state observations containing all relevant board information at the current time. Each layer in the third dimension of the matrix represents a feature similar to the state representation in Silver et al., 2016. Deep Line Wars also features possibilities of making an abstract state-space representation such as a heat-map.

Using heat maps enables the encoding of five-dimensional state information to only three dimensions, red, green, and blue (RGB). Figure 4.2 demonstrates the state from a player

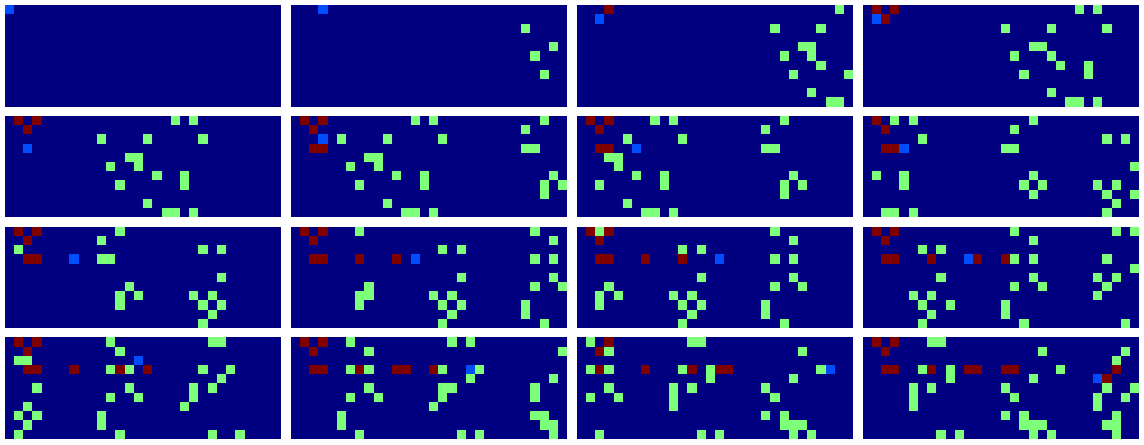


Figure 4.2: State abstraction using heat maps. Red pixels illustrate friendly units and buildings. Green illustrates enemy units and buildings, and light blue illustrates the mouse cursor.

using colored heat maps, with red pixels as friendly buildings, green pixels as enemy units, and light blue pixels as the mouse cursor. It is also possible to reduce the state-space to a one-dimensional matrix using gray-scale intensities. This is useful because Convolutional Neural Networks (CNN) take significantly less time to train when reducing the state-space. The state is not fully described heat maps as economics, health, and income are not encoded. However, extending the heat map to encode 24 bits of vector data in a single pixel is possible and can fully encode the state-space.

Action-space

Action-spaces in Deep Line Wars are highly configurable using the game API.[3] The default action-space features 12 discrete actions selections, as seen in Table 4.1. Specifically, the action is a one-hot vector, where 1 indicates to act, and 0 does not perform the action. Table 4.2 illustrates the continuous action-space and has 4 actions that are any real number between the specified bounds. For example, action 1.321 for the "Unit Send" action in the continuous action-space translates to the "send_militia" action in discrete space.

Table 4.2: The continous action-space in Deep Line Wars.

Action	**Min Value**	**Max Value**	**Description**
Move Mouse X	0.0	1.0	0.0 is left while 1.0 is far right
Move Mouse Y	0.0	1.0	0.0 is top and 1.0 is bottom
Unit Send	0.0	4.0	Each whole number is a unit
Unit Build	0.0	3.0	Each whole number is a building

[3]The complete game API definitions are located in the action_space.py source file.

Table 4.1: The discrete action-space in Deep Line Wars. The action column denotes the action name with a corresponding description column.

Action	Description
cursor_left	Move cursor one step to the left
cursor_right	Move cursor one step to the right
cursor_up	Move cursor one step up
cursor_down	Move cursor one step down
send_militia	Spawn a militia unit
send_footman	Spawn a footman unit
send_grunt	Spawn a grunt unit
send_armored_grunt	Spawn an armored grunt unit
build_basic_tower	Build a basic tower at the cursor
build_fast_tower	Build a fast tower at the cursor
build_faster_tower	Build a faster tower at the cursor
no_action	Do nothing

Baseline Results

Table 4.3: The table presents baseline results for selected Deep Line Wars map sizes. The map size is written width × height, and for the score, WR denotes Win-Rate and IC is the gold income after the match ends.

Map Size	Algorithm	Opponent	Score	Source
30x11	Rule-Based	Random	WR-92.1%	Paper B
30x11	DQN	Random	WR-63.5%	Paper B
30x11	Rule-Based	Random	IC-830	Paper B
30x11	DQN	Random	IC-645	Paper B
11x11	DQN	Rule-Based	IC-550	Paper E
11x11	PPO	Rule-Based	IC-290	Paper E
11x11	DVAE (RNN)	Rule-Based	IC-350	Paper E
30x30	DQN	Rule-Based	IC-110	Paper G
30x30	PPO	Rule-Based	IC-91	Paper G
30x30	DVAE (SAFE)	Rule-Based	IC-500	Paper G
12x11, 22x22, 40x30, 64x64		Competition	WR	See Table 6.5

Table 4.3 presents the baseline results of the Deep Line Wars environment from experiments in Paper B, Paper E, and Paper G. Specifically, we demonstrate that DQN, PPO,

and DVAE beats the rule-based agent implementation.

4.1.3 Summary

Deep Line Wars is a simple but advanced Real-time (strategy) game simulator that fills the gap between Atari 2600 (Bellemare et al., 2013) and Starcraft II (Vinyals et al., 2017). We show that algorithms such as DQN and PPO can beat hard-coded expert agents in the game using implemented observation and action-spaces. Deep Line Wars supports both continuous and discrete action-spaces, and it is possible to use heat maps in place of raw pixel observations to reduce observation dimensionality. Finally, the Deep Line Wars have a configurable interface to adjust income rates, unit properties, and the game execution speed.

4.2 Deep Maze

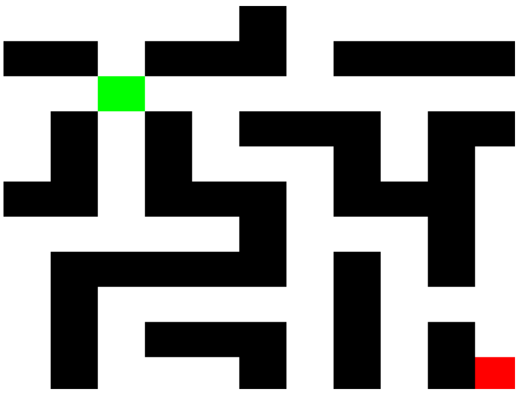
The Deep Maze is a flexible learning environment for controlled research in exploration, planning, and memory for reinforcement learning algorithms. Maze solving is a well-known problem used heavily throughout the RL literature (Sutton et al., 1999) but is often limited to small and fully-observable scenarios. The problem is framed as a Markov Decision Process (MDP) in the fully observable case. Partial Observable Markov Decision Process (POMDP) is used for mazes with partial observability (see Section 2.1 for its definition). The Deep Maze game environment adds over 540 pre-configured scenarios, including mazes that are only partial visible (learning to navigate POMDP's). At the time of this contribution, there were no similar peer-reviewed maze research environments to the best of our knowledge. Later, Chevalier-Boisvert et al., 2019 proposed Gym-MiniGrid, featuring comparable characteristics to Deep Maze.[4]

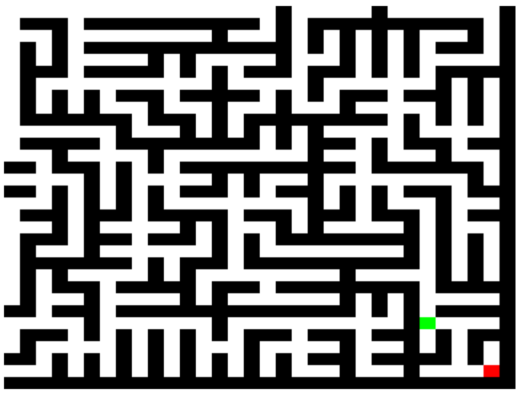
4.2.1 Motivation

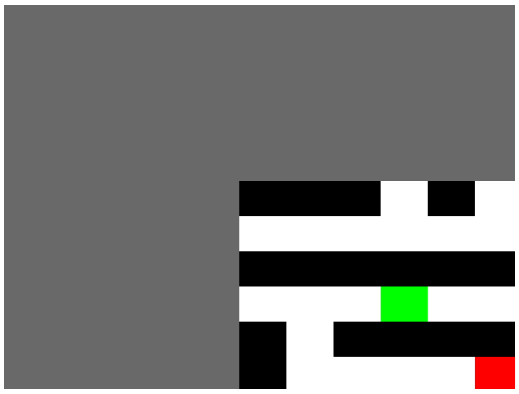
Motivation 1 This contribution motivates the use of maze environments, providing a platform for RL research. Specifically, maze environments address the need to study RL agents' exploration-exploitation trade-off, planning, and memory, which proves valuable for mastering RTS games (Ontanon, 2013).

Motivation 2 Deep Line Wars does not sufficiently address all aspects of an RTS game. This contribution motivates to create Deep Maze, which can fill the research gap as there are no existing solutions for grids with POMDP settings.

[4]Deep Maze is open-source and freely available at https://github.com/cair/deep-maze


(a) A small maze with fully observable state-space.


(b) A large maze with fully observable state-space.


(c) A partially observable maze where the agent has 3 tiles of vision.

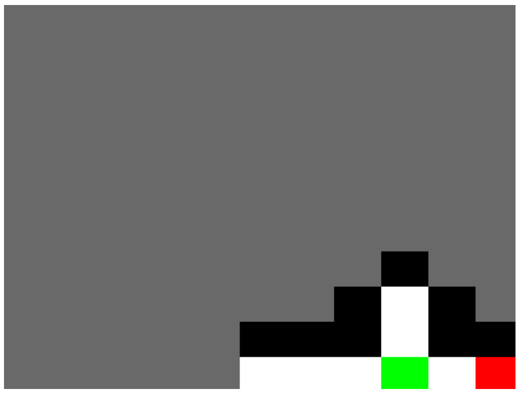

(d) A partially observable maze where the agent has a ray-tracing vision.

Figure 4.3: Overview of four distinct maze scenarios using Deep Maze.

4.2.2 Baseline Results

Table 4.4 shows the baseline results for Deep Maze from our work. Specifically, the Deep Maze environment demonstrates capabilities in the DVAE algorithm to reconstruct maze-environments in Paper D, showing promising results in navigating mazes from imaginary data. We detail the results in the peer-reviewed work from Paper D and Paper H.

4.2.3 Design Specifications

Figure 4.3 illustrates selected scenarios in the Deep Maze, ranging from small-scale MDP's (fully observable mazes) to large-scale POMDP's (partially observable mazes). The Deep Maze features custom game mechanics such as dynamic goal positions and dynamically changing maze structures (i.e., moving walls). Preprocessing of data is essential so that agents can extract features from the input data. Deep Maze has a built-in state representation for imaging and raw state matrices, similar to Deep Line Wars. The game engine is modularized and has a software development kit that facilitates the devel-

Table 4.4: The table presents baseline results for selected Deep Maze map sizes. The map size is written width $\times$ height. The score is percentage-based, where 100% denotes the optimal path. The episodes column denotes how many times the experiment ran. Note that for the TRPO-D and PPO-$\hat{D}$ algorithm, we experiment with replay-buffer in on-policy algorithms, effectively training them in an offline manner.

Map Size	**Algorithm**	**Episodes**	**Score**	**Source**
11x11	DQN-$\hat{D}$	1K	94.56%	Paper D
11x11	TRPO-$\hat{D}$	1K	96.32%	Paper D
11x11	PPO-$\hat{D}$	1K	98.71%	Paper D
11x11	DQN-D	1K	98.26%	Paper D
11x11	TRPO-D	1K	99.32%	Paper D
11x11	PPO-D	1K	99.35%	Paper D
21x21	DQN-$\hat{D}$	1K	64.36%	Paper D
21x21	TRPO-$\hat{D}$	1K	78.91%	Paper D
21x21	PPO-$\hat{D}$	1K	89.33%	Paper D
21x21	DQN-D	1K	84.65%	Paper D
21x21	TRPO-D	1K	92.11%	Paper D
21x21	PPO-D	1K	96.41%	Paper D
11x11-NoWalls	DVAE (CostNet)	100	100%	Paper H
11x11-NoWalls	PPO	100	52%	Paper H
11x11-NoWalls	DQN	100	77%	Paper H

opment of new scenarios and game logic modes. The design of Deep Maze focuses on supporting nearly any possible scenario combinations in the realm of maze solving.[5]

The Deep Maze game environment presents the following scenarios types:

Traditional MDP Traditional MDP mode generates a maze structure randomly with a specified seed. The state observations are fully visible.

POMDP Similar to the traditional MDP mode, but the state observations are partially visible. The POMDP mode supports radius-based and ray-tracing-based vision.

Limited POMDP In extension to the POMDP mode, the Limited game mode reveals the full state observation for a few frames before behaving like a POMDP. This mode is especially suited for memory-based RL research.

Timed Limited (PO)MDP Extending the Limited POMDP, the Timed version requires task completion before time runs out.

[5]The Deep Maze is open-source and publicly available at https://github.com/CAIR/deep-maze.

All pre-configured scenario setups have a variable map size ranging between 2×2 and 56×56 tiles.

4.2.4 Summary

Deep Maze is a versatile maze game environment for researching planning, memory, and model-based algorithms in reinforcement learning. We contribute a platform that supports traditional observation schemes following the MDP framework, including game configuration for observations following the POMDP framework. We show that it helps study dynamics models. We provide baseline results for future research and demonstrate that algorithms learn successfully in the tested environment setups.

4.3 Deep RTS

The Deep RTS game research environment enables studying planning, reasoning, and control at different difficulty levels. Deep RTS aims to narrow the research gap between Micro RTS (Ontanon et al., 2013) and StarCraft II (Vinyals et al., 2017). Figure 4.4 illustrates the game visuals. Deep RTS compose games as scenarios that define a particular goal to win the game. For primitive scenarios, the goal is to gather a set amount of gold before the agent receives a reward signal and the game terminates. On the other hand, complex scenarios support observability, environmental danger, and other opponents. It is possible to define delayed actions and rewards to increase the task's difficulty.[6]

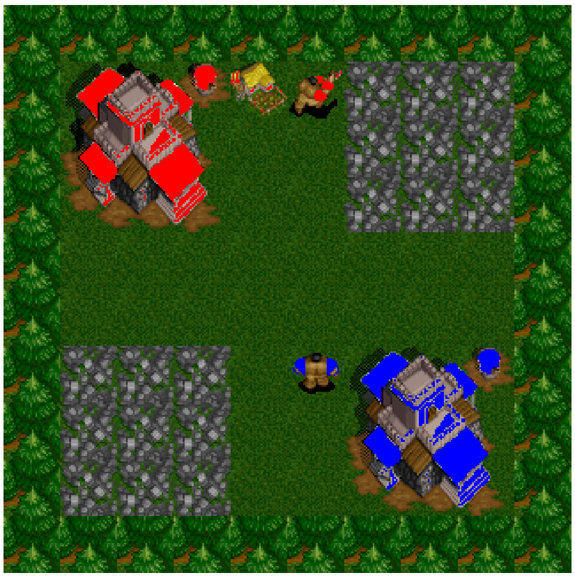


Figure 4.4: Demonstration of the Deep RTS Deathmatch (10x10-2-FFA) scenario where two participants battle towards defeating the opposing player.

[6]Deep Line Wars is open-source and freely available at https://github.com/cair/deep-rts

4.3.1 Motivation

Motivation 1 The Deep RTS game environment enables research at different difficulty levels in planning, reasoning, and control. The motivation behind creating Deep RTS is that the Deep Line Wars environment (Section 4.1) from Paper B cannot alone fill the gap between Micro RTS (Ontanon et al., 2013) and StarCraft II (Vinyals et al., 2017), specifically in long-horizon planning and complex reasoning.

Motivation 2 The existing solutions are not sufficiently flexible, which Tian et al., 2017 discover simultaneously. Specifically, Micro RTS does not support complex game settings and has only a few units and buildings available. On the other hand, StarCraft II requires immense computing power to train RL algorithms in an acceptable timeframe. This contribution motivates to build an environment that supports deterministic and non-durative game settings for the simplest configurations, up to highly complex stochastic environments with durative gameplay.

Motivation 3 Existing solutions, such as Micro RTS and StarCraft II, are not straightforward to customize for non-standard experimentation. This motivates the creation of a highly customizable scenario engine to build custom environments trivially. Deep RTS supports the OpenAI Gym abstractions through our specially crafted Python API and can easily integrate with Python and C++ applications.

4.3.2 Design Specifications

Game Objective

The Deep RTS objective is to build a base consisting of a town-hall, and then strive to expand the base using gathered resources to gain the military upper hand. Military units conduct attacks where the primary goal is to demolish the opponent's base. Players start with a worker unit. The primary objective of the worker units is to expand the base offensive, defensive and gather natural resources found throughout the game world. Buildings can further spawn additional units that strengthen the player's offensive capabilities. All opponent units must be destroyed for a player to reach the terminal state.

A regular RTS game can be represented in three stages: early-game, mid-game and late-game. Early-game is the gathering and base expansion stage. The mid-game focuses on the military and economic superiority, while the late-game stage is usually a deathmatch between players until the game ends.

Table 4.5: A partial list of configuration flags for the Deep RTS game engine.[7]

Config Name	Type	Description
instant_town_hall	Bool	Spawn Town-Hall at game start.
instant_building	Bool	Non-durative Build Mode.
instant_walking	Bool	Non-durative Walk Mode.
harvest_forever	Bool	Harvest resources automatically.
auto_attack	Bool	Automatic retaliation when being attacked.

Game Mechanics

The game mechanics of the Deep RTS are flexible and can be adjusted before a game starts. Table 4.5 shows a list of the available game setting parameters. An important design choice is to allow actions to affect the environment without temporal delay. The game engine runs tick-based and has a default base tick-rate of 10. The base-tick rate is the minimal time any action takes to perform, but constructing buildings takes longer. All tick-timers are adjustable in the configuration file. For each game-loop iteration, the tick counter increments by one, and actions evaluate promptly. Using a tick-rate-based game engine allows running high-resolution tick-rates that closely resemble the StarCraft II dynamic and low-resolution tick-rates to mimick the Micro RTS engine.

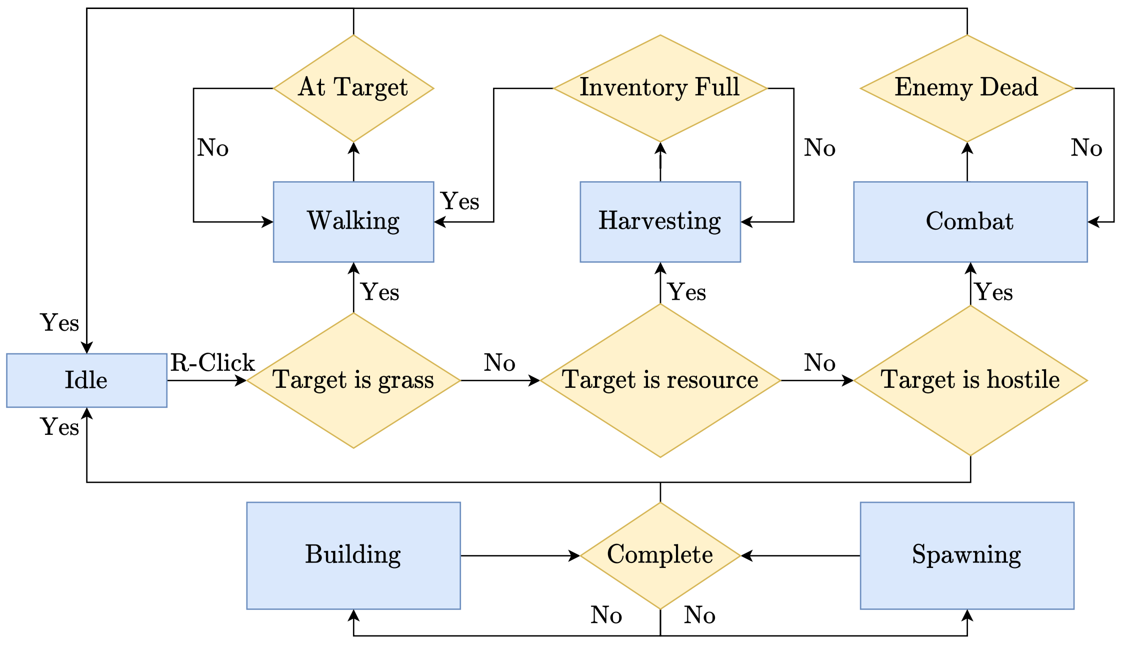


Figure 4.5: Unit state evaluation based on actions and current state

All game entities (units and buildings) have a state machine that determines their current state. Figure 4.5 illustrates the state-machine that evaluates for every game tick. Units and buildings start in the "Spawning" state and transition to the "Idle" state after the spawning process. The Idle state is the default state for all units and buildings. There are primarily three resources, gold, lumber, and oil, available for workers to harvest. The value range is practically limited to the number of resources on the game map. The food and unit limits ensure that the player does not excessively produce units.

Action-space

The Deep RTS action-space separates into two abstract levels. The first level is actions that directly trigger a state transition: right-click, left-click, move-left, and select-unit. The second layer of abstraction is actions that combine actions from the previous layer, typically *select-unit* $\rightarrow$ *right-click* $\rightarrow$ *right-click* $\rightarrow$ *move-left*. The benefit of such abstractions is that algorithms can focus on specific areas within the game-state and build hierarchical models that specialize in tasks (planning). Deep RTS initially features 16 different actions in the first layer and six actions in the last abstraction layer, but it is trivial to add additional actions.

Scenarios

Because Deep RTS targets various reinforcement learning problems, game scenarios such as resource gathering, military, and defensive tasks present only a fraction of the complexity in a full RTS game. Table 4.6 demonstrates a majority of the present scenarios in Deep RTS, where:

- The first six scenarios mimicks classical StarCraft II RTS game mode. The map sizes vary between 10x10 to 31x31, with the ability to have six active players in a free-for-all setting.
- The *Solo-Score* scenario features an environment where the objective is to maximize score and minimize used time. The score is predetermined as a ratio of gathered resources and damage done, but it is easily modified if required.
- *Solo-Resources* is a game mode that focuses on resource gathering. The agent must find a balance between base expansion and resource gathering to gather as many resources as possible optimally (shortest amount of time).
- *Solo-Army* is a scenario where the primary goal is to expand the military forces quickly and launch an attack on the enemy player. The enemy player is either idle or uses a hard-coded strategy.
- *Gold-Collect* and *Lumber-Collect* aim towards optimization of the shortest-path problem. The objective is to maximize the specified resource by minimizing the

number of walked steps.

- *Lava-Maze* is a maze map where the aim is to avoid lava tiles. If a player enters the lava tiles, the game terminates with a negative reward. The objective is for the player to reach the middle part of the map and harvest gold without suffering damage. See Figure 6.25 for a snapshot of this game scenario.

Table 4.6: The table shows a partial list of available scenarios in the Deep RTS game environment. The score column denotes the reward function objective where W/R is shorthand for Win-Ratio and numbered score follows a scalar output function.

Scenario Name	**Description**	**Game Length**	Score	**Map Size**
10x10-2-FFA	2-Player game	600-900 ticks	W/R	10x10
15x15-2-FFA	2-Player game	900-1300 ticks	W/R	15x15
21x21-2-FFA	2-Player game	2000-3000 ticks	W/R	21x21
31x31-2-FFA	2-Player game	6000-9000 ticks	W/R	31x31
31x31-4-FFA	4-Player game	8000-11k ticks	W/R	31x31
31x31-6-FFA	6-Player game	15k-20k ticks	W/R	31x31
Solo-Score	Score Accumulation	1200 ticks	0-1K	10x10
Solo-Resources	Resource Harvesting	600 ticks	0-1K	10x10
Solo-Army	Army Accumulation	1200 ticks	0-1K	10x10
Gold-Collect	Resource Accumulation	1000 ticks	0-1K	10x10
Lumber-Collect	Resource Accumulation	1000 ticks	-500-0	10x10
Lava-Maze	Lava Avoidance	1000 ticks	0-113	30x30

State Space and Graphics

The Deep RTS game visuals have several optional backends available in Blend2D, SFML, and PyGame. These are graphical rendering frameworks. Figure 4.6 shows the graphics of Deep RTS, where Blend2D is the primary backend for drawing graphics. It renders significantly faster than the alternatives because it can render purely using the CPU (software-rendering). Although it is common to use hardware-rendering with GPUs in modern graphics, it poses a challenge because the frame buffer resides in the GPU memory, making it costly to transfer frames from GPU to CPU memory. The Deep RTS game aims primarily for algorithmic play, but manual control is possible using the PyGame and SFML backend. There are several methods of representing the game-state observations. The first method is to use images of the game graphics or the 3-D matrix representation. Each layer in the matrix representation depicts a feature, i.e., the units player health at layer 0 and the owner of the units at layer 1. There is support for 24 distinct features

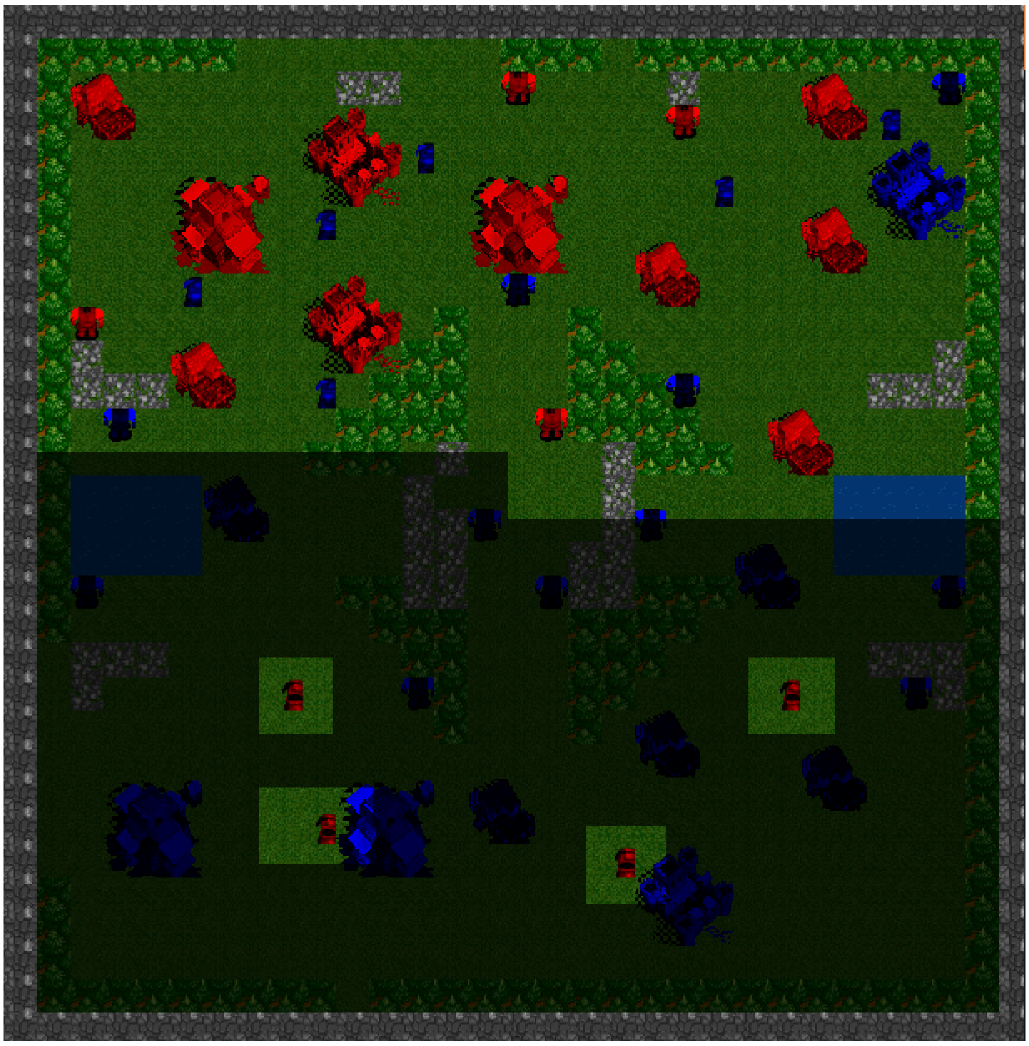


Figure 4.6: The image demonstrates a complex scenario with fog of war in the Deep RTS environment. The filter is manually added for demonstration intents because the game only supports opaque fog of war.

which the programmer can choose freely. In the context of RL, it is possible to model the state-space as partially observable, meaning that only parts of the game are visible, as seen in Figure 4.6. Partial observability adds fog of war to the game, meaning that areas are hidden where the player lacks map control.

4.3.3 Baseline Results

Table 4.7: A summary of baseline results for selected Deep RTS scenarios. The 15x15-2-FFA scenario measures score in win-rate (%) while others use a scoring function, where values are seen in Table 4.6.

Environment	Algorithm	Score	Source
15x15-2-FFA	DQN vs Random	75%	Paper C
15x15-2-FFA	DQN vs Rule	50%	Paper C
Lumber-Collect	DQN	-179	Paper E
Lumber-Collect	DVAE (RNN)	-14	Paper E
Lumber-Collect	PPO	-2	Paper E
Gold-Collect	DQN	226	Paper G
Gold-Collect	PPO	384	Paper G
Gold-Collect	DVAE (SAFE)	498	Paper G
Gold-Collect	DQN	389	Paper H
Gold-Collect	PPO	551	Paper H
Gold-Collect	DVAE (Cost)	669	Paper H
Lava-Maze	A2C	91	Table 6.6
Lava-Maze	DQN	98	Table 6.6
Lava-Maze	RAINBOW	99	Table 6.6
Lava-Maze	IMPALA	100	Table 6.6
Lava-Maze	PPO	96	Table 6.6
Lava-Maze	DVAE (SAFE)	97	Table 6.6
Lava-Maze	S-ORACLE	94	Table 6.6

Many of the presented articles evaluate algorithm reward performance using the Deep RTS game environment. In addition to our peer-reviewed work, many student projects use Deep RTS to study algorithms for solving RTS games[8]. Table 4.7 shows results from peer-reviewed contributions from Paper C, Paper E, Paper G, and Paper H, including the latest work in Paper M. The scoring model is found in Table 4.6.

4.3.4 Summary

Deep RTS is a novel RTS game environment for research in reinforcement learning algorithms. We show that the Deep RTS game environment has high-performance, with few

[8]A non-exhaustive list of projects using the Deep RTS engine is located atdocs/PROJECTS.md in the Deep RTS GIT repository.

game mechanics that negatively impact the update and frame-rate performance. The most resource-hungry mechanic of Deep RTS is the pathfinding algorithm, but it is possible to deactivate by configurations, effectively reducing the game execution time. We show that reinforcement learning algorithms outperform hard-coded strategies and prove that the built systems are suitable for reinforcement learning research. Deep RTS supports mini-games or scenarios that enable research in specific characteristics of agents like long-term memory, planning, and decision safety. The aim is that Deep RTS can evolve to become a central platform for RTS game AI control research. We provide baseline results for future research in Table 4.7.

4.4 Deep Warehouse

Safety during learning in RL has been a less prevalent priority in recent years than improving the performance of existing non-safe algorithms. We argue that this may be due to the high cost of physical systems to experiment on and that the RL research community primarily uses games as a benchmarking tool. Naturally, this encourages maximizing the agent reward performance by trial and error because there is no consequences for failed actions. Training algorithms in real-world environments have severe safety challenges and suffer from low sampling speeds (Botvinick et al., 2019). Deep Warehouse has a wide range of configurations. Furthermore, it is the only open-source implementation that aims to simulate proprietary automated storage and retrieval systems to the best of our knowledge. We also refer to the environment as the ASRS-Lab. [9]

In the context of warehousing, an Automated Storage and Retrieval Systems (ASRS) is a composition of computer programs working together to maximize the incoming and outcoming throughput of items. An item can be any type of material, for example, computer hardware. Using an ASRS in logistics has many benefits, including high scalability, increased efficiency, reduced operating expenses, and operation safety. This work primarily considers a cube-based ASRS environment which can be thought of as a 3-dimensional asymmetrical rectangle with items stored depth-wise. Taxi agents collect and deliver items to delivery points on the rectangle's uppermost layer and usually work alongside other agents. A computer program controls the taxi agent that reads its sensory data to determine the following action. Although these systems are far better than manual labor warehousing, there is still significant improvement potential in the current state-of-the-art. Most ASRS are manually crafted expert systems, which due to the high complexity of the multi-agent ASRS, only performs sub-optimally (Roodbergen and Vis, 2009). One such example is the AutoStore warehouse system, one of the world-leading companies in ASRS. Their approach is to build and optimize their warehouse system virtually using

[9]Deep Warehouse is open-source and freely available at https://github.com/cair/deep-warehouse

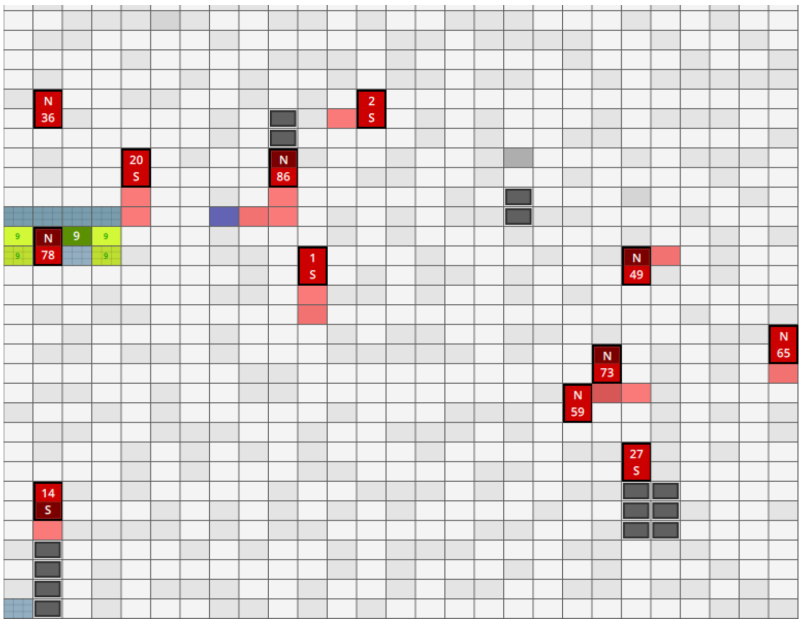


Figure 4.7: ©Autostore AS. The AutoStore simulation environment replicates the conditions of physical installations with minimal error margins. Taxi agents are red 2x1 rectangles with corresponding cart orientation (N = North, S = South) and a unique identifier number. Single red squares denote the path which a taxi "locks" during path execution. The blue square is the destination tile. The dark gray tiles are item boxes that the taxis have placed on top of the grid, where lighter gray gradients represent the depth of item boxes in the grid. The green and yellow tiles are the delivery station for items.

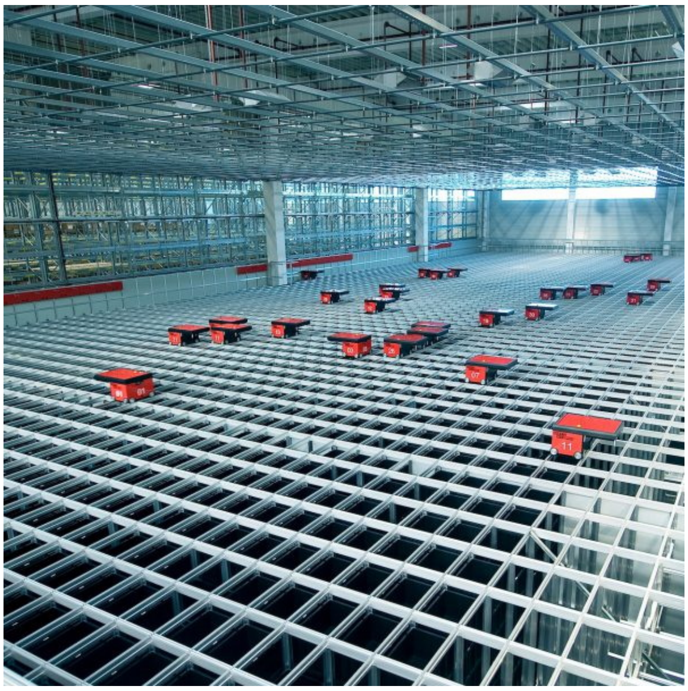


Figure 4.8: ©Autostore AS. Illustration of a physical installation of an Autostore System. The red taxi agents gather items from the bins in the grid and deliver them to delivery points.

a highly accurate simulation environment (Figure 4.7). They can build a highly efficient industry-grade warehouse system from simulations, as seen in Figure 4.8. The challenge is that these systems have a substantial state-space complexity (See Section 4.6.1), meaning that it is nearly impossible to create hand-crafted algorithms with global optimality. For this reason, it is natural to investigate if RL can provide new directions for safe and more efficient algorithms for grid warehousing.

4.4.1 Motivation

It is well known that the training of algorithms in real-world environments is complicated for several reasons mentioned below, which causes non-deterministic side-effects.

Motivation 1 In real-world environments, there is no option to accelerate the sampling speed to increase training speed since the training speed depends on real-world time.

Motivation 2 Reinforcement learning builds on trial and error, which is not applicable in mission-critical systems as an error can have catastrophic consequences. A majority of used environments measure the risk-neutral performance of the agent.

Motivation 3 In real-world environments, additional uncertainty factors can drastically change the observable state-space. Most RL algorithms can adapt sufficiently to slight changes, but with the risk of policy collapse for drastic changes.

Motivation 4 A deterministic simulation environment system is challenging to replicate in the real world. Because real-world environments are susceptible to external forces, such environments are prone to become stochastic. For example, if the power grid goes offline or an earthquake disturbs the operations, these are factors that training data does not usually reflect in simulated environments. These factors cause challenges to guarantee safety during training in real-world environments.

4.4.2 Design Specifications

The Deep Warehouse environment is implemented with flexible state, action, and reward representations with safe reinforcement learning in mind. There are many categories of ASRS in the real world, and building an environment flexible enough to accommodate all requirements for any system is challenging. Deep Warehouse could successfully reconstruct shuttle-based, aisle-based, and grid-based warehouses. For this work, we consider grid-based ASRS.

Figure 4.9 illustrates the observable state-space from a two-dimensional point of view. In a simple cube-based ASRS, the environment consists of (B) passive and (C) active

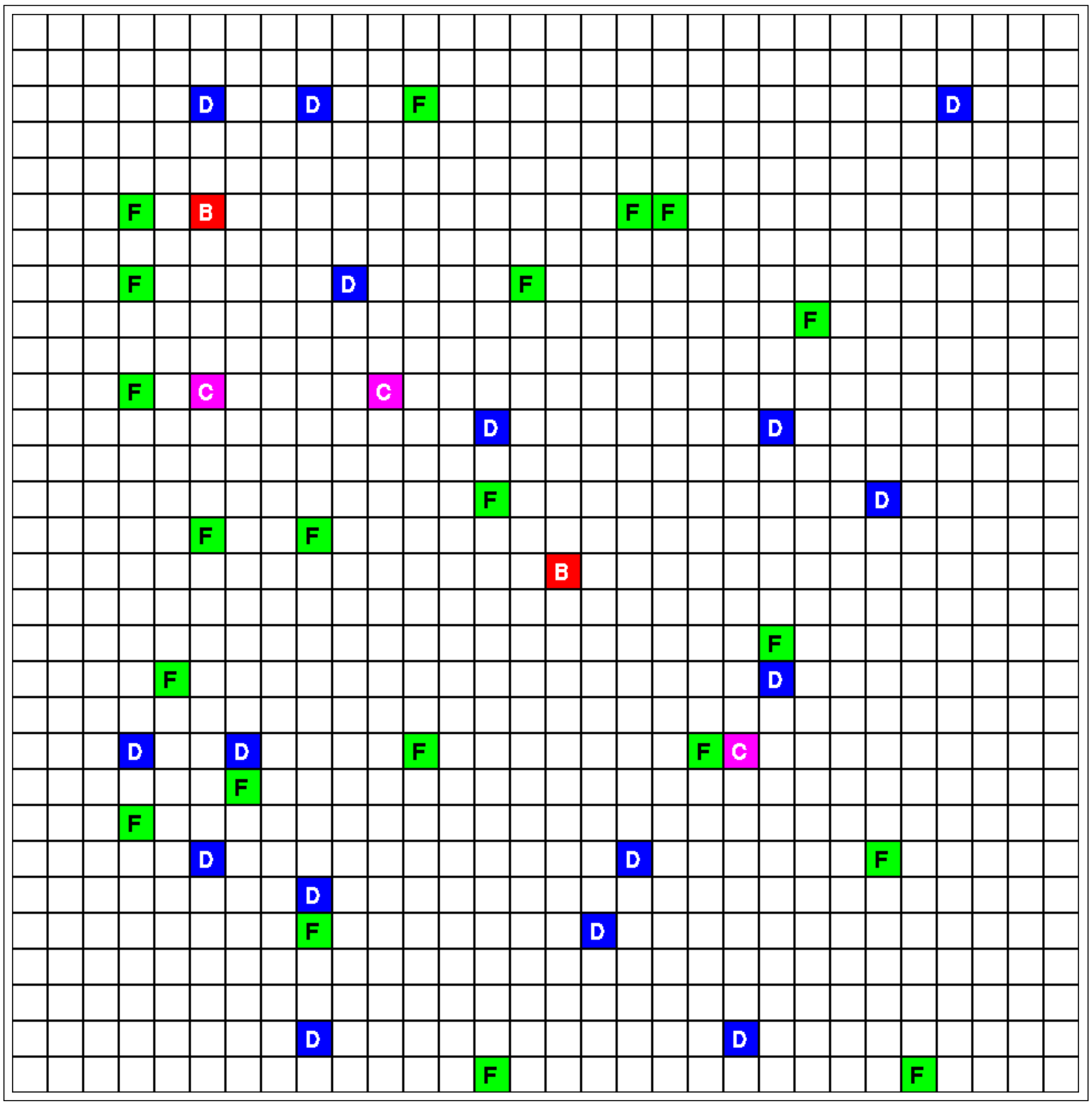


Figure 4.9: Deep Warehouse. In a simple cube-based ASRS, the environment consists of (B) passive and (C) active delivery points, (D) pickup points, and (F) taxis (Paper G).

delivery points, (D) pickup points, and (F) taxis. The goal is to find a positive terminal state using minimal time with limited actions. One episode of the environment is defined as follows. The (taxi) agent starts at an arbitrary position on the plane. At the same time, the agent receives a *retrieve order* from the ASRS *scheduling system*. This order describes a target location for items to be retrieved. The agent must now reach the target location in minimal time using its controls. Considering that there are many other agents on the plane, the control task is challenging to learn because each action has a significant risk of collision with other agents and the outer bounds of the grid system. When an agent enters a target position, it is rewarded and is assigned a *delivery task* from the scheduling system. The agent must now move to the designated location described by the delivery task. When the agent reaches its destination, a large reward is given.

A taxi can move using a discrete or continuous controller. The agent can increase or decrease motor thrust in the discrete mode and move in either direction, excluding the diagonals. For the continuous mode, all of these actions are floating-point numbers between (off) 0 and (on) 1, giving a significantly harder action-space to learn. The simulator also features a continuous mode for the state-space, where actions are performed asynchronously to the game loop. The environment supports custom modules for item scheduling systems, item sorting, agent controllers, and reward functions.

A notable benefit of Deep Warehouse is that it can accurately model real-world warehouse environments at high speed. The Deep Warehouse environment runs an order of magnitudes faster on a single high-end processing unit compared to real-world systems. The reward performance is measured by comparing the number of actions a taxi performs in the real environment versus the virtual environment. The environment can be distributed on many processing units to increase the computational performance further. Our benchmarks suggest that the simulator can achieve 1 million state samples per second during the training of deep learning models using High-Performance Computing (HPC) schemes.

4.4.3 Baseline Results

Table 4.8: Description of the baseline results in Deep Warehouse. The environment naming convention is width $\times$ height $\times$ number of agents. In these experiments, the max score of 30x30x20 is 500, 41x41-100 is 200, and 11x11x10 is 500.

Environment	Algorithm	Score	Source
30x30-20	DQN	198	Paper E
30x30-20	PPO	293	Paper E
30x30-20	DVAE (RNN)	495	Paper E
41x41-100	DQN	163	Paper G
41x41-100	PPO	200	Paper G
41x41-100	DVAE (SAFE)	200	Paper G
11x11-10	DVAE (SAFE-PPO)	0	Paper I
11x11-10	DVAE (SAFE-DQN)	391	Paper I
11x11-10	DQN	500	Paper I
11x11-10	PPO	500	Paper I
30x30-20	DVAE (SAFE-PPO)	421	Paper I
30x30-20	DVAE (SAFE-DQN)	391	Paper I
30x30-20	DQN	496	Paper I
30x30-20	PPO	139	Paper I

Table 4.8 presents the baseline results of Deep Warehouse. Additional results on safety are found in Figure 7 and Figure 8 in Paper M. The table is the combined results of Paper E, Paper G, and Paper I, demonstrating the effectiveness of model-free approaches and the DVAE algorithm. The environment column describes grid size (width x height) and the number of taxi agents in the grid (after the dash, namely, 20, 100, 10, and 20). The algorithm column is which algorithm achieves the corresponding score. The score measures how efficiently the agents can retrieve and deliver items in the grid.

4.4.4 Summary

The Deep Warehouse environment is especially suited for studying AI Control and reinforcement learning agents for industry-near ASRS. Although the Deep Warehouse does not behave identical to a real-world system, it adequately represents the perceived reality from empirical studies to verify capabilities, training times, decision safety, and agent reward performance. The Deep Warehouse is versatile with support for discrete and continuous action-spaces, and the programmer can optionally customize the observation space to better fit specific algorithms. The environment is configurable, meaning grid size and depth, number of agents, and taxi physics can be adjusted to match real-world installations better. Finally, we provide baseline results in Table 4.8 for future studies.

4.5 CaiRL Environment Suite

The design goal of CaiRL is to have interoperability with OpenAI Gym with significantly better CPU performance and the flexibility to support environments, agnostic to programming languages. Compatibility with OpenAI Gym is central to CaiRL enabling, developers and researchers to use CaiRL and OpenAI GYM side-by-side, without significant amendments to existing code. [10]

CaiRL is a new reinforcement learning environment toolkit for high-performance experiments. By designing such a toolkit, reinforcement learning becomes more affordable due to reduced execution costs and strives to reach more sustainable AI. A bi-effect of these goals is that experiments run significantly faster, and most CPU cycles on training AI instead of evaluating game-states. The CaiRL environment toolkit supports classical RL problems such as, but not limited to:

Classical Environments: Cart-Pole, Acro-Bot, Mountain-Car, and Pendulum

New Environments: e.g., Deep RTS (Paper C), Deep Line Wars (Paper B), Deep Maze (Paper D), Deep Warehouse (Paper G) and the X1337 Space Shooter (not peer-reviewed).

Flash Games: Over 1 000 Flash games are available for experimentation[11].

Java Games: e.g., Micro RTS and Showdown AI Competition.

4.5.1 Motivation

Motivation 1 Current environments are primarily implemented in high-level languages. The consequence is that environments use additional CPU cycles, more computational power, which makes the environment run slower. Therefore, it is crucial to guide the future development of game environments towards more efficient methods. This contribution is motivated by the need for a standardized high-performance toolkit for building reinforcement learning research environments.

Motivation 2 Few works actively report the climate emission produced by environments or algorithms. This contribution motivates a standardized system to report climate

[10] CaiRL is open-source and freely available at https://github.com/cair/rl

[11] The full list of available game environments are found at https://flashrl.per-arne.no/

emissions for the RL experiments. Furthermore, the contributions aim to significantly reduce the CPU load of game environments to actively reduce the climate footprint, save training, and testing time.

Motivation 3 Many existing game environments are written in languages other than Python, making it challenging to train RL agents efficiently and seamlessly because of custom interoperability layers. This contribution motivates standardizing an interoperability layer between programming languages such as Java and C++.

Motivation 4 The Ph.D. work presents many new environments, namely FlashRL in Paper A, Deep Line Wars in Paper B, Deep Maze in Paper D, Deep RTS in Paper C, and Deep Warehouse in Paper E. This contribution motivates reproducing these environments collectively in a highly-efficient toolkit so that our contributions stand complete and accessible for future RL research.

4.5.2 Design Specifications

CaiRL uses C++ with highly performant fast-paths such as Single instruction, multiple data (SIMD) for vectorized calculation that fits into the processor registry in a single instruction. The design mimics OpenAI Gym but relies on templating and const expressions that enable calculations to evaluate at compile-time instead of runtime.

Module Layer

CaiRL has a modular design with little cross-dependencies between module categories to decrease compile times. CaiRL splits modules into three categories:

- spaces for action and observation definitions,
- environments for wrappers of supported environments,
- utilities for enriching the framework, such as the tournament module.

Figure 4.10 shows the CaiRL framework with a streamlined dependency structure, where the primary dependency lies in the CaiRL Env class. The CaiRL env class is the glue that binds together the environment definition and logic. The spaces module defines classes for different types of action and state spaces, e.g., n-dimensional matrices and one-hot vectors. The environment module is the collection of implemented games in CaiRL. The utility module contains many convenience tools for communicating with remote environments through Websockets and setting up tournaments for experimental setups. CaiRL environments are C++ code that compiles down to machine code and wraps to a Python-compatible binary using Pybind11.

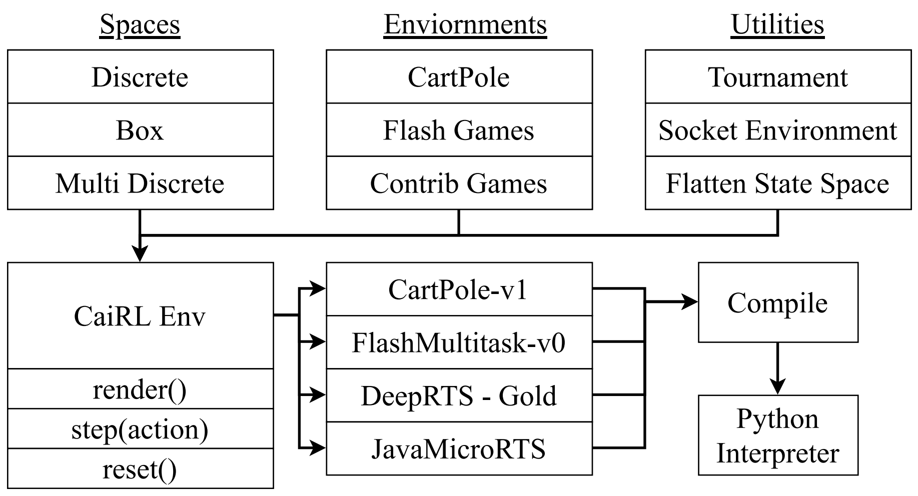


Figure 4.10: Brief overview over components in CaiRL. The framework splits code into modules similar to the OpenAI Gym framework: spaces, environments, and utilities. The modules interface with the CaiRL Env class, from which all environments derive. The environments compile to a shared library and are callable from python code.

Interaction Layer

There are two ways of building reinforcement learning experiments with CaiRL using C++ directly or through the Python-to-C++ bindings. CaiRL runs efficiently in both programming languages, but there is a computational cost to run the Python interpreted and translate calls between C++ and Python. The primary goal of the API design is to match the Gym API closely, enabling developers to choose CaiRL or OpenAI Gym for common game environments effortlessly.

Listing 4.1 Minimal Example of CaiRL-CartPole-v1 in C++

```
1  e =  Flatten<TimeLimit<200,CartPoleEnv>>()
2  for(int ep = 0; ep < 100; ep++){
3   e.reset();
4   int terminal, steps = 0;
5   while(!terminal){
6    steps++;
7    const auto [s1, r, t, info] =
8    e.step(e.action_space.sample());
9    auto obs = e.render();
10   terminal = t;
11  }
12 }
```

As seen in Listing 4.1, the C++ interface is similar to OpenAI Gym. The difference is that CaiRL modules use template classes, as seen in line 2. Templates in C++ define a class that can evaluate much of the program logic during compile-time. There are considerable runtime benefits because code initialization resolves during compile-time at the cost of longer compile times. Another downside of templates is reduced flexibility because it is impractical to create environment definitions during runtime, which is required when building Python-based environment wrappers. However, it is possible to define runtime-defined environments in Python at the cost of lower performance.

A very central component of CaiRL is the ability to run experiments natively in Python. This becomes possible through exposing C++ code through wrappers using the Pybind11 library. Pybind11 is a library that provides functions for translating Python to C++ system calls automatically (and vice versa). Using the Python wrapper code, there is no need for C++ knowledge, and it is possible to both use and extend CaiRL solely using Python. The Python interface is similar to the C++ interface but focuses more on compatibility with the OpenAI Gym interface.

Listing 4.2 Minimal Example of OpenAI and CaiRL CartPole-v1 in Python

```
1  #e =  gym.make("CartPole-v1")
2  e =  cairl.make("CartPole-v1") # Use CaiRL
3  for ep in range(100):
4  e.reset()
5  terminal, steps = 0
6  while not terminal:
7  steps++
8  a = e.action_space.sample()
9  s1, r, t, info = e.step(a)
10 obs = e.render()
11 terminal = t
```

Listing 4.2 illustrates the use of CaiRL in Python compared to OpenAI Gym. In particular, the only change that is required to switch between the two frameworks is to comment on Line 1 and comment out Line 2.

FlashRL

Paper A presents preliminary work on providing an RL platform for Flash Games and is further pursued in CaiRL. The first draft of this method uses external applications to execute the Flash runtime and uses a virtual frame buffer in combination with remote access via VNC. CaiRL and Paper M extend this work with the ability to run Flash games without requiring third-party applications, and compared to FlashRL, performs significantly better. CaiRL extends the LightSpark Flash emulator for Actionscript 3 and falls back to GNU Gnash for ActionScript 2. CaiRL features a repository of over 1300 Flash games for conducting AI and RL research.

Affordable and Sustainable AI

AI is a constantly growing field of research, and with the active focus on Deep Learning, the computational requirements increase sharply (Vinyals et al., 2019). Deep Learning models range from a few thousand parameters up to several billion parameters that require carefully tuning with algorithms such as stochastic gradient descent. Therefore, computing power is an essential part of achievable performance for trainable models. The same applies to Deep Reinforcement Learning, but it also requires live data sampling from an environment, making it more compute demanding. It is safe to conclude that running experiments becomes exponentially more expensive and is against more sustainable AI. CaiRL aims to minimize the cost of reinforcement learning by reducing environment execution time. In essence, this has the positive side-effect of reducing the carbon emission footprint in RL significantly compared to existing solutions and is presented further in Section 6.5.5.

Design Discussion

A graphics accelerator takes charge of evaluating program code into a graphical representation through a renderer. The graphics accelerator runs either as a software implementation such as program code or a hardware implementation such as graphics cards with specialized electronics for graphics rendering. It is natural to assume that hardware-accelerated graphics yield the most performance because of task specialization, but this is not always the case because the cost to copy the frame buffer from GPU to CPU memory is often more expensive in simple 2D graphics compared to the CPU render-time.

According to Mileff and Dudra, 2012, software rendering in modern CPU chips performs 2-10x faster due to specialized bytecode instructions. The study concludes that although GPU can render frames faster, provided that the frame permanently resides in GPU memory. Having frames in the GPU memory is impractical for machine learning applications because of the copy between CPU and GPU. The authors in Mendel and Bergström, 2019 propose using Single instruction, multiple data (SIMD) optimizations to improve game performance. SIMD extends the CPU instruction set for vectorized arithmetic to increase instruction throughput. The authors find that using SIMD instruction increases performance by over 80% compared to traditional CPU rendering techniques.

The findings in these studies suggest that software acceleration is beneficial in some graphic applications, and similarly, we find it useful in a reinforcement learning context. Empirically, we see that software rendering performs better for simple 2D and 3D graphic applications due to the high-latency copy operation needed between GPU and CPU. Much of the CaiRL performance gains lie in the fact that software rendering, while slower for advanced games such as StarCraft, significantly outperforms hardware rendering for simple graphics. One alternative to improve performance in hardware rendering is to use pixel buffer objects or equivalent implementations. A Pixel Buffer Object (PBO) is a buffer storage which allows the user to retrieve frame buffer pixels asynchronously while a new frame buffer draws to the screen frame buffer. In particular, copying pixels without PBO is slower because rendering must halt while the buffer is read.

CaiRL aims to increase the game repository over time and encourage submissions of new environments. For this reason, it is essential to recommend implementation methods that yield the highest performance. We find that software rendering is the best choice for simple 2D and 3D-based games, and in complex 3D games such as StarCraft II, the programmer should implement PBO-based rendering if it is possible to gain access to the rendering context.

4.5.3 Results

Performance Evaluation

The CaiRL toolkit primarily measures the performance increase compared to OpenAI Gym and quantifies the potential to reduce climate emissions in RL research. We report that DQN successfully beat the Multitask environment with early-stopping, as seen in Paper L, Figure 7. For performance gains of using CaiRL, Table 4.9 demonstrates the effectiveness compared to OpenAI Gym in training and execution time. We test classical RL environments such as CartPole, where CaiRL variants are code ports to C++. The CaiRL and OpenAI Gym columns present the execution time in milliseconds for 100 trials of DQN.

Table 4.9: The performance in execution-time of CaiRL, compared to OpenAI Gym. We test classical RL environments, namely CartPole, Acrobot, MountainCar, and Pendulum. CaiRL versions are reimplementations in C++. The CaiRL and OpenAI Gym columns present the execution time in milliseconds for 100 trials of DQN.

Environment	**CaiRL**	**OpenAI Gym**	**Source**
CartPole-v1	**37K**	49K	Paper L
Acrobot-v1	**63K**	86K	Paper L
MountainCar-v1	**41K**	89K	Paper L
Pendulum-v1	**46K**	87K	Paper L

Carbon Emission Evaluation

The last decade of machine learning innovations has become increasingly more computational heavy with the inception of deep learning models. Machine learning becomes a

Table 4.10: The DQN hyperparameters for the carbon emission experiment

Hyperparameter	**Value**
Discount	0.99
Units	32, 32
Activation	ELU
Optimizer	Adam
Loss Function	Huber
Batch Size	32
Learning Rate	3e-4
Target Update Freq	150
Memory Size	50 000
Exploration Start	1.0
Exploration Final	0.01

more substantial problem for the climate because of the carbon emissions of the power draw. It is possible to quantify carbon emissions using CO_2-equivalent (CO_2-eq), a standardized measure used to express the global-warming potential of various greenhouse gases as a single scalar value, which we use in our estimations.

This section addresses the following question:

Question: *Is CaiRL a better alternative for lowering carbon emissions in RL research than OpenAI Gym?*.

To begin answering this question, we run experiments with the novel experiment-impact-tracker from Henderson et al., 2020b. The experiment-impact-tracker is a drop-in method to track energy usage, carbon emissions, and compute utilization of the computer system. The authors propose and encourage researchers to create more sustainable AI. Our experiments run a DQN agent on the classical control environment CartPole-v1 in CaiRL and OpenAI Gym. We compare toolkits using the console-only version and the graphical variant. The experiment runs 1 000 000 times in the console version and 10 000 times in the graphical version.[12]. Finally, the DQN parameters are shown in Table 4.10.

Table 4.11 shows that CaiRL has a considerably lower carbon emission compared to OpenAI Gym. CaiRL has 20.89 times less carbon emission in the console variant than OpenAI Gym. The graphical experiment shows a more significant difference with over 147 578 times lower carbon emissions throughout the test period. OpenAI Gym has high emission rates because the frame buffer draws to screen before the frame is accessible to system

[12]The experiment code is accessed at https://github.com/cair/rl/blob/main/cairl/examples/benchmarks/cartpole_v0.py

Table 4.11: Description of the total carbon emission values and power consumption used during the experiments. The carbon emission is measured in CO2/kg, and the power draw is measured in milliwatt-hour (mWh).

Measurement	**Environment**	**CaiRL**	**Gym**	**Ratio**
CO2/kg	Console	**0.000014**	0.000067	20.8955
CO2/kg	Graphical	**0.000051**	0.075265	147578.431373
Power (mWh)	Console	**0.000319**	0.001483	21.5104
Power (mWh)	Graphical	**0.001131**	1.673959	148006.9849

memory. The emission rate measurements subtract the DQN training time usage with the total time used to only account for the environment runtime costs, effectively excluding the DQN training time. We outline the DQN hyperparameters in Table 4.10. Based on our empirical experiments, we find CaiRL is a better alternative for lowering carbon emissions in RL research than OpenAI Gym.

4.5.4 Summary

CaiRL is a novel platform for RL research and aims to reduce experiment execution time to reduce budget costs and carbon emissions. CaiRL outperforms OpenAI Gym implementations significantly while also being compatible with existing OpenAI Gym codebases. We demonstrate how CaiRL is used, and we draft recommendations for defining graphical interfaces in games to reduce rendering time. Furthermore, we illustrate that CaiRL supports many programming languages, including C++, Java, Python, and ActionScript 2 and 3. CaiRL supports over 1000 games in ActionScript, Several C++ games, Micro RTS, and Showdown in Java and supports OpenAI Gym Python games out of the box. In the evaluations of CaiRL, we demonstrate superior performance and a positive reduction in the carbon footprint when training Q-Learning agents compared to OpenAI Gym.

4.6 Summary

This chapter contributes five novel environments to the body of science, specifically Deep Line Wars, Deep Maze, Deep RTS, Deep Warehouse, and CaiRL. Table 4.12 outlines the resulting source code. The primary motivation for creating a comprehensive test bench is that existing literature has complexity or performance gaps.

Table 4.12: An overview of the source-code repositories of our environment contributions. Additional code can be found at https://github.com/topics/per-arne

Environment	**Paper**	**Location**
Deep RTS	Paper C	https://github.com/cair/deep-rts
CaiRL	Paper L	https://github.com/cair/rl
FlashRL	Paper A	https://github.com/cair/FlashRL
Deep Line Wars	Paper B	https://github.com/cair/deep-line-wars
Deep Line Wars 2	Paper B	https://github.com/cair/deep-line-wars-2
Deep Maze	Paper D	https://github.com/cair/deep_maze
Deep Warehouse	Paper M	https://github.com/cair/deep-warehouse

4.6.1 Game Complexity

Figure 4.11 shows a comprehensive chart of state-space complexity in our game contributions and other well-known environments. For example, the simplest environments in StarCraft have a state-space from 10^{1685} to 10^{36000} (Ontanon et al., 2013). The state-space lower bound calculation is written $10^{\log(||\text{tile-combinations}^{\text{width*height}}||)}$, where width and height denote absolute logarithm of the multiplied map dimensions, tile-combinations for the number of placable units. Micro RTS has similar estimates as the Deep RTS environment because it features seven different units, building types, and dynamic map size. The problem with Micro RTS is that it runs only in Java and is significantly slower than Deep RTS. Because of a considerable state complexity in the environments, we can only provide conservative lower bounds, but we expect the upper bound to be significantly higher.

Case Study: ASR Systems

Following the study of Beckschäfer et al., 2017, we compare the game environments in Figure 4.11 to real-world ASRS complexity. The authors study Grid-based Warehouse systems, similar to Deep Warehouse, specifically testing hand-crated algorithms. While the authors do not estimate the state-space complexity of their problem, we can derive a lower bound from their results and written descriptions. For example, if one considers a 20x20 grid with 4987 bins, the grid height is approximately 13 squares. Assuming that every bin is unique, a static grid's state-space complexity is $13^{(20^2)} =\sim 10^{445}$. Additionally, we can compute the number of sorting and pickup robots, which is a large part of the surface plane configuration:

$$\frac{(\text{width} \times \text{height})!}{((\text{width} \times \text{height}) - \text{robots}) \times \text{sorting-robots}! \times \text{pickup-robots}!}. \tag{4.1}$$

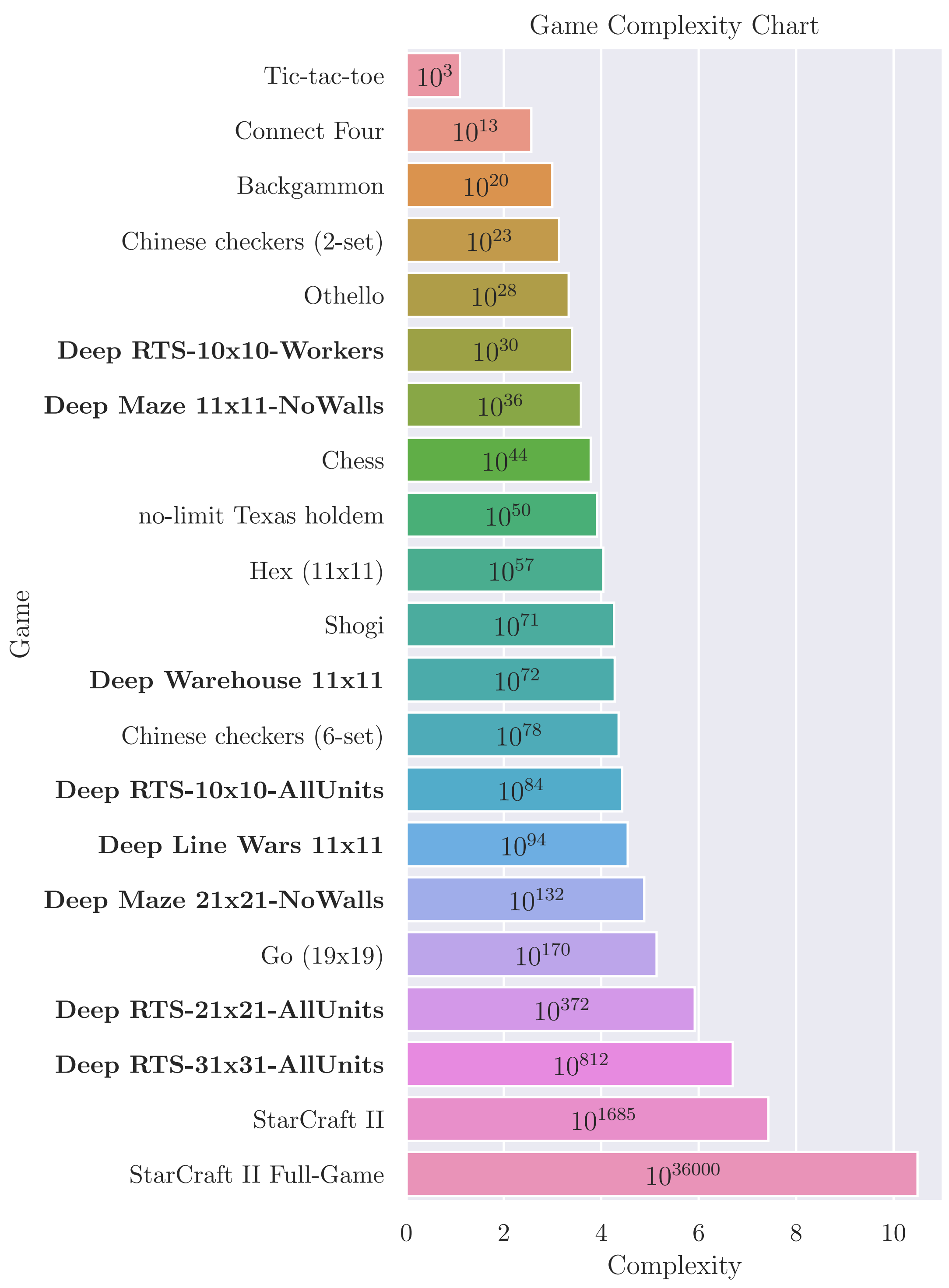


Figure 4.11: Estimations of the state-space complexity in a range of games. The y-axis is in logarithmic scale, and the centered bar-text is the state-space complexity. State-space complexity is the number of state combinations that exist for the game. The bold items are our contributions where Deep RTS is from Paper C, Deep Line Wars is from Paper B, Deep Maze Paper D, and Deep Warehouse is from Paper E.

Following this equation, we find that the state-space lower bound for ten pickup robots and ten sorting robots in a 20x20 grid with 4987 bins is $\sim 10^{445} + 10^{38} \sim 10^{445}$, which is comparable to the Deep RTS-21x21-AllUnits environment. Consequently, on the extreme, Autostore installations can reach a grid size of $255 \times 255 \times 19$ and operate over 400 concurrent robots. Such a system has a lower bound state-space estimation of $\sim 10^{83150} + 10^{1120}$, which exceeds the StarCraft II Full-Game environment.

4.6.2 Performance

Table 4.13: FPS comparison of selected environments. For environments with 60 or 144 FPS, it is difficult to (1) uncap the game loop or (2) render above the screen refresh rate.

Environment	Frame per second	Source
ALE	6,500	Bellemare et al., 2013
Malmo Platform	60-144	Johnson et al., 2016
ViZDoom	8,300	Kempka et al., 2016
DeepMind Lab	1,000	Beattie et al., 2016
OpenAI Gym	60	Brockman et al., 2016
Stratagus	60-144	Ponsen et al., 2005
Micro RTS	11,500	Ontanon, 2013
TorchCraft	2,500	Synnaeve et al., 2016
ELF	36,000	Tian et al., 2017
SC2LE	60-144	Vinyals et al., 2017
Deep RTS	24,000-7,000,000	Paper C
Deep Warehouse	110 000-1 551 000	Paper E
Deep Maze	530 000-956 000	Paper D
Deep Line Wars 1	200 000-400 000	Paper B
Deep Line Wars 2	5 000 000-25 000 000	Paper B

There is a substantial difference between the performance in games targeted research and those aimed towards gaming. Table 4.13 shows that the frame-rate difference ranges from 60 to 25 000 000 for selected environments. A high frame rate is essential because some exploration algorithms often require a quick assessment of future states through forward-search. Some of the existing environments (e.g., Micro RTS, TorchCraft, ELF, and SC2LE) have good FPS but cannot compete with the framerate of our contributions. There are various reasons for lower FPS, but the most prominent reason is that the game environments have frame and game-loop locks that were difficult to disable. Otherwise, games like StarCraft II (SC2LE) have a high computation requirement. Our environments dominate the FPS counter because all environments use software rendering to prevent frame buffer copies and are written in efficient low-level C++.

Chapter 5

Algorithm Contributions

This chapter introduces three novel reinforcement learning algorithms, the Dreaming Variational Autoencoder (DVAE), Deep Variational Q-Networks (DVQN), the Observation Reward Action Cost LEarning *Ensemble* (ORACLE), and variations of these. We present the algorithms ordered by contribution date. ORACLE and DVAE aim to improve sample efficiency and increase the safety in RL algorithms following a model-based approach. DVQN move towards interpretability of latent spaces and latent space clustering for multi-policy hierarchical RL algorithms. The aim is to create an RL algorithm that has improved safety, is more sample efficient, improves towards the larger goal of solving industry-grade, safety-critical systems, and succeeds in playing RTS games.

The presentation of algorithm contributions is structured as follows. Algorithms are introduced, following a description of the algorithm and motivations to improve such algorithms. We thoroughly present the algorithm's architecture and theory, which builds momentum for the evaluations in Chapter 6. This chapter presents research from Paper D, Paper E, Paper F, Paper G, Paper H, Paper I, Paper J, Paper K, and Paper M.

5.1 Dreaming Variational Autoencoder (DVAE)

The Dreaming Variational Autoencoder (DVAE) is a model-based reinforcement learning approach for safe and efficient learning. In model-based RL, the goal is to learn a behavioral policy using a known or unknown dynamics model. A dynamics model describes the transition probabilities of the environment $\mathcal{T}(s_{t+1}, r_t | s_t, a_t)$ (See Section 2.2.4). This work considers the model unknown and must be learned before deriving a behavioral policy. While model-based RL is traditionally risk-neutral, it can act as a safety precaution because it can potentially learn good strategies fully offline without exploring the ground truth environment and is the driving motivation behind the DVAE algorithm.

5.1.1 Motivation

Motivation 1 Recent studies in the literature concurrently conclude that model-based approaches have significantly better sample efficiency. This motivates further study on solving RTS games (Walsh et al., 2010). Paper A, Paper B, and Paper C focuses on model-free methods, and the conclusion from our studies is that these methods require immense computational power to learn policies that accomplish good performance (Thompson et al., 2020).

Motivation 2 Industry applications primarily employ algorithms designed by hand. It is natural to assume that it is difficult to achieve global optimality in complex systems with many dynamic variables. RL learns fully autonomous, and recent literature demonstrates above human performance in many game environments, at the cost of extensive trial-and-error (Mnih et al., 2015). Industrial systems are often costly in production, making it difficult to train algorithms directly. These concerns motivate research towards methods that learn the dynamics of the environment by observing existing expert systems in live industry installations. The learned dynamics can then fuel the learning of RL algorithms fully offline.

Motivation 3 Most RL algorithms are risk-neutral, meaning they do not aim to learn objectives with minimal failure. In extension to Motivation 2, this contribution motivates to improve the safety of DVAE to make RL more applicable to industry applications.

5.1.2 The Dreaming Variational Autoencoder

Algorithm 1: The Dreaming Variational Autoencoder - Observe Routine

```
1  Function DVAE-Observe($\mathcal{T}$, $\mathcal{N}$) : $\mathcal{D}$
       Init: Experience-Replay $\mathcal{D}$ to capacity $\mathcal{N}$
2      for i = 0 to N_EPOCHS do
3          Observe starting state, $s_0 \sim \mathcal{N}(0, 1)$
4          while $s_t$ not TERMINAL do
5              $a \sim \Omega_\theta(\mathcal{S} = s_t)$ ;                       // Observe action
6              $s_{t+1}, r_t, t_t = \mathcal{T}(s_t, a_t)$ ;              // Observe transition
7              $\mathcal{D}(s_t, a_t, r_t, s_{t+1}, terminal)$ ;            // Store experience
8              $s_t \leftarrow s_{t+1}$
9          end
10     end
       Return: $\mathcal{D}$
11 end
Assumption: Expert-system $\Omega$
```

DVAE is an end-to-end solution for predicting probable future states $\hat{s}_{t+n}$ where *hat* denotes a predicted state, t is the timestep, and n is the nth prediction. The goal is to predict states from an arbitrary state-space $\mathcal{S}$ using state-action pairs explored prior to s_{t+n} and

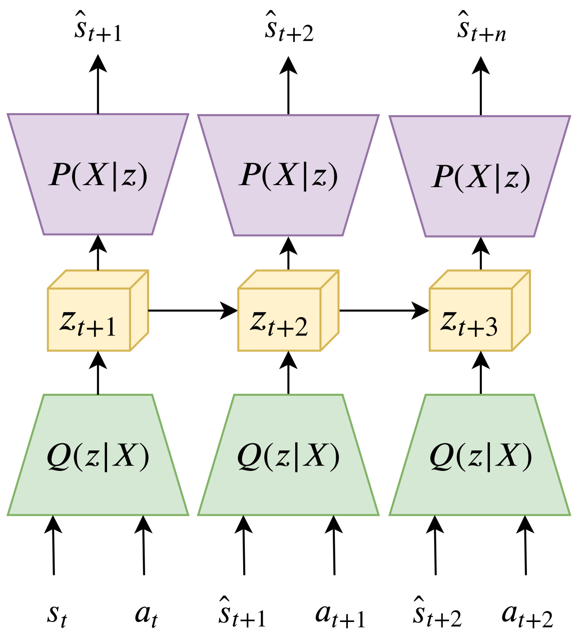


Figure 5.1: A graphical model of the DVAE model. The model takes state s_t (for POMDP's, $s_t = o_t$) and action a_t pairs as input and outputs the encoded latent space variables z_t. The latent space variables represent a compressed representation of the transitioned state (next state) $\hat{s}_t$. The decoder network takes the future state latent variables z_{t+1} and outputs a visual representation of the future state. $\mathcal{Q}(z_t|X)$ is the encoder, z_t is latent space variables, and $\mathcal{P}(X|z_t)$ is the decoder. DVAE can also use LSTM's to improve prediction towards longer time horizons.

Algorithm 2: The Dreaming Variational Autoencoder - Train Routine

```
   Init: Artificial Experience-Replay $\hat{\mathcal{D}}$ to capacity $\mathcal{N}$
   Init: Encoder $\mathcal{Q}(z_t|s_t, a_t; \theta)$
   Init: Decoder $\mathcal{P}(\hat{s}'|z_t; \theta)$
   Init: DVAE model $\hat{\mathcal{T}}_\theta(\hat{s}', \hat{r}_t|s_t, a_t) = \mathcal{P}(\hat{s}'|z_t)|\mathcal{Q}(z_t|s_t, a_t))$
1  Function DVAE-Dynamics-Train($d_t$): $\hat{\mathcal{T}}$
2      $s_t, a_t, r_t, s_{t+1} \leftarrow d_t$ ;                        // Expand ER sample
3      $z_t \leftarrow \mathcal{Q}(s_t, a_t)$ ;                                // Encode $X$
4      $\hat{s}' \leftarrow \mathcal{P}(z_t)$ ;                  // Decode $z_t$ into future state
5      Optimize model using stochastic gradient descent
6  end
7  Function DVAE-Dynamics-Sample($\mathcal{D}$): $\hat{\mathcal{D}}$
8      for $d_i$ in D do
9          $s_t, a_t, r_t, s_{t+1} \leftarrow D_i$ ;                    // Expand ER sample
10         $X = s_t, a_t$
11         $z_t \leftarrow \mathcal{Q}(X)$ ;                                   // Encode $X$
12         $\hat{s}' \leftarrow \mathcal{P}(z_t)$ ;              // Decode $z_t$ into future state
13         $\hat{\mathcal{D}}(\hat{s}_t, a_t, r_t, \hat{s}', terminal)$;           // Store experience
14         $\hat{s}_t \leftarrow \hat{s}'$
15     end
       Return: $\hat{\mathcal{D}}$
16 end
17 while DVAE-Dynamics model is not trained do
18     for $d_i$ in D do
19         Train dynamics model DVAE-Dynamics-Train($d_t$)
20     end
21 end
22 while Model-free algorithm is not trained do
23     Choose action $a_t$ from policy $\pi_\theta(a_t|s_t)$
24     Execute $a_t$ at state $s_t$ using dynamics model $\hat{\mathcal{T}}_\theta(\hat{s}', \hat{r}_t|s_t, a_t)$
25     Perform algorithm specific policy update $\pi_{\theta'} \leftarrow \pi_\theta$
26 end
   Outcome: Learned dynamics model $\hat{\mathcal{T}}_\theta$
27 Hyperparameters: See Table 5.1.
```

a_t. Figure 5.1 depicts a simplistic graphical model of the DVAE model. A state s_t and action a_t is input to the encoder $Q(z|X)$, which predicts the latent state of the environment dynamics $z_t \dots z_{t+n}$. Then, using the latent state and action can transition the state to $z_t + 1$, and so forth. The decoder $P(X|z)$ reconstructs the observed state and is part of the reconstruction objective of the model.

The algorithm splits into two steps seen in Algorithm 1 and Algorithm 2. The observation algorithm works as follows.

Line 1 The DVAE-Observe function takes the environment $\mathcal{T}$ and Experience Replay (ER) buffer with capacity $\mathcal{N}$ as input arguments. The function returns the ER buffer $\mathcal{D}$ on completion. The ER buffer initializes the capacity to $\mathcal{N}$.

Line 2-3 Iterate for N_EPOCHS (hyperparameter) and every epoch observe the initial starting state of the environment

Line 4-9 (L4) While the environment is not in a terminal state. (L5) Let the expert-system Ω sample actions, (L6) Let the environment transition to s_{t+1}, and (7) store the state-observation, action, reward, and terminal state in ER buffer $\mathcal{D}$.

The training algorithm (Algorithm 2) works as follows.

Init Initialize weights θ for encoder and decoder in the DVAE model $\hat{\mathcal{T}}$.

Line 1-7 (L1) Define the *DVAE-Dynamics-Train* function that takes an ER sample as input and returns the trained dynamics model $\hat{\mathcal{T}}$ (Equation 2.13). (L2) Unpacks the ER sample and (L3) inputs to the encoder. The decoder predicts the estimated future state $\hat{s}'$. The model updates weights using stochastic gradient descent.

Line 7-16 (L7) Define the *DVAE-Dynamics-Sample* function that uses the ER buffer from Algorithm 1. (L8) For all samples in $\mathcal{D}$, (L9) expand the tuple (L10) and encode state s_t and action a_t to X. (L11) Encode X to the latent state variable z_t and (L12) decode to the estimated future state $\hat{s}'$. (L13) Store the estimations in the ER buffer along with ground truth reward, action, and terminal state. (L14) update state variable.

Line 17-21 Train the dynamics model using samples from the ER buffer until the model has *sufficiently low* prediction error. The error function is problem dependant and is defined separately.

Line 22-26 Using the trained dynamics model, follow the standard training scheme of RL algorithms but query the dynamics model in place of the ground truth environment. Update the policy accordingly.

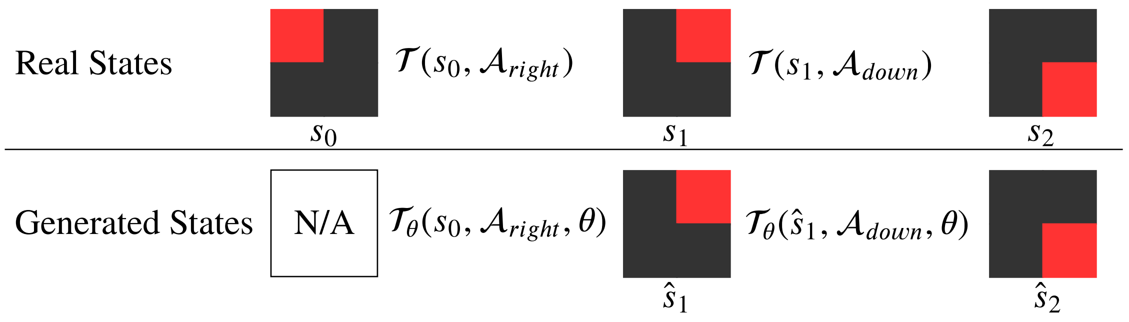

Real States
$\mathcal{T}(s_0, \mathcal{A}_{right})$
$\mathcal{T}(s_1, \mathcal{A}_{down})$
s_0
s_1
s_2
Generated States
N/A
$\mathcal{T}_\theta(s_0, \mathcal{A}_{right}, \theta)$
$\mathcal{T}_\theta(\hat{s}_1, \mathcal{A}_{down}, \theta)$
$\hat{s}_1$
$\hat{s}_2$

Figure 5.2: DVAE algorithm for generating states using $\hat{\mathcal{T}}_\theta$ versus the real transition function $\mathcal{T}$. First, a real state is collected from the replay-memory. DVAE can then produce new states from the current trajectory τ using the state-action pairs. θ represent the trainable model parameters.

Figure 5.2 illustrates how the algorithm generates sequences of artificial trajectories using $\hat{\mathcal{T}}_\theta = \mathcal{P}(X|\mathcal{Q}(z|X))$, where $z = \mathcal{Q}(z|X)$ is the encoder, and $\hat{\mathcal{T}}_\theta = \mathcal{P}(X|z)$ is the decoder. With state s_0 and action $\mathcal{A}_{right}$ as input, the algorithm generates state $\hat{s}_1$, which in the table is similar to the real state s_1. With the next action input, $\mathcal{A}_{down}$, DVAE generates the next state $\hat{s}_2$ where the goal is that DVAE the prediction is equal to s_2. Note that this is without ever observing state s_1. Therefore, the DVAE algorithm needs initialization with the initial state, s_0. The DVAE algorithm primarily uses a one-step prediction scheme but can optionally serve multi-step predictions, discovered concurrently by Hafner et al., 2020.

For DVAE to perform well, the requirement is that the environment has coherent and sufficient observability to make learning the hidden dynamics possible. In other words, the environment cannot be hidden like the pong example that we exemplify in Section 2.1.1. DVAE can predict trajectories of imaginary data using one-step predictions by calling the transition function recursively. $\tau = \hat{s}_1, a_1, \hat{s}_2, a_2, \hat{s}_3, a_3 = \mathcal{T}_\theta(\mathcal{T}_\theta(\mathcal{T}_\theta(s_0, \mathcal{A}_{a_0}), \mathcal{A}_{a_1}), \mathcal{A}_{a_2})$, is a example of 3-step trajectory. According to empirical evidence, the DVAE algorithm can predict imaginary data accurately for approximately 3-timestep. However, compounding errors makes it difficult to predict longer state sequences, as demonstrated in evaluations in Section 6.1. In addition to the original DVAE model, several contributions extend the model to better support complex environments, longer horizon predictions, and safety in Paper F (Generative Adversarial Networks and Stochastic Weight Averaging), Paper G (Safety), and Paper H (CostNet)

5.1.3 Generative Adversarial Networks

Figure 5.3 illustrates the DVAE-GAN extension. This model introduces the *generator* G and *discriminator* D. The motivation follows an adversarial approach proposed in

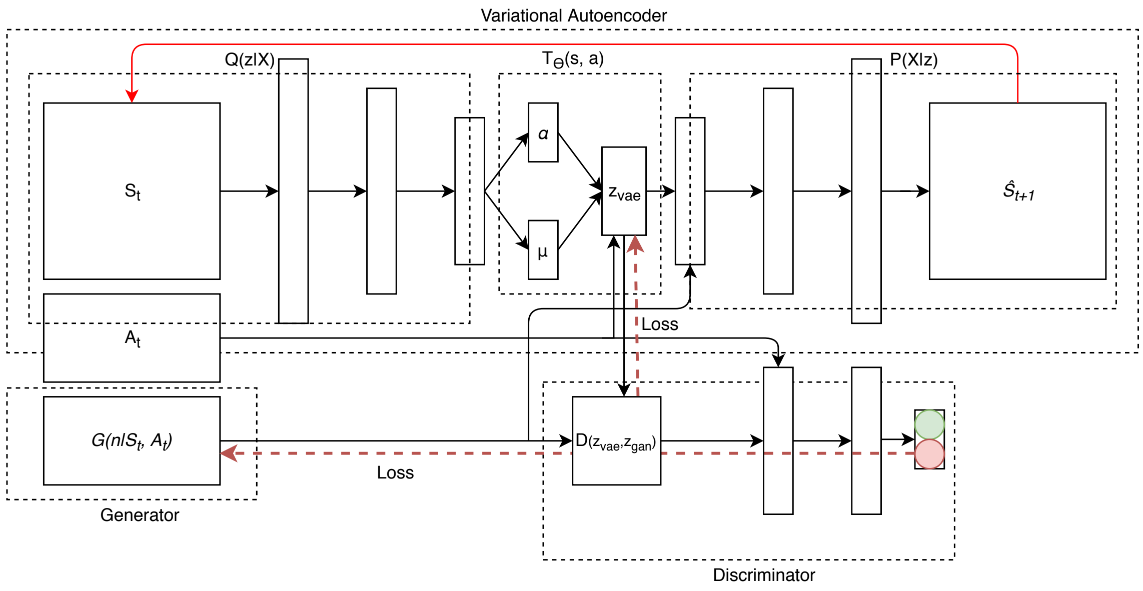


Figure 5.3: The proposed DVAE-GAN architecture. While the original VAE architecture persists, as seen in Figure 5.1, a new generative adversarial networks component is added for increased generalization across the latent space.

Makhzani et al., 2015. The generator $G(n|S_t, A_t)$ generates samples from Gaussian distributions, conditioned on current state and action to predict the latent space distribution z_{gan}. The discriminator $D(z_{vae}, z_{gan})$ is a neural network that predicts the validity of the input, in this case, if the latent space variable is from the ground-truth distribution. A min-max game between the generator and the discriminator fuels learning. The generator minimizes its error towards the real latent space, and the discriminator learns to distinguish between the real and fake latent distributions. The algorithm samples from the VAE latent space distribution model $z_{vae} = \alpha_t + (\mu_t \times N(0,1))$. The discriminator evaluates z_{vae}'s authenticity, updates weights and biases according to the confidence in the prediction. The loss function of the discriminator is combined with the original VAE loss, which empirically demonstrated better stability of the VAE, reducing posterior collapse frequency drastically.

5.1.4 Stochastic Weight Averaging

SWA is a novel approach to ensemble learning. The objective is to widen the global optima space to increase the chance of finding and persisting during stochastic gradient descent updates. The author claims that it trivializes the optimization of the optimization problem and improves the model generalization (Izmailov et al., 2018).[1] Compared to other ensemble learning techniques, SWA only requires a single model where snapshots

[1]The author further demonstrated effectiveness in *Improving Stability in Deep Reinforcement Learning with Weight Averaging*, but the work is not peer-reviewed. However, Paper F demonstrates similar findings.

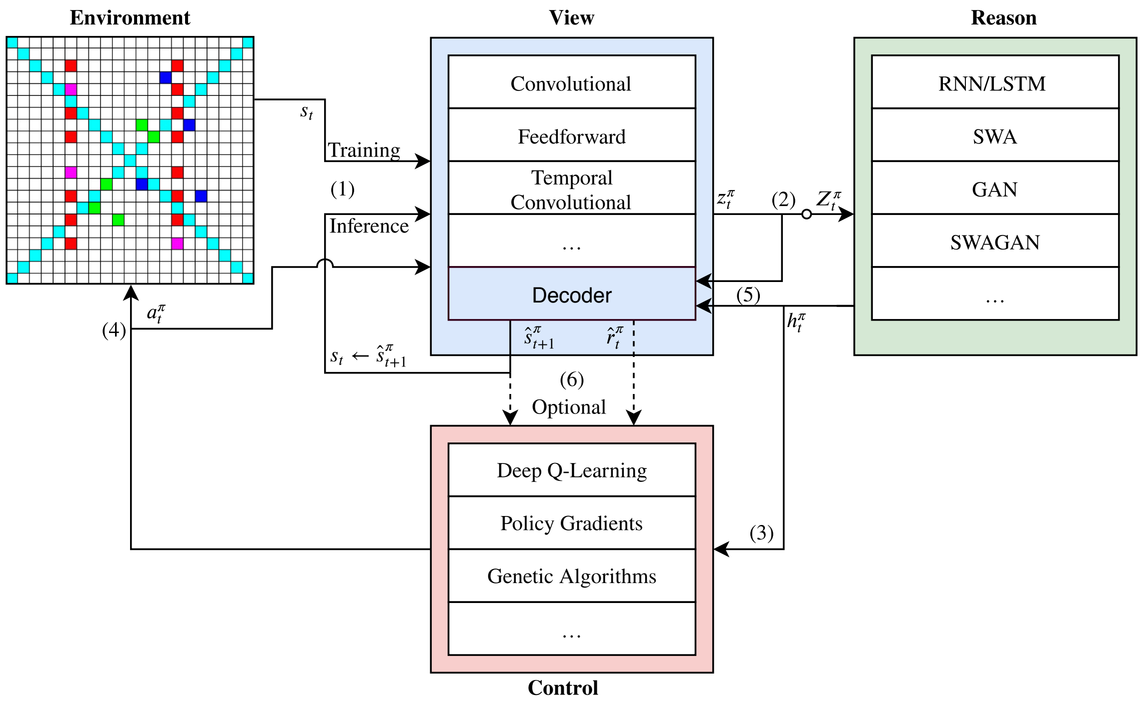


Figure 5.4: The view-reason-control (VRC) architecture of the module-based DVAE variant. The model compresses an input state s_t in the view module before evaluating the time-dependant latent variables h_t in the reasoning module. The latent variables evaluate actions used to transition the environment to the next timestep. Additionally, the DVAE can train without real environment interactions by directly sending predicted state and action to the view module, effectively starting a new timestep.

are stored every n epochs that are averaged every m epochs. We use a cyclical learning rate and average the weights for each training iteration, creating the DVAE model (Smith, 2015). SWA is sensible because the compounding error for predicting longer horizons increases variance and generally decreases model stability. Due to the larger optima space, it is more likely that the algorithm recovers from local optima during training.

5.1.5 Module-Based DVAE Architecture

The original DVAE architecture from Paper D has several challenges with modeling complex state spaces. We add several extensions to the model to address these challenges to improve performance across various environments. These extensions include VAE's, LSTMs, and fine-tuned variations. Paper E generalizes the model into three individual modules, forming the View, Reason, and Control (VRC) model. The VRC model embeds all improvements into a single model and learns which algorithms to use under certain conditions in an environment using grid-search strategies. Figure 5.4 shows an overview of the proposed VRC where the in-figure numbering represents the following:

1. A state s_t is observed. This observation stems from the real environment and the dynamics model during training at inference time. The observation is encoded in the view component (e.g., via AE or GAN) and outputs an embedding z at time t w.r.t policy π.

2. The reason component learns the time dynamics between state sequences. Encoded states are accumulated into a buffer $Z_t = \{z_{t-n} \dots z_t\}$ and are then used to predict the hidden-state h_t w.r.t the encoded state sequence. The reason component typically consists of a model with an RNN-like structure that generalizes well on sequence data.

3. The hidden state is then used to evaluate an action using policy π, and

4. is sent to the environment and the view for the next iteration.

5. The decoder prepares the hidden-state h_t and encoded state z_t, producing the succeeding state $\hat{s}_{t+1}$. The prediction is then used in the next iteration as current state s_t, leading to (1). As an optional mechanism, the controller can use the output from the decoder instead of the hidden state information. This is beneficial when working with model-free algorithms such as DQN (Mnih et al., 2015).

5.1.6 Safe Dreaming

The Safe Dreaming Variational Autoencoder (S-DVAE) aims to increase agents' safety in environments with catastrophic state outcomes. S-DVAE is a model-based RL approach for safe and efficient learning. S-DVAE learns a dynamics model similar to DVAE but emphasizes using an expert system to learn the dynamics model. The algorithm models the problem as a CMDP (Section 2.3.2) with a combination of risk-directed exploration and curiosity.

A dynamics model learns the transition dynamics of the real environment. The model gathers experience by observing an expert system to learn these transitions. Expert systems usually exist in industrial applications, requiring minimal effort to train the dynamics model. Also, the expert system has a high likelihood of already making safe decisions but often operates with sub-optimal performance. Therefore, RL algorithms are well equipped for decision-making in industry-near environments towards improving performance and safety with expert system guidance.

Training model-free RL algorithms on top of the DVAE model is safe because it is no longer required to balance the exploration-exploitation trade-off. It is also sample efficient because it only requires training on environmental samples during the dynamics model learning. Deep Q-Networks from 2015 is well suited because it is an off-policy

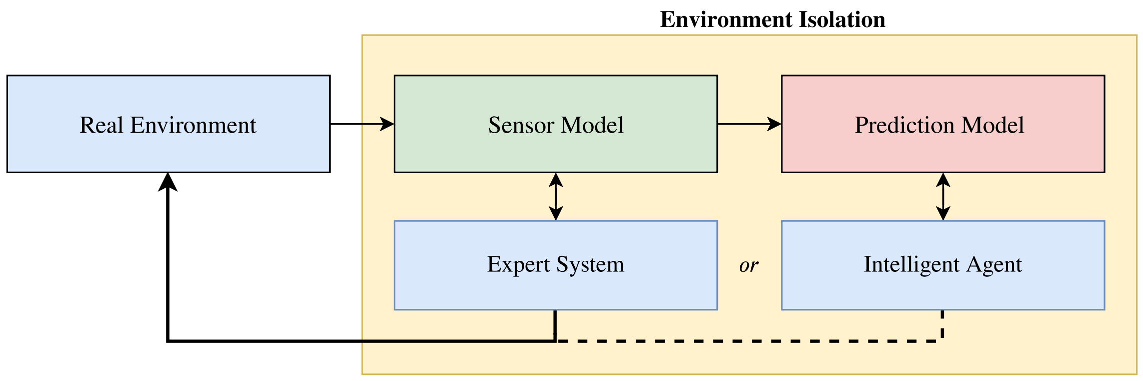


Figure 5.5: Learning Strategy. The general idea of S-DVAE is to isolate the agent training to reduce the risk of catastrophic behavior in the real environment. The dynamics model observes the sensors of the real environment and estimates its transition function. The intelligent agent uses the dynamics model to train in an offline setting without the risk of making mistakes in the real world. After training, the algorithm is deployed to the real environment, with significantly less chance of entering catastrophic states.

algorithm with convergence guarantees. Therefore, combining the S-DVAE algorithm and model-free algorithms ensures that learning is performed safely without the risk of entering catastrophic states, or causing damage to the real-world environment.

Training duration depends on the problem and should rely on some mechanic to determine the learning stopping criteria. As the algorithm learns an optimal policy for the dynamics model, it gradually transitions to make actions in the real environment, based on the rate of catastrophic states. At such a time when the agent interacts directly with the real environment, it is possible to enter catastrophic states. The algorithm combats exploration in uncertain state-space regions using curiosity. Several methods in literature use curiosity to combat local optima policies but can similarly provide a mechanism to quantify nearby states' uncertainty and, therefore, avoid such states efficiently (Chiang et al., 2019). We name the strategy negated curiosity and is a reward bonus to reduce the exploration of state-space regions with high uncertainty. This way, the fully deployed algorithm will behave cautiously when the movement towards novel states appears or if the environment is changed dynamically.

The training procedure illustrated in Figure 5.5 works as follows. (1) The dynamics model observes and learns the real environment using a sensor model. The same sensor model is the expert system's interface for decision-making. (2) The intelligent agent (e.g., an RL agent) interacts with the dynamics model and improves the behavioral policy. (3) At the point where the intelligent agent is sufficiently trained, it can replace existing expert systems with comparable safety and performance. (4) If desirable, the intelligent agent can train further in the real-world environment to potentially improve reward performance.

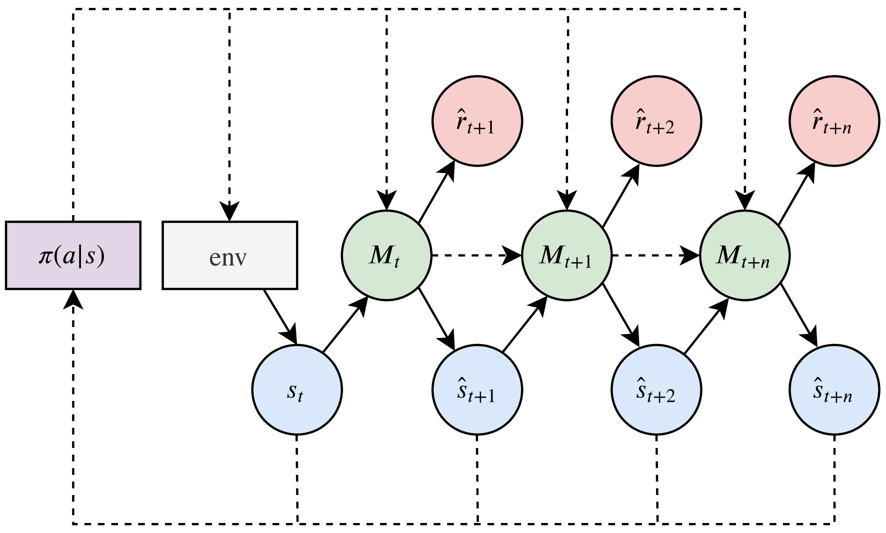


Figure 5.6: Unrolling predictions for S-DVAE. The agent observes a state from the environment or the dynamics model for each timestep. Furthermore, the agent makes an action that transitions to the next state with the corresponding reward. M denotes the dynamics model where $\hat{s}$ and $\hat{r}$ is the predicted state and reward.

The execution of S-DVAE is shown in Figure 5.6 and works as follows. The policy $\pi(a|s)$ predicts action a for the observed state s. The first action is sent to the real environment to produce an initial state $s_t = s_0$. The dynamics model M process the initial state s_t and initial action a_t and outputs predicted future state $\hat{s}_{t+1}$ and reward $\hat{r}_{t+1}$. The reward is used for policy updates during training and the state for further action prediction. The policy predicts action $\hat{a}_{t+1}$ and evaluates the next-step state $\hat{s}_{t+2}$. The procedure continues until the algorithm meets the stopping criteria.

Figure 5.7 shows the S-DVAE architecture. The s-encoder transforms raw input data into a meaningful and compact feature embedding (latent variables). S-DVAE uses VAE's primarily for this task, but other methods are also applicable, such as generative adversarial networks (GAN). Depending on the environment and the input data, it is possible to visualize the embedding $z_x \in \mathbb{Z}$ and manually tune the latent variable values.

The t-encoder learns the transition function $\mathcal{T}\colon \mathcal{S} \times \mathcal{A} \to \mathcal{S}$. The t-encoder model computes the future state embedding z_{t+1} based on previous latent space variables from the view module $Z_t = \{z_{t-n} \dots z_t\}$. The π denotes the policy under which S-DVAE operates. S-DVAE shows empirically that LSTM's performs best for learning future state latent variables.

The control policy $\pi(s|a)$ is responsible for interaction with the environment and the dynamics model (s-encoder and t-encoder). The control is the primary module for making safe actions and progressing the learning in the right direction. S-DVAE uses DQN with optimization constraints, risk-directed exploration, and negated curiosity. The negated

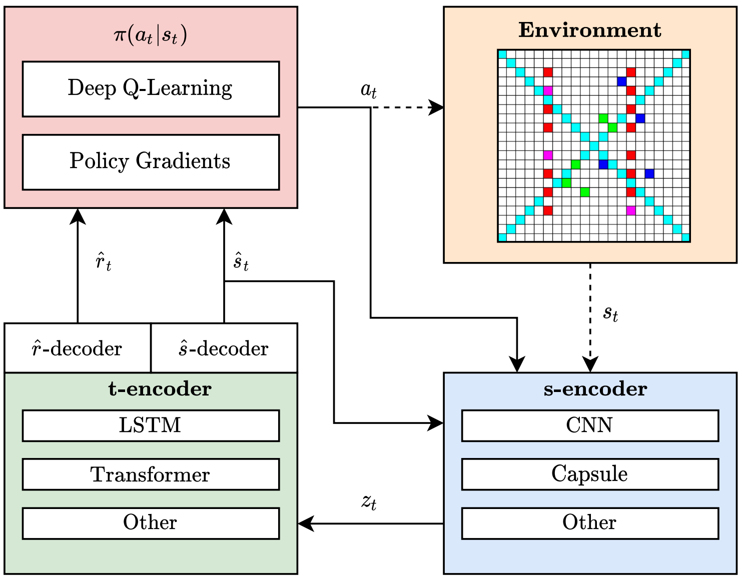


Figure 5.7: Detailed architecture for S-DVAE. Dotted lines illustrate that the procedure is optional. The algorithm is modular so that compatible algorithms and schemes are usable and problem independent. The s-encoder transforms raw input data into a meaningful and compact feature embedding. The t-encoder evaluates the temporal relationship between latent space variables. The S-DVAE architecture is similar to Figure 5.4, but algorithms ($\pi(a_t|s_t)$) have additional safety constraints.

curiosity act as the constrained criterion for the MDP. The input to the algorithm is a raw-state, commonly a high-dimensional data structure. The benefit of the S-DVAE architecture is that the t-model finds latent variables that represent the state with the order of magnitudes less complexity. S-DVAE also enables initial training fully offline in a *dream* version of the real environment.

Exploration and policy update constraints

There are significant improvements to S-DVAE, compared to DVAE for exploration and policy updates to find safer policies. S-DVAE uses risk-directed exploration (Edith et al., 2005) by modifying the action-selection strategy, seen in Equation 2.18. The policy updates have a set of constraints, omitting particular updates to prevent the algorithm from moving towards unsafe behaviors.

The algorithm receives feedback (rewards) during learning from the real-world environment. All actions in the action-set have some feedback, although only the selected action's feedback is seen during exploration. S-DVAE assumes that all actions that the agent does make, are unsafe actions. This way, the algorithm gradually maps the unsafe policy space, as illustrated in Figure 2.4. It is important to note that this mapping does not influence the agent's choices when learning the dynamics model. If the agent revisits a state in another episode, the agent may select new actions. Such an event labels the new action safe for positive rewards. Depending on how much the algorithm samples from observing an expert system, the better understanding the dynamics model gets for the state-risk mapping of the state-space.

Curiosity-driven Safety

Using expert systems to explore and train off-policy reduces the need for RL algorithms to perform live decision-making to learn a dynamics model. Collecting observations prior to RL decision-making increases the accuracy of the dynamics model and entertains the idea of deriving curiosity-based exploration using the dynamics model. Two very central works on the topic are Pathak et al., 2017; Schmidhuber, 2010.

Curiosity-driven exploration is composed of extrinsic (the environment) and intrinsic (curiosity) rewards, where the agent is encouraged to enter unexplored states. Curiosity is demonstrates to work well in problems where the trajectory to the goal-state is highly non-linear. The agent might come in situations where negative rewards must be endured to prevent local optima. The problem is that curiosity is counterproductive in decision safety because we do not wish to explore areas with little information. The agent effectively makes safer decisions by reformulating the agent's objective to be *curious about known* states. The *negated-curiosity* effect encourages the agent to stay in states where the dynamics model has low uncertainty. For each dynamics model prediction, we can calculate the error, which is the difference between the predicted state and the actual state (the state observed by the agent). The model knows little about the consequences of doing the action for predicted states with high error, indicating that the action may lead to a catastrophic state. Curiosity is the mean squared error of the predicted future state features $\hat{\mathcal{T}}(\hat{s_{t+1}}|s_t, a_t)$ and the ground truth future state $\mathcal{T}(s_{t+1}|s_t, a_t)$ where

$$C_u(\mathcal{T}, \hat{\mathcal{T}}) = \frac{1}{2}||\hat{\mathcal{T}}(\hat{s_{t+1}}|s_t, a_t) - \mathcal{T}(s_{t+1}|s_t, a_t; \mathcal{P})||_2^2, \tag{5.1}$$

defines the curiosity vector C_u that quantifies how curious the agent is for performing a particular action. In curiosity-driven exploration, the goal is to pursue states that maximize curiosity, but we aim to minimize C_u for actions with high uncertainty for safe exploration. In our approach, the weighted curiosity vector adds to the action probability distribution such that

$$\mathcal{U}(s, a) = \mathcal{U}_{ri+Cu}(s, a) = \mathcal{U}_{ri} + \alpha C_u \tag{5.2}$$

where α is the risk-aversion parameter and $\mathcal{U}$ extends the risk-directed exploration bonus from Equation 2.18.

The updated utility is then compatible with Q-Learning updates using a neural network function approximator with weight θ and the Q-Network. The network is trained by sequentially minimizing the loss function $L_i(\theta_i)$ where i denotes the iteration, such that,

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim p(\cdot)}\Bigg[\big(y_i - Q(s,a;\theta_i)\big)^2\Bigg], \tag{5.3}$$

where $y_i = \mathbb{E}_{s_{t+1}\sim\mathcal{E}}[\mathcal{U} + \gamma \max_{a'} Q(s_{t+1}, a'; \theta_{i-1})|s,a]$ is the target for iteration i and $p(s,a)$ is the behavior distribution (Mnih et al., 2015). Finally, the standard differentiated loss, w.r.t to the weights θ denoted,

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim p(\cdot);s_{t+1}\sim\mathcal{E}}\Bigg[\bigg(\mathcal{U} + \gamma \max_{a'} Q(s_{t+1}, a'; \theta_{i-1}) - Q(s,a;\theta_i)\bigg)\nabla_{\theta_i} Q(s,a;\theta_i)\Bigg], \tag{5.4}$$

where $\mathcal{U}$ is the modified risk-reward from Equation 5.2.

Analysis of convergence guarantees

S-DVAE combines several approaches that previous work has shown to have convergence properties. The algorithm models the problem as an MDP, which is proven to have convergence properties in several works (Feinberg et al., 2014; Feinberg and Lewis, 2018; Haddad and Monmege, 2014; Santos and Rust, 2004). The Markov property is especially interesting, and the proof is detailed well in Hairer, 2016. The S-DVAE algorithm uses CMDP's and is proved to have convergence properties for the discounted case used in this work (Altman, 1999).

Tabular Q-Learning is known to converge as time goes towards T, but deep learning variants, specifically DQN, have primarily empirical success. There are efforts such as Fan et al., 2019 that prove theoretical convergence for simplified DQN, but no proof for the general case. The proposed approach is based primarily on empirical observations regarding using neural network estimators for the dynamics model. DVAE uses a similar approach to Ha and Schmidhuber, 2018a; 2018b; Hafner et al., 2019 where the dynamics model encoder constructs a variational bound on the data log-likelihood:

$$\begin{aligned} lnM_d(s_1 : T) &\triangleq ln \int \prod_{t=1}^{T} M(s_t|s_{t-d}) M(s_t|s_t^o) ds_{1:T} \\ &\geq \sum_{t=1}^{T}\Bigg(\underbrace{\mathbb{E}_{q(s_t|s_t^o))}[lnM(s_t^o|s_t)]}_{\text{reconstruction}} \\ &\quad \underbrace{- \mathop{\mathbb{E}}_{M(s_{t-1}|s_{t-d})q(s_{t-d}||s^o \leq t-d)}[KL[q(s_t|s_t^o \leq t) \| M(s_t|s_{t-1})]]}_{\text{multi-step prediction}}\Bigg) \end{aligned} \tag{5.5}$$

where s^o denote unprocessed states. We refer the reader to 2019 for the derivation. The curiosity bonus used in the proposed algorithm works well empirically, but there is no

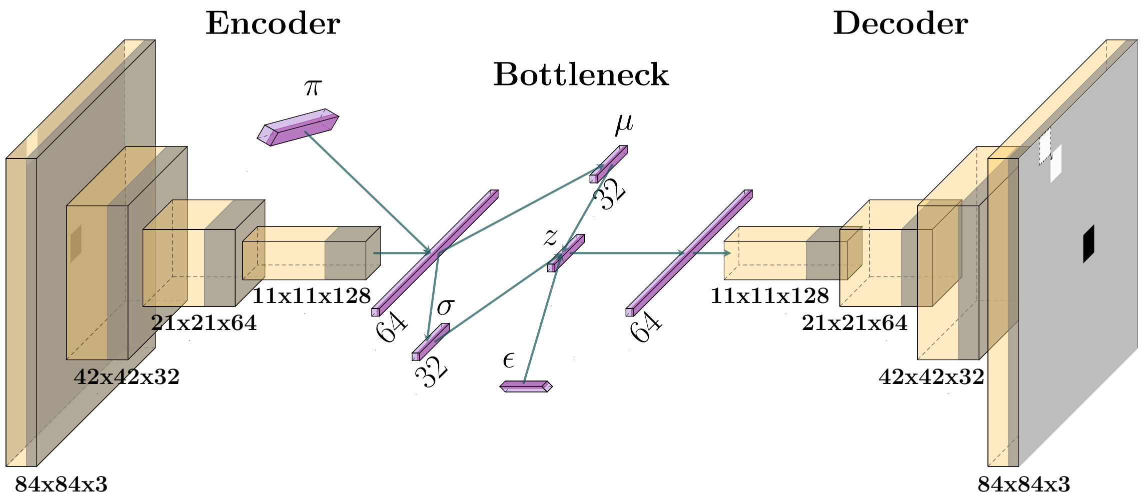


Figure 5.8: The encoder and latent-decoder architecture for learning a compact representation of states. The model is similar to a conditional variational autoencoder.

proof of convergence to the best of our knowledge. S-DVAE demonstrates empirical safety and reward performance through trial and error, but theoretical convergence remains future work.

5.1.7 CostNet for Goal-Directed Reinforcement Learning

CostNet is a combination of three disciplines in Deep Learning, (1) GDRL (Koenig and Simmons, 1996), (2) Model-Based RL (Sutton and Barto, 2018), and (3) Variational Autoencoders (Kingma and Welling, 2013) and form a novel approach for learning the cost between states modeled after an MDP, using the DVAE model for state-predictions. The algorithm accumulates training data using expert systems or random sampling. For systems where safety is a priority, it is advised to perform sampling according to manually defined risk constraints at the cost of increased sample complexity (Paper F).

The initial training phase involves training a dynamics model of the environment. Recent work indicates that state-of-the-art models suffer from severe policy drift after a few predictions, and DVAE-CostNet is no exception to the rule (Andersen et al., 2018b; Ha and Schmidhuber, 2018a; Janner et al., 2019). Therefore, the problem is redefined to learning only the one-step predictions. Figure 5.8 illustrates the proposed structure for the encoder-latent-decoder model for CostNet. The model is a convolutional variational autoencoder with three layers of convolutions before the latent-vector computation. The input is an observation o_t. The latent space z_t forms from an estimated μ, and σ, mean and standard-deviation respectively, from Gaussians. The $\epsilon \sim N$ denotes sampling with the reparametrization trick, as described in Kingma and Welling, 2013. On the right-hand side, the estimated latent variable z_t reconstructs into the future state $\hat{o}_{t'}$. The input is an image of an arbitrary state. The hidden layers are convolutions with 32, 64, and 128

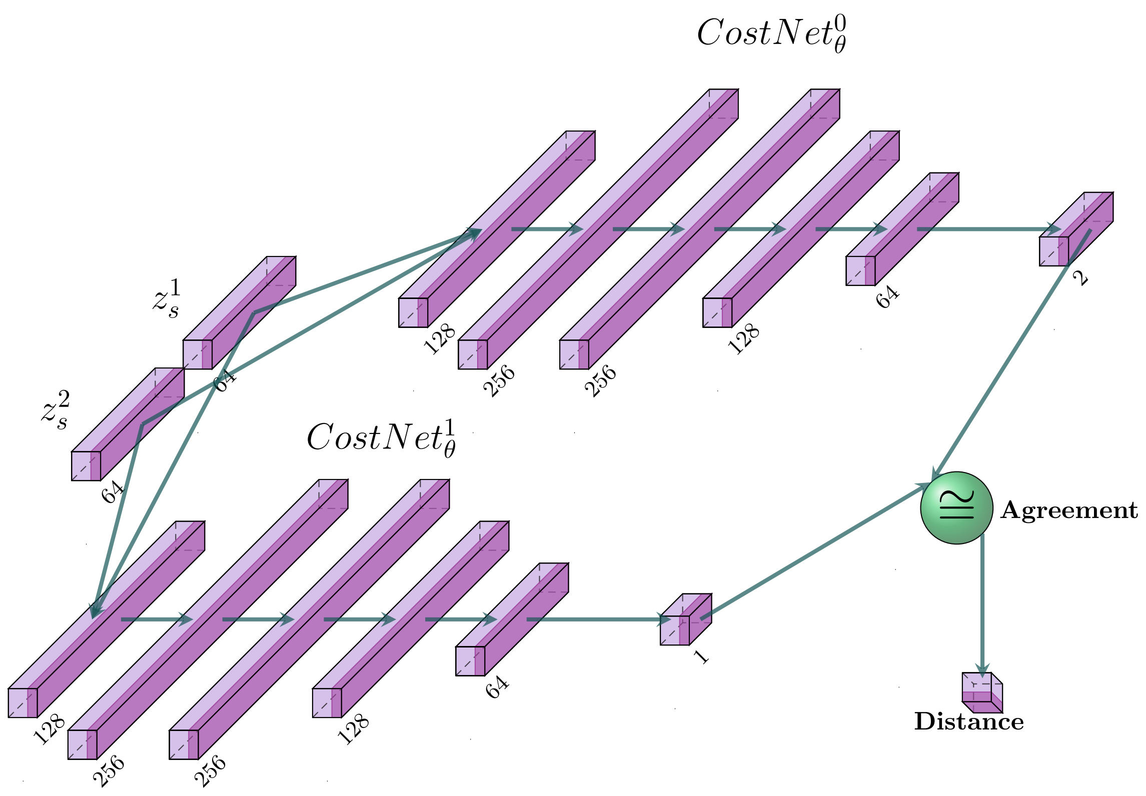


Figure 5.9: The proposed DVAE-CostNet architecture. In-total two inputs, z_s^1 and z_s^2, represent encoded states (see Figure 5.8. The inputs are sent through two streams (models), CostNet$_\theta^0$ and CostNet$_\theta^1$, and learns using two objectives. Both networks must agree on the answer for gradients to contribute positively during training. The training is completed when both networks predict the same state to be closest to the goal state. The hidden layers are standard fully-connected with ReLU activation. The output for CostNet$_\theta^0$ activates with softmax, and CostNet$_\theta^1$ with sigmoid activation.

filters, a kernel size of 2, and a stride of 2 with Rectified Linear Unit (ReLU) activation. The latent-vector size is 64 neurons, but it is highly advised to fine-tune these hyper-parameters as the required embedding capacity varies on the problems state-complexity.

Figure 5.9 shows the proposed architecture for the DVAE-CostNet algorithm and consists of two models with different objectives. The first model, CostNet$_\theta^0$, predicts which of the two states are closest to the goal, $state_A$ or $state_B$. The output is a vector that describes the probability of $state_A$ and $state_B$ being closest to the goal. The second model, CostNet$_\theta^1$, predicts the absolute distance to a goal state as a real number between 0 and 1, where 0 is at the goal state, and 1 is at the maximum possible distance. Both networks train using Mean Squared Error (MSE) loss, where the labels stem from the experience buffer and the distance label from a backtracking algorithm. The predictions are considered correct (reliable) when there is an agreement between both networks, i.e., that CostNet$_\theta^0$ correctly predicts which of $state_A$ or $state_B$ is closest, and CostNet$_\theta^1$ predicts the actual distance.

To exemplify, consider the inputs z_s^1 ($state_A$) and z_s^2 ($state_B$) where z_s^1 is closest to the goal state. In this case, the first index in the vector from the CostNet$_\theta^0$ prediction should be the largest signal, and the predicted distance from CostNet$_\theta^1$ for z_s^1 should be less for the similar prediction z_s^1. If this is in place, we claim that the models agree. When the network agreements are consistent, the training is considered complete.

5.1.8 Hyperparameters

Table 5.1: Tunable parameters in DVAE. Note that this is the complete list for DVAE with its extensions. The hyperparameter column names the specific parameter, the Values column is which data type, the Selected column is the proposed setting, and the comment column summarizes the hyperparameter function.

Hyperparameter	**Values**	**Selected**	**Comment**
Batch Size	$\mathbb{Z}^+$	16	Number of sequence batches
Buffer Size	$\mathbb{Z}^+$	90 000	Replay buffer
Optimizer		Adam	
Learning Rate	$\mathbb{R}$	1e-08	Low Learning rate to improve stability.
Latent Leaps	$\mathbb{Z}^+$	3	The number of multi-step predictions.
Dynamics Model RNN		LSTM	The first version does not use RNN. Extensions use LSTM.
Activation Functions		ReLU	
Reward Prediction	$\mathbb{B}$	1	The first version does not predict rewards.
Cost Prediction	$\mathbb{B}$	1	The first version does not predict costs.
w	$\mathbb{R}$	0.5	Weighted entropy, Paper G Eq 10.
α	$\mathbb{R}$	0.99	Risk-Awareness Paper G Eq 11.

Table 5.2: Hyper-parameters of DVAE-CostNet algorithm. The parameter column is the particular hyperparameter, and the value column is the proposed default value.

Parameter	**Value**
Learning Rate (DQN)	0.01
Discount Factor (DQN)	0.95
ER-Size (DQN)	5000
Optimizer	Adam
Optimizer Learning Rate	0.001
Drift-Threshold ψ	0.3

5.1.9 Summary

DVAE is a novel approach that aims towards safer and more sample-efficient RL for RTS and industry-near applications. DVAE is primarily a variational autoencoder (2013). Still, we investigate several approaches to improve the method, namely using GAN, SWA, CostNet, and modifying the objective towards more safe learning of model-free approaches using a learned DVAE model as the training environment. The challenge with DVAE is to learn stable models for longer time horizons. The study finds that only three steps predictions produce satisfactory results. However, 3 step predictions provide 77% improvement in sample efficiency compared to traditional model-free algorithms.

5.2 Deep Variational Q-Networks (DVQN)

The Deep Variational Q-Networks (DVQN) extends the Deep Q-Networks algorithm that finds good policies in an organized latent space. The algorithm is especially suited for clustering the state-space and learning multiple policies in a singular clustering model (See Section 7.5). Empirically, the algorithm shows comparable reward performance to traditional model-free deep Q-Network variants. DVQN combines two emerging algorithms, VAE (2013) and DQN (Mnih et al., 2015). This work is published in Paper J with the title *Interpretable Option Discovery using Deep Q-Networks and VAEs*.

5.2.1 Motivation

Motivation 1 VAEs are highly expressive models that usually model data as Gaussians. From prior research, it is clear that data in the latent space of VAE's correlates highly (e.g., similar data is close). This motivates using VAE's to create an RL algorithm that can cluster the state-space, enabling different behaviors for individual clusters.

Motivation 2 Option-based RL is a hierarchical approach that enables algorithms to encode/learn several objectives separately. Combined with Motivation 1, this contribution motivates to design a foundation for algorithms that can learn to automatically discover options following the seminal work of Sutton et al., 1999.

5.2.2 Deep Variational Q-Networks

In traditional Deep Q-Networks, hidden layers are treated as a black box. On the contrary, the objective of the VAE is to reconstruct the input and organize the latent-vector so that similar (data) states are adjacently modeled as a Gaussian distribution.

By introducing a VAE mechanism into the algorithm, we expect better interpretability

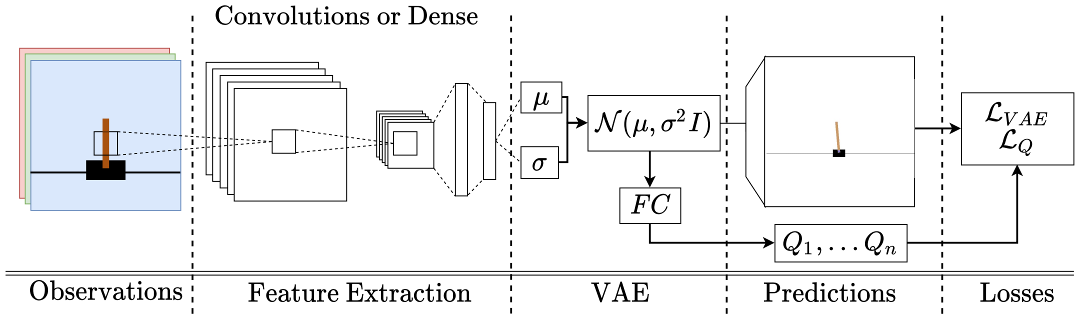


Figure 5.10: The deep variational Q-Networks architecture.

for creating options in RL, which is the primary motivation for this contribution. VAEs are, in contrast to DRL, involved with the organization of the latent space representation and are commonly used to generate clusters of similar data with t-distributed stochastic neighbor embedding (t-sne) or Principal Component Analysis (PCA) (Zheng et al., 2017). The DVQN algorithm introduces three significant properties. First, the algorithm fits the data as a Gaussian distribution. This reduces the policy space, which in practice reduces the probability of the policy drifting away from global minima. Second, the algorithm is generative and does not require exploration schemes such as ϵ-greedy because of the reparametrization noise during training. Third, the algorithm can learn the transition function and, if desirable, generate training data directly from the latent space parameters, similar to the work of Ha and Schmidhuber, 2018a.

Figure 5.10 illustrates the architecture of DVQN. The architecture follows general trends in similar RL literature but has notable additions. First, features are extracted from the state input, typically using convolutions for raw images and fully connected for vectorized input. The extracted features are forwarded to a fully connected intermediate layer of a user-specified size, commonly between 50 to 512 neurons. The intermediate layer splits into two streams representing the variance μ and standard deviation σ and is used to sample the latent-vector using a Gaussian distribution. The latent-vector is forwarded to the decoder for state reconstruction and the Q-Learning stream for action-value (Q-value) optimization. The decoder and Q-Learning streams have the following loss functions:

$$\mathcal{L}_{VAE} = \text{MSE}(s, \hat{s}) + D_{KL}[q_\psi(z|s) \| p_\theta(z|s)] \tag{5.6}$$

$$\mathcal{L}_{DQN} = \text{Equation 2.11} \tag{5.7}$$

$$\mathcal{L}_{DVQN} = c_1 \mathbb{E}_{\sim q_\psi(z|s)}[\mathcal{L}_{VAE}] + c_2 \mathbb{E}_{s,a,s_{t+1},D \sim r}[\mathcal{L}_{DQN}]. \tag{5.8}$$

The algorithm loss function $\mathcal{L}_{DVQN}$ is composed of two local objectives: $\mathcal{L}_{DQN}$ and $\mathcal{L}_{VAE}$. In the VAE loss, the first term is the mean squared error between the input s_t and its reconstruction $\hat{s}_t$. The second term is regularization using KL-distance to minimize the latent distribution and a Gaussian distribution *distance*. The DQN loss (Equation 5.7) is a traditional Deep Q-Networks update, as described in Mnih et al., 2015. Both terms in $\mathcal{L}_{DVQN}$ weights with a constant $c \in \{0 \dots 1\}$.

Algorithm 3: DVQN: Minimal Implementation

```
Init: Environment
Init: DVQN model $\pi$
Init: Experience-Replay Buffer $D_\pi$
1 for $i$ = 0 to N_EPISODES do
2     $D_\pi \leftarrow$ Collect samples from environment using policy $\pi$ via the generative
        policy sampling.
3     Train model $\pi$ on a mini-batch from $D_\pi$ with objective from Equation 5.8
4 end
```

Algorithm 4 describes the general routine for the DVQN algorithm. First, the environment is initialized. Second, the DVQN model from Figure 5.10 is initialized with the desired hyperparameters, and third, the ER buffer is created. The algorithm samples actions from the generative policy for exploration and stores these as MDP tuples in the experience replay for a specified number of episodes. After each episode, the algorithm samples mini-batches from the experience replay and performs parameter updates using stochastic gradient descent. The (loss function) optimization objective is described in Equation 5.8. The process repeats until the algorithm converges.

5.2.3 Hyperparameters

Table 5.3: Algorithm and hyperparameters used in the experiments. For the Rainbow algorithm, we used the same hyperparameters described in 2018. The DDQN had target weight updates every 32k frames.

Algorithm	**DQN**	**DDQN**	**Rainbow**	**DVQN (ours)**
Optimizer	Adam			RMSProp
Learning Rate	0.003			0.000025
Activation	ReLU			ELU
Batch Size	32			128
Replay Memory	1m			
Epsilon Start	1.0			N/A
Epsilon End	0.01			N/A
Epsilon Decay	0.001 (Linear)			N/A
Gamma	0.95			
Q-Loss	Huber			MSE

During the experiments, we found DVQN to be challenging to tune. Initially, the algorithm used ReLU as activation but was discarded due to vanishing gradients resulting in divergence for policy and reconstruction objectives. Using Exponential Linear Units (ELU), we found the algorithm to be significantly more stable during the experiments, and it additionally did not diverge if training continued after convergence. Table 5.3 shows the hyperparameters used in our experiments where most of the parameters are adopted from prior work. Recognize that the DVQN algorithm does not use ϵ-greedy methods for exploration. This is because random sampling is done during training in the VAE part of the architecture. In general, the algorithm tuning works well across all of the tested domains, and better results can likely be achieved with extended hyperparameter searches.

5.2.4 Summary

We present the Deep Variational Q-network, a novel algorithm for learning generative latent space distribution policies. The learned latent space is particularly useful for clustering states that are close to each other for discovering options automatically. The DVQN algorithm can perform comparably to traditional Deep Q-Networks in the tested environments. DVQN does not provide the same training stability and is significantly harder to fine-tune than traditional Deep Q-Networks algorithms. For instance, network capacity is increased. As a result, the algorithm takes longer to train, and during the experiments, only the RMSprop optimizer (Nair and Hinton, 2010) with a small step size provided convergence. Additionally, the exponential linear units from Clevert et al., 2015 positively affected stability. The DVQN contributes a novel approach for options discovery in hierarchical reinforcement learning algorithms on the positive side.

The combination of VAE and reinforcement learning algorithms has exciting properties. Under optimal conditions, the latent space should follow a Gaussian distribution where policy evaluations always provide optimal state-action values since these are the built-in properties of the latent space in any VAE. The difference between traditional Deep Q-Networks and DVQN primarily lies in reducing sparsity in the latent space. Deep Q-Networks do not provide a latent space structure reflecting distances between states but rather a distance between Q-values (Mnih et al., 2015). By using KL-regularization from VAE, low state-to-state is encouraged. Another benefit of VAE is that we sample from a Gaussian distribution to learn μ and σ, which is especially satisfying for algorithms with off-policy sampling and therefore eliminates the need for (ϵ-greedy) random sampling.

Section 6.3 demonstrates the effectiveness of DVQN to structure latent spaces to human-interpretable formats and shows comparable reward performance to traditional model-free approaches.

5.3 Observation Reward ActionCost LEarning (ORACLE)

The Observation Reward Action Cost Learning Ensemble (ORACLE) algorithm is a novel end-to-end architecture for training model-free algorithms on a dynamics model learned through observations of arbitrary environments.

5.3.1 Motivation

Motivation 1 During our findings in the DVAE Model, we found several shortcomings in learning expressive models, especially for advanced environments such as Deep-RTS and StarCraft II. This motivates new directions towards building a more expressive and capable model for model-based RL.

Motivation 2 DVAE uses traditional VAE's from Kingma and Welling, 2013 at its core, prone to posterior collapse. State-space models combined with deep learning techniques, on the other hand, demonstrate less chance of catastrophic local minima (Fraccaro, 2018). This motivates using states-space models to learn dynamics in place of VAE's.

Motivation 3 Driven by Motivation 2, this contribution motivates the study of VQ-VAE's in place of VAE's. This contribution motivates a study of VQ-VAE's because they demonstrate superior performance compared to VAE's and does not suffer from posterior collapse. Several of our contributions have sought to experiment with VQ-VAE's (Paper G, Section 6 and Paper H, Section 7).

5.3.2 Observation Reward Action Cost LEarning

ORACLE combines state-of-the-art deep learning techniques; Stochastic Recurrent State Space Models (SRSSM) from 2018, VAE from Kingma and Welling, 2013, and VQ-VAE from Razavi, van den Oord, and Vinyals, 2019. The vision is that observations from the environment compress to a compact variable, evaluate the next-step latent variable, and decompress the next-step observation from the latent-state approximation. The overall goal of ORACLE is to learn the dynamics model $\mathcal{T}(s_{t+1}|s_t, a_t)$, where the approximation model denotes $\hat{\mathcal{T}}(s_{t+1}|s_t, a_t)$.

ORACLE uses deterministic convolutional encoders, stochastic deconvolutional decoders, SRSSM's, and a model-free RL policy. Figure 5.11 shows the complete ORACLE architecture. There are two variants of ORACLE, the risk-neutral variant from Paper K and a safety-aware variant in Paper M. The difference is that Paper K uses a traditional reward function. In contrast, Paper M uses several reward-shaping techniques for safe RL.

5.3.3 Encoder and Decoder

This work primarily attends to high-dimensional observations such as images. ORACLE uses convolutional neural networks to reduce pixel input dimensionality efficiently, and for simple raw numerical inputs, a regular fully-connected network is sufficient.

The encoder takes observations o_t as input and outputs a compressed state x_t. It is a three-layer deterministic convolutional neural network (CNN) that aims to extract information from pixel observations o_t and reduce dimensionality before reshaping the compressed representation to a vector x_t. Using CNN is first proposed in the seminal work of LeCun et al., 1998. The encoder (and decoder) are learned using the MSE (cross-entropy) between the predicted observation $\hat{o}_{t+1}$, and *after-the-fact* observations o_{t+1},

$$\mathcal{L}_{OBS} = -\mathbb{E}[log p(o|z)]. \tag{5.9}$$

The decoder takes the latent variables z_t as input and outputs the decompressed observation $\hat{o}_{t'}$. It is a deconvolution neural network that upsamples the compressed latent variables to the same dimensions of o_t. The final convolutional layer predicts μ and σ to parameterize Gaussian distributions for every output pixel $\hat{o}_{t'}$. Empirical testing demonstrates significant reconstruction performance benefits when using stochastic decoders because it learns data variability better, resulting in far more accurate predictions than

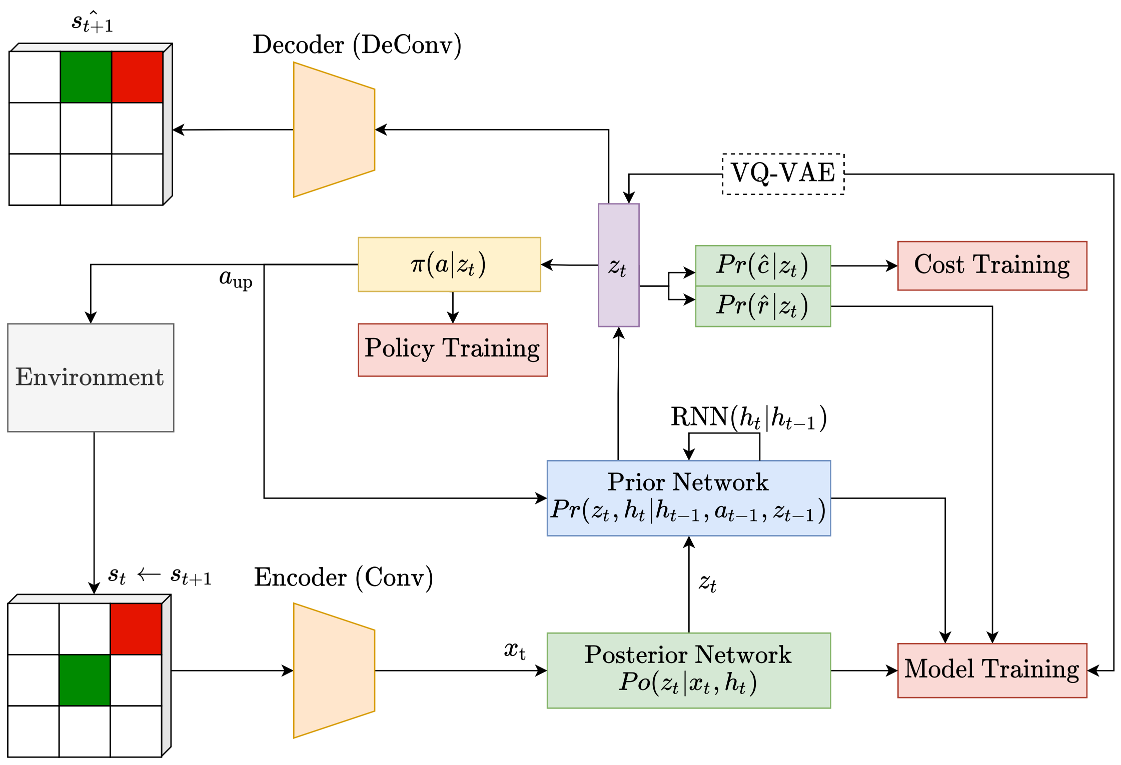


Figure 5.11: The ORACLE model. The environment produces an initial state s_t that feeds into a ConvNet. The flattened representation x_t is fed into the posterior distribution and produces a latent vector z_t. Concurrently, the prior distribution, which aims to learn the dynamics model, uses the previous hidden state to predict a belief of the latent vector following the intuition from Equation 2.4. The latent vector is the backbone for predicting a reward and the state-cost. The policy uses trains concurrently where actions are made to the real environment. Every component of the model trains jointly where each *training block* (e.g., the colored squares) plays a part in the final optimization objective.

deterministic decoders (Rybkin et al., 2021). There are no particular differences during inference as we sample the mean, which roughly equates to the mean value of deterministic neural networks (Fraccaro, 2018).

5.3.4 Dynamics Model

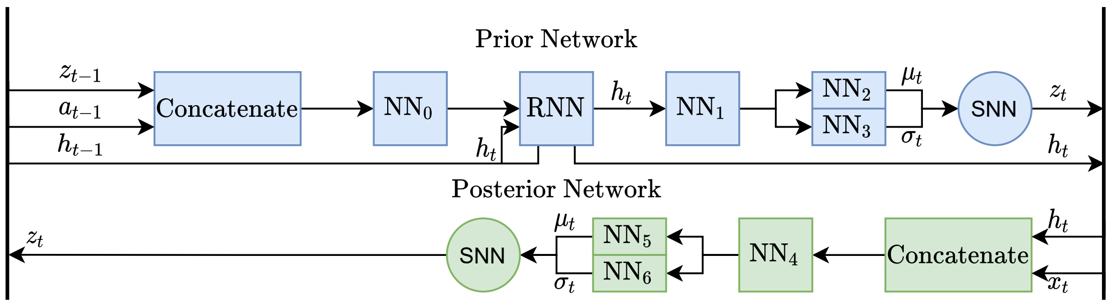


Figure 5.12: Detailed overview of the dynamics model forward-pass from Figure 5.11. The prior network uses last-step information, along with an action (the action that leads up to state s_t is denoted a_{t-1}), and outputs the computed hidden-state and the sampled latent-state variable z_t. For LSTM, we store the cell state along with the hidden state. The posterior network takes in the observed information x_t and our prior belief state h_t and predicts the *informed* latent-state variable z_t. Although both latent-state variables are denoted z_t, the prior latent-state variable is denoted $\hat{z}_t$ during training.

ORACLE estimates the true MDP transition function (Equation 2.2), detailed in Section 2.2.4. ORACLE uses deep neural networks to estimate the transition function using VAE's (Kingma and Welling, 2013) and SRSSM's (Fraccaro, 2018) to create a highly expressive probabilistic model. ORACLE models two distributions, a prior model and a posterior model, where the goal is to transfer posterior knowledge to the prior distribution using optimization. Figure 5.12 depicts the forward pass of ORACLE after encoding the observation o_t to a more compact representation x_t.

The prior network (generative network) predicts the next state latent space variables z_t using information from the previous timestep; the previous latent variables z_{t-1}, current action a_{t-1}, and the hidden-state h_{t-1}. ORACLE uses an LSTM architecture but is compatible with Gated Recurrent Unit (GRU) (Cho et al., 2014) or Liquid-Time Constant (LTCRNN) networks (Hasani et al., 2021). When using LSTM, it is crucial to preserve the cell state between training and inference. The primary goal of the RNN layer is to transition the hidden state h_{t-1} to the future hidden state h_t. In MDP's, the RNN model learns the transition function matrix from Equation 2.2. Finally, the stochasticity in the transition function is the parametrization of Gaussian's μ and σ to allow sampling of the latent-state variables z_t.

The posterior network (inference model) is simpler than the generative network. First, it concatenates the hidden-state h_t, the encoded observation x_t, and forwards to a feed-forward layer (NN$_4$) before parameterizing Gaussian distributions, similar to the prior network. Finally, it predicts *informed* (*after-the-fact*) latent space variables z_{t+1}.

The forward-pass of the dynamics model is as follows where the prior model performs the following operations:

1. Compute $u_t = \text{concat}(z_{t-1}, a_{t-1})$.
2. Compute RNN state $h_t = \text{RNN}(u_t)$.
3. Parameterize mean $\mu_t = \text{NN}_2(h_t)$ diagonal covariance matrix $\sigma = \text{NN}_3(h_t)$.
4. Sample from Gaussian distributions $\text{SNN}(z_t|h_t;\theta) \sim \mathcal{N}(h_t;\mu,\sigma)$.

All steps are performed for every sample and form our prior belief state of the latent variables. As seen in Equation 5.11, the posterior model depends on previous hidden-state h_{t-1}, action a_{t-1}, and the encoded state-observation x_t. The posterior model can be summarized to the following procedure:

1. Compute $u_t = \text{concat}(h_t, x_t)$.
2. Parameterize mean $\mu_t = \text{NN}_5(u_t)$ diagonal covariance matrix $\sigma = \text{NN}_6(u_t)$.
3. Sample from Gaussian distributions $\text{SNN}(z_t|h_t;\psi) \sim \mathcal{N}(x_t;\mu,\sigma)$.

Training takes inspiration from previous work in Paper D using variational inference (Kingma and Welling, 2013) with the extension of using SRSSM's from Fraccaro, 2018. VAE's and SRSSM's are highly expressive model classes for learning patterns in time series data and system identification (e.g., learning dynamics model from observed data) (Doerr et al., 2018). The algorithm trains similarly to VAE's using amortized variational inference since $po(z_t|x_t;\theta) = \int_{z_t} \frac{po(x_t|z_t;\theta)po(z_t;\theta)}{po(x_t)} dz$ is intractable (Zhang, Butepage, et al., 2019). The generative model (prior) pr and the inference model (posterior) po is expressed,

$$\textbf{Prior Model} : pr(z_t, h_t|h_{t-1}, a_{t-1};\theta) \tag{5.10}$$

$$\textbf{Posterior Model} : po(z_t|x_t, h_t;\theta). \tag{5.11}$$

Equation 5.10 is the prior distribution that attempts to learn parameters θ that best fit the posterior distribution (Equation 5.11) by minimizing the Kullback Leibler Divergence (KL) distance. The intuition is that the posterior model learns dynamics of the MDP (environment) through observations o_t while the prior distribution must learn indirectly through optimization of the parameters θ. To make the optimization tractable, we use the

ELBO (Jordan et al., 1999; Kingma and Welling, 2013) such that the optimization term for the SSM (generative and inference network) becomes,

$$\mathcal{L}_{SSM} = \mathcal{L}_{OBS} - D_{KL}[po(z|x;\theta)|pr(z;\theta)]. \tag{5.12}$$

5.3.5 Categorical Latents

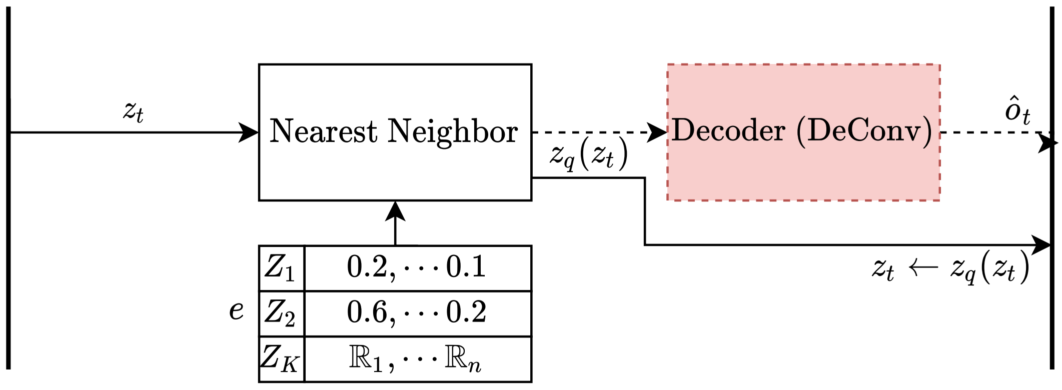

z_t
Nearest Neighbor
$z_q(z_t)$
Decoder (DeConv)
$\hat{o}_t$
$z_t \leftarrow z_q(z_t)$
e
Z_1 | $0.2, \cdots 0.1$
Z_2 | $0.6, \cdots 0.2$
Z_K | $\mathbb{R}_1, \cdots \mathbb{R}_n$

Figure 5.13: A detailed overview of the vector quantization network for mapping latent space variables to categories after the generative network has predicted a continuous latent space variable. The VQ-VAE module is depicted in Figure 5.11 and attempts to map variables from the generative network (prior) to categories. The dashed lines illustrate the process during training, while solid lines illustrate inference time computation. The algorithm finds an appropriate category using the nearest neighbor.

Following the work in Razavi, van den Oord, and Vinyals, 2019, ORACLE uses a variation of the VQ-VAE architecture, illustrated in Figure 5.13. The VQ-VAE network categorizes latent variables similar to Paper H and enables policy selection of specific areas in the state-space. Abstract policy selection is a promising direction towards automating and generating options in hierarchical RL as proposed in the Options framework by the seminal work of Sutton et al., 1999. Using VQ-VAE demonstrates benefits for planning and predictive learning (Ozair et al., 2021) and does not suffer from a posterior collapse in contrast to VAE's. We discovered that combining SRSSM's and VQ-VAE shows more consistent convergence, reduces the risk of posterior collapse and recovers well from post mortem posterior collapse if SWA is used to optimize the model parameters.

Inference and training are straightforward where the sampled latent variables from the SRSSM mapped to a categorical codebook $Z_1 \cdots Z_K$ with K possible categories of latent space variables (Table 5.4 for hyperparameters). VQ-VAE outputs the categorical latent vector closest to the continuous-space SRSSM latent-vector $z_t \leftarrow z_t^k$. Training occurs jointly with the rest of the model,

$$\mathcal{L}_{VQ} = ||sg\left[z_e\right] - e||_2^2 + \beta_{\text{vq}}||z_e - sg\left[e\right]||_2^2, \tag{5.13}$$

where e is the codebook. The first term is the codebook alignment loss which updates the selected category closer to the SSM latent vector. The sg denotes the stop-gradient operator, which prevents gradients computation from flowing through the SSM model. This is because we are only concerned with updating the codebook. The second term moves the SSM latent-vector towards the codebook, but because SRSSM trains concurrently, it is desirable to reduce the weight $\beta_{\text{vq}} \sim 0.05$, in contrast to Van Den Oord et al., 2017 that propose a value in the range $0.25 \leq \beta_{\text{vq}} \leq 2.0$.

5.3.6 Rewards, Risks, and Costs

The learned dynamics model is the foundation for reward and cost estimates as part of the feedback that fuels the learning of model-free RL agents. The feedback signal originates from the environment (e.g., the designer creates the reward signal), uncertainty-based risk quantization, and distance costs. The distance-cost metric measures the distance between the present state observation and a possible positive terminal state observation. Risk calculation follows the risk-sensitivity scheme (Section 2.3.1) and expresses the return function according to Equation 2.19. ORACLE learns the reward function $\hat{\mathcal{R}}$ by minimizing the squared loss between predicted and actual reward and parametrizes a Gaussian distribution. The cost is learned separately and relies upon reaching a terminal state during exploration. When a terminal state is reached, experiences from the start state to the goal state are labeled in a temporal difference manner (e.g., how many steps it took from state s_{t-n} until the terminal state s_t). The risk measure is a side-effect from model uncertainty in the reward estimates $Risk = \text{Var}(\hat{\mathcal{R}})$ and gives the reward function,

$$\mathcal{R}_{oracle} = \mathcal{R}(z) - Var(\hat{\mathcal{R}}) + (1 - C(z;\theta)). \tag{5.14}$$

The risk-aware approach in ORACLE comprises two terms that reduce risk during decision-making in training and inference. The first term in Equation 2.18 is the inference-time risk reduction. During learning, the agent utilizes Equation 2.18 for action selection and Equation 2.19 with $\omega = \text{Var}(\hat{\mathcal{R}}(z_t))$. The second term is the cost function $C(z_t))$ that predicts the normalized distance to a positive goal state. sg denotes the stop-gradient operator. To optimize and learn the reward function and cost function, we minimize the following,

$$\mathcal{L}_{Rew} = \underbrace{\mathbb{E}[\log pr(\mathcal{R}|z_t;\theta)]}_{reward-loss} + \underbrace{\text{sg}(\mathbb{E}[\log pr(C|z_t;\theta)])}_{cost-loss}. \tag{5.15}$$

5.3.7 Actor Policy Ensemble

The actor ensemble's primary objective is to make informed decisions that lead to a safe trajectory to a goal state. ORACLE uses an ensemble of model-free algorithms, specifically A2C, DQN, RAINBOW, IMPALA, and PPO. These algorithms behave differently but share the fundamental goal of maximizing rewards. Reward maximization fuels the motivation for reward function modeling using Equation 5.14. Specifically, all actor algorithms optimize towards the risk-aware reward function, and the goal is that the algorithms empirically demonstrate better safety during the exploration and exploitation of learned dynamics. The actor ensemble is defined,

$$\pi_{ensemble} = \text{MV}(\{\pi_{A2C}, \pi_{DQN}, \pi_{RAINBOW}, \pi_{IMPALA}, \pi_{PPO}\}). \tag{5.16}$$

MV is the majority vote operator, and $\pi_{ensemble}$ is the policy that selects action a_t according to the voted action. $e\pi_{A2C}$ denotes a policy trained using A2C, $e\pi_{DQN}$, a policy trained with DQN, and so on. Recall that policies select actions according to the risk-adjusted utility function in Equation 2.18 from Edith et al., 2005, but use the predicted reward $\hat{\mathcal{R}}$ instead of the environment reward signal $\mathcal{R}$. The ensemble of policies cannot analytically guarantee safety during learning but empirically shows a promising step towards safer algorithms.

5.3.8 Algorithm Description

Algorithm 4: ORACLE Training Routine

Init: Environment

Outcome: Learned dynamics model $z_{t+1} = f(z_t, h_t, a_t)$

Outcome: Learned ensemble of policies π = { $\pi_1 \cdots \pi_n$ }

Hyperparameters: Table 5.4.

```
1 while dynamics model is not trained do
2     Train $po_\theta(\hat{x}_{t+1}, z_{t+1} | x_t, s_{t+1}, a_t; \theta)$ ;        // Equation 5.17
3 end
4 while model-free ensemble not trained do
5     Choose action $a_t$ from policy
6     Execute $a_t$ at state $s_t$ and get $z_{t+1}, r_t, c_t$ via dynamics model
       $po_\theta(z_{t+1}, r_{t+1}, c_{t+1} | z_t, a_t)$
7     Perform policy update
8 end
```

Algorithm 4 shows pseudo-code for training ORACLE. ORACLE has two training procedures. First, the ORACLE sample the environment using a policy. Second, the algorithm proceeds using one of two schemes. Scheme #1 uses the agent ensemble for exploration and trains concurrently. Scheme #2 uses an expert system to pre-train the dynamics model. Using Scheme #1, each algorithm in the ensemble trains either directly as the samples are observed or stored in a buffer for fully offline training. After learning the dynamics model, the second training procedure for a model-free algorithm begins. The programmer can use any model-free algorithm but should note that the training procedure is different for off and on-policy algorithms. Optimally, the algorithm should utilize a replay buffer and sample actions from an external policy, following Scheme #1 to learn off-policy algorithms. In contrast, on-policy algorithms should train directly without such storage. When the model-free agent has reached a sufficient level of reward performance, the ensemble is ready for making decisions in the ground truth environment. In this implementation, we use majority voting to select a decision and sample randomly when there is no policy consensus.

5.3.9 Tuning and Training ORACLE

The ORACLE algorithm has many hyperparameters for tuning training stability and reward performance, seen in Table 5.4. Our finding is that ORACLE is relatively robust in default hyperparameter configuration but requires additional tuning for StarCraft II and Deep RTS. We test LTC and GRU extensively, but LSTM performs best overall in the experiments. Another notable hyperparameter choice is the Adaptive Gradient Clipping (AGC) algorithm, a novel approach to clip the gradient from historical norms (Seetharaman et al., 2020). Additionally, clipping the gradients for values out of bounds of -100.0 and 100.0 demonstrates better training stability, but the downside risk is to get stuck in a local optimum. The ORACLE model optimizes all objectives (Equation 5.9, 5.12, 5.13, and 5.15) jointly,

$$\mathcal{L}_{ORACLE} = \mathcal{L}_{OBS} + \mathcal{L}_{Rew} + \gamma\mathcal{L}_{SSM} + \mathcal{L}_{VQ}, \tag{5.17}$$

using Stochastic Weight Averaging (SWA) with the AdamW optimizer (Loshchilov and Hutter, 2019). see Section 5.1.4. The Latent Leap (LL) hyperparameter determines how many steps into a potential future the algorithm predicts. Specifically, it is expected that the accuracy of predictions will reduce as the LL parameter becomes larger due to compounding errors. We found that ORACLE works well with LL=30 for the tested environments.

5.3.10 Summary

We present ORACLE as a novel approach towards safer and more sample-efficient RL for RTS and industry-near applications. In contrast to related literature, ORACLE makes no assumptions or premises for prior knowledge at the cost of not guaranteeing safety analytically. Using an expert system (Scheme #2) to pre-train the dynamics model significantly decreases the likelihood of entering catastrophic states. Section 6.5 shows that the actor ensemble performs well within a presumable safe state-space set, without external guidance if pretraining of the dynamics model is allowed. The ORACLE can be summarized to the following procedure:

Table 5.4: Set of tunable parameters in ORACLE. In addition to this list, we can adjust the model complexity, such as neuron count and hidden layer count. The hyperparameter column names the specific parameter, the Values column is which data type, the Selected column is the proposed setting, and the comment column summarizes the hyperparameter function.

Hyperparameter	**Values**	**Selected**	**Comment**
Batch Size	$\mathbb{Z}^+$	48	Number of sequence batches
Sequence Size	$\mathbb{Z}^+$	48	Number of frames in a sequence
Buffer Size	$\mathbb{Z}^+$	9 000	Replay buffer
Reward Scaling	$\mathbb{R}$	1.0	Scaling of the reward objective
Cost Scaling	$\mathbb{R}$	1.0	Scaling of the cost objective
VQ Scaling	$\mathbb{R}$	0.1	Scaling of the VQ objective
KL Scaling	$\mathbb{R}$	1.0	Scaling of the KL objective (KL-β)
KL Minimum Nats	$\mathbb{R}$	3.0	Minimal information loss
Optimizer		AdamW	AdamW improves generalization. See 2019
Gradient Clipping	$\mathbb{R}$	100.0 $\pm$	Clip gradients to increase learning stability
Adaptive GC	$\mathbb{B}$	1	Based on the history of gradient norms (Seetharaman et al., 2020)
Learning Rate	$\mathbb{R}$	0.0001	Low Learning rate to improve stability.
Latent Leaps	$\mathbb{Z}^+$	30	The number of leaps into future states.
Dynamics Model RNN		LSTM	
Activation Functions		ELU	
Enc/Dec Neurons	$\mathbb{Z}^+$	1024	
Stochastic Reward	$\mathbb{B}$	1	Sample rewards under Gaussian assumptions
Stochastic Costs	$\mathbb{B}$	1	Sample costs under Gaussian assumptions
Discount Factor γ	$\mathbb{R}$	0.96	The discount factor used in value-based algorithms
Risk-Entropy Weight w (Eq 2.16)	$\mathbb{R}$	0.6	Risk-Directed Exploration entropy weight
Risk-Utility Function ρ (Eq 2.18)	$\mathbb{R}$	0.75	Weight of risk in utility function.
Risk function weight β (Eq 2.19)	$\mathbb{R}$	0.40	Weight of the risk functipn ω.
VQ-VAE weight β_{vq}	$\mathbb{R}$	0.05	Weight of VQ-VAE encoding updates.

1. Read state observation o_t
2. Extract features of the observation x_t
3. Compute latent-state using x_t for the prior and posterior $\hat{z}_t$
4. (Optional) Compute categorical latent-state for latent space using VQ-VAE architecture z_t
5. Predict using latent-state z_t
    - Decode latent-state in decoder and sample possible future observation $\hat{o}_t$
    - (Optional) Predict action for policy ensemble given latent-state using majority voting $\pi_{ensemble}(a_t|z_t)$
    - (Optional) Predict reward of the latent-state $\hat{r}_t$
    - (Optional) Predict cost of the separate model using latent-state $\hat{c}_t$
6. Compute gradients jointly following Equations 5.9, 5.12, 5.13, 5.15, and separately for the policy ensemble and cost network. Use the Adam optimizer with SWA and gradient clipping.

5.4 Summary

The previous sections present three algorithms for improving sample efficiency and safety in model-based RL for RTS games and industry applications. RTS games and industry applications are couples tightly because they have similar complexities and need for decision safety. In RTS games, the agent loses may lose the game when acting unsafe, and in industry, the agent may enter catastrophic system states that damage humans and equipment. DVAE aims to improve sample efficiency using observations of expert systems and address safety through reward shaping and policy constraints (CMDP's).

ORACLE similarly expands on this notion but focuses more on reward shaping to decrease the need for manual constraint designing. ORACLE and DVAE have different trade-offs. DVAE guarantees safety when using sufficient constraints such as Lyapunov functions or barrier functions. The downside is that DVAE becomes complex with multitudes of model and environment assumptions, which is not practical for industrial applications. On the other hand, ORACLE makes no such assumptions at the cost of not having safety guarantees. However, empirical testing can demonstrate safety, and for some industrial applications, a combination of ORACLE and an expert system governor to restrict catastrophic behavior could be sufficient.

The last contribution is the DVQN algorithm that studies the latent space structure towards interpretability and automatic Options selection in hierarchical RL. Although this contribution does not address the topic adequately, it builds momentum for the work on using VQ-VAE's in ORACLE, which enables multi-policy learning through categorizing latent variables. However, this approach is not covered in this dissertation but remains detailed for future work

Chapter 6

Contribution Evaluation

The proposed algorithms are model-based approaches that aim to learn models that can accurately express environment dynamics and to make safer decisions towards better integration with safety-critical industrial applications. This chapter presents the empirical evaluations and results of the algorithms in Chapter 5 using the environments from Chapter 4. We organize the chapter into sections for major components of our contributions, namely:

1. Risk-neutral DVAE algorithm for model-based RL,
2. safety-aware S-DVAE algorithm for industry applications,
3. the DVQN algorithm towards interpretable and automatic option discovery,
4. ORACLE for RTS game and industry applications, and
5. S-ORACLE for safer learning of model-based RL

6.1 DVAE Evaluations

The goal of DVAE is to learn a dynamics model that can train reinforcement learning agents with minimal ground truth sampling. DVAE has several extensions, DVAE-GAN, DVAE-SWA, and DVAE-SWAGAN, aiming to improve the model stability so that DVAE can produce better quality outputs and latent spaces for environments with continuous or sparse state-spaces. The methods are detailed in Section 5.1. This section focuses on empirical evaluations and experiments for the DVAE, DVAE-GAN, DVAE-SWA, and DVAE-SWAGAN algorithms.

6.1.1 Hypotheses

We would like to evaluate the following hypotheses for the DVAE algorithm.

Hypothesis DVAE 1 *DVAE can accurately model simple environments such as the Deep Maze without walls.*

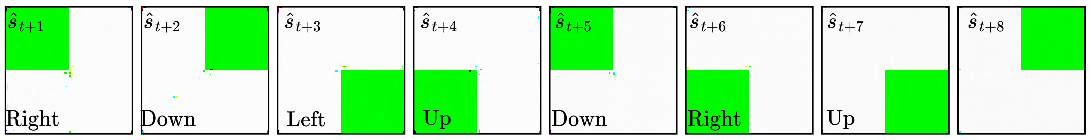


Figure 6.1: For the 2×2 scenario, only 50% of the environment is explored, leaving artifacts on states where the model is uncertain of the transition function. In more extensive examples, the player disappears, teleports, or gets stuck in unexplored areas. The upper-right corner denotes the nth predicted state, e.g., $\hat{s}_{t+n}$.

Hypothesis DVAE 2 *DVAE models complex environments such as Deep Line Wars and Sonic The Hedgehog without difficulties.*

Hypothesis DVAE 3 *DVAE outperforms traditional RL methods in reward performance while having higher sample efficiency.*

6.1.2 Standard DVAE

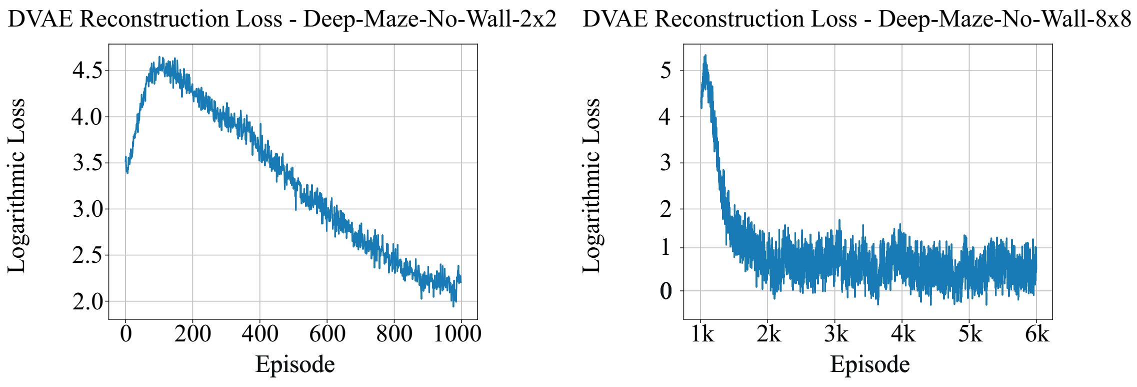
The *standard* DVAE algorithm is proposed in Paper D and is our first contribution to model-based RL. The *vanilla* DVAE algorithm uses a standard VAE to represent the dynamics model that predicts states $\hat{s}_{t+1}$ that fuel training in model-free algorithms such as DQN.

6.1.2.1 Reconstruction - Deep Maze-No-Wall

The algorithm must generalize over many similar states to model a large state-space for the deep maze environment. DVAE aims to learn the transition function $\mathcal{T}(s_t, a_t)$, bringing the state from s_t to s_{t+1}. We use the Deep Maze environment because it provides simple rules with a controllable state-space complexity. Also, we can omit the importance of reward for some scenarios.

We trained the DVAE model on two *Deep Maze-No-Wall* scenarios using grid-size 2×2 and 8×8. The encoder and decoder use the same convolution architecture as Pu et al., 2016 and trains the algorithm for 1000 episodes in the 2×2 scenario and 6000 episodes in the 8×8 scenario. The difference in episodes reflects the task's difficulty, and it is expected that DVAE converges faster to simpler environments. For the encoding of actions and states, we concatenate the flattened state-space and action-space, having a fully-connected layer with ELU activation before calculating the latent space. We used the Adam optimizer (Kingma and Ba, 2015) with a learning rate of 3e-05 to update the parameters. To calculate the loss, we used the same loss function as proposed by Kingma and Welling, 2013. The rest of the hyperparameters are the defaults, as seen in Table 5.1.

Figure 6.2 illustrates the loss during training of the DVAE algorithm in the No-Wall Deep



(a) Training loss for Deep-Maze-2 $\times$ 2. (b) Training loss for Deep-Maze-8 $\times$ 8.

Figure 6.2: The training loss for DVAE in the 2×2 No-Wall and 8×8 deep maze scenario. The experiment runs for 1000 episodes for 2×2 and 5000 episodes for 8×8. To demonstrate the effectiveness of VAE's interpolation capacity, we mask the bottom half of the diagonal of the observable state-space for 2×2 (POMDP style). For the 8×8, the state is fully observable.

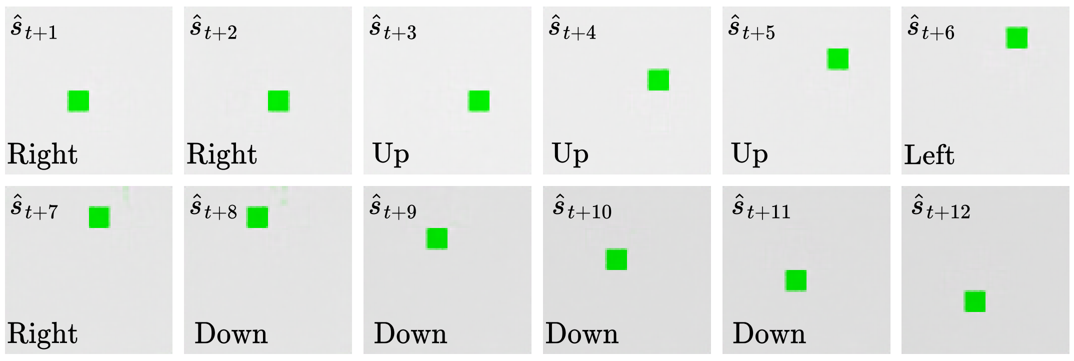


Figure 6.3: Results of 8×8 Deep Maze modeling using the DVAE algorithm. We simplify the experiments using no reward signals to guide the agent. The upper-left caption describes current state, $\hat{s}_t \ldots \hat{s}_{t+12}$, while the bottom-left caption is the action that transition the state $s_{t+1} = \hat{\mathcal{T}}(\hat{s}_t, a_t)$.

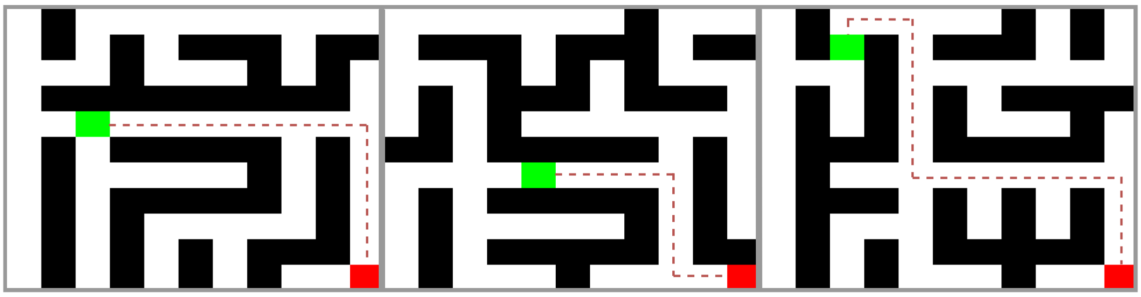


Figure 6.4: A typical deep maze of size 11×11. The lower-right square indicates the goal state, the dotted line is a retrace of the predicted optimal path for that maze, while the final square represents the player's current position in the state-space. The controller agent is DQN, TRPO, and PPO (from left to right).

Maze scenario. In the 2×2 scenario, DVAE trains on only 50% of the state-space, which results in noticeable graphics artifacts in the future state predictions, seen in Figure 6.1. In the 8×8 environment, the algorithm trains on all possible states. We observe in Figure 6.3 that the image quality and the capabilities of DVAE are significantly better for the fully observable case.

6.1.2.2 Performance - Deep Maze-No-Wall

The goal of the performance evaluation is to determine the empirical performance of off-policy model-free RL agents using an experience-replay buffer generated from the DVAE algorithm ($\hat{\mathcal{D}}$ from Algorithm 1). We evaluate the performance in the more complex environments Deep-Maze-11×11 and Deep-Maze-21×21. Table 6.1 compares the reward performance of DQN (Mnih et al., 2015), TRPO (Schulman et al., 2015), and PPO (Schulman et al., 2017) using the DVAE generated $\hat{\mathcal{D}}$ to tune the parameters. Note, however, that TRPO and PPO are on-policy algorithms and, the generated states must be generated online so that the algorithm remains *on-policy-ish* (i.e., uses a buffer-size of 1).

Figure 6.4 illustrates three maze variations of size 11×11, where the agent has learned the optimal path. We see that the best performing algorithm, PPO (2017), beats DQN and TRPO using either $\hat{\mathcal{D}}$ or $\mathcal{D}$. The DQN-$\hat{\mathcal{D}}$ agent did not converge in the 21×21 environment. However, value-based algorithms could likely struggle to map inaccurate states with graphical artifacts generated from the DVAE algorithm. These artifacts significantly increase the state-space, but empirical data suggest that on-policy algorithms perform better on noisy state-spaces.

Table 6.1: Results of the deep maze 11×11 and 21×21 environment, comparing DQN (Mnih et al., 2015), TRPO (Schulman et al., 2015), and PPO (Schulman et al., 2017). The optimal path yields a score of 100%, while no solution yields 0%. Each of the algorithms ran 10000 epochs for both map sizes. *Converged Epoch* represents at which epoch the algorithm converged during training.

Environment	Algorithm	Avg Perf.	Convergence Episode
11×11	DQN-$\hat{\mathcal{D}}$	94.56%	9314
11×11	TRPO-$\hat{\mathcal{D}}$	96.32%	5320
11×11	PPO-$\hat{\mathcal{D}}$	98.71%	3151
11×11	DQN-$\mathcal{D}$	98.26%	4314
11×11	TRPO-$\mathcal{D}$	99.32%	3320
11×11	PPO-$\mathcal{D}$	99.35%	2453
21×21	DQN-$\hat{\mathcal{D}}$	64.36%	N/A
21×21	TRPO-$\hat{\mathcal{D}}$	78.91%	7401
21×21	PPO-$\hat{\mathcal{D}}$	89.33%	7195
21×21	DQN-$\mathcal{D}$	84.63%	8241
21×21	TRPO-$\mathcal{D}$	92.11%	4120
21×21	PPO-$\mathcal{D}$	96.41%	2904

6.1.3 Recurrent Neural Networks

Using Recurrent Neural Networks improves the temporal capabilities of DVAE. We discover this combination in Paper D. The study expands further in Paper E.

6.1.3.1 Reconstruction - Deep Line Wars

Figure 6.5 illustrates the state quality during training of DVAE in a total of 6000 epochs. Both players draw actions randomly. The algorithm understands that the player units can be located in any tiles after only 50 epochs. At 1000 epochs, we observe that the DVAE makes significantly better predictions of the probability of unit locations (i.e., some units show more densely in the output state). At the end of the training, the DVAE algorithm can predict towers and unit locations at any given timestep during the game epoch.

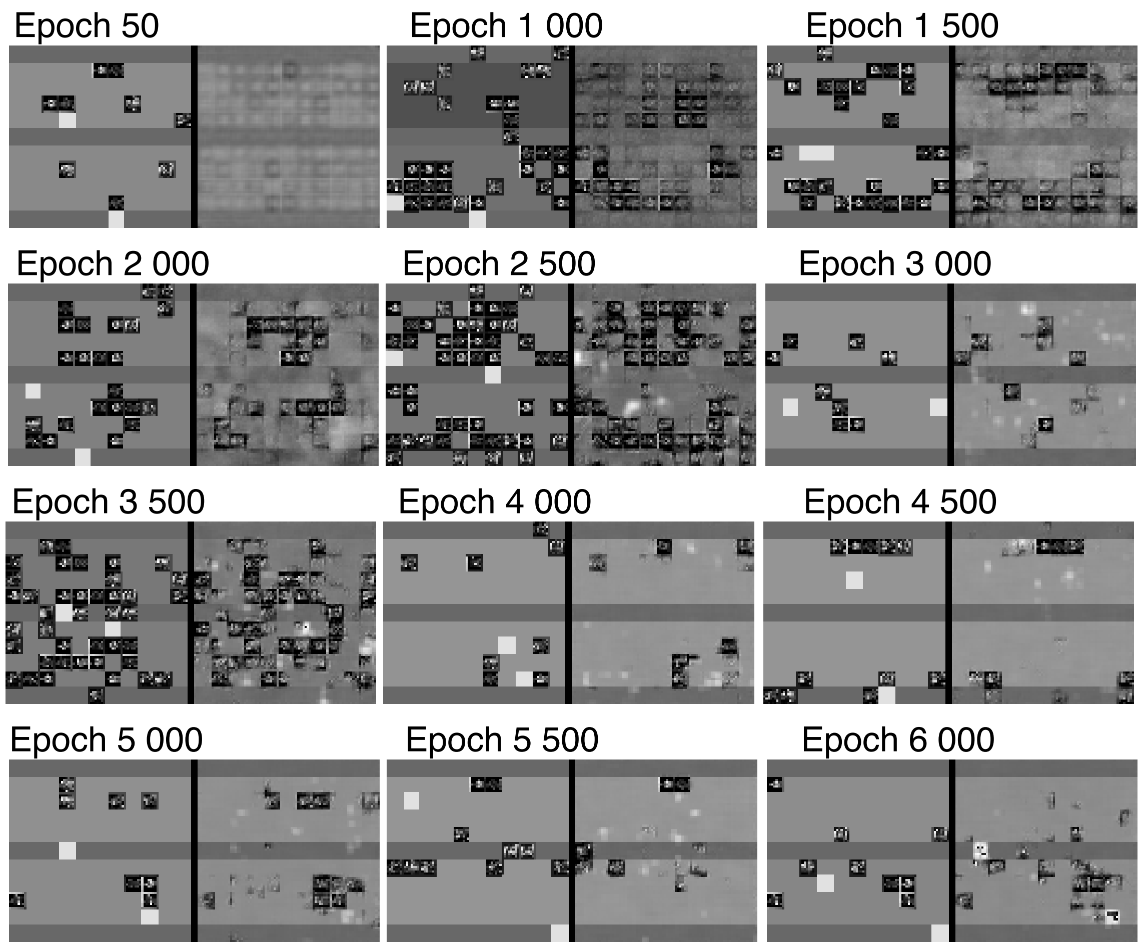
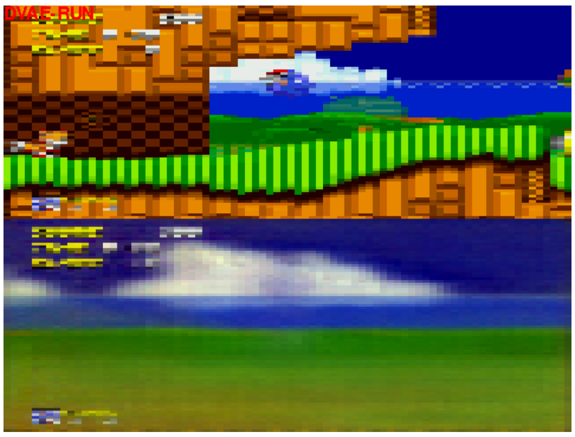
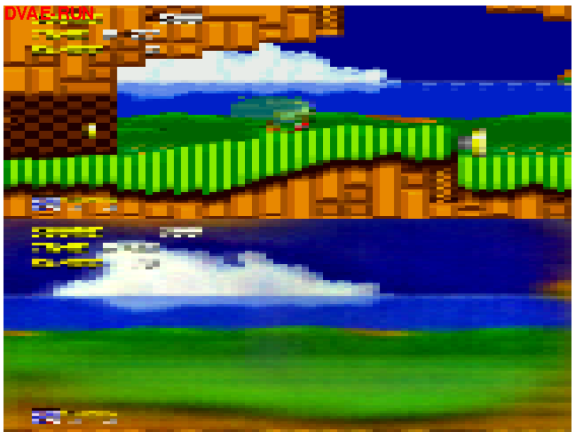


Figure 6.5: The DVAE algorithm applied to the deep line wars environment. Each epoch illustrates the quality of generated states in the game, where the left image is the real state s_t, and the right image is the predicted state $\hat{s}_t$.

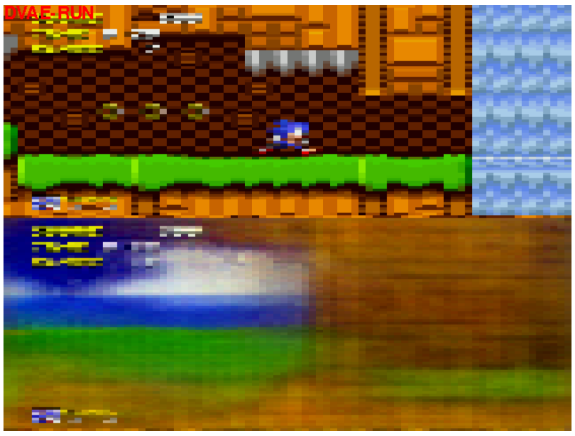
6.1.3.2 Reconstruction - Sonic The Hedgehog 2



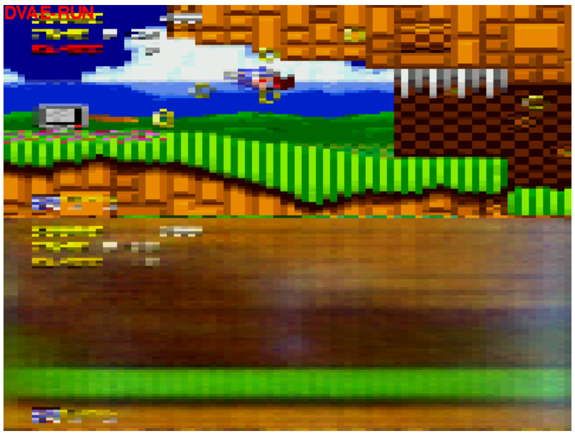
(a)



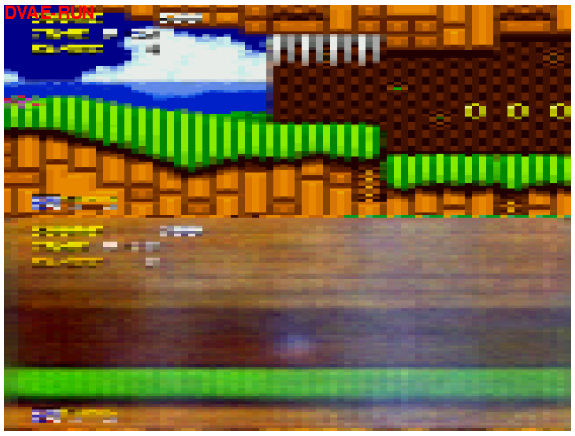
(b)



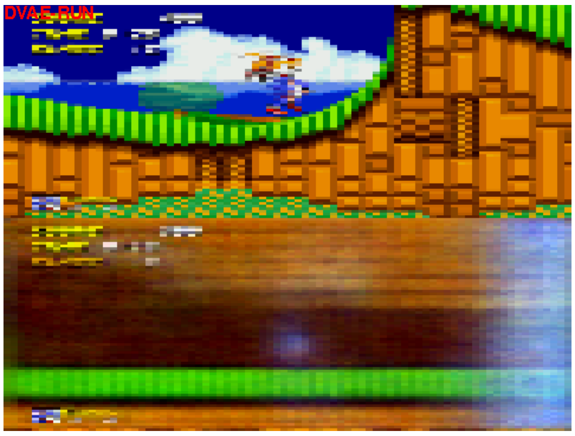
(c)



(d)



(e)



(f)

Figure 6.6: Dreaming in Sonic The Hedgehog 2 game from Sega Genesis. The figure demonstrates the capability of DVAE to reconstruct animated games, and as clearly seen, the algorithm fails to reconstruct states that are close to the ground truth. The upper image is the ground truth s_t, and the bottom is the n-step prediction $\hat{s}_{t+3}$.

Figure 6.6 demonstrates the prediction capabilities of DVAE for more complex games. As clearly seen, DVAE has difficulties predicting future states in Sonic the Hedgehog 2. The first row shows the ground truth observations, while the second row shows the corresponding predictions. A common problem with VAE's is that the signal from observations is too weak, resulting in the decoder ignoring the latent variables. We can think of the phenomena as the model learning to produce output independently of the observation space. This results in highly general output, which is essentially a mean of all observations seen thus far. The problem is infamously characterized as a posterior collapse, seen in the second row of Figure 6.6a. We further study this problem in Section 6.4.3 in the ORACLE algorithm, which demonstrates a significant reconstruction performance improvement than DVAE.

6.1.3.3 Performance Evaluation

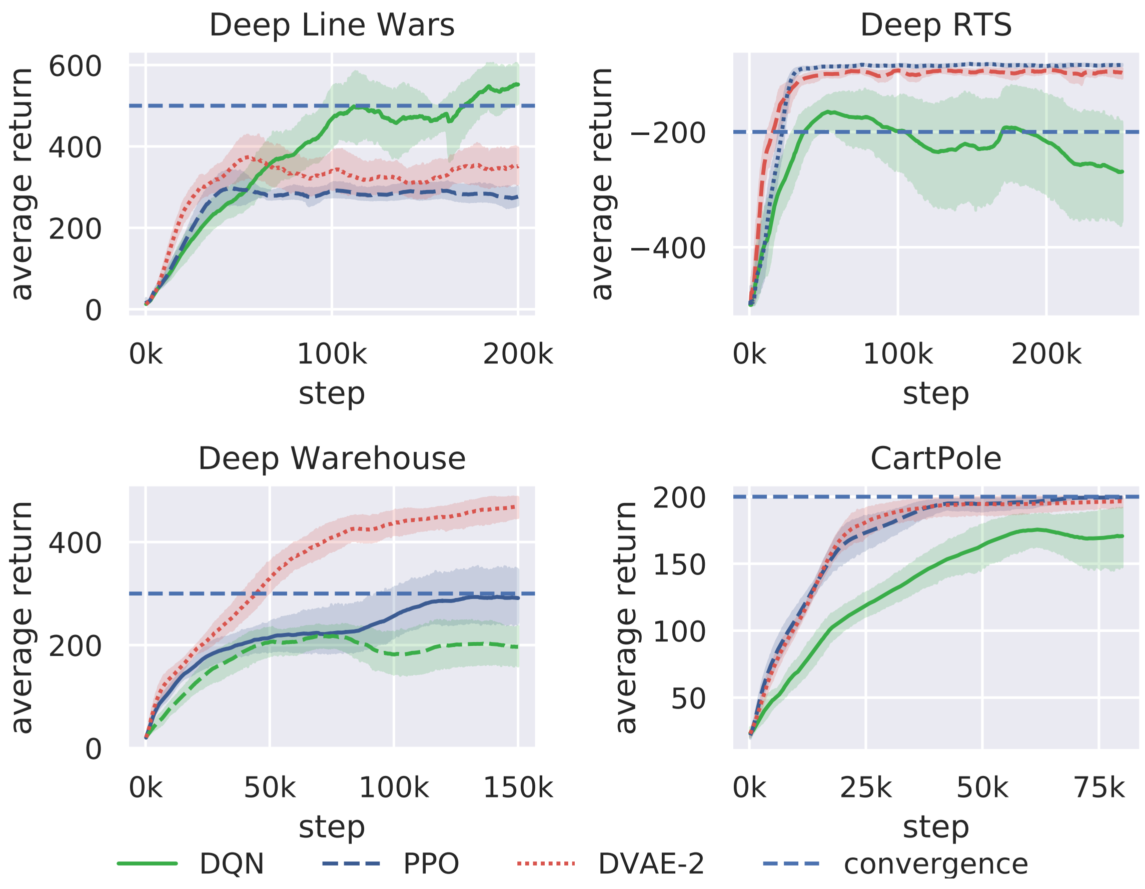


Figure 6.7: We compare DVAE with RNN (DVAE-2 in the figure) using two baseline algorithms, DQN and PPO. The solid curve illustrates the mean of 12 trials, and shaded regions are the standard deviation between all trials. The x-axis shows the number of episodes, and the y-axis shows the average return.

Figure 6.7 demonstrates the average reward performance of DVAE in four tasks; Deep RTS Paper C, Deep Warehouse Paper F, Deep Line Wars Paper B, and CartPole-v1 (Brockman et al., 2016). We now evaluate the results in an orderly fashion.

Deep Line Wars The DQN policy outperforms the DVAE and PPO policy in the Deep Line Wars-11 $\times$ 11 discrete action-space environment. DVAE uses PPO as the decision policy and sees a marginal improvement over traditional PPO in a model-free setting yielding better reward performance and better sample efficiency. DQN quickly learned the correct Q-values due to the small environment size.

Deep RTS The Deep RTS Wood-Collect experiment tasks the agent to harvest 500 wood resources before the timestep limit is exhausted. The environment scores performance from -500 to 0, where 0 is the best possible score. For every wood harvested, the score increased by 1. The agent masters the task if the score is above -200 score at the time of the terminal state. DVAE outperforms the baseline algorithms in sample efficiency but falls behind PPO in reward performance.

Deep Warehouse The DVAE algorithm outperforms PPO and DQN in sampling and reward performance during 150 000 game steps. The score function is a counter of how many tasks the agent has performed during the episode. If the agent collects and retrieves 300 packages, the agent has sufficient reward performance to beat many handcrafted algorithms in ASRS. The environment is multi-agent, and in this experiment, we used a 30 $\times$ 30 grid with 20 taxis running the same policy.

CartPole-v1 DVAE and PPO have similar reward performance, but DVAE has marginally better sample efficiency after 25 000 timesteps. Using the VRC architecture (Section 5.1.5), the algorithm works best with Convolutional + LSTM and Temporal Convolution and GAN for continuous control tasks (see Figure 5.4). PPO and DVAE use the same hyper-parameters and are therefore directly comparable. We use PPO as the policy for DVAE but observe that DVAE is more sample efficient and performs equally good or better than model-free PPO in all tested scenarios.

6.1.4 Generative Extensions

DVAE-SWA, DVAE-GAN, and DVAE-SWAGAN aim to better the reconstruction capabilities than DVAE in continuous state-spaces, such as the Deep Line Wars environment. We evaluate these extensions using the action policy of Deep Line Wars-11 $\times$ 11 with PPO as the action policy. The extension Stochastic Weight Averaging (SWA) is shown to improve training stability in supervised learning, and Generative Adversarial Networks (GAN) is a generative model which has demonstrated good reconstruction performance in generating high-resolution images.

6.1.4.1 Training Evaluation

Figure 6.8 shows the training loss comparison of DVAE, DVAE-SWA, DVAE-GAN, and DVAE-SWAGAN for 1000 epochs (x-axis), where the y-axis describes the loss value. DVAE-SWA, DVAE-GAN, and DVAE-SWAGAN have significantly lower training loss than DVAE, where DVAE-SWAGAN is the superior model. This results in significantly better prediction accuracy for environments with animations, such as Deep Line Wars.

6.1.4.2 Reconstruction - Deep Line Wars

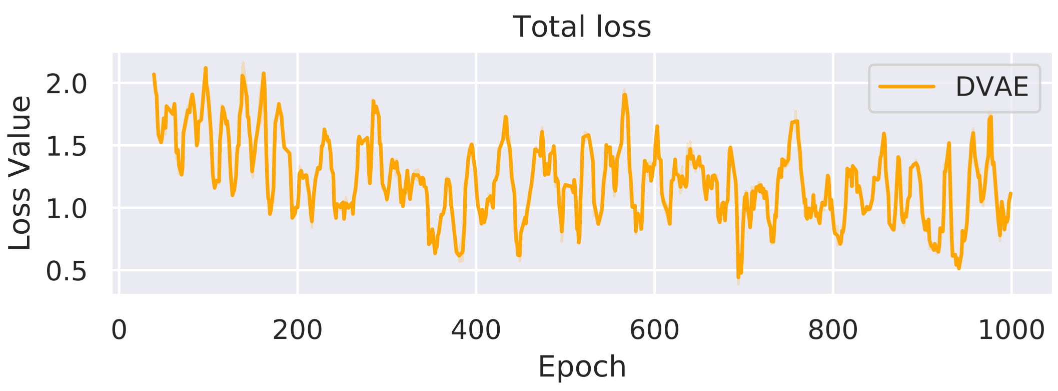
Figure 6.9 demonstrates the reconstruction capabilities of DVAE-SWAGAN for the Deep-Line-Wars 11×11 environment. The DVAE-SWAGAN algorithm sampled states after training for 1000 epochs, as seen in Figure 6.8. Despite that the DVAE-SWAGAN method still suffers from posterior collapse as in DVAE, the RL agents still learn a policy capable of beating random agents. We believe this is because similar states often represent similar value functions. Compared to Figure 6.5, DVAE-SWAGAN performs significantly better, considering it provides more clear imaging and can reconstruct several dimensions (RGB) and, while not perfect, can predict the position of towers reasonably well.

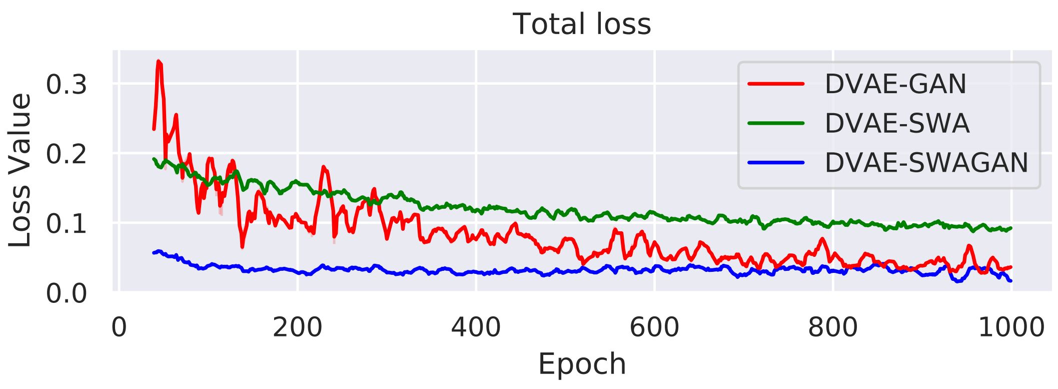
6.1.5 CostNet

The main objective of the CostNet extension is to predict distances between two states in an environment or MDP. We test the DVAE-CostNet algorithm in four environments, CartPole-v1 from 2016, DeepRTS-GoldCollect Paper C, and DeepMaze StaticNoWalls Paper D. The experiments compare DVAE-CostNet to DQN (Mnih et al., 2015) and PPO (Schulman et al., 2017) for 1000000 timesteps during 100 experiments for statistical analysis of the results.

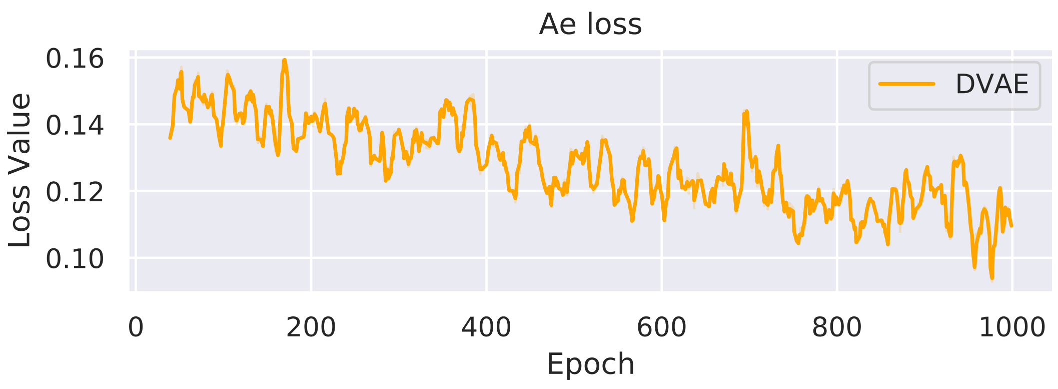
6.1.5.1 Evaluation Setup

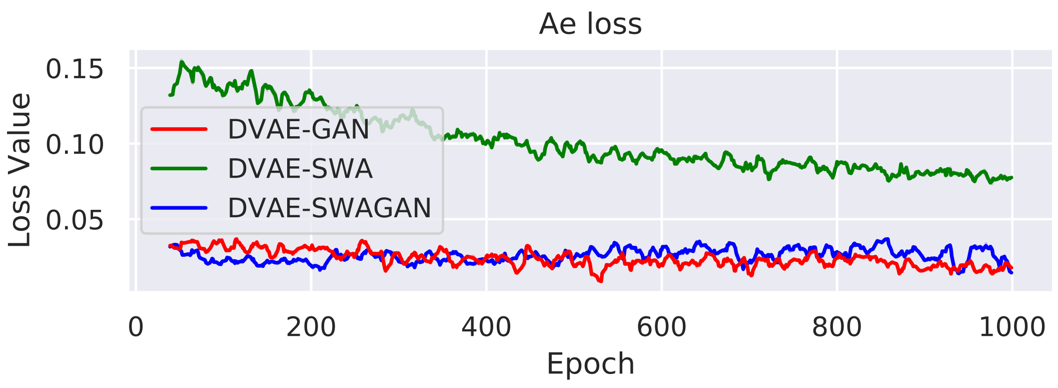
The hyperparameters for DVAE-CostNet are found in Table 5.2. Figure 6.10 shows the environments used in the experiments. The first environment is CartPole, a common benchmark for exploratory reinforcement learning research. The objective is to balance a pole on a cart for 500 timesteps at which the episodes end. The second environment is DeepRTS- GoldCollect, a simple environment where the goal is to accumulate as much gold as possible for 5 minutes. The optimal episodic reward for this environment is set to 1000. Finally, the DeepMaze-StaticNoWalls environment is an 11×11 grid structure



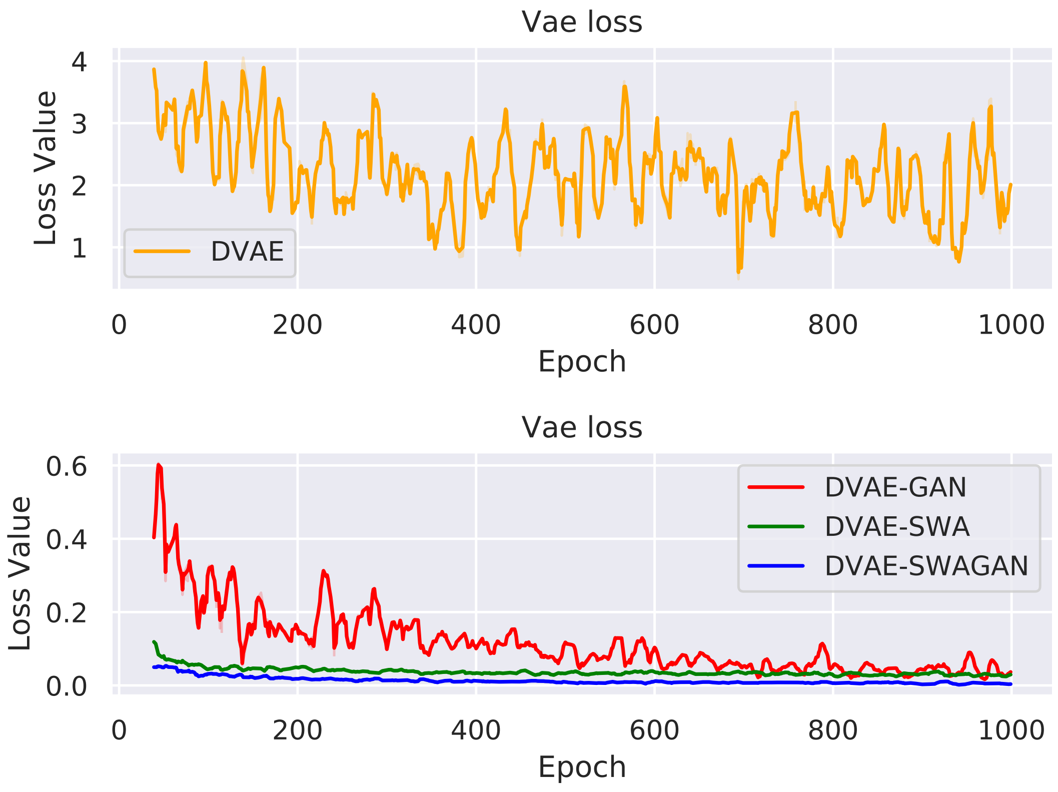


(a) Total Loss for the tested algorithms.





(b) Autoencoder loss (L2) for the algorithms.



(c) Variational Autoencoder regularizer (KL) for the tested algorithms.

Figure 6.8: Training of DVAE compared to the DVAE-SWA, DVAE-GAN, and DVAE-SWAGAN extension. (a) describes the total training loss of the model for the respective models. The total training loss includes reconstruction and regularization. (b) denotes the reconstruction loss, which typically indicates how well the output corresponds to the true observation. (c) is the regularizer term measuring the KL-divergence of the prior and the posterior distributions.

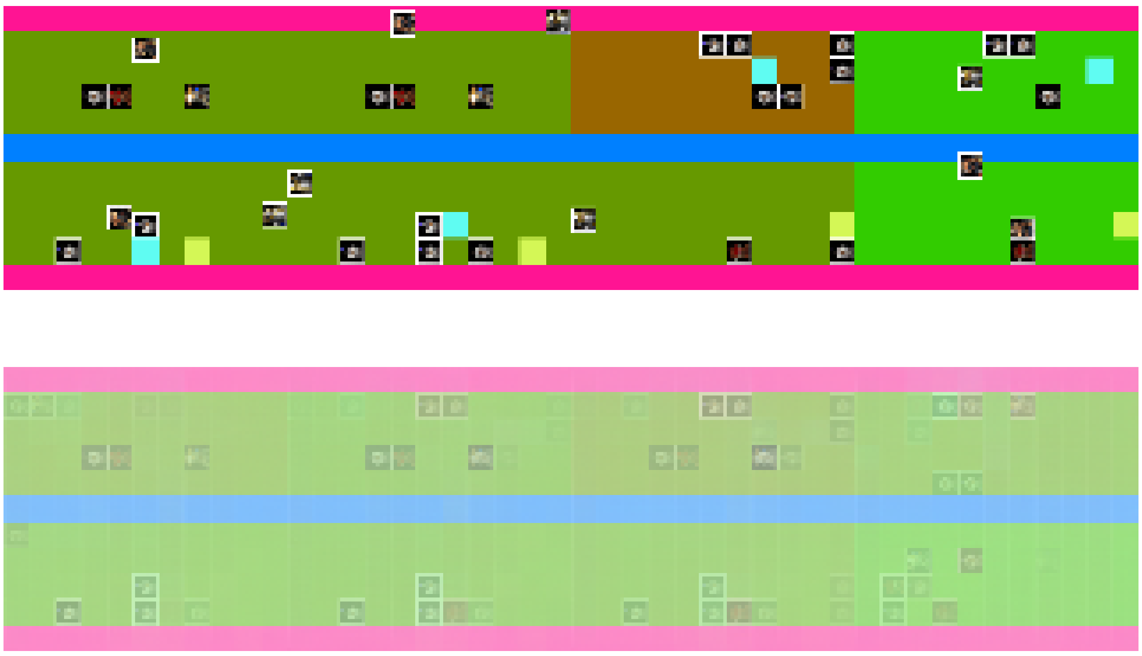


Figure 6.9: The first row represents the ground truth future state, and the second row is the predicted future state using DVAE-SWAGAN. The predictions are 3-step future state predictions.

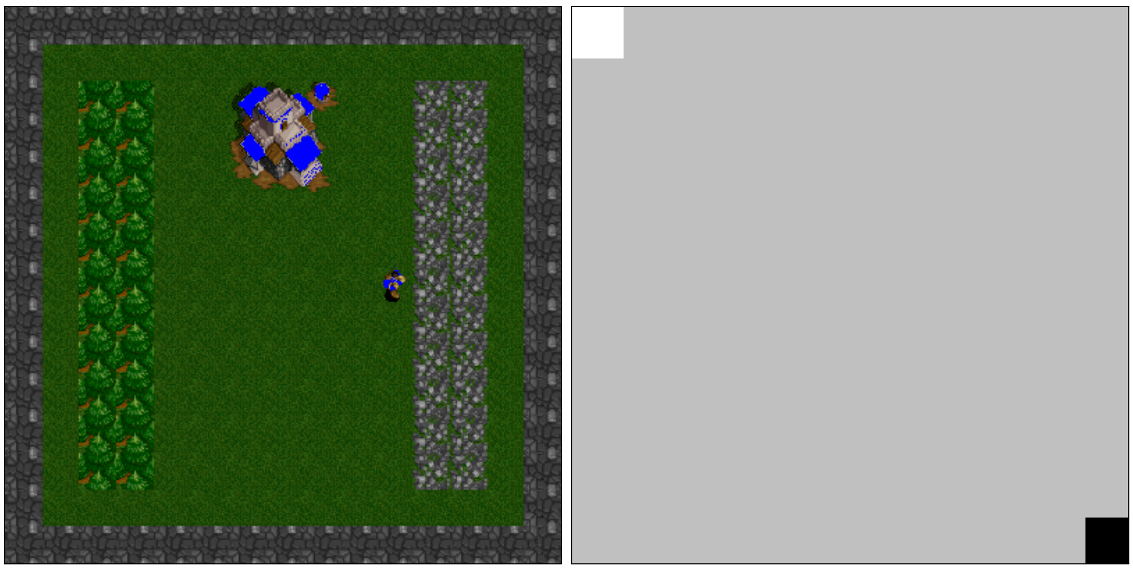


Figure 6.10: Illustration of the experiments. (a) DeepRTS-GoldCollect. Gather as much gold as possible in a timeframe of 5 minutes. (b) DeepMaze. The player (white) must enter the terminal state (black) in the shortest possible time.

where the goal is located at a fixed position. The reward for DeepMaze is the length of the maze because the agent and goal are located at opposite corners.

6.1.5.2 Empirical Evaluations

Figure 6.11 compares the performance of DVAE-CostNet against two competing algorithms, DQN and PPO. Every experiment runs for 100 episodes for 1 million timesteps. DVAE-CostNet shows outstanding performance compared to fully model-free variants in two of three environments. In CartPole, PPO works best, but DVAE-CostNet closely follows. The experiments show that increasing the drift-threshold ψ also decreases performance and is an indication that CostNet impacts performance positively. Several parameters for the drift-threshold parameterψ are tested, but a value of 0.3 seems stable across several environments. PPO algorithm uses the parameters defined in 2017 and DQN in Mnih et al., 2015. DVAE-CostNet shows significantly better performance across all environments in terms of variance, seen clearly in the DeepMaze environment results. The primary reason is that the algorithm starts with a relatively good idea of the underlying environment dynamics from learning the dynamics model.

In terms of reward performance, the DVAE-CostNet agent starts at near-optimal performance in some environments, such as the DeepRTS-GoldCollect environment. There are still challenges to be investigated, such as preventing divergence if the policy is already doing good behavior. Another problem is that DVAE-CostNet demands initial data from expert systems which, is challenging to guarantee in any environment. Regardless of these challenges, the algorithm is a good leap in the right direction. DVAE-CostNet with a modified DQN reward function significantly increases the agent's reward performance, especially in more complex environments such as Deep RTS.

6.1.6 Summary

We demonstrate that the DVAE algorithm, with its extension, can predict future states well in many environments. The algorithm falls short when modeling highly complex environments such as the Deep Line Wars and Sonic the Hedgehog 2. According to the performance evaluations, DVAE can perform comparably to the PPO algorithm in terms of agent reward performance with the added benefit of significantly increasing the sample efficiency.

Hypothesis DVAE 1 *DVAE can accurately model simple environments such as the Deep Maze without walls.*

Evaluation: According to the evidence in Figure 6.1 and Figure 6.3, it is safe to assume that DVAE can accurately model fully-deterministic, simple environments.

Hypothesis DVAE 2 *DVAE models complex environments such as Deep Line Wars and Sonic The Hedgehog without difficulties.*

Evaluation: According to the evidence in Figure 6.5 and Figure 6.6, it is not likely that DVAE can model other complex environments better.

Hypothesis DVAE 3 *DVAE outperforms traditional RL methods in reward performance while also having higher sample efficiency.*
Evaluation: According to our evaluation, DVAE can outperform traditional RL methods using the learned latent space as the foundation for policy learning, as seen in Figure 6.7.

Computational Power

The most decisive factor for training the DVAE algorithm in more complex problems is the availability of computational power. Training a deep learning model is a complex endeavor that requires millions of parameters to update many times to find a good point on the optimization plane. According to AI pioneer Richard S. Sutton:

"The biggest lesson that can be read from 70 years of AI research is that general meth-

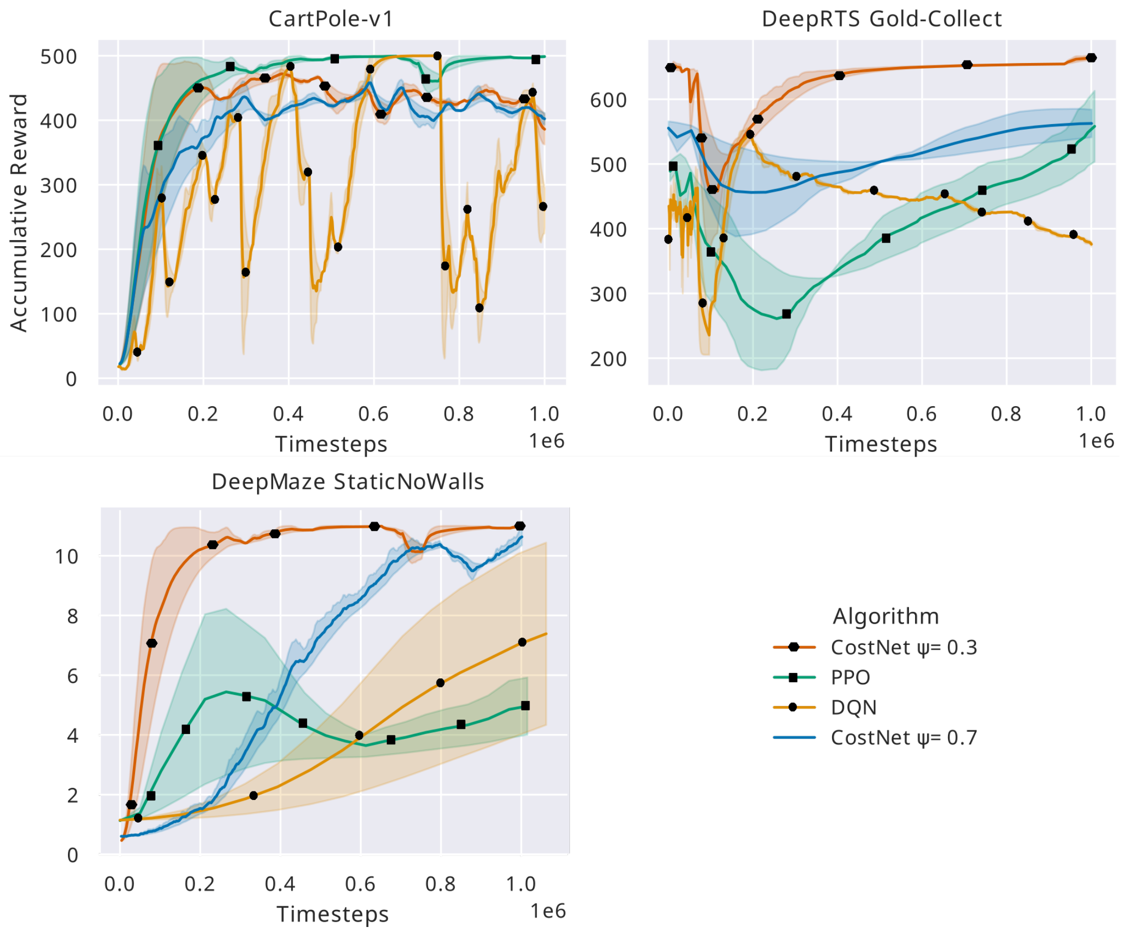


Figure 6.11: A comparison of PPO (square), DQN (circle), CostNet (unmarked and diamond) performance in CartPole, DeepRTS, and Deep Maze environment. The y-axis shows the accumulated reward, and the x-axis is at which timestep.

ods that leverage computation are ultimately the most effective, and by a large margin." (Sutton, 2019)

Based on observations from our study, our interpretation of his statement is that algorithms that can saturate computational power well (e.g., guided brute force searches) demonstrate superior reward performance. DVAE trains using two NVIDIA 2080 RTX TI GPU cards that, if tuned properly, can operate at approximately 26.9 TFLOPS. Compared to our computational power, AlphaStar from Vinyals et al., 2019 consumes a total of 6720 TFLOPS considering 16 TPU3 units for a single agent. The AlphaStar algorithm trains using more than a single agent for several weeks. These observations combined paint a picture that the leading indicator for successive algorithms is the available computational power, not the model design. Considering these factors, we believe that the DVAE algorithm, set aside its weaknesses, is a good step in the right direction for an expressive prediction model that requires little computational power compared to similar work.

6.2 S-DVAE Evaluations

The goal of the Safe Dreaming Variational Autoencoder (S-DVAE) evaluations is to demonstrate that S-DVAE can learn an accurate dynamics model and learn good policies while behaving safely during learning. S-DVAE uses risk-directed exploration and curiosity and integrates with DQN (Mnih et al., 2015) (Section 5.1.6). We compare our algorithm to RAINBOW (DQN) and PPO. The section evaluates the S-DVAE in various environments, including the popular Atari 2600 Learning Environment (Bellemare et al., 2013), Deep RTS (Paper C), Deep Line Wars (Paper B), and Deep Warehouse (Paper E) for industry-near approximations.

6.2.1 Hypotheses

We carefully address the following hypotheses for the S-DVAE algorithm.

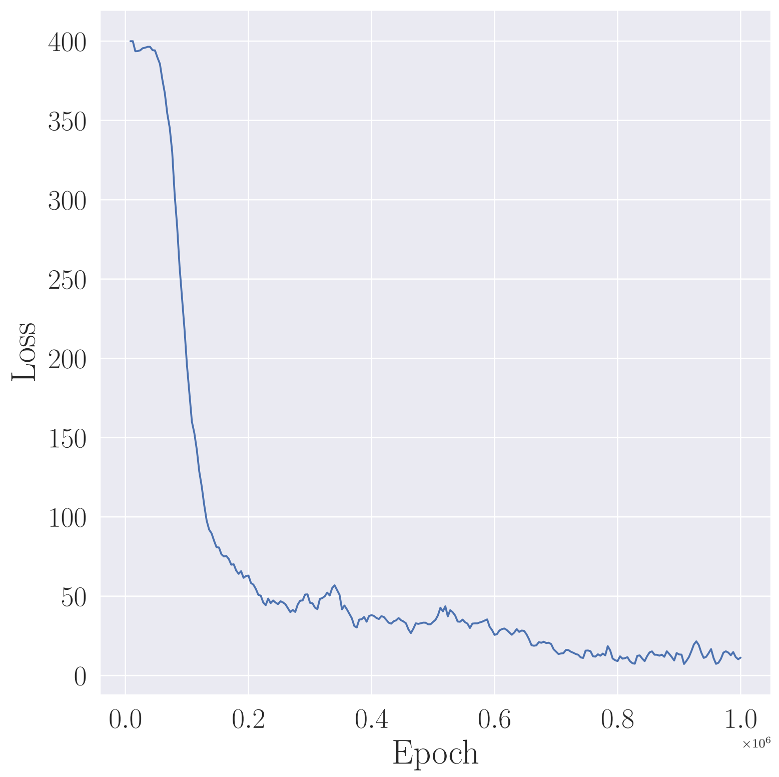
Hypothesis S-DVAE 1 *S-DVAE is a safer option than DVAE and model-free approaches.*

Hypothesis S-DVAE 2 *S-DVAE guarantees safety during training.*

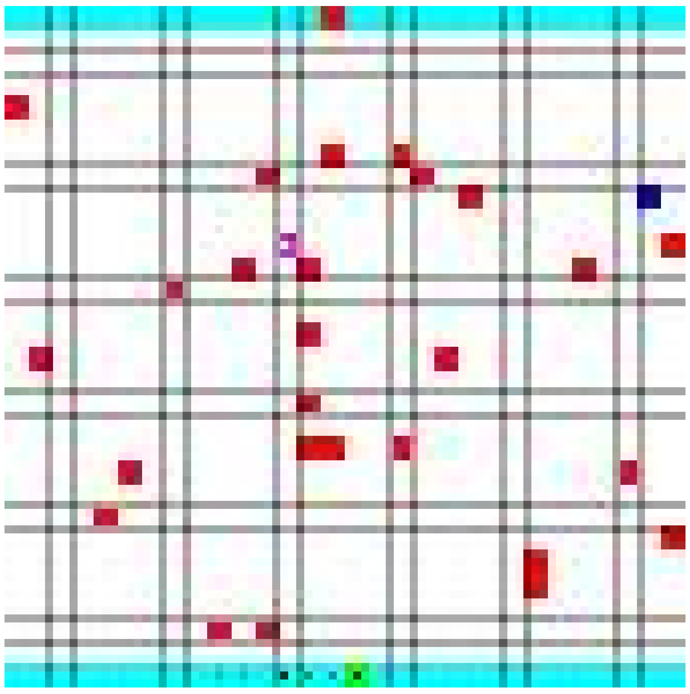
Hypothesis S-DVAE 3 *Tuning the S-DVAE algorithm is trivial to tune most tested environments.*

6.2.2 Dynamics Model Evaluations

The dynamics model's objective is to learn environment dynamics and features to mimic the environment behavior accurately. Figure 6.12a illustrates the average training loss



(a) Average dynamics model M reconstruction loss. The loss function is the mean-squared error between s_t and $\hat{s}_t$.



(b) Decoder output after training. The decoder output of the dynamics model after 5 hours of training without speed acceleration in the Deep Warehouse-41x41 environment.

Figure 6.12: S-DVAE dream constructions in the Deep Warehouse game environment with the corresponding reconstruction loss. The goal is to reconstruct states which are visually similar to Figure 4.9.

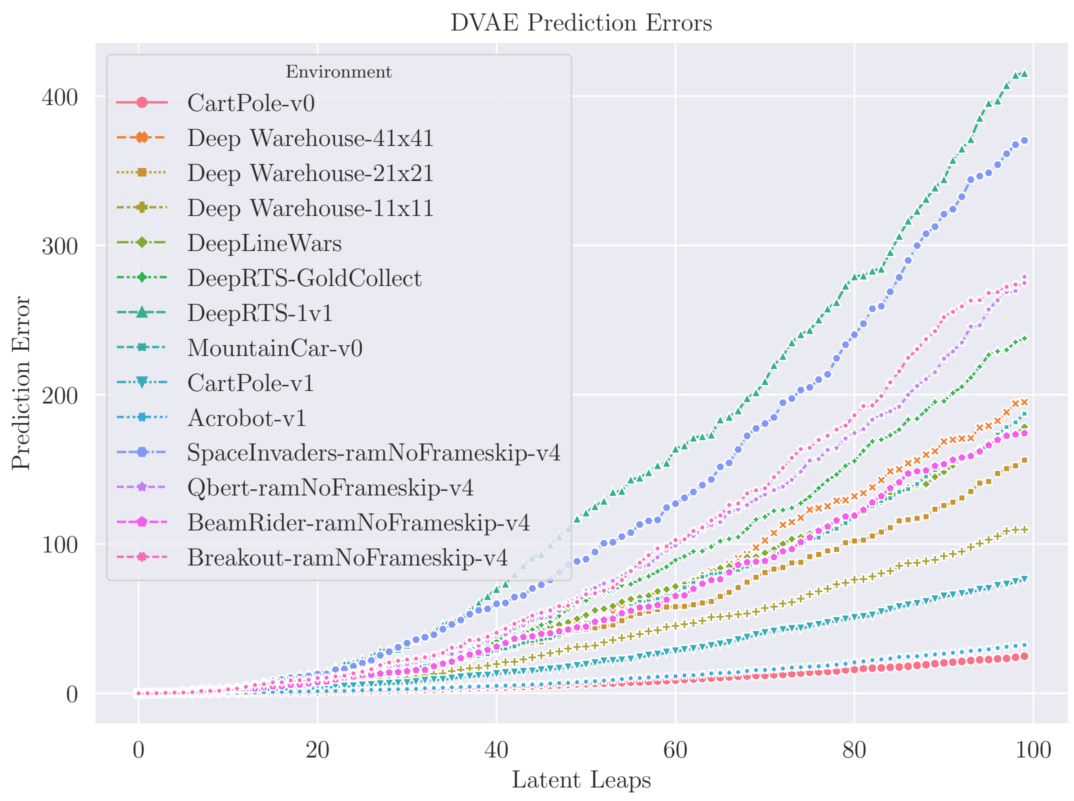


Figure 6.13: Cumulative Prediction Error. The y-axis shows the pixel error where each number represents a 2-dimensional error. For example, an error of 32 means that 32 $\times$ 32 pixels have incorrect values. The x-axis is how many predictions in the future are made without interaction with the real environment (how many states in the future has the algorithm "dreamed"

.

for tested environments. The x-axis ranges from 0 to 1 000 000 timesteps, and the loss is according to Equation 5.4, where lower values indicate better *dream* predictions. The trend is for the loss to start high and quickly reduce to only minor weight adjustments during training. These minor weight adjustments play a significant role in learning accurate embeddings, as illustrated by Figure 6.12b. The dynamics model uses hand-crafted expert systems designed to perform well in the specific environment during evaluations and experiments.

A way to measure the prediction capabilities of the dynamics model is to investigate the cumulative prediction error. Figure 6.13 illustrates this cumulative prediction error for all tested environments. The experiments show that the prediction error tends towards exponential growth when the dynamics model predicts longer horizons. Table 6.2 shows that the dynamics model has an error of $\sim$256 (the observations are $256 \times 256 \times 1$) which means that every pixel in the prediction is incorrect, making it difficult to use for training model-free algorithm.

Table 6.2: Exponential cumulative prediction error. The cumulative prediction error increases exponentially for all environments. The table shows that the exponential growth is consistently less extreme for simple environments. The numbers in the header present the n-th future state.

Environment	**10**	**25**	**50**	**75**	**100**
Deep Warehouse-11x11	0.34	2.08	7.47	14.91	25.10
Deep Warehouse-21x21	0.36	1.91	7.53	16.62	28.75
Deep Warehouse-41x41	0.43	2.98	10.50	25.06	43.02
Deep Line Wars	0.54	3.81	13.73	29.41	50.58
DeepRTS-1v1	2.72	16.29	65.98	143.02	255.99
DeepRTS-GoldCollect	0.69	4.55	19.75	46.17	78.00
Acrobot-v1	0.42	2.04	9.15	19.56	34.86
BeamRider-ramNoFrameskip-v4	1.77	9.33	33.99	75.94	135.49
Breakout-ramNoFrameskip-v4	1.52	7.20	28.84	67.37	110.48
CartPole-v0	0.31	1.52	6.13	13.62	22.58
CartPole-v1	0.35	1.50	6.66	14.53	26.01
MountainCar-v0	0.65	3.66	15.31	30.73	49.95
Qbert-ramNoFrameskip-v4	0.77	4.20	18.23	38.69	63.38
SpaceInvaders-ramNoFrameskip-v4	1.04	4.98	21.96	53.11	89.28

It is sensible to restrict the prediction horizon for advanced or difficult environments. The downside of limiting the prediction horizon is that the algorithm cannot train fully offline. However, the algorithm successfully reduces the volume of real training data needed to converge model-free approaches by magnitudes.

The dynamics model successfully learns several environments sufficiently, including ASRS-Lab-21x21 (Deep Warehouse), CartPole, and Deep Line Wars. A potential improvement in the model is to tune the α, w, and learning rate to improve other environments' accuracy. However, hyperparameters remain problem-specific and must be carefully tuned.

6.2.3 Agent Failure Rate

The failure rate measurement counts the number of negative rewards the agent receives during an episode. The environment has a negative reward for catastrophic states and positive rewards for other states. Recall that the algorithm should interpret the MDP with constraints and label catastrophic states accordingly (Figure 2.4).

Figure 6.14 illustrates the failure-rate for S-DVAE with three hyperparameter configurations, $\alpha = 0.99, w = 0.01$, $\alpha = 0.7, w = 0.3$, $\alpha = 0.5, w = 0.5$. Recall that higher α and lower w values account for safety-aware behavior. Safer configurations of S-DVAE clearly impacts the rate by which the algorithm makes mistakes.

The algorithm does not always learn good policies, such as in the DeepRTS environment. The reason is that the reward function does not represent the goal, and further investigations discover that this is indeed the case for DeepRTS-1v1. S-DVAE outperforms PPO and DQN significantly for the DeepRTS-GoldCollect environment, with a more direct (and more straightforward) reward function.

S-DVAE increases safety significantly for most of the environments tested in these evaluations. The results from Figure 6.14 show a consistent decrease in failures when tuning the safety-awareness sensitivity hyperparameter, α, and w. The benefits of having high values for the safety-awareness parameters are that it increases action safety, but on the other hand, slows down convergence and increases the chance of getting stuck in local optima.

6.2.4 Agent Reward Performance

S-DVAE has comparable performance to DQN and PPO in terms of accumulating reward. Figure 6.15 shows the performance after S-DVAE is pre-trained on the dynamics model. Experiments use DeepLineWars, DeepRTS-GoldCollect, Acrobot, CartPole, and Deep Warehouse-41x41. Furthermore, the figure demonstrates that S-DVAE trains successfully to sufficient behavior level and can train additionally in the real environment after initial

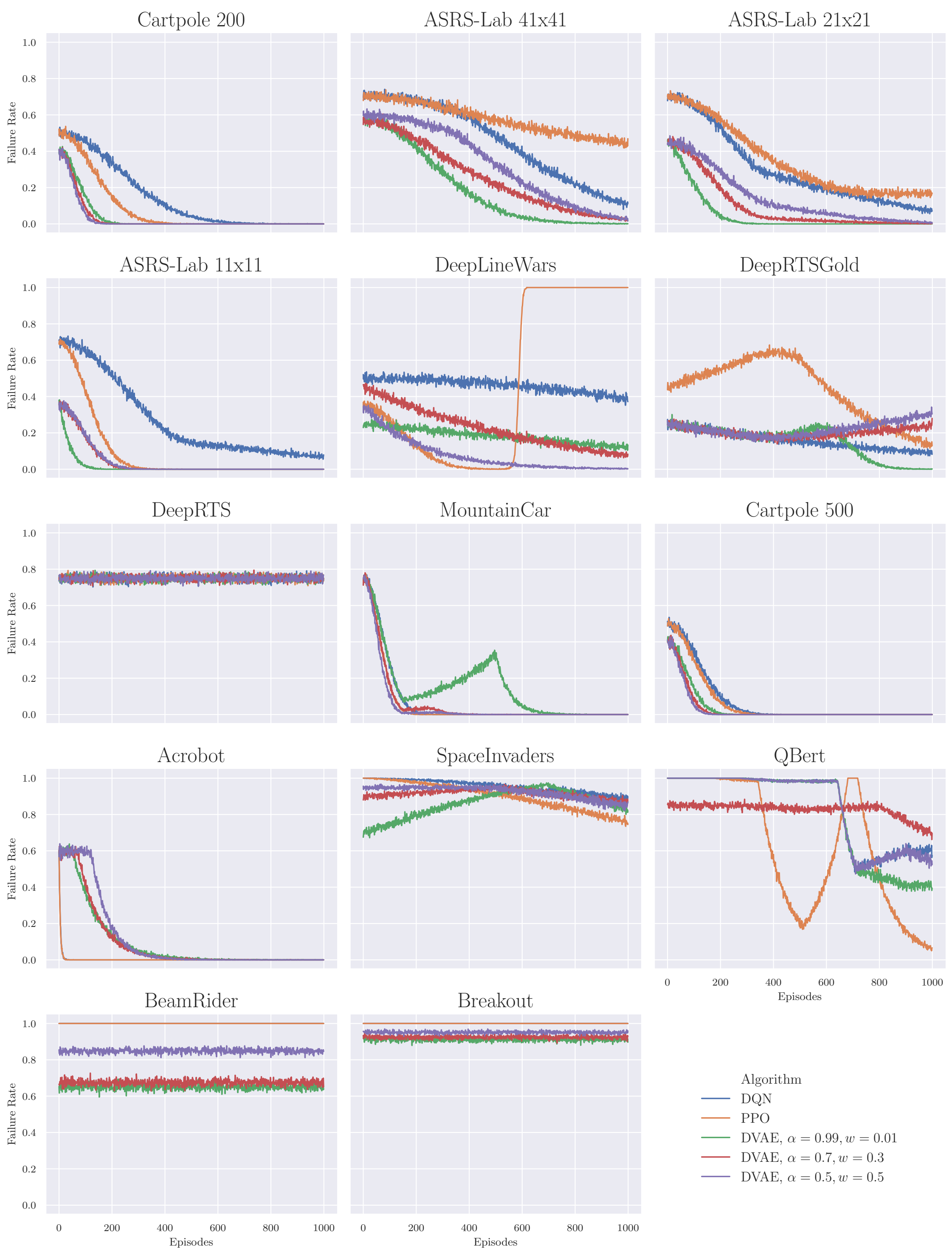


Figure 6.14: Agent failure rate. We evaluate the rate at which an agent fails during trials across various environments. The x-axis illustrates the episode number, and the y-axis is the rate in percentage. Each environment is averaged over 100 trials for 1000 episodes. We compare three safety configurations of DVAE against DQN (Mnih et al., 2015) and PPO (Schulman et al., 2017)

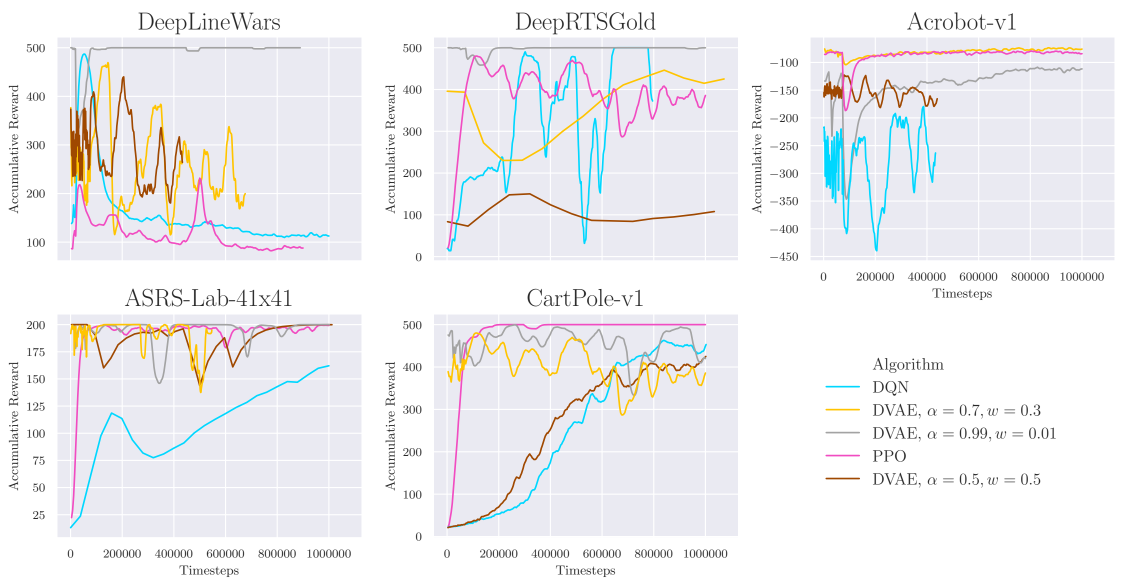


Figure 6.15: Behavioral agent performance. S-DVAE shows good performance when accumulating reward (y-axis) during training for 1 million timesteps (x-axis). The experiment was averaged across 100 runs and, due to execution time, limited to only a subset of the environments

training.

S-DVAE is not always stable when training in complex environments, as seen in the DeepLineWars experiments. Out of 100 trials, the S-DVAE configuration using $\alpha = 0.7$ and $\alpha = 0.5$ diverged dramatically, which early-stopped the experiment before reaching 1 million timesteps.

The pretraining was done using a horizon of 40 frames for 2 million timesteps. In practice, this only results in 50000 timesteps in the real environment, resulting in magnitudes lower risk of failures. However, sensitivity to hyperparameters is a significant issue that limits the algorithm from functioning well throughout all tests without extensive hyperparameter tuning.

6.2.5 Summary

Given the empirical evidence and evaluation, we find that S-DVAE can improve decision safety in numerous games, including Deep RTS, Deep Line Wars, and industry-near applications such as Deep Warehouse. We can conclude our evaluation hypotheses as follows.

Hypothesis S-DVAE 1 *S-DVAE is a safer option than DVAE and model-free approaches.*

Evaluation: According to the empirical evidence, the methods introduced in S-DAVE improve safety. By tuning the hyperparameter α and w, we demonstrate that

S-DVAE gradually becomes more safety-aware, as seen in Figure 6.14. However, it is not exclusively safer for all environments, but our observation is that tuning the hyperparameters alleviate instabilities. Perhaps the most considerable challenge present in both DVAE and S-DVAE is posterior collapse, causing the algorithm to fail to behave safely.

Hypothesis S-DVAE 2 *S-DVAE guarantees safety during training.*

Evaluation: S-DVAE cannot guarantee safety during training, and we discuss this further in our conclusion. Our approach provides safety hints to the agent, pushing exploration and decisions towards states that the dynamics model deems safer. However, to guarantee safety in RL, current literature utilizes policy constraints following many mathematical assumptions about model stability and accuracy, which is not realistic for industry applications.

Hypothesis S-DVAE 3 *Tuning the S-DVAE algorithm is trivial to tune for most tested environments.*

Evaluation: The S-DVAE algorithm has two safety hyperparameters, α, and w. Although there are few hyperparameters, it is still now fully known which factors determine the weight of these parameters. However, we have discovered three configurations that appear to work well for the tested environments, as illustrated in Figure 6.14.

6.3 DVQN Evaluations

We present experiments and perform empirical evaluations in four traditional RL environments to demonstrate the effectiveness of the DVQN algorithm. The goal of the DVQN algorithm is to learn structured latent spaces that are interpretable to humans and can cluster into multiple behavioral policies. The algorithm is further detailed in Section 5.2. We show that the algorithm can organize the latent space by observation similarities while maintaining comparable reward performance to model-free Deep Q-Networks algorithms.

6.3.1 Hypotheses

We carefully address the following hypotheses for the DVQN algorithm.

Hypothesis DVQN 1 *DVQN can learn interpretable observation/state-spaces.*

Hypothesis DVQN 2 *DVQN allows clustering of observation spaces towards automatic option-based learning.*

Hypothesis DVQN 3 *DVQN provides interpretability while also maintaining a healthy agent reward performance.*

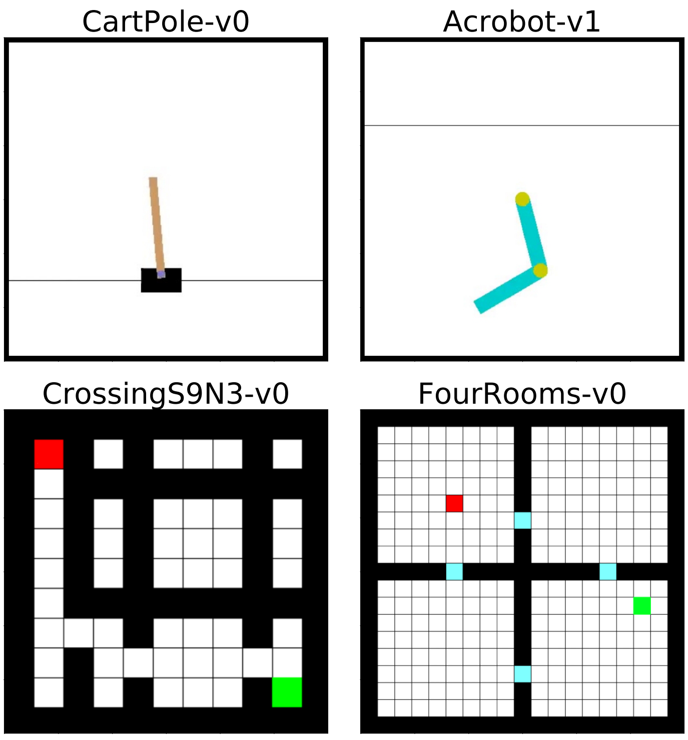


Figure 6.16: The experiment test-bed contains the following environments; CartPole-v0, Acrobot-v1, CrossingS9N3-v0, and FourRooms-v0

6.3.2 Evaluation Description

We evaluate DVQN in four environments; CartPole-v0, Acrobot-v1, CrossingS9N3-v0, and FourRooms-v0. Figure 6.16 shows a brief overview of the environments. These environments are trivial to solve using model-free reinforcement learning. This makes them excellent for conducting experiments with the sole focus of visualizing the learned latent space.

The FourRooms-v0 environment is especially suited for option-based RL and can solve the problem in a fraction of time steps compared to model-free RL. Although the DVQN algorithm does not use an options-based algorithm, the primary goal is to organize the latent space so that it is possible to extract meaningful and interpretable options automatically. The scope for this evaluation is to provide comparable reward performance to variants of Deep Q-Networks (Chen et al., 2016; Hessel et al., 2018; Mnih et al., 2015). DVQN benchmarks against vanilla DQN, DDQN, and Rainbow.

FourRooms-v0 and CrossingS9N3-v0 are a grid-world environment where the objective is to reach the terminal-state cell (In the lower right of the image in both environments). In FourRooms-v0, the agent has to enter several doors and only complete a part of the goal for each door it enters. FourRooms-v0 is ideal for option-based reinforcement learning because each door is considered a sub-goal. While the environment is solvable by many deep reinforcement learning algorithms, option-based RL is more efficient. The agent

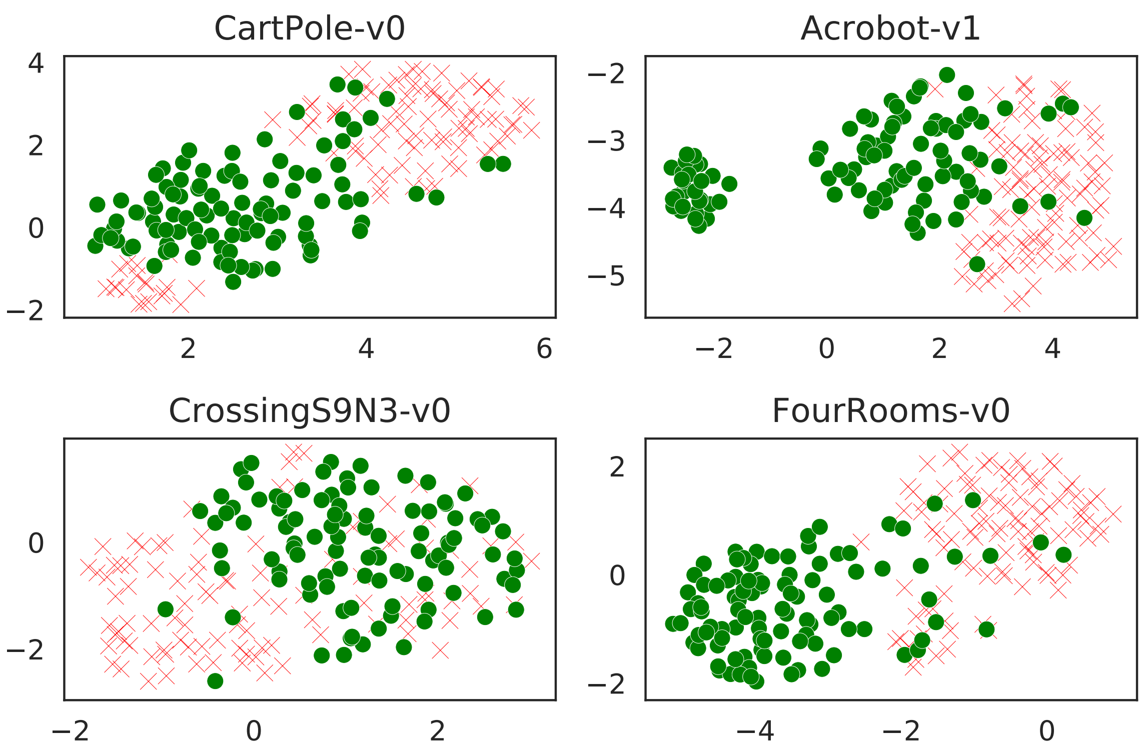


Figure 6.17: The learned latent space for all of the tested environments. DVQN successfully trivializes selecting options, as seen in the well-separated state clusters. The circular points illustrate states with positive rewards, and the cross illustrates negative rewards.

receives small negative rewards for moving and positive rewards for entering the goal-state (global) or the doors (local). The Crossing is a simpler environment where the agent has to learn the shortest path to the goal state. The agent can move one cell per time step in any direction in both grid environments.

To further show that the algorithm works in simple control tasks, we perform experiments in CartPole-v0 and Acrobot-v1. The objective in CartPole-v0 is to balance a pole on a cart. Each step generates a positive reward signal while receiving negative rewards if the pole falls below an angle threshold. The agent can control the direction of the cart at every time step. The Acrobot-v1 has a similar aim to control the arm to hit the ceiling in a minimal number of time steps. The agent receives negative rewards until it reaches the ceiling. The CartPole-v0 and Acrobot-v1 environments origins from Brockman et al., 2016, while CrossingS9N3-v0 and the FourRooms-v0 origins from Chevalier-Boisvert et al., 2018.[1]

6.3.3 Interpretability Evaluations

An attractive property of our model is that the latent space is human interpretable. As seen in Figure 6.17, the DVQN algorithm can produce clustered latent spaces for all tested en-

[1]A community-based scoreboard can be found at https://github.com/openai/gym/wiki/Leaderboard.

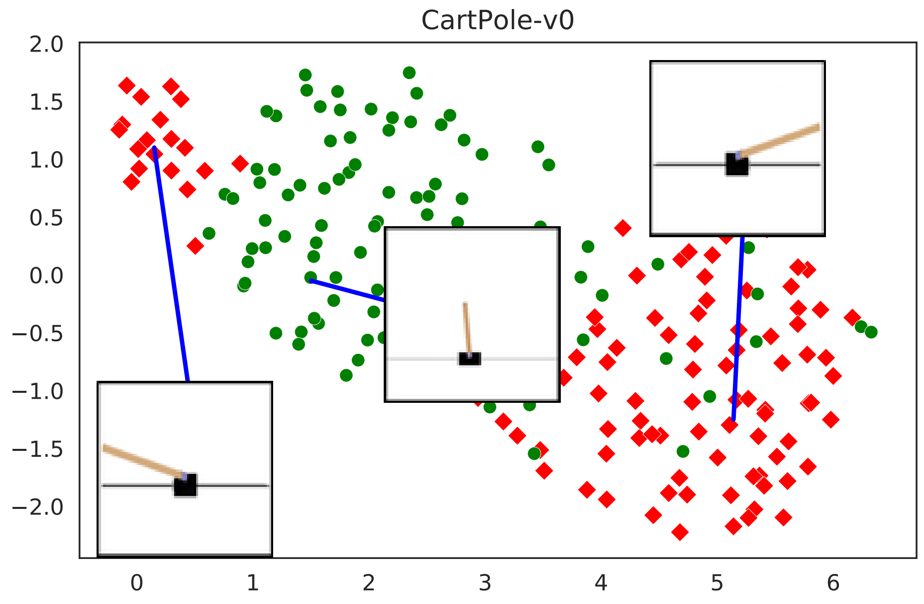


Figure 6.18: The relationship between states and the latent space for the CartPole-v0 environment. DVQN can separate each angle, left, middle, and right into separable clusters, especially useful in option-based reinforcement learning. Additionally, the visualization of the latent space that the Q-head uses to sample actions is trivial to interpret.

vironments. For example, in the CartPole-v0 environment, there are three clusters where two of them represent possible terminal states, and one represents states that give a reward. To fully utilize the capabilities of DVQN, the latent space can be used to generate options for identified clusters to promote different behavior for these regions in the observation space.

The DVQN algorithm can also provide interpretable state safety. As seen in Figure 6.17, it is possible to visualize which states are safe (positive feedback) and which are unsafe (negative feedback). A prerequisite for using such a method in industry is a simulation environment that can visit a fraction of unsafe states. The dynamics model can then interpolate between states and predict state safety in unvisited states from this fraction.

Figure 6.18 illustrates the visualization of the latent space representation in CartPole-v0. We find that each cluster represents a specific position and angle of the pole. The latent space interpolates between these state variations, which explains its shape. Although the clusters are imperfect, it is trivial to construct separable clusters with high precision. These clusters enable the automatic construction of initiation and termination boundaries for options-based policies.

6.3.4 Performance evaluation

Figure 6.19 illustrates a comparison of reward performance between state-of-the-art Deep Q-Networks variants and the proposed DVQN algorithm. The reward performance mea-

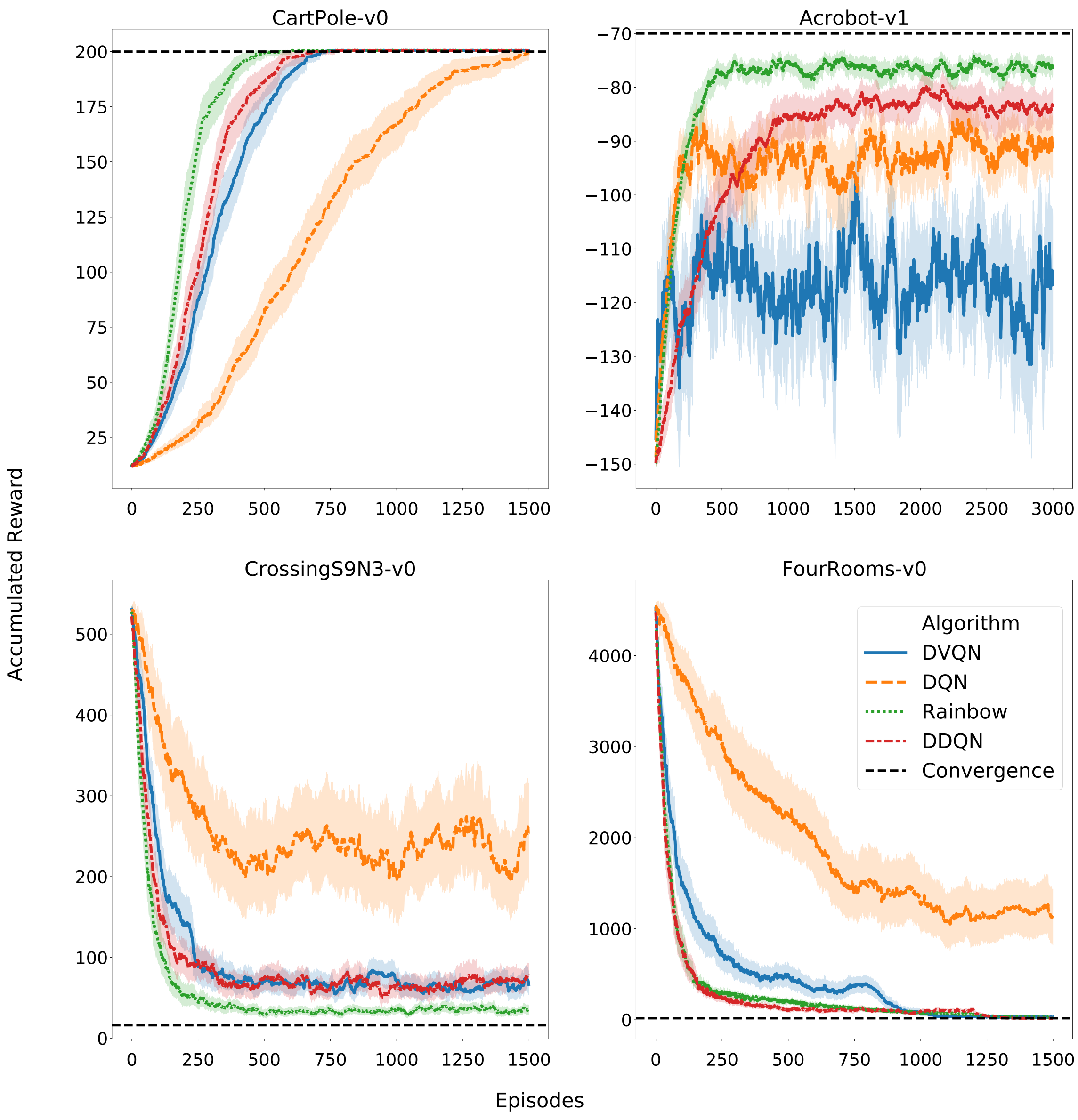


Figure 6.19: The accumulative sum of rewards of the DVQN compared to other Q-Learning-based methods in the experimental environments. Our algorithm performs better than DQN from Mnih et al., 2015, and performs comparably to DDQN from van Hasselt et al., 2015 and Rainbow from Hessel et al., 2018. We define an episode threshold for each environment (x-axis) and accumulate the agent rewards as the performance metric (y-axis) for CartPole-v0 and Acrobot-v1. The scoring metric in CrossingS9N3-v0 and the FourRooms-v0 is based on how many steps the agent used to reach the goal state.

surement is the average of 100 trials over 1500 episodes for CartPole-v0, CrossingS9N3-v0, FourRooms-v0, and 3000 episodes for Acrobot-v1. The goal of the Acrobot-v1 is to reach a score of -70, In CrossingS9N3-v0 and FourRooms-v0, the optimal score is 0, and for CartPole-v0, the optimal score is 200. The DVQN algorithm performs better than DQN and shows comparable reward performance to DDQN and Rainbow. DVQN cannot find a good policy in the Acrobot-v1 environment but successfully learns a good visual representation of the latent space. In general, the DVQN algorithm is significantly harder to train because it requires the algorithm to find a good policy within a Gaussian distribution. We found this to work well in most cases, but it required fine-tuning hyper-parameters. The algorithm is also slower to converge, but we improved training stability by increasing the batch size and decreasing the learning rate.

6.3.5 Summary

The empirical evaluation suggests that DVQN can learn generative policies from a latent space distribution. The learned latent space is particularly useful for clustering states that are close to each other for *discovering options automatically*. The DVQN algorithm can perform comparably to traditional Deep Q-Networks In the tested environments.

Hypothesis DVQN 1 *DVQN can learn interpretable observation/state-spaces.*

> **Evaluation:** DVQN can learn interpretable observation spaces according to Figure 6.17. Figure 6.18 is particularly captivating because it clearly shows that the latent space is structured to mimic the game objective closely.

Hypothesis DVQN 2 *DVQN allows clustering of observation spaces towards automatic option-based learning.*

> **Evaluation:** Figure 6.18 suggests that it is feasible to cluster the learned latent space such that it is possible to generate options in a hierarchical RL setting automatically. However, this remains future work.

Hypothesis DVQN 3 *DVQN provides interpretability while also maintaining a healthy agent reward performance.*

> **Evaluation:** Figure 6.19 demonstrates that DVQN has comparable reward performance to DDQN and Rainbow in a majority of the environments, except for the Acrobot-v1 environment. Future work hopes to address this by fully utilizing the learned options to achieve better reward performance.

6.4 ORACLE Evaluations

Observation Reward Action Cost Learning Ensemble (ORACLE) aims to learn a dynamics model capable of multi-step predictions for complex environments. ORACLE uses SSMs trained with AVI. The ORACLE algorithm is detailed further in Section 5.3. We evaluate whether ORACLE can perform well across many different environments and outperforms existing RL. The evaluation takes place in CartPole-v1, Deep RTS Deathmatch, and HalfCheetahPyBulletEnv-v0 from the open-source PyBullet physics engine.

6.4.1 Hypotheses

We carefully address the following hypotheses for the ORACLE algorithm.

Hypothesis ORACLE 1 *ORACLE performs better than state-of-the-art model-based and model-free algorithms.*

Hypothesis ORACLE 2 *ORACLE can navigate RTS and industry applications without significant drawbacks.*

Hypothesis ORACLE 3 *ORACLE can reconstruct and predict trajectories in animated environments.*

6.4.2 Hyperparameter and Sample Efficiency

These experiments focus on tuning the hyperparameters of the ORACLE algorithm to improve sample efficiency while maintaining acceptable reward performance. The study investigates if SWA, VQ, Latent Leaps, and AGC can have an advantageous effect on sample efficiency and if certain algorithm combinations show improved performance. We report notable findings and discuss choosing the correct hyperparameters for different environment types. We run the experiments five times for all environments and average the results. Experiments run for 1 000 000 steps or until the algorithm reaches a convergence score. If the algorithm fails to converge in time, we report the score as N/A. Finally, we use an ensemble of model-free algorithms for decision making, which we detail further in Section 6.4.5.

Table 6.3 illustrates the sample efficiency measured as convergence rate for different hyperparameter settings with separate dynamics models per environment. The results clearly show that using LL=30, VQ=off, AGC=on, and SWA=on is the best choice for CartPole-v1. The DeepRTS Deathmatch and HalfCheetahPyBulletEnv-v0 demonstrate good reward performance when enabling VQ. We conclude that VQ performs worse in CartPole because it is a far simpler environment, meaning the algorithm never learns the problem slower because of the added model complexity. This makes sense, as the VQ architectures

Table 6.3: The experiment setup for the limited hyperparameter search. The table clearly shows that a latent-leap of 30 is superior in reaching a convergence score for all tested environments.

CartPole-v1				
SWA	**VQ**	**LL**	**AGC**	**Convergence Step**
on	on	10	on	N/A
on	on	30	on	755 000
on	on	60	on	N/A
on	**off**	**30**	**on**	**390 000**
off	on	30	off	825 000

DeepRTS Deathmatch				
SWA	**VQ**	**LL**	**AGC**	**Convergence Step**
on	on	10	on	N/A
on	**on**	**30**	**on**	**600 000**
on	on	60	on	N/A
on	off	30	on	N/A
off	on	30	off	N/A

HalfCheetahPyBulletEnv-v0				
SWA	**VQ**	**LL**	**AGC**	**Convergence Step**
on	on	10	on	N/A
on	**on**	**30**	**on**	**725 000**
on	on	60	on	N/A
on	off	30	on	N/A
off	on	30	off	N/A

double the number of trainable parameters in the model. The algorithm cannot generalize well environments with a few steps before the termination state. The primary function of the VQ layer is to allow for encoding multiple environments in the same dynamics models. For this reason, we proceed with experiments training the algorithm using the same dynamics model for all environments.

Table 6.4 illustrates the sample efficiency of ORACLE when using the same model for all environments. This experiment aims to see if feeding the latent vector into a VQ to structure the latent space categorically. The results clearly show that ORACLE can generalize across several environments using the same parameters, using a VQ layer after the generative network.

Table 6.4: The data depicts the average performance using a single dynamics model to learn all environments. The table clearly shows that enabling VQ has a positive effect on sample efficiency, and without, the environment can not converge before the step limit has passed.

SWA	VQ	LL	AGC	Average Convergence Step
on	on	10	on	895 000
off	**on**	**30**	**on**	**565 000**
on	on	60	on	N/A
on	off	10	on	N/A
on	off	30	on	N/A
on	off	60	on	N/A
on	on	10	off	N/A
on	on	30	off	695 000
on	on	60	off	N/A
off	on	10	on	N/A
off	on	30	on	596 000
off	on	60	on	N/A
off	on	10	off	N/A
off	on	30	off	650 000
off	on	60	off	N/A

We recommend the following strategy for tuning the ORACLE algorithm based on the empirical evidence. For simple environments with less than 1 000 timesteps before forced termination, we recommend disabling VQ and using LL=30. If the environment exceeds 1 000 timesteps, enable VQ. When training the algorithm on all environments, we recommend having all hyperparameters enabled using LL=30.

6.4.3 Reconstruction - Sonic The Hedgehog 2

ORACLE can, similar to DVAE, predict artificial trajectories from a given start state. DVAE had severe problems with posterior collapse, training instabilities, and poor capability to reconstruct longer time horizons. We improve these problems in ORACLE. Figure 6.20 shows the resulting reconstruction capabilities after training where the controller is a hard-coded strategy. The first row is the live game at time t, the second row is the state at time $t - 30$, and the third row is the predicted state at time t, given the state in the second row. We can clearly see that the ground truth state (first row) and the predicted state (third row) are similar. There are some differences in the location of the Tails

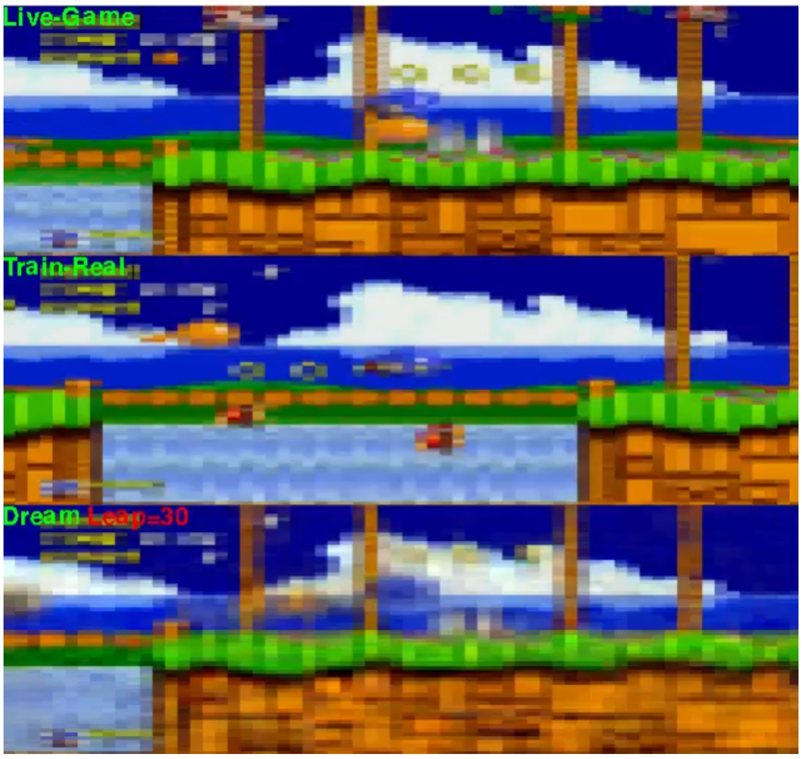
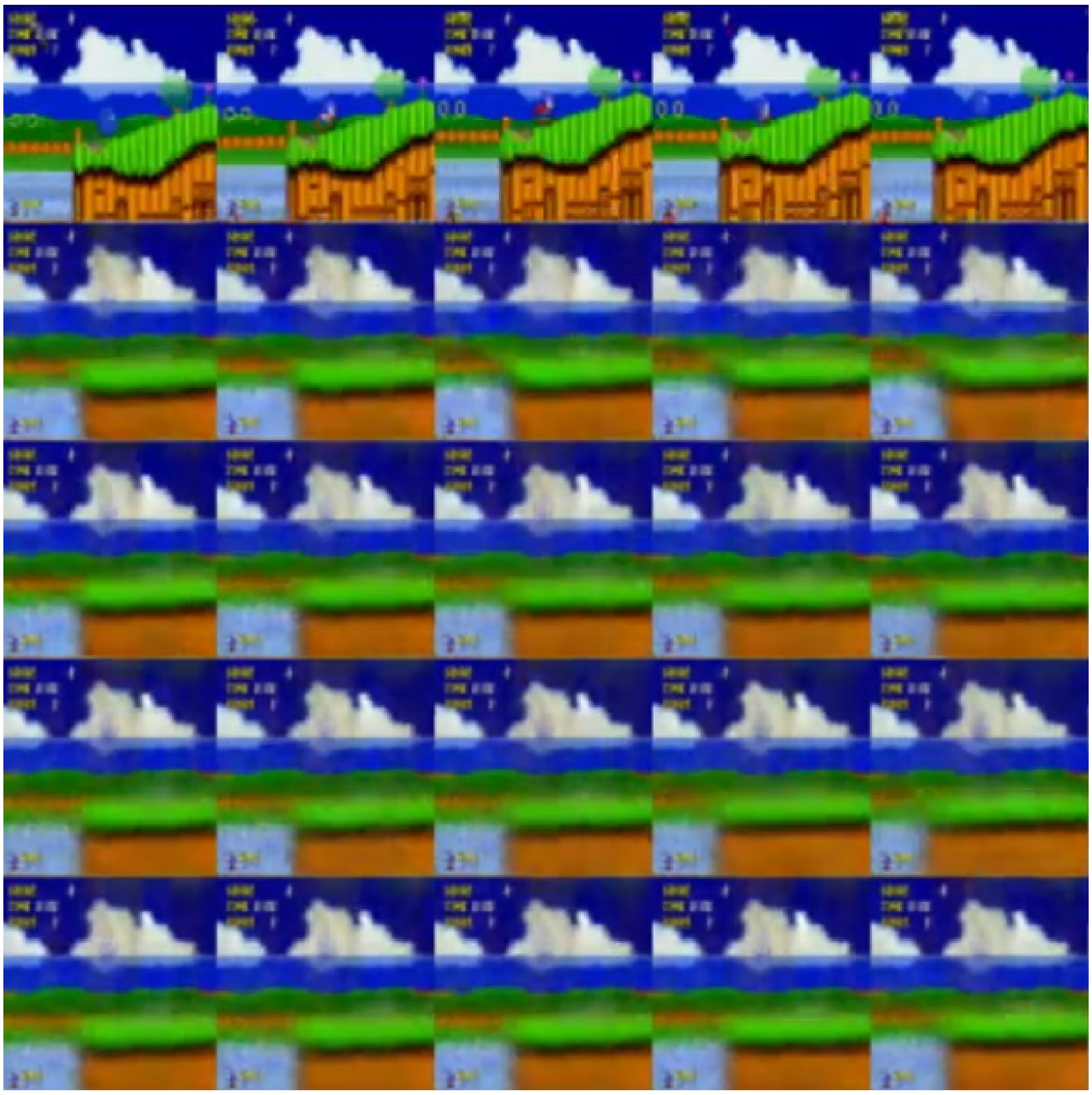


Figure 6.20: Observation of the Sonic the Hedgehog 2 state and the corresponding 30-step prediction (LL=30). The first row is the ground truth state. The second row is a past ground truth state s_{t-30}, and the bottom row is the predicted state at s_t.

character (orange ball) and the piranha in the water, where the prediction suggests there is a piranha above water. However, compared to Figure 6.6, ORACLE shows significantly better reconstruction capabilities than DVAE.

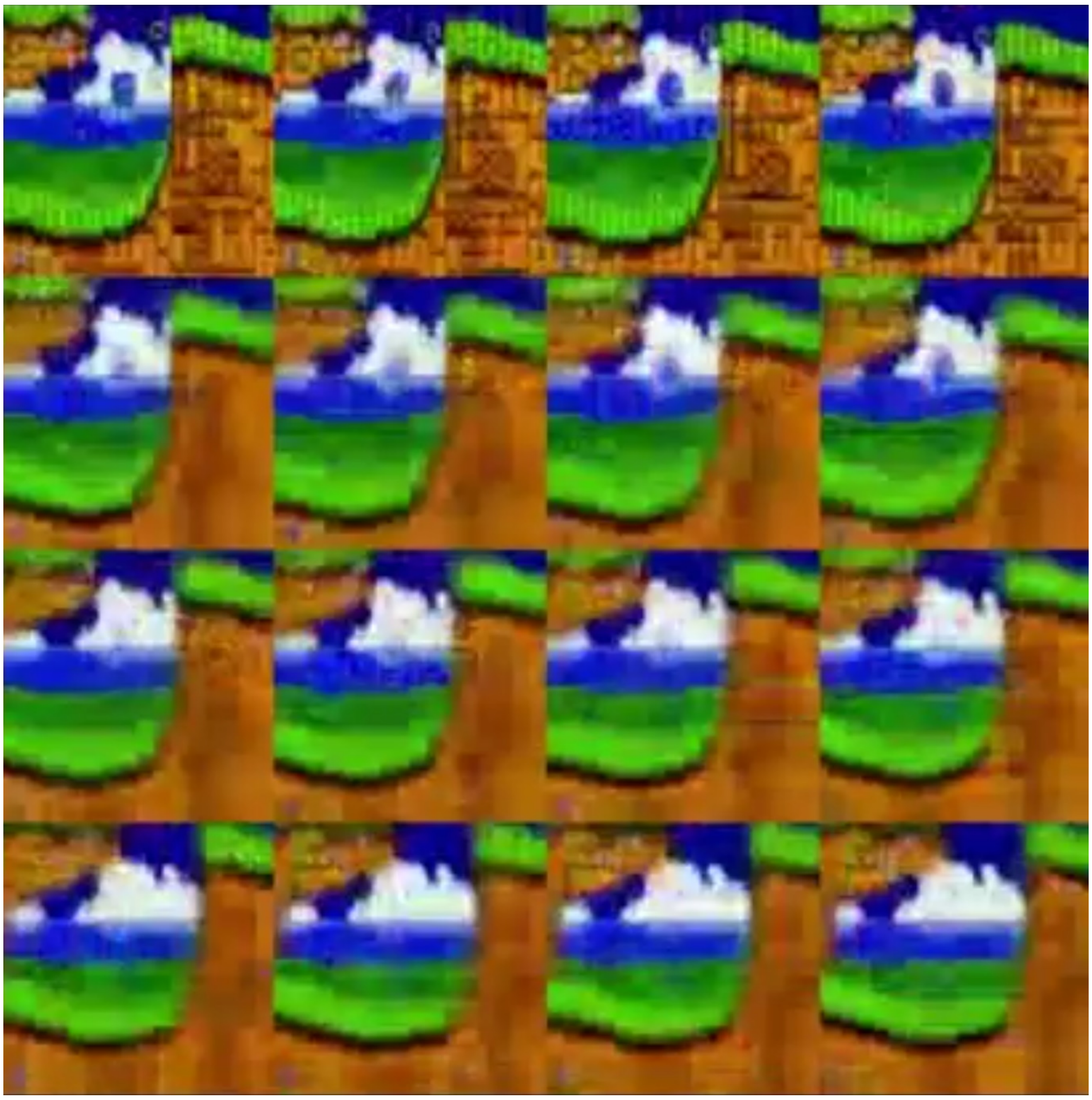
We extend the study to include multiple predictions per sample (i.e., generate variations of the same data), as shown in Figure 6.21. The top row is the ground-truth state, where the remaining rows are predictions. The trajectory starts at the first available ground truth state (upper left frame) and uses the previous prediction to predict the next. Figure 6.21a shows that there are slight variations between each trajectory. There are slight variations, typically between rows 2 and 5, where the Sonic has walked further in the row 5 trajectory (e.g., Sonic has progressed more to the right). Figure 6.21b similarly shows that ORACLE can generate plausible stochastic variations. Row 2 shows sonic in a similar position as the ground truth trajectory. The other rows illustrate trajectories where Sonic is moving downwards, more easily seen when observing the grass in the uppermost corner of each prediction.

The challenge with ORACLE is that it takes substantially longer to train. The training time for training an accurate model takes approximately 48 hours using an Nvidia 2080TI in Sonic the Hedgehog 2 for a single level. DVAE trains significantly faster but lacks the long-horizon prediction capabilities in ORACLE.



(a) Sample #2

Figure 6.21: Demonstration of Sonic the Hedgehog 2 (top row) and corresponding predictions for four trajectories. The predicted trajectories start before the top left most ground truth state.



(b) Sample #3

Figure 6.21: Demonstration of Sonic the Hedgehog 2 (top row) and corresponding predictions for four trajectories. The predicted trajectories start before the top left most ground truth state.

6.4.4 Performance - Deep Warehouse

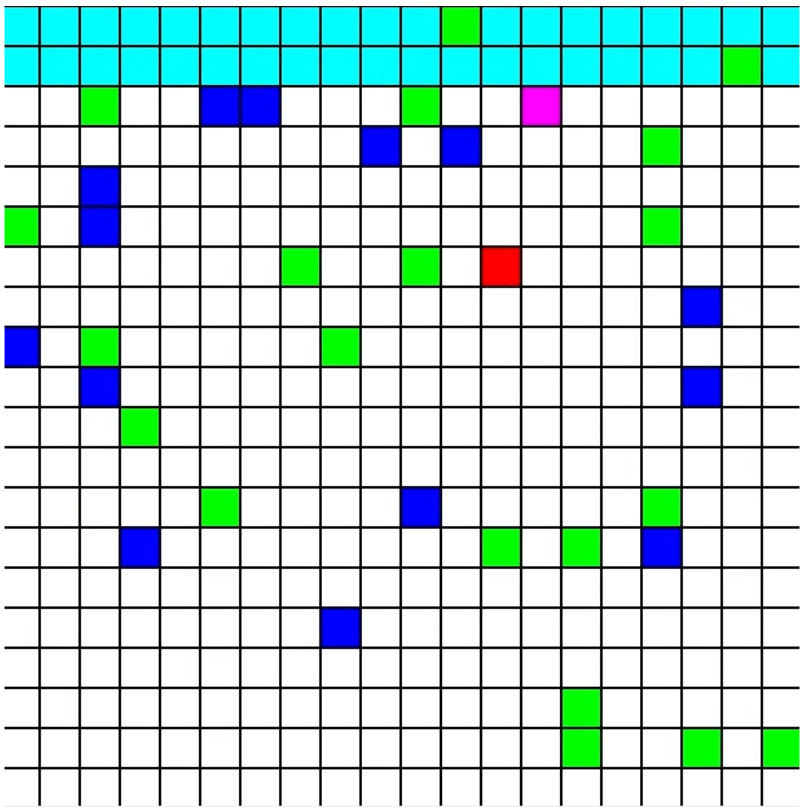


Figure 6.22: A Deep Warehouse 20x20 environment with 30 agents. The ORACLE algorithm controls taxis using the same policy. The observation input marks the selected taxi to identify better the agent that action is sent.

We measure the performance of ORACLE in the Deep Warehouse environment using a single DQN for decision-making. The goal of the agents (green squares) is to pick up assigned items (blue squares are active items) and deliver them to the designated delivery point (pink is active, red is the inactive delivery points) at which the episode terminates. The agents can collide or crash in the outer walls, which resets the agent to the spawn area (cyan). The agent receives negative rewards for every step, a medium reward for picking up items, and a large reward for delivering the item to the correct pickup point.

Figure 6.23 shows four experiment graphs from training ORACLE in the Deep Warehouse 20x20 (Figure 6.22) environment using DQN as the decision-making policy. The deliveries max indicates how many agents can deliver their items before reaching a terminal state. The reward signal gradually increases, suggesting that the algorithms learn to fit the reward function. We also see that the episode length decreases concurrently with increasing rewards, meaning that the agents find the goal state (or collides) faster. The deliveries max charts indicate how many agents were abler to deliver packages before reaching a terminal state. We can achieve between 15 and 25 deliveries, whereas 30 is when the policy finds optimal behavior. These findings fuel the need for further study in

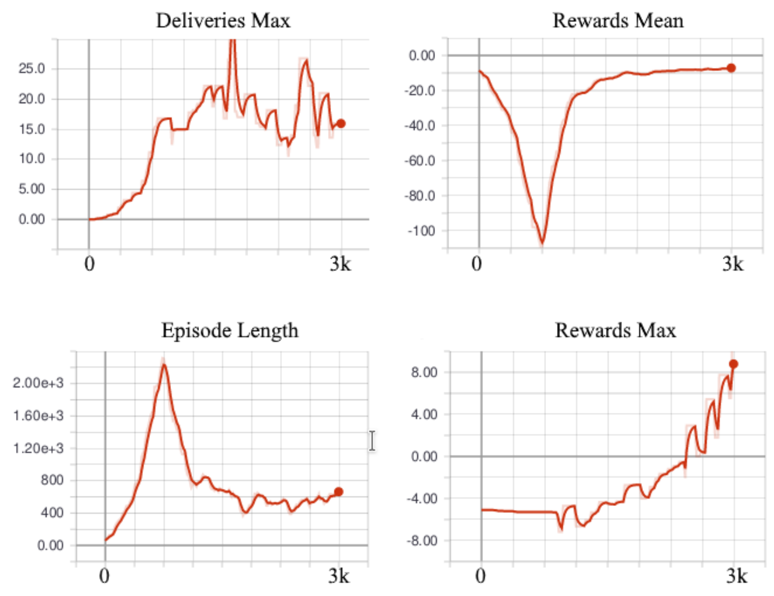


Figure 6.23: Performance chart for ORACLE in Deep Warehouse using DQN as the decision-making policy. The x-axis is the episode number.

agent safety, which we aim to address in Section 6.5.

6.4.5 Comparative Reward Performance

The comparative performance evaluation aims to understand how the ORACLE performs compared to state-of-the-art model-based and model-free methods and how to tune the algorithm for different environments. The experiments compare ORACLE to the model-free algorithms RAINBOW (2018), and PPO (Schulman et al., 2017), and Dreamer (Hafner et al., 2020) for model-based methods. The learning ORACLE policy ensemble consists of PPO, DQN, RAINBOW, A3C, and VPG and performs majority voting for evaluated actions. In the case of a draw, we randomly select one of the actions. The experiments run five times where the evaluation metric is the ratio between score and steps. [2] For the comparison, we use reference hyperparameters found in Hessel et al., 2018 for RAINBOW and Schulman et al., 2017, respectively.

Figure 6.24 shows that ORACLE outperforms the tested model-free and model-based approaches for the selected environments. A dependent factor on ORACLE's reward performance is the latent-leap parameter which represents how many steps the algorithm *leaps into the future* before resampling the real environment. Specifically, we see that a latent leap of 30 is a good compromise between sample efficiency and reward performance.

[2]We make the reader aware that the experiments are compute-heavy, hence few experiment iterations. In total, the experiments take $\sim$ 5 days of wall-clock time to train on consumer-level hardware.

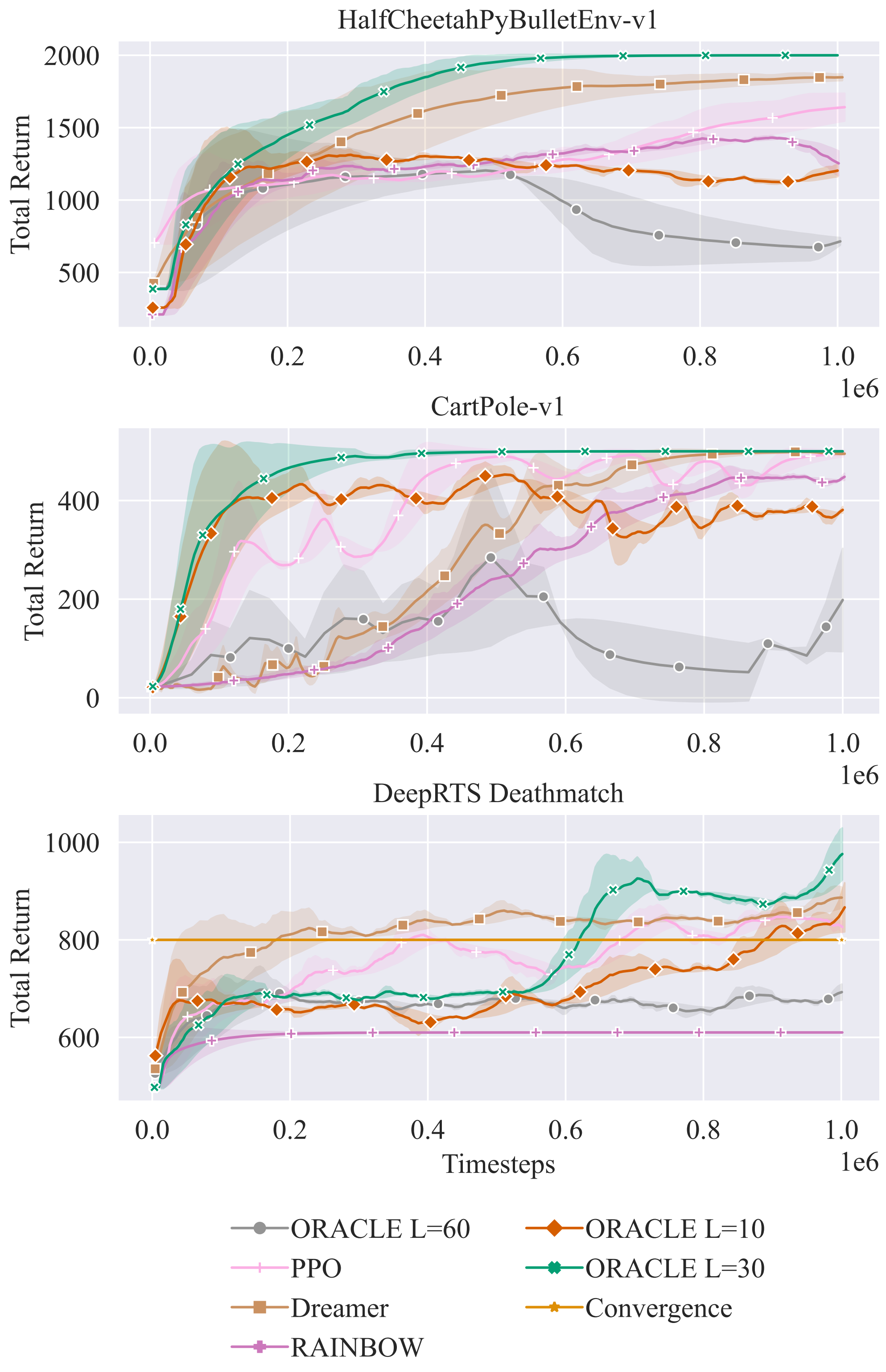


Figure 6.24: The accumulated reward performance and sample efficiency of PPO, RAINBOW, Dreamer, and ORACLE. We observe that in all environments, ORACLE outperforms the state-of-the-art algorithms with L set to 30. The x-axis describes the environment step, while the y-axis describes the average return.

6.4.6 Design Conclusion

We next discuss the findings in the hyperparameter tuning experiments and the comparative performance evaluation to understand better why ORACLE outperforms prior approaches.

Specifically, we think that ORACLE has strength in capturing large state-spaces and has a good ability to generalize well over sparse datasets. The dynamics model is primarily continuous-space with a combination of deterministic and stochastic variables. A VQ layer between the continuous probabilistic latent vector transforms the latent vector into discrete space before reconstructing the output. In parallel to our work, Ozair et al., 2021 have the same conclusion and show outstanding results on solving chess, outperforming previous methods. We conclude that:

- using VQ in combination with VAE and SRSSM provides a powerful enhancement to model robustness, but it falls short when used for more simple problems,
- generally, we see that ORACLE is best suited for larger problems with more than 1 000 timesteps, and
- it remains an open question to justify the combination of VAE, SRSSM, and VQ analytically.

6.4.7 Summary

We present a novel model-based reinforcement learning algorithm for improving sample efficiency. We test the algorithm in popular game environments and present state-of-the-art results in Deep RTS Deathmatch and HalfCheetahPyBulletEnv-v1 compared to Dreamer, PPO, and RAINBOW.

Hypothesis ORACLE 1 *ORACLE performs better than state-of-the-art model-based and model-free algorithms.*
Evaluation: After a comprehensive hyperparameter search and empirical experiments (Figure 6.24), ORACLE performs better in the tested environments. However, we believe that the tested algorithms could potentially see similar improvements with extensive hyperparameter tuning for the tested algorithms. Nonetheless, ORACLE demonstrates excellent reward performance and significantly better reconstruction performance than DVAE from Paper D.

Hypothesis ORACLE 2 *ORACLE can navigate RTS and industry applications without significant drawbacks.*
Evaluation: This study demonstrates that ORACLE is a good algorithm for

RTS games, where it reaches a close maximum score after 1 000 000 timesteps in the Deep RTS Deathmatch environment. Furthermore it masters CartPole-v1 and HalfCheetahPybulletEnv-v1. This means that ORACLE may become a good algorithm for safe RL in industry applications while maintaining good reward performance. The most influential lesson from this study is that complex models require extensive tuning to work well and for ORACLE to succeed, it takes significant effort to reach good reward performance.

Hypothesis ORACLE 3 *ORACLE can reconstruct and predict trajectories in animated environments.*

Evaluation: According to empirical evidence shown in Figure 6.21a-6.20, ORACLE can accurately reconstruct and predict trajectories in animated environments like Sonic the Hedgehog 2. ORACLE improves significantly from prior work using the DVAE algorithm to predict 30-steps compared to 3-step using DVAE. The challenge with Observation Reward Action Cost Learning Ensemble (ORACLE) is that it takes substantially longer to train the algorithm making it costly to train the model for many problems.

6.5 S-ORACLE Evaluations

We present a thorough evaluation of the ORACLE with safety extensions, S-ORACLE. S-ORACLE extends ORACLE to concern decision-making safety at training and test-time. The algorithm uses goal-directed RL, risk-sensitive RL, and risk-directed exploration, which we detail in Section 2.3. The algorithm extensions are detailed 5.3.6. The evaluations are two-fold, where the first set of experiments focuses on safety and the second set of experiments focuses on agent reward performance. We evaluate S-ORACLE in four Deep RTS, ELF, Micro RTS, StarCraft II environments, and the industry-near environment Deep Warehouse. The safety evaluations compare S-ORACLE to PPO and DVAE. We use five model-free algorithms for the agent reward performance evaluations, A2C, DQN, RAINBOW, IMPALA, PPO, using standard hyperparameters from the respective literature. Additionally, we test with the DVAE algorithm from Paper G.

6.5.1 Hypotheses

We carefully address the following hypotheses for the S-ORACLE algorithm.

Hypothesis S-ORACLE 1 *S-ORACLE is safer than risk-neutral algorithms and DVAE.*

Hypothesis S-ORACLE 2 *S-ORACLE are more climate-friendly compared to model-free methods.*

Hypothesis S-ORACLE 3 *S-ORACLE outperforms model-free algorithms in terms of agent reward performance.*

6.5.2 Experimental Setup

Experiments allocate one GPU per algorithm from a pool of 2x 2080 TI and 2x 1080 TI cards, where 1080 TI cards have 37% longer train time to compensate for slower training speeds. For climate footprint estimations, we limit the GPU power draw to 250w and divide the total energy consumption of the 1080 TI-based experiments to compensate for the additional time. Model-free algorithms train for 24 hours and model-based trains for 16 hours to demonstrate efficiency. Reward performance is tested for several episodes, where the number of episodes is specified per environment due to computational limitations.

The experiment's goal is to demonstrate S-ORACLE reward performance and safety performance. Concurrently, a baseline is constructed for future work. Experiments show that S-ORACLE generalizes well for multiple problems, although the algorithm still has improvement potential in reward performance, sample efficiency, and safety. The results also systematically report S-ORACLE's energy and carbon footprint compared to other tested algorithms, as suggested by Henderson et al., 2020a.

6.5.3 Safety Evaluation

S-ORACLE aims to learn an agent agnostic feedback signal that directs the risk-neutral agents towards safer trajectories during training. S-ORACLE learns in a two-objective process where the first objective is to learn the dynamics model and the second is to learn an actor ensemble (see Section 4). S-ORACLE formalize two training schemes for training the dynamics model:

1. **Scheme #1**. Train dynamics model and actor ensemble concurrently, balancing the exploration-exploitation trade-off for risk management.
2. **Scheme #2**. Train dynamics model using external knowledge, using existing expert systems known to operate safely in the environment and subsequently, train actor ensemble offline using dynamics model before carefully evaluate safety in live systems. This approach is seen as successful in prior work Paper F and Paper G.

The first method is the most versatile because it does not require any knowledge *a priori* to solve the problem. However, the first method is at increased risk of entering catastrophic states. The second method improves empirical safety but relies on external systems to succeed. This Section aims to empirically demonstrate the safety performance of S-ORACLE compared to PPO in Scheme #1 and DVAE in Scheme #2. The experiments

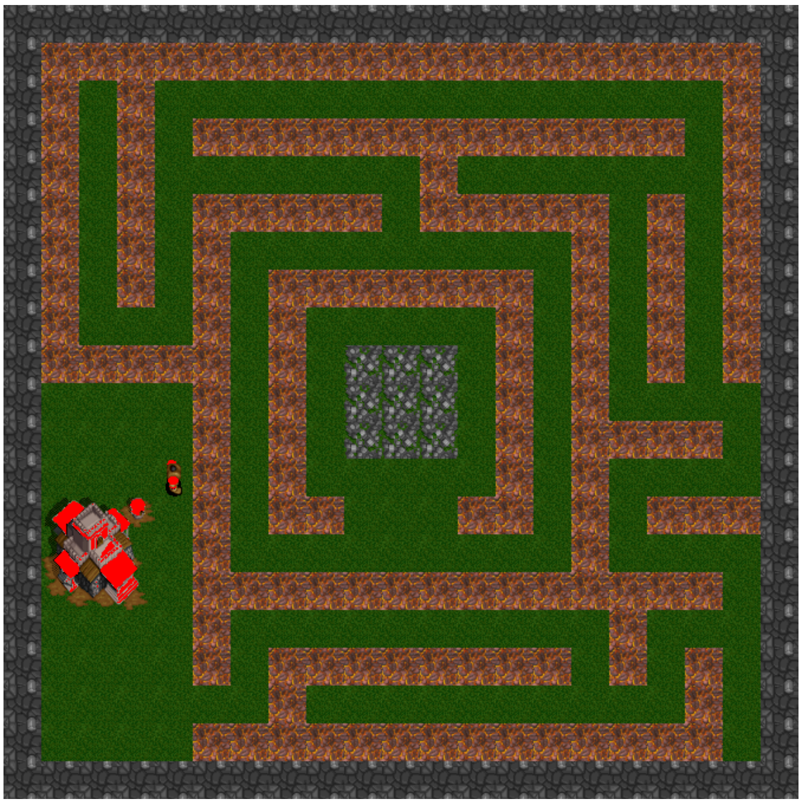


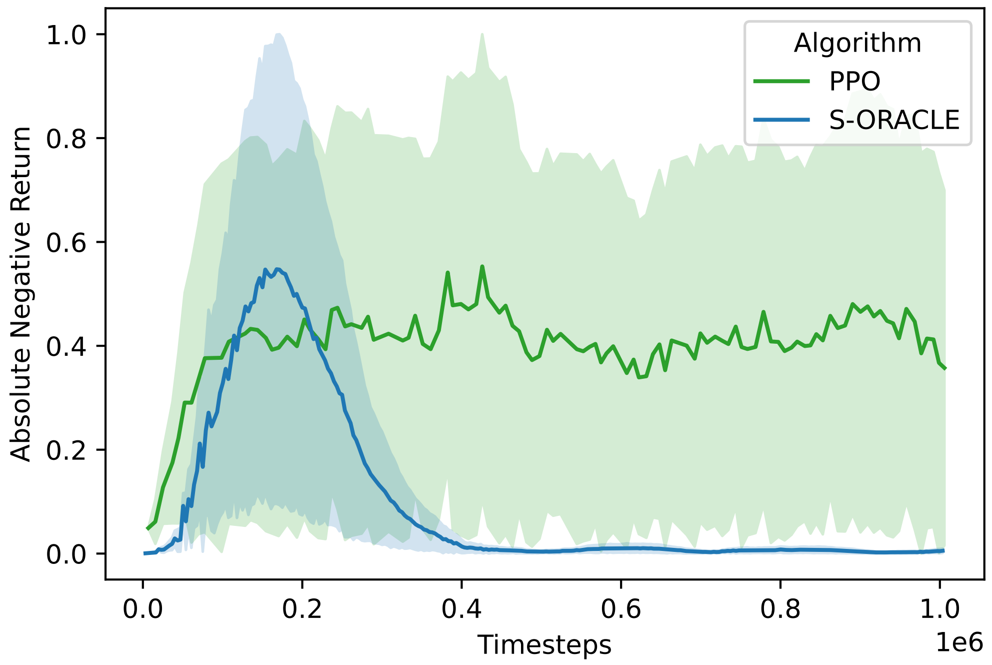
Figure 6.25: A graphical observation of the Deep RTS Lava Environment. The goal is for the player (unit) to reach the middle section of the map without walking in lava.

measure the ratio of incoming absolute negative returns to determine safety, normalized between $0.0 \leq x \leq 1.0$. For every 100 000 timesteps, the algorithm evaluates for 100 episodes, and the measured min-max variance is illustrated using shades in the plots. S-ORACLE is evaluated in two environments using the described training schemes:

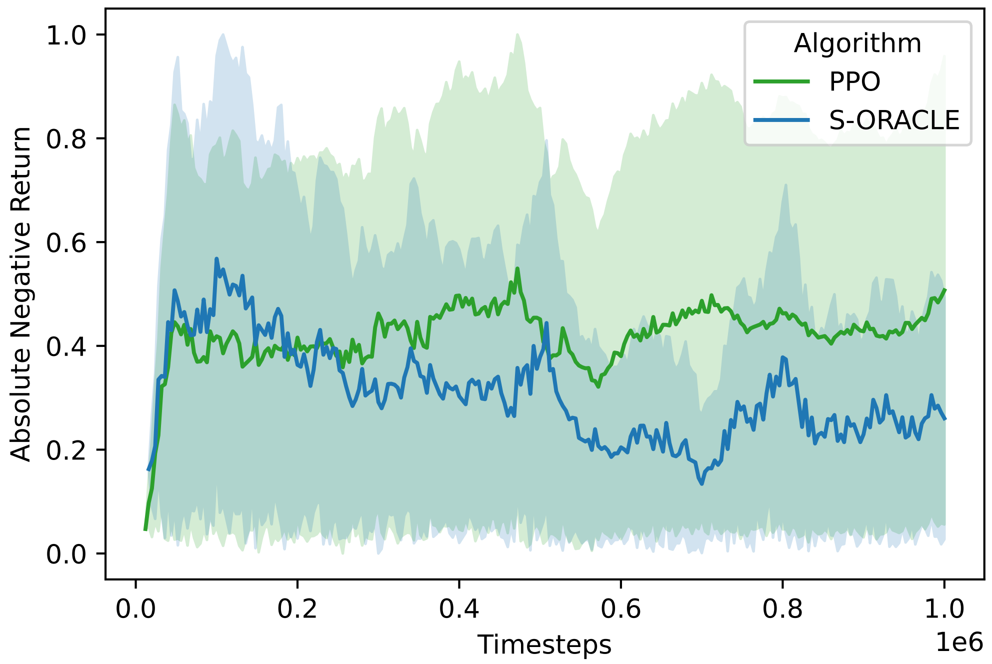
1. **The Deep RTS lava environment** is seen in Figure 6.25. The agent performs safe behavior when avoiding lava states. The goal is to reach the gold in the middle of the map.

2. **Deep Warehouse** logistics challenge. Transport items from source to destination as fast as possible while avoiding collision with other agents. The algorithm controls a single agent, while the remaining agents use a manhattan distance-based heuristic. Depending on map size, the agent is part of a larger logistics system with other taxi agents. See Section 4.4 for additional details about the Deep Warehouse environment.

In Scheme #1, the algorithms balance the exploration-exploitation trade-off during dynamics model training. Therefore, they are more susceptible to error states because of missing knowledge about the dynamics. Figure 6.26a-6.27b shows the safety violations

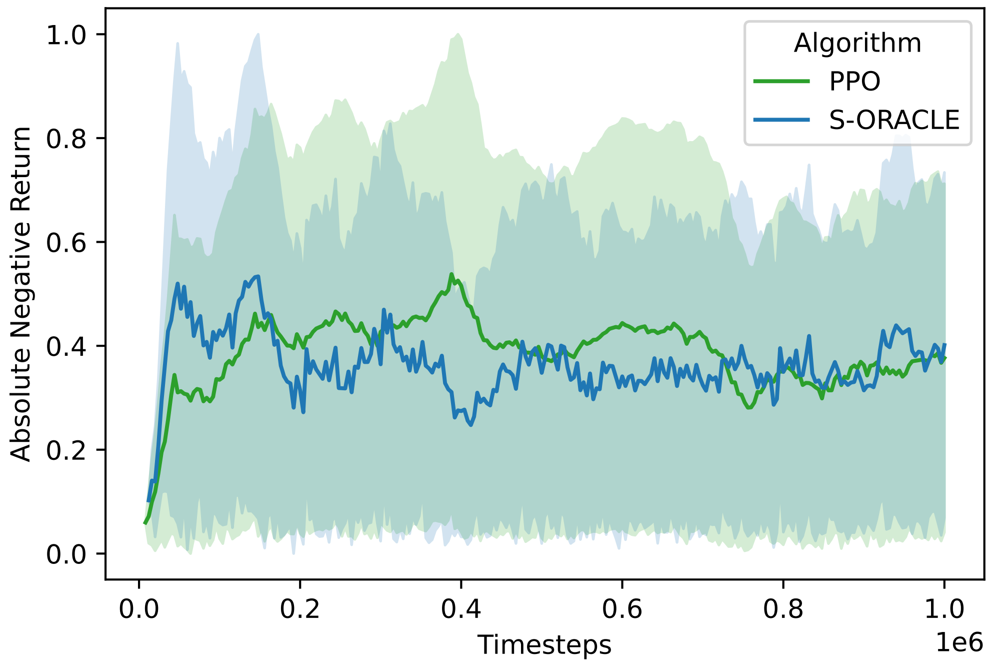


(a) Deep RTS Lava Maze.

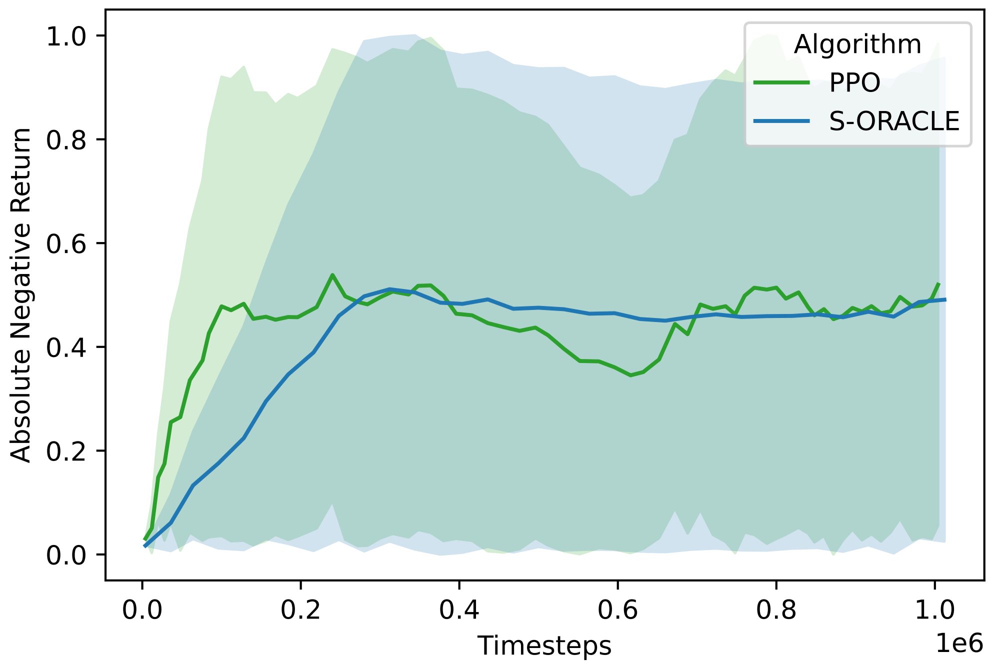


(b) Deep Warehouse 11×11.

Figure 6.26: Safety violations rate following training Scheme #1. The y-axis is the mean absolute negative return where the absolute negative return is averaged per 10 000 timesteps.



(a) Deep Warehouse 22×22.



(b) Deep Warehouse 41×41.

Figure 6.27: Safety violations rate following training Scheme #1. The y-axis is the mean absolute negative return where the absolute negative return is averaged per 10 000 timesteps.

ratio in Deep RTS and Deep Warehouse, respectively. In the Deep RTS Lava environment, it is clear that S-ORACLE learns to avoid lava, and for Deep Warehouse 11×11 (Figure 6.26b), the S-ORACLE gradually reduces the rate of negative returns compared to PPO. Similarly, for Deep Warehouse 22×22 and 41×41 (Figure 6.27a and Figure 6.27b), S-ORACLE accumulate fewer negative rewards, but the effect seems to diminish for more complex state-spaces. Collectively, the figures demonstrate that a risk-neutral algorithm (PPO) explores negative states at a relatively uniform rate. At the same time, the risk-averse agent (S-ORACLE) impacts the number of visited error-states significantly less.

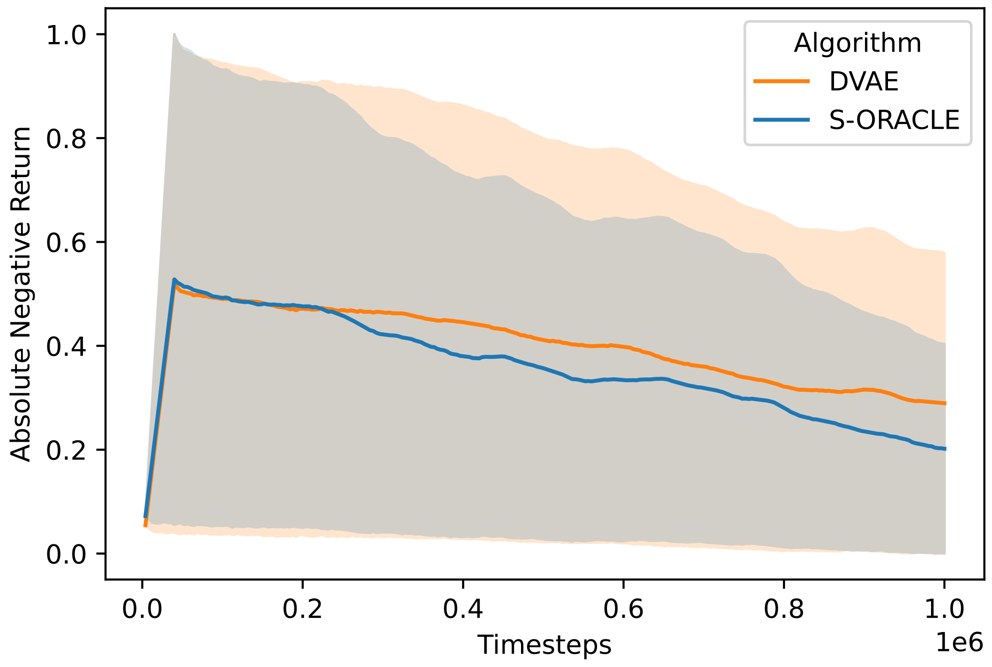
In Scheme #2, the dynamics model is trained from observations of an expert system before training using the dynamics model occurs for the agent algorithm. The second approach demonstrated better safety awareness, naturally, because the algorithm learned much of the environment through the dynamics model before making actions live. Figure 6.28-6.28 shows the safety-awareness of S-ORACLE compared to DVAE, clearly showing a downwards trend in the mean absolute negative return. However, as the environment becomes more complex (Figure 6.29b), the effect seems to diminish in contrast to Figure 6.28b.

The empirical evidence clearly shows that S-ORACLE improves safety in Scheme #1 and Scheme #2 compared to model-free RL algorithms and the prior work of DVAE. Specifically, as the algorithm learns the environment dynamics, the agent receives fewer negative rewards than other tested environments.

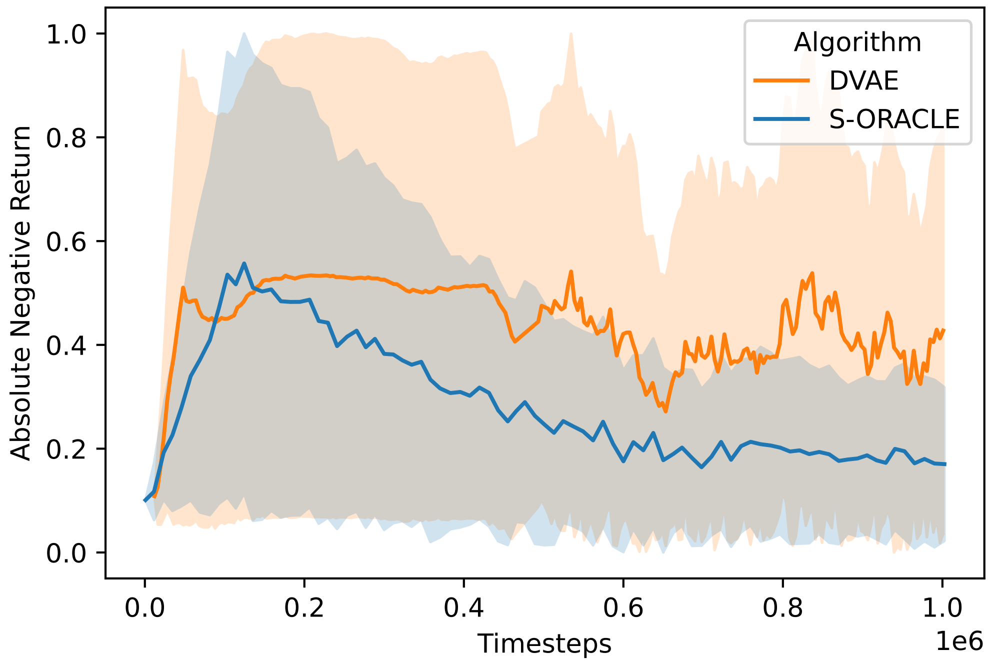
6.5.4 Performance Evaluations

Deep Line Wars. Table 6.5 shows the reward performance of the tested algorithms against the hard-coded expert agent. Each algorithm runs 1000 times and is averaged over. Specifically, we observe that all tested algorithms perform well, even for large maps. The S-ORACLE agent performs comparably to the tested model-free algorithms and outperforms all tested algorithms on average. The action-space comprises 4 actions for cursor position, 4 build actions, and 4 spawn unit actions.

Deep RTS. We evaluate the algorithms for 100 episodes in the Deep RTS Lava Maze problem and the Deep RTS Deathmatch between the tested algorithms. For the maze-like environment, results are averaged. Table 6.6 shows the reward performance of tested algorithms in the maze environment. All algorithms perform well, where the best algorithm, IMPALA, consistently found the best path after training. The other algorithms perform in the $\sim 95\% \pm 5$ range. Table 6.7 shows that S-ORACLE outperforms 5 out of 7 algorithms and scores comparably to PPO in 1v1 matches. We observed that S-ORACLE plays conservatively and wins by strate-

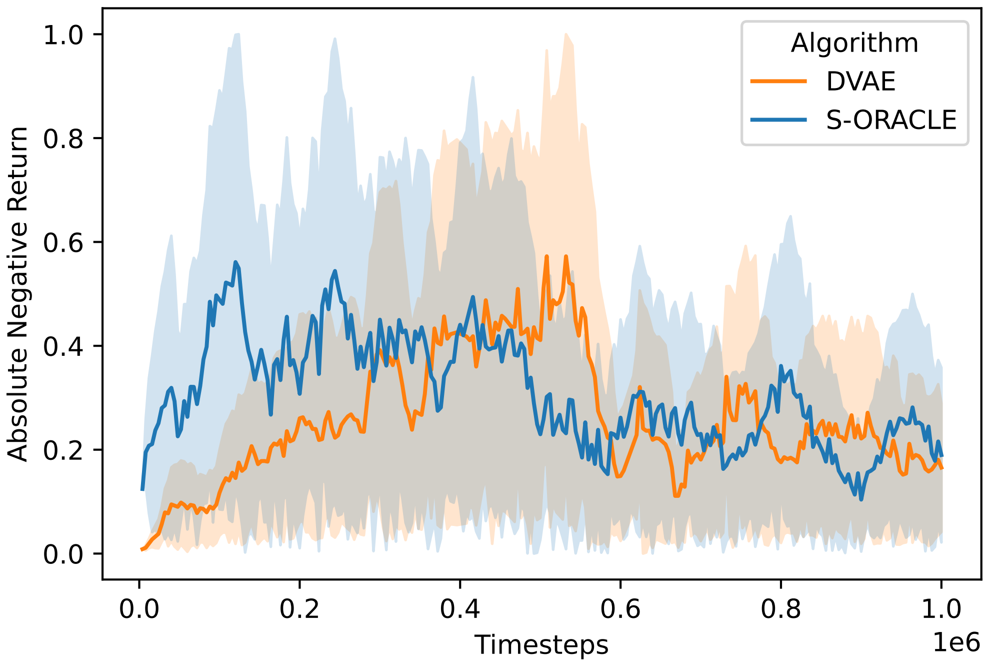


(a) Deep RTS Lava Maze.

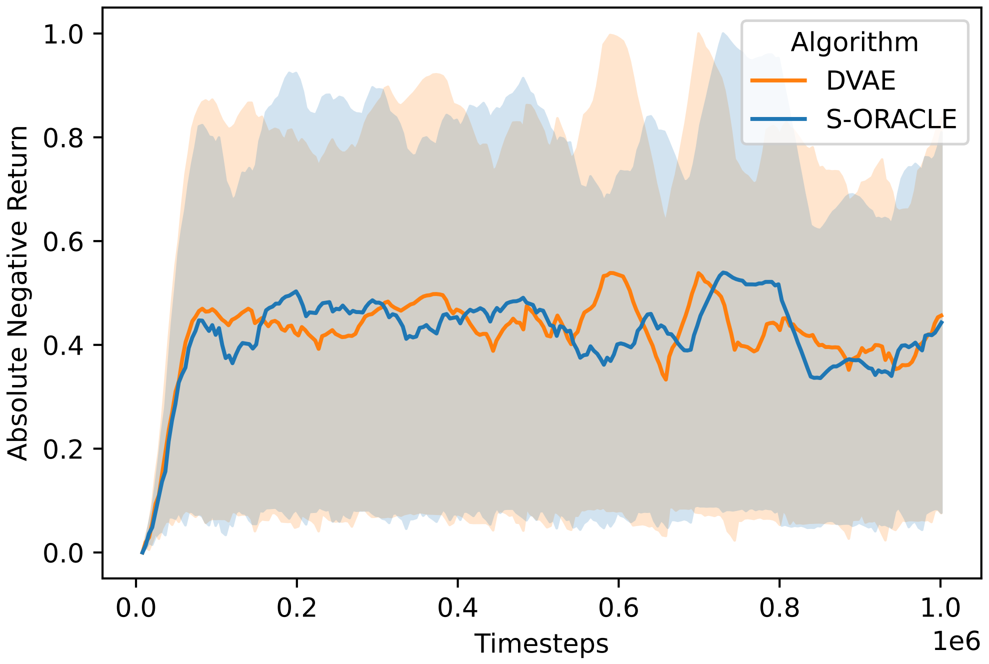


(b) Deep Warehouse 11×11.

Figure 6.28: Safety violations rate following training Scheme #2. The y-axis is the mean absolute negative return where the absolute negative return is averaged per 10 000 timesteps.



(a) Deep Warehouse 22×22.



(b) Deep Warehouse 41×41.

Figure 6.29: Safety violations rate following training Scheme #2. The y-axis is the mean absolute negative return where the absolute negative return is averaged per 10 000 timesteps.

gically blocking the opponent's path with houses during experiments. In the long run, the conservative behavior wins because the cost of units is more expensive than houses. Additionally, houses allow the players to build larger armies that trivially defeat opponents with limited housing. The agent had access to 13 discrete actions at every timestep and received positive rewards for victories and zero during the game.

ELF: Mini-RTS. The partial observability of Mini-RTS makes the learning task significantly harder to master compared to Deep RTS. Table 6.8 shows that S-ORACLE average 83% win ratio but had difficulties defeating the AI-Simple algorithm consistently. The Mini-RTS action-space comprises 9 strategic actions in which the agent can construct complex strategies. We observe that the average-length game is approximately 3500 ticks where a positive reward is given for victories, a negative reward for defeats, and zero rewards in other states. Algorithms are tested for 100 episodes.

Micro RTS. Perhaps the most widely used environment for competitions is the Micro RTS environment (Ontanon, 2013). We test each algorithm for 100 episodes against strategies presented in the IEEE Conference on Games (COG-2019) competition. Table 6.9 shows the results, where each row illustrates the win rate against the column algorithm. We see that S-ORACLE outperforms the other algorithms, closely followed by DVAE, PPO, and IMPALA. The action and state-spaces vary between maps but are thoroughly described in Huang et al., 2021.

StarCraft II. Perhaps the most compute-heavy experiment is the StarCraft II environments, albeit we only focus on mini-games of the original game. In particular, we use the mini-games from Vinyals et al., 2017 already standardized in literature, and we adopt these findings for our comparison. As seen in Table 6.10, no algorithm remains dominant in all environments, but the professional player from DeepMind (DM) (2017). However, the S-ORACLE algorithm scores on average better than the model-free approaches, likely because of the ensemble technique when voting for actions. The action-space is challenging to implement because it comprises several hundred actions for every timestep. For this reason, we choose to adopt the same method as in H. Hu and Wang, 2020.

6.5.5 Sustainability Report

Table 6.11 reports the contribution of $\frac{CO2}{kg}$ in the experiments following Henderson et al., 2020a. Algorithm reward performance is tested in 41 experiments and six safety experiments for PPO, DVAE, and S-ORACLE. Note that the model-based approaches have a

Table 6.5: Deep Line Wars results. Each agent (row) plays against the built-in *expert* agent at different map sizes (column). The last column is the average reward performance for all map sizes. The cell values represent a win ratio ranging from 0 to 1.

Algorithm | Map	**12x11**	**22x22**	**40x30**	**64x64**	**Avg**
A2C	1.00 ±0.0	0.96 ±0.01	0.86 ±0.07	0.69 ±0.3	0.88
DQN	1.00 ±0.0	0.92 ±0.03	0.89 ±0.1	0.85 ±0.08	0.92
RAINBOW	1.00 ±0.0	0.95 ±0.03	0.90 ±0.08	0.64 ±0.01	0.87
IMPALA	1.00 ±0.0	0.96 ±0.04	0.95 ±0.04	0.85 ±0.11	0.94
PPO	1.00 ±0.0	0.99 ±0.01	0.95 ±0.01	0.66 ±0.18	0.90
DVAE	1.00 ±0.0	0.99 ±0.01	0.89 ±0.1	0.77 ±0.05	0.91
S-ORACLE	1.00 ±0.0	0.98 ±0.01	0.97 ±0.01	0.87 ±0.07	**0.96**

Table 6.6: The results of the Deep RTS Maze environment. The environment defines reward function as $r_t = optimalRoute \times 2 - t$. The results are averaged over 100 episodes. The cell values represent accumulated scores ranging from 0 to 113.

Algorithm	**Score**
A2C	91 ±5
DQN	98 ±9
RAINBOW	99 ±4
IMPALA	100 ±2
PPO	96 ±6
DVAE	97 ±11
S-ORACLE	94 ±5

Table 6.7: DeepRTS 1v1 results. The row is the win rate of the respective algorithm against the column-wise algorithm for 100 episodes. The cell values represent a win ratio ranging from 0 to 1.

Algorithm	A2C	DQN	RAINBOW	IMPALA	PPO	DVAE	S-ORACLE	Avg
A2C	-	0.35 ±0.11	0.23 ±0.05	0.44 ±0.11	0.14 ±0.05	0.39 ±0.11	0.36 ±0.05	0.32
DQN	0.65 ±0.03	-	0.45 ±0.02	0.36 ±0.05	0.25 ±0.03	0.36 ±0.32	0.25 ±0.14	0.39
RAINBOW	0.77 ±0.02	0.55 ±0.15	-	0.59 ±0.05	0.47 ±0.07	0.55 ±0.05	0.45 ±0.02	0.56
IMPALA	0.56 ±0.07	0.64 ±0.11	0.41 ±0.08	-	0.45 ±0.04	0.59 ±0.22	0.36 ±0.18	0.50
PPO	0.86 ±0.07	0.75 ±0.01	0.53 ±0.11	0.55 ±0.02	-	0.52 ±0.05	0.56 ±0.03	**0.63**
DVAE	0.61 ±0.25	0.64 ±0.05	0.45 ±0.05	0.41 ±0.06	0.48 ±0.05	-	0.25 ±0.02	0.47
S-ORACLE	0.64 ±0.11	0.75 ±0.02	0.55 ±0.09	0.64 ±0.03	0.44 ±0.02	0.75 ±0.02	-	**0.63**

training budget of 16 hours, and model-free algorithms have 24 hours. The Figure shows that model-based approaches utilize time significantly more, considering that total CPU

Table 6.8: ELF: Mini-RTS results. Each experiment runs for 100 episodes and is averaged over. The cell values represent a win ratio ranging from 0 to 1.

Algorithm	**AI-Simple**	**AI-Hit-and-run**	**Average**
A2C	0.85 ±0.07	0.86 ±0.08	0.86
DQN	0.56 ±0.05	0.85 ±0.02	0.71
RAINBOW	0.75 ±0.11	0.88 ±0.05	0.82
IMPALA	0.66 ±0.02	0.96 ±0.09	0.81
PPO	0.78 ±0.04	0.97 ±0.11	**0.88**
DVAE	0.73 ±0.05	0.99 ±0.05	0.86
S-ORACLE	0.67 ±0.09	0.98 ±0.04	0.83

Table 6.9: Micro RTS averaged results for all tested mini-games. Results for individual environments are found in the appendix of Paper M. The cell values represent a win ratio ranging from 0 to 1.

Agorithm	**RandomBiasedAI**	**NaiveMCTS**	**Tiamat**	**Droplet**	**UTS_Imass**	**Win Ratio**
A2C	0.89 ±0.01	0.86 ±0.06	0.84 ±0.11	0.79 ±0.14	0.84 ±0.13	0.84
DQN	0.91 ±0.04	0.86 ±0.12	0.79 ±0.03	0.82 ±0.17	0.7 ±0.26	0.82
RAINBOW	0.89 ±0.1	0.88 ±0.06	0.83 ±0.08	0.79 ±0.21	0.79 ±0.11	0.83
IMPALA	0.88 ±0.05	0.86 ±0.09	0.86 ±0.05	0.87 ±0.06	0.85 ±0.12	0.86
PPO	0.91 ±0.02	0.92 ±0.02	0.83 ±0.07	0.84 ±0.05	0.83 ±0.07	0.86
DVAE	0.90 ±0.07	0.87 ±0.09	0.80 ±0.18	0.90 ±0.07	0.84 ±0.15	0.86
S-ORACLE	0.90 ±0.06	0.88 ±0.02	0.86 ±0.02	0.84 ±0.04	0.86 ±0.05	**0.87**

and GPU time is close to model-free algorithms. The reason for better utilization is that model-based RL algorithms better saturate the GPUs when training using the dynamics model.

According to (Vinyals et al., 2019), for AlphaStar to beat professionals in StaCraft II, it trains 12 agents for 1056 hours using 32 third-generation Tensor Processing Units (TPU's). TPUv3 has a power consumption of 0.450kWh, which results in 14.4kWh for the TPU's. Excluding the rest of the system (CPU, memory, disks, and so on), it takes $14.4 * 1056 = 15206.4$kW to train AlphaStar without any model tuning. Considering a carbon intensity of 25, it takes 608.256 $\frac{CO2}{kg}$ climate emissions to train AlphaStar compared to S-ORACLE with only 6.77 $\frac{CO2}{kg}$ for a multitude of tested simpler environments.

6.5.6 Summary

This work investigates whether the safer model-based reinforcement learning algorithm S-ORACLE has (1) better reward performance, (2) sample efficiency, (3) more climate-

Table 6.10: StarCraft II results. The A2C, A3C, and DM are results from relevant literature, and the remainder is novel results in this work. We ran the experiments ten times and averaged the results. The cell values represent the total accumulated return. The table adopts results from H. Hu and Wang, 2020 for A2C, Zha et al., 2021 for A3C, and Vinyals et al., 2017 for the DeepMind (DM) results.

Environment	**A2C**	**A3C**	**DM**	**DQN**	**RAINBOW**	**IMPALA**	**PPO**	**ORACLE**	**S-ORACLE**
MoveToBeacon	21.3	24	26	26	30	32	35	24	29
DefeatRoaches	72.5	47	41	100	81	91	75	60	77
BuildMarines	0.55	0.6	138	0	0	2	8	12	2
CollectMineralShards	81	45	133	3	12	41	53	55	58
CollectMineralAndGas	3320	371	6880	3978	3911	4251	4102	5212	5102
FindAndDefeatZerglings	22.1	25	46	45	21	23	19	29	35
DefeatBanelingsAndZerglings	56.8	43	729	62	20	423	251	305	530
Average Score	510.6	79.3	**1141.8**	602	582.1	694.7	649	813.8	**833.2**

Table 6.11: This work contributed 55.08 kg of CO2eq to the atmosphere and used 1377.0 kWh of electricity. The columns describe the following from left to right: (1) Number of experiments performed, (2) Average hours of CPU time per experiment, (3) Average hours of GPU time per experiment, (4) Total CPU time for all experiments, (5) Total CPU time for all experiments, (6) Total kW for all experiments, and finally, the total $\frac{CO2}{kg}$ contribution of the experiments.[3]

Algorithm	**Num Exp**	**Avg CPU H.**	**Avg GPU H.**	**Tot CPU H.**	**Tot GPU H.**	**Tot. kW**	$\frac{CO2}{kg}$
A2C	41	13.53	21.41	554.73	877.81	233.04	9.32
DQN	41	11.71	20.22	480.11	829.02	219.02	8.76
RAINBOW	41	19.24	15.74	788.84	645.34	180.66	7.23
IMPALA	41	15.55	14.55	637.55	596.55	164.76	6.59
PPO	45	16.22	19.22	729.90	864.90	234.11	9.36
DVAE	45	14.33	11.24	644.85	505.80	142.25	5.69
S-ORACLE	49	13.11	12.53	642.39	613.97	169.23	6.77

friendly, and (4) safer than traditional model-free deep RL algorithms. Empirical evidence suggests that 2-4 hold except for 1, where the algorithm is on-par or inferior to model-free approaches. Similar conclusions are found in (Janner et al., 2019), emphasizing having smaller rollout horizons for better model prediction. This Section details further 1 to 4 and concludes the findings of this work.

Hypothesis S-ORACLE 1 *S-ORACLE is safer than risk-neutral algorithms and DVAE.*

Evaluation: Figure 6.26 to Figure 6.27 suggests that S-DVAE is safer than PPO. S-ORACLE is significantly safer in Deep RTS Lava Maze and Deep Ware 11x11 and has lower variance in the experiments. However, the results indicate that the safety margin diminishes for more complex environments such as Deep Warehouse 22x22 and Deep Warehouse 41x41. S-ORACLE shows similar results to S-DVAE, where it outperforms for simpler environments and where the safety effect diminishes for more complex environments, as seen in Figure 6.29.

Hypothesis S-ORACLE 2 *Compared to model-free algorithms, S-ORACLE is more climate-friendly.*
Evaluation: Table 6.11 shows that S-ORACLE trains are more climate-friendly compared to A2C, DQN, RAINBOW, and PPO but are marginally less efficient compared to DVAE and IMPALA.

Hypothesis S-ORACLE 3 *S-ORACLE outperforms model-free algorithms in terms of agent reward performance.*
Evaluation: S-ORACLE beats the other algorithms in Deep Line Wars, Deep RTS 1v1, and Micro RTS. There are several reasons for this. First, S-ORACLE is extensively tuned from work in Paper K and Paper M, whereas we use the reference values for the other algorithms. The second reason is that S-ORACLE has the benefits of optimizing the objective more aggressively towards safer actions. The positive effect is that the algorithm can avoid many scenarios where risk-neutral algorithms go in disastrous states. However, this is likely highly dependent on the environment, and we believe that S-ORACLE might fail in environments where other approaches thrive. Nonetheless, S-ORACLE performs empirically well in RTS Games.

6.6 Summary

We present results for our contributions, the Deep Variational Q-Networks (DVQN), Dreaming Variational Autoencoder (DVAE), Observation Reward Action Cost Learning Ensemble (ORACLE), and its decision-safe variants. We present results for our contributions, the Deep Variational Q-Networks (DVQN), Dreaming Variational Autoencoder (DVAE), Observation Reward Action Cost Learning Ensemble (ORACLE), and its decision-safe variants. We test our algorithms in the proposed game environments, Deep Line Wars, Deep Maze, Deep RTS, and Deep Warehouse, demonstrating that algorithms learn good strategies. However, there is still room for algorithms to perform better. Therefore, we believe that the experiments can function as a platform for RL research in RTS games towards mission-critical ASRS industry systems.

We demonstrate that the Deep Variational Q-Networks (DVQN) can learn latent embeddings that visually resemble the target environment. The CartPole-v0 experiment (Figure

6.18) empirically demonstrates the effectiveness of our algorithm. The embedding clearly separates into three clusters: one for when the pole is stable and two for pole angles in either direction. This work builds a foundation for innovations in hierarchical reinforcement learning because it is possible to create separate behavior policies for every cluster. The study also demonstrates the effectiveness of continuous embeddings (Gaussians) and presses for future study in categorical embeddings, which we later studied in the ORACLE algorithm with VQ-VAE.

The Dreaming Variational Autoencoder (DVAE) evaluations show that our algorithm can predict simple environments accurately. We show that DVAE can fully learn the DeepMaze-8x8-NoWall environment (Figure 6.3) and, combined with model-free algorithms, can navigate autogenerate mazes effectively (Figure 6.4). We show that adding RNN's boosts the reconstruction capabilities of DVAE but suffers from posterior collapse in environments with huge state-spaces, such as Sonic The Hedgehog. DVAE demonstrates empirically good reward performance, where it performs better or comparably to DQN and PPO in Deep Line Wars, Deep RTS, Deep Warehouse, and CartPole. We also suggest that stochastic weight averaging improves the learning stability, effectively boosting the predictive capabilities (Figure 6.9). Finally, we propose DVAE combined with CostNet for predicting distance from the current observation state to an estimated goal observation state. Results show that our method performs better than PPO and DQN in CartPole, DeepRTS, and Deep Maze (Figure 6.11).

Observation Reward Action Cost Learning Ensemble (ORACLE) proceeds the DVAE algorithm with new fundamentals to dynamics modeling. Specifically, we formalize the algorithm as a state-space model using recurrent neural networks and stochastic neural networks following the amortized variational inference training scheme. We show that our approach performs significantly better than DVAE and traditional model-free algorithms. We perform an extensive hyperparameter tuning experiment and show that training ORACLE on a single environment, SWA=on, VQ=on, LL=30, and AGC=on, works best for most environments. If ORACLE should learn a policy for multiple environments, disabling SWA for multi-policy training works best.

We show that our algorithms can operate more safely using the proposed safety mechanism, risk-directed exploration, and risk-sensitive decision-making. We show that our algorithms perform well in Deep Line Wars, DeepRTS, ELF, Micro RTS, and StarCraft II while also using resources, therefore, having less climate emission than model-free algorithms (Table 6.11). The reduced compute time corresponds well to the increased sample efficiency, as empirically verified in Figure 6.26. Although our approach cannot guarantee safety, the algorithm shows a good sign of reducing the failure rate. Therefore, our contribution is a good step towards versatile reinforcement learning in RTS and industry applications.

Chapter 7

Conclusion and Future Work

This thesis proposes a novel deep model-based reinforcement learning method towards mastering RTS games and improving agents' decision safety in industry-like environments. We approach the challenges with three divisions of labor. The first challenge in Chapter 4 is to fill the gap of missing environments in literature for RTS game research. We contribute six novel environments, Deep Line Wars, Deep Maze, Deep RTS, Deep Warehouse, CaiRL, and FlashRL, that are especially suited for reinforcement learning research, having a standardized interface of communications through the OpenAI Gym toolkit. The second challenge in Chapter 5 addresses model-based reinforcement learning towards better sample efficiency and better decision safety during learning. The final challenge in Chapter 6 empirically verifies our environments' usefulness and concurrently demonstrates our proposed algorithms' effectiveness.

7.1 Research Questions

We categorically conclude the research questions of this Ph.D. study towards more sample efficient algorithms for RTS games and more decision safe RL algorithms for industry-near applications.

Question 1: *To what extent can we improve the complexity gap for game environments in reinforcement learning research?*

Conclusion: We propose six new environments for reinforcement learning research. Our contributions serve toward creating environments with quantifiable state-space, which is a longstanding gap in available game environments (Figure 4.11). There is still room for new environments towards further state-space diversity and problem difficulty. For this reason, we propose CaiRL, a high-performance experiment toolkit that provides standardized methods to implement high-performance environments interoperable with existing environments toolkits. We study game environments design using high-level versus low-level languages (Section 4.6.2). From our results, we can conclude that SIMD CPU instruction sets and software rendering to avoid frame buffer copies are more efficient to use for environments with primitive graphics (CaiRL - Design Discussion). We empirically demonstrate that our environment contributions are suited for reinforcement learning research and provide significantly better reward performance than prior solutions (Table 4.13).

Question 2: *How can the sample efficiency of model-free reinforcement learning algorithms be increased to acquire good behavioral policies faster in complex RTS games?*

Conclusion: We introduce two model-based approaches, DVAE and ORACLE, and compare these against state-of-the-art model-based and model-free algorithms. We compare the sample efficiency of our model-based approach DVAE and ORACLE against state-of-the-art model-free algorithms such as PPO and DQN. Experiments empirically verify that learning a dynamics model using data from existing expert systems, then using the same dynamics model to train model-free algorithms, significantly reduces the samples needed to learn a good policy. We find that the sample efficiency dramatically increases when training traditional model-free algorithms using a dynamics model compared to traditional sampling techniques in the ground-truth environment. Specifically, we show that DVAE and ORACLE outperform other algorithms (Section 6.5.4) when operating within a limited computational budget. Furtermore, it converges faster than traditional model-free reinforcement learning because of the improved efficiency.

Question 3: *How can RTS game environments support reinforcement learning towards the goal of real-world industrial applications?*

Conclusion: We study existing literature on RTS games and industry applications separately and identify general environment characteristics. Finding such connections is essential because it indicates that RTS games can be directly used in research towards studying industry AI Control, which drastically lowers the cost of research. Our study suggests that both disciplines feature sparse rewards, partial observability, have high dimensional state-spaces (Figure 4.11) where it is required to plan safe decisions over longer time horizons. This analysis introduces the Deep RTS game environment and Deep Warehouse industry environment and shows that using the same algorithms and hyperparameters demonstrates similar learning curves and reward performance. Therefore, we conclude that using RTS games for research can support the development of novel AI control systems in industrial applications.

Question 4: *To what extent are deep Variational Autoencoder (VAE)s effective as a mechanism to learn the dynamics of virtual environments?*

Conclusion: We introduce DVAE, a model-based reinforcement learning approach for estimating a dynamics model using Variational Autoencoder (VAE)s to model the observation space. We show that deep VAEs can learn accurate dynamics of a virtual environment when supplied with sufficient training data. We empirically demonstrate that DVAE can predict future states in the Deep Line Wars environment with gradual improvement during training, demonstrating that it can successfully map state to latent state encodings without posterior collapse. For more complex game environments such as Sonic the Hedgehog 2, DVAE struggles to learn a good policy because there is too much similarity between state observations which causes the learning to collapse for even short-horizon predictions. Our study shows that VAEs can learn the dynamics of virtual environments and improve with extensions such as recurrent neural networks and optimization with stochastic weight averaging and is also later verified by Chien and Chiu, 2021; Gregor, Papamakarios, et al., 2019.

Question 5: *How can state-space modeling combined with recurrent neural networks for planning be used to learn the dynamics of a composite game environment?*

Conclusion: We introduce a novel model-based reinforcement learning named ORACLE that learns the dynamics of an environment through active exploration or passive observation. The model is a leap towards more sample-efficient algorithms to accommodate the lack of volumetric data in real-world systems. We use state-of-the-art state-space modeling techniques to form ORACLE for end-to-end learning of dynamic models in games and industry environments. We show that using state-space models boosts the prediction capabilities compared to DVAE. We perform an extensive hyperparameter search on core components of the ORACLE model and find that the model performs well for 30-step predictions and has mixed results when using separate models per environment. Using a single ORACLE model for multiple environments further demonstrates that disabling SWA, enabling VQ, and using AGC is the best combination of components in the tested environments (Table 6.3). Furthermore, our study shows that ORACLE performs better than DVAE and state-of-the-art model-free reinforcement learning in composite game environments such as StarCraft II.

Question 6: *How can we improve the decision safety of risk-neutral model-free reinforcement learning algorithms?*

Conclusion: We introduce a novel combination of decision safety mechanisms towards improving safety in reinforcement learning agents. We show that it is possible to extend ORACLE, DVAE, and model-free algorithms to better account for risk and decision safety. This is done using a combination of risk-directed exploration, following a risk-sensitive reward scheme described in Equation 2.19. Unique to DVAE is the CMDP's ability to limit the policy search space to eliminate unsafe policies combined with an inverted curiosity reward bonus to discourage exploration in uncertain parts of the state-space. We empirically show that the safe DVAE variant predicts states accurately while having comparable reward performance in a majority of the tested environments. It does so with significantly better decision safety than traditional model-free reinforcement learning algorithms, as Figure 6.14 demonstrates. The S-ORACLE variant shows better decision safety than DVAE and PPO (Section 6.5.3). ORACLE and DVAE integrate well with existing model-free algorithms. Furthermore, the decision safety improves significantly following the proposed training schemes. Therefore, a model-free reinforcement learning algorithm can benefit greatly from using the proposed models to train safer agents.

Question 7: *How can we apply deep reinforcement learning algorithms to industry-like applications without the risk of damaging humans and real-world equipment?*

Conclusion: Safety is perhaps the most challenging trait to master in an RL setting because it is not naturally embedded in the RL framework's foundation. Perhaps the first hint is found in the word *reinforcement*, which is the concept of *building upon something* or *encouraging or establishing a belief or pattern of behavior*. The RL literature often depicts the analogy of a child in a behavioristic way that through sensory stimuli such as vision, hearing, smell, feeling, and taste, the child learns through trial and error. However, when comparing RL algorithms to a child, we must strip away all its sensory information and memory and feed it noisy and sparse information about the world. If so, the child (algorithm) has very few sensory sensations, and at the beginning of time t does not know the environment. It seems nearly impossible to learn anything without trial and error. Therefore, the child analogy may be inadequate. Drawing parallels to ORACLE and DVAE, the algorithm is expected to perform erroneous decisions during learning. However, our study suggests the challenge is best approached using existing algorithms or expert systems already operational in the target environment (Section 6.5.3, Scheme #2). We show that using such a training scheme holds sufficient environment statistics to learn a dynamics model that can train model-free algorithms to make safe decisions. The experiments show that using expert-system knowledge to train the dynamics model works empirically well in the Deep Warehouse environment.

7.2 Environments

We have proposed six novel environments for research in reinforcement learning agents for RTS and industry-like applications. We empirically verify our new environments by training state-of-the-art reinforcement learning methods and systematically measure the achieved score and reward. The proposed environments help fill the state-space complexity gap in available reinforcement learning environments and are strictly open-source for peer collaboration and cross-verification of experiments. The environment source code can be found at https://github.com/topics/per-arne. We show that our proposed algorithms can learn behaviors in simple versions of the proposed environments, but it is more challenging to master more difficult scenario settings. The algorithm learns effective strategies using the same hyperparameters in composite game problems such as StarCraft II. We present baseline results for all environments, and using Deep Line Wars, we demonstrate that DVAE outperforms state-of-the-art algorithms in map sizes up to 30x30 (Table 4.3). We show that the Deep Maze environment is useful for studying pathfinding in RTS agents. Experiments show that our approach, the DVAE-CostNet, performs better than model-free approaches (Table 4.4). We present the most versatile RTS environment for RL research in recent literature to the best of our knowledge. We show that our algorithm performs safer while having comparable reward performance (Table 4.7) to state-of-the-art algorithms. Furthermore, we present the Deep Warehouse environment for industry-near testing of automated storage and retrieval systems. We show that our model-based approach, DVAE performs better than state-of-the-art model-free algorithms in most of the tested warehouse sizes (Table 4.8). Finally, we collect all of our game environment contributions in a highly efficient experiment engine, CaiRL, that demonstrates significantly better CPU efficiency. CaiRL provides an easy-to-use framework that is operable with the OpenAI Gym toolkit, with the benefit of substantially reducing experiment runtimes and the positive side-effect of reducing climate emission of machine learning research. Specifically, we show that CaiRL has 20.89 times less carbon emission than OpenAI Gym in console-based applications (no graphics). The graphical experiment shows a more significant difference with a reduction of 147 578 times the carbon emissions. This is because OpenAI Game implementations are locked to 60 frames per second (Table 4.11) using much more computational resources than is needed. The overall conclusion from our study on environments is that by carefully crafting environments with efficient techniques, experiment time is drastically reduced, saving time, allowing for more complex models to train faster, reducing the computational cost and hence, economic costs. Finally, it contributes towards more sustainable climate emissions in machine learning research.

7.3 Model-Based Reinforcement Learning Algorithms

With the Dreaming Variational Autoencoder (DVAE), Deep Variational Q-Networks (DVQN), and Observation Reward Action Cost Learning Ensemble (ORACLE), we introduce three model-based reinforcement learning approaches towards better agents for RTS games and industry-like applications. There are many similarities with the DVAE and ORACLE algorithms, but our study indicates a significant difference in training speed. For simple environments, we found that DVAE converges significantly faster, and because the model has less components, it also utilizes the GPU better than ORACLE. Below we will present conclusions regarding our proposed models.

The Dreaming Variational Autoencoder (DVAE) combines VAEs and reinforcement learning to learn a dynamics model that accurately reproduces state observations in the game and real-world environments. We demonstrate that the algorithm is effective in simple environments, such as the Deep Line Wars and Deep Maze, and can accurately predict 3-step future observations in deterministic environments. Our model combines traditional model-free reinforcement learning algorithms, such as Deep Q-Networks. We show that learning a dynamics model using DVAE can effectively train model-free algorithms through empirical observations. We also found that the DVAE model performs poorly in environments with complex state-spaces. To remedy this, we propose extensions to improve the expressive capabilities of the model. We extend the DVAE model with recurrent neural networks, significantly boosting the learning of temporal dependencies between states. Furthermore, we discover that it is possible to combine GAN and VAE where the latter is the generator to produce more accurate samples. Further studies found that using Stochastic Weight Averaging (SWA), an ensemble-like training technique stabilized training and empirically reduced posterior collapse frequency. Finally, we combine DVAE with the CostNet model for estimating distances between state observations in an MDP. We show that these extensions improve reward performance, although there is still the need for manual hyperparameter tuning depending on the problem. We present sane default hyperparameters as used throughout our study. The reward performance experiments show that DVAE is an effective algorithm in industry-like applications. Specifically, DVAE can converge to 200 points in large 41x41 Deep Warehouse environments compared to 160 points for Deep Q-Networks (Figure 6.15).

Deep Variational Q-Networks is a novel reinforcement learning algorithm. The goal is to create an interpretable latent space to cluster similar areas of the observed state-space. We aim to empirically study if VAEs can function as a model for automatically creating initialization and termination signals for options in hierarchical reinforcement learning using the Options framework. We show that our method can structurally model observed state-spaces visually and that it is trivial to construct clusters that can act as separate behaviors for multi-level policies (Figure 6.18). We demonstrate that the algorithm, which fuels learning of a Q-function policy, performs better than DQN and shows comparable reward performance to DDQN and Rainbow but is less sample efficient. We conclude that it is feasible to study the method further, which builds momentum to study the variational vector-quantization technique that became essential for multi-environment policies in the ORACLE algorithm.

Observation Reward Action Cost Learning Ensemble extends DVAE in the direction of state-space modeling in combination with the VAE and recurrent neural networks to form a highly expressive model for long-horizon predictions. Our model uses state-of-the-art stochastic recurrent state-space models to learn high dimensional state-spaces from StarCraft II, Deep RTS, Deep Warehouse. We show that our model can learn to predict trajectories of 30 steps ahead and demonstrate that in combination with existing model-free algorithms can significantly increase sample efficiency during learning. Our experiments show that ORACLE requires hyperparameter tuning for individual environments but can also learn good policies in multiple environments using a singular model if using vector-quantization. We compare ORACLE to state-of-the-art model-free algorithms, and experiments show that it scores 14% better in HalfCheetahPyBulletEnv-v1, converges 60% faster in CartPole-v1, and scores 11% better in Deep RTS Deathmatch.

7.4 Safe Reinforcement Learning

Safety is crucial for an algorithm that is supposed to operate in industry-like production environments outside the research lab. We propose several novel methods in decision safety for reinforcement learning agents that are empirically verified in several experiments. DVAE, demonstrates that policy constraints reduce the risk of entering catastrophic states by altering the reward signal using risk-directed exploration and inverted curiosity. Our results show that the proposed safer learning scheme reduces failure in most tested environments and performs more safely than traditional model-free reinforcement learning (Figure 6.14). We conclude that this approach by no means is the solution to a perfectly safe algorithm, but it progresses the literature towards safer methods. We extend the safety study to the ORACLE algorithm and propose following a risk-sensitive reward signal using the variance of predicted rewards. In combination with the CostNet mechanism, we form a three-component reward signal (Equation 5.14) that significantly reduces the agent failure rate in combination with the actor ensemble. As a secondary measure, we propose two schemes for training the algorithm where Scheme #2 provides additional safety through passively learning a dynamics model for pretraining the actor module. A key contribution in this work is that our methods do not rely on model assumptions and are generally model agnostic, meaning that they can integrate directly with or without model-based learning. Our vision is to perfect the ORACLE algorithm so that agents can learn without relying on theoretical assumptions that do not work in practice and can scale to complex real-world environments.

7.5 Future Work

This Ph.D. work presents several new methods in deep reinforcement learning, which opens up for a future study to improve AI Control agents' learning in RTS games and industry applications. This chapter summarizes our contributions to four topics; environments, dynamics modeling, decision safety, and applied IAI.

Environments: We have proposed a series of environments compatible with the OpenAI Gym toolkit through the CaiRL toolkit. All contributions are open source and form a foundation for future work on improving the environments to reach a broader audience in the AI research community. While all proposed environments provide improvements, it should be noted that the possible future work of CaiRL includes more efficient use of SIMD instruction sets and future porting of fundamental reinforcement learning problems to the platform. This can reduce the execution time of environments by orders of magnitude and further reduce the climate footprint of machine learning. There is an increasing uptake in focus to minimize the climate footprint of algorithms, but little work has been done on environment execution. Therefore, we hope to increase the attention towards developing and maintaining efficient game environments for RL research.

Model-Based Reinforcement Learning: We have extensively studied the use of VAEs and variational inference to learn a dynamics model for training traditional model-free algorithms. Our work clearly shows that this approach works well. However, it still has several shortcomings that, if addressed, should significantly improve the quality of predictions, increase sample efficiency, and consequently increase the agent reward performance after training. Perhaps the most notable challenge is to prevent posterior collapse. Several approaches are appealing for addressing posterior collapse. The work from Razavi, van den Oord, Poole, et al., 2019 proposes δ-VAE, a framework for selecting variational families that prevent posterior collapse without modification of the training objective (ELBO). A list of future research directions to eliminate this problem in the DVAE and ORACLE is found at https://github.com/sajadn/posterior-collapse-list. The ORACLE algorithm empirically performs better than DVAE due to the recurrent state-space model. Recent work proposed a new class of time-continuous recurrent neural network models called Liquid Time-constant Networks (Hasani et al., 2021). Our preliminary study suggests that these networks can better learn temporal data to increase the expressive capabilities of ORACLE for longer prediction sequences.

Hierarchical Reinforcement Learning with Options: We present DVQN for continuous space interpretability of state-space and study categorical latent spaces in ORACLE. This work builds momentum for a branch of novel RL option discovery methods that enable learning multi-level policies and potentially allow for more reusable RL algorithms where different behavioral policies are learned separately. Specifically, we demonstrate that combining clustering with DVQN enables separate behaviors, and for ORACLE, we demonstrate that using VQ-based models can better encode multiple environments.

Decision Safety: We present several novel techniques for improving safety in model-based reinforcement learning agents during training and inference. Our method primarily focuses on reward shaping techniques that guide the agent to goal states more safely. However, it cannot guarantee safety. On the other hand, CMDP's, as initially used in the DVAE model, can provide theoretical guarantees for convergences given that the model is known or can be learned accurately. Existing work has seen success with using Lyapunov functions. We wish to investigate if it is possible to model the problem similarly, with fewer assumptions. One appealing approach is Han et al., 2021, which shows how RL with UUB guarantee can be applied to control dynamic systems with safety constraints. The authors show that the proposed method can achieve superior performance in maintaining safety.

Industry-Near Environments: The proposed algorithms demonstrate good empirical reward performance in virtual industry-near environments, but we have not had the opportunity to test our algorithms in real-world systems. The hope is that future work can include resources to test the algorithm in grid-like storage systems, alternatively using miniature physical toy installations. This can provide valuable data on how to train algorithms in environments where external energy can create stochastic behaviors in an otherwise deterministic environment.

Bibliography

Altman, E. (1999). *Constrained Markov decision processes* (1st ed.). CRC Press.

Andersen, P.-A., Goodwin, M., & Granmo, O.-C. (2017a). FlashRL: A Reinforcement Learning Platform for Flash Games. *Norwegian Informatics Conference*.

Andersen, P.-A., Goodwin, M., & Granmo, O.-C. (2017b). Towards a Deep Reinforcement Learning Approach for Tower Line Wars. In M. Bramer & M. Petridis (Eds.), *Artificial intelligence xxxiv* (pp. 101–114). Springer International Publishing. https://doi.org/10.1007/978-3-319-71078-5_8

Andersen, P.-A., Goodwin, M., & Granmo, O.-C. (2018a). Deep RTS: A Game Environment for Deep Reinforcement Learning in Real-Time Strategy Games. *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, 1–8. https://doi.org/10.1109/CIG.2018.8490409

Andersen, P.-A., Goodwin, M., & Granmo, O.-C. (2018b). The Dreaming Variational Autoencoder for Reinforcement Learning Environments. In Max Bramer & M. Petridis (Eds.), *Artificial intelligence xxxv* (XXXV, pp. 143–155). Springer, Cham. https://doi.org/10.1007/978-3-030-04191-5_11

Andersen, P.-A., Goodwin, M., & Granmo, O.-C. (2019). Towards Model-Based Reinforcement Learning for Industry-near Environments. In M. Bramer & M. Petridis (Eds.), *Artificial intelligence xxxvi* (pp. 36–49). Springer International Publishing. https://doi.org/10.1007/978-3-030-34885-4_3

Andersen, P.-A., Goodwin, M., & Granmo, O.-C. (2020a). CostNet: An End-to-End Framework for Goal-Directed Reinforcement Learning. In *Artificial intelligence xxxvii* (pp. 94–107). Springer, Cham. https://doi.org/10.1007/978-3-030-63799-6_7

Andersen, P.-A., Goodwin, M., & Granmo, O.-C. (2020b). Increasing Sample Efficiency in Deep Reinforcement Learning using Generative Environment Modelling. *Expert Systems*. https://doi.org/10.1111/exsy.12537

Andersen, P.-A., Goodwin, M., & Granmo, O.-C. (2020c). Interpretable Option Discovery using Deep Q-Learning and Variational Autoencoders. *3rd International Conference on Intelligent Technologies and Applications*, *1382*, 127–138. https://doi.org/10.1007/978-3-030-71711-7_11

Andersen, P.-A., Goodwin, M., & Granmo, O.-C. (2020d). Safer Reinforcement Learning for Agents in Industrial Grid-Warehousing. *6th International Conference on Machine Learning, Optimization, and Data Science (LOD)*, *12566 LNCS*, 169–180. https://doi.org/10.1007/978-3-030-64580-9_14

Andersen, P.-A., Goodwin, M., & Granmo, O.-C. (2020e). Towards Safe Reinforcement-learning in Industrial Grid-warehousing. *Information Sciences*, *537*, 467–484. https://doi.org/10.1016/j.ins.2020.06.010

Andersen, P.-A., Goodwin, M., & Granmo, O.-C. (2021a). CaiRL: A High-Performance Reinforcement Learning Environment Toolkit. *IEEE Conference on Games*, 10.

Andersen, P.-A., Goodwin, M., & Granmo, O.-C. (2021b). ORACLE: End-to-End Model Based Reinforcement Learning. In *Artificial intelligence xxxviii* (pp. 1–14). Springer, Cham. https://doi.org/10.1007/978-3-030-91100-3_4

Andersen, P.-A., Goodwin, M., & Granmo, O.-C. (2021c). Towards Safe and Sustainable Reinforcement Learning for Real-Time Strategy Games. *Artificial Intelligence*.

Andersen, P.-A., Kråkevik, C., Goodwin, M., & Yazidi, A. (2016). Adaptive task assignment in online learning environments. *ACM International Conference Proceeding Series*, *13-15-June*. https://doi.org/10.1145/2912845.2912854

Annasamy, R. M., & Sycara, K. (2018). Towards Better Interpretability in Deep Q-Networks. *Proceedings, The Thirty-Third AAAI Conference on Artificial Intelligence*. http://arxiv.org/abs/1809.05630

Arulkumaran, K., Deisenroth, M. P., Brundage, M., & Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, *34*(6), 26–38. https://doi.org/10.1109/MSP.2017.2743240

Azar, M. G., Piot, B., Pires, B. A., Grill, J.-B., Altché, F., & Munos, R. (2019). World Discovery Models. *ArXiv e-prints*, (Mar).

Azizzadenesheli, K., Liu, W., Lipton, Z. C., & Anandkumar, A. (2018). Sample-Efficient Deep RL with Generative Adversarial Tree Search. *Workshop at the thirty-fifth International Conference on Machine Learning (ICML)*.

Bagnell, J. A., Ng, A. Y., & Schneider, J. G. (2001). Solving Uncertain Markov Decision Processes. *Carnegie Mellon Research Showcase*.

Baldi, P. (2012). Autoencoders, Unsupervised Learning, and Deep Architectures. *ICML Unsupervised and Transfer Learning*. https://doi.org/10.1561/2200000006

Bangaru, S. P., Suhas, J., & Ravindran, B. (2016). Exploration for Multi-task Reinforcement Learning with Deep Generative Models. *arxiv preprint arXiv:1611.09894*. http://arxiv.org/abs/1611.09894

Barriga, N. A., Stanescu, M., & Buro, M. (2017). Game Tree Search Based on Non-Deterministic Action Scripts in Real-Time Strategy Games. *IEEE Transactions on Computational Intelligence and AI in Games*, 1–1. https://doi.org/10.1109/TCIAIG.2017.2717902

Barto, A., Mahadevan, S., & Lazaric, A. (2003). *Recent Advances in Hierarchical Reinforcement Learning* (tech. rep.). PIGML Seminar-AirLab.

Beattie, C., Leibo, J. Z., Teplyashin, D., Ward, T., Wainwright, M., Küttler, H., Lefrancq, A., Green, S., Valdés, V., Sadik, A., Schrittwieser, J., Anderson, K., York, S., Cant, M., Cain, A., Bolton, A., Gaffney, S., King, H., Hassabis, D., ... Petersen, S. (2016). DeepMind Lab. *arxiv preprint arXiv:1612.03801*. http://arxiv.org/abs/1612.03801

Beckschäfer, M., Malberg, S., Tierney, K., & Weskamp, C. (2017). Simulating Storage Policies for an Automated Grid-Based Warehouse System. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *10572 LNCS*, 468–482. https://doi.org/10.1007/978-3-319-68496-3_31

Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2013). The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*, *47*, 253–279. https://doi.org/10.1613/jair.3912

Bellman, R. (1952). On the Theory of Dynamic Programming. *Proceedings of the National Academy of Sciences*, *38*(8), 716–719. https://doi.org/10.1073/PNAS.38.8.716

Berkenkamp, F., Krause, A., & Schoellig, A. P. (2021). Bayesian optimization with safety constraints: safe and automatic parameter tuning in robotics. *Machine Learning 2021*, 1–35. https://doi.org/10.1007/S10994-021-06019-1

Berkenkamp, F., Turchetta, M., Schoellig, A. P., & Krause, A. (2017). Safe Model-based Reinforcement Learning with Stability Guarantees. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in neural information processing systems 30* (pp. 908–918). Curran Associates, Inc. https://papers.nips.cc/paper/6692-safe-model-based-reinforcement-learning-with-stability-guarantees

Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., de Oliveira Pinto, H. P., Raiman, J., Salimans, T., Schlatter, J., ... Zhang, S. (2019). Dota 2 with Large Scale Deep Reinforcement Learning. *CoRR*, *abs/1912.0*. http://arxiv.org/abs/1912.06680

Blundell, C., Uria, B., Pritzel, A., Li, Y., Ruderman, A., Leibo, J. Z., Rae, J., Wierstra, D., & Hassabis, D. (2016). Model-Free Episodic Control. *arxiv preprint arXiv:1606.04460*. http://arxiv.org/abs/1606.04460

Botvinick, M., Ritter, S., Wang, J. X., Kurth-Nelson, Z., Blundell, C., & Hassabis, D. (2019). Reinforcement Learning, Fast and Slow. *Trends in cognitive sciences*, *23*(5), 408–422. https://doi.org/10.1016/j.tics.2019.02.006

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). OpenAI Gym. *arxiv preprint arXiv:1606.01540*. http://arxiv.org/abs/1606.01540

Buesing, L., Weber, T., Racaniere, S., Eslami, S. M. A., Rezende, D., Reichert, D. P., Viola, F., Besse, F., Gregor, K., Hassabis, D., & Wierstra, D. (2018). Learning and

Querying Fast Generative Models for Reinforcement Learning. *Workshop at the thirty-fifth International Conference on Machine Learning (ICML)*. http://arxiv.org/abs/1802.03006

Campos, P., & Langlois, T. (2003). Abalearn: Efficient self-play learning of the game abalone. *INESC-ID, neural networks and signal processing group*.

Chen, W., Zhang, M., Zhang, Y., & Duan, X. (2016). Exploiting meta features for dependency parsing and part-of-speech tagging. *Artificial Intelligence*, *230*, 173–191. https://doi.org/10.1016/j.artint.2015.09.002

Cheng, R., Orosz, G., Murray, R. M., & Burdick, J. W. (2019). End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks. *33rd AAAI Conference on Artificial Intelligence, AAAI 2019, 31st Innovative Applications of Artificial Intelligence Conference, IAAI 2019 and the 9th AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019*. https://doi.org/10.1609/aaai.v33i01.33013387

Chevalier-Boisvert, M., Lahlou, S., Nguyen, T. H., Bahdanau, D., Willems, L., Bengio, Y., & Saharia, C. (2019). Babyai: A platform to study the sample efficiency of grounded language learning. *7th International Conference on Learning Representations, ICLR 2019*.

Chevalier-Boisvert, M., Willems, L., & Pal, S. (2018). Minimalistic Gridworld Environment for OpenAI Gym.

Chiang, I.-H., Huang, C.-M., Cheng, N.-H., Liu, H.-Y., & Tsai, ·. S.-C., givenun=0. (2019). Efficient Exploration in Side-Scrolling Video Games with Trajectory Replay. *The Computer Games Journal 2019 9:3*, *9*(3), 263–280. https://doi.org/10.1007/S40869-019-00089-X

Chien, J. T., & Chiu, Y. C. (2021). Stochastic Temporal Difference Learning for Sequence Data. *Proceedings of the International Joint Conference on Neural Networks*, *2021-July*. https://doi.org/10.1109/IJCNN52387.2021.9534155

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*. https://doi.org/10.3115/v1/d14-1179

Chow, Y., Nachum, O., Duenez-Guzman, E., & Ghavamzadeh, M. (2018). A Lyapunov-Based Approach to Safe Reinforcement Learning. *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 8103–8112. https://dl.acm.org/doi/abs/10.5555/3327757.3327904

Chua, K., Calandra, R., McAllister, R., & Levine, S. (2018). Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.), *Advances in neural information processing systems 31* (pp. 4754–4765). Curran Associates, Inc.

Churchill, D., Lin, Z., & Synnaeve, G. (2017). An Analysis of Model-Based Heuristic Search Techniques for StarCraft Combat Scenarios. *Workshop in Artificial Intelligence and Strategy Games*, 8–14. https://aaai.org/ocs/index.php/AIIDE/AIIDE17/paper/view/15916

Clevert, D.-A., Unterthiner, T., & Hochreiter, S. (2015). Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). *The International Conference on Learning Representations 16*. http://arxiv.org/abs/1511.07289

Commandeur, J. J., Koopman, S. J., & Ooms, M. (2011). Statistical software for state space methods. *Journal of Statistical Software*, *41*(1). https://doi.org/10.18637/jss.v041.i01

de S. Braga, A. P., & Araújo, A. F. R. (1998). Goal-Directed Reinforcement Learning Using Variable Learning Rate. Springer, Berlin, Heidelberg. https://doi.org/10.1007/10692710_14

Debnath, S., Sukhatme, G., & Liu, L. (2018). Accelerating Goal-Directed Reinforcement Learning by Model Characterization. *IEEE International Conference on Intelligent Robots and Systems*, 8666–8673. https://doi.org/10.1109/IROS.2018.8593728

del R. Millán, J. (1995). Reinforcement learning of goal-directed obstacle-avoiding reaction strategies in an autonomous mobile robot. *Robotics and Autonomous Systems*, *15*(4), 275–299. https://doi.org/10.1016/0921-8890(95)00021-7

Ding, Z., & Dong, H. (2020). Challenges of Reinforcement Learning. *Deep Reinforcement Learning: Fundamentals, Research and Applications*, 249–272. https://doi.org/10.1007/978-981-15-4095-0_7

Doerr, A., Daniel, C., Schiegg, M., Duy, N.-T., Schaal, S., Toussaint, M., & Sebastian, T. (2018). Probabilistic Recurrent State-Space Models. In J. Dy & A. Krause (Eds.), *Proceedings of the 35th international conference on machine learning* (pp. 1280–1289). PMLR. http://proceedings.mlr.press/v80/doerr18a.html

Eddy, S. R. (2004). What is a hidden Markov model? *Nature Biotechnology 2004 22:10*, *22*(10), 1315–1316. https://doi.org/10.1038/nbt1004-1315

Edith, L. L., Melanie, C., Doina, P., & Bohdana, R. (2005). Risk-directed Exploration in Reinforcement Learning. *IJCAI 2005 Workshop on Planning and Learning in A Priori Unknown or Dynamic Domains*.

Fan, J., Wang, Z., Xie, Y., & Yang, Z. (2019). *A Theoretical Analysis of Deep Q-Learning* (tech. rep.). Princeton University.

Fathy, I., Aref, M., Enayet, O., & Al-Ogail, A. (2010). Intelligent online case-based planning agent model for real-time strategy games. *Proceedings, Intelligent Systems*

Design and Applications (ISDA), 2010 10th International Conference on IEEE, 445–450. https://doi.org/10.1109/ISDA.2010.5687225

Feinberg, E. A., Kasyanov, P. O., & Zgurovsky, M. Z. (2014). Convergence of value iterations for total-cost MDPs and POMDPs with general state and action sets. *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, 8. https://doi.org/10.1109/ADPRL.2014.7010613

Feinberg, E. A., & Lewis, M. E. (2018). On the convergence of optimal actions for Markov decision processes and the optimality of (s , S) inventory policies. *Naval Research Logistics*, *65*(8), 619–637. https://doi.org/10.1002/nav.21750

Fraccaro, M. (2018). *Deep latent variable models for sequential data* (Doctoral dissertation). Technical University of Denmark. DTU Compute. https://orbit.dtu.dk/en/publications/deep-latent-variable-models-for-sequential-data

García, J., & Fernández, F. (2015). A Comprehensive Survey on Safe Reinforcement Learning. *Journal of Machine Learning Research*, *16*, 1437–1480.

Gaskett, C. (2003). Reinforcement learning under circumstances beyond its control. *Proceedings of the international conference on computational intelligence for modelling control and automation*. http://www.his.atr.co.jp/cgaskett/

Geibel, P., & Wysotzki, F. (2005). Risk-Sensitive Reinforcement Learning Applied to Control under Constraints. *Journal of Artificial Intelligence Research*, *24*, 81–108. https://doi.org/10.1613/jair.1666

Gosavi, A. (2009). Reinforcement learning: A tutorial survey and recent advances. *INFORMS Journal on Computing*, *21*(2), 178–192.

Gregor, K., Jimenez Rezende, D., Besse, F., Wu, Y., Merzic, H., & van den Oord, A. (2019). Shaping Belief States with Generative Environment Models for RL. In H. Wallach, H. Larochelle, A. Beygelzimer, F. Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems 32* (pp. 13475–13487). Curran Associates, Inc. http://papers.nips.cc/paper/9503-shaping-belief-states-with-generative-environment-models-for-rl.pdf

Gregor, K., Papamakarios, G., Besse, F., Buesing, L., & Weber, T. (2019). Temporal difference variational auto-encoder. *7th International Conference on Learning Representations, ICLR 2019*.

Ha, D., & Schmidhuber, J. (2018a). Recurrent World Models Facilitate Policy Evolution. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.), *Advances in neural information processing systems 31* (pp. 2450–2462). Curran Associates, Inc. http://papers.nips.cc/paper/7512-recurrent-world-models-facilitate-policy-evolution.pdf

Ha, D., & Schmidhuber, J. (2018b). World Models. *arxiv preprint arXiv:1803.10122*. https://doi.org/10.5281/zenodo.1207631

Haddad, S., & Monmege, B. (2014). Reachability in MDPs: Refining convergence of value iteration. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *8762*, 125–137. https://doi.org/10.1007/978-3-319-11439-2_10

Hafner, D., Lillicrap, T., Ba, J., & Norouzi, M. (2020). Dream to Control: Learning Behaviors by Latent Imagination. *Proc. 8th International Conference on Learning Representations, ICLR'20*, 1–26. https://openreview.net/forum?id=S1lOTC4tDS

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., & Davidson, J. (2019). Learning Latent Dynamics for Planning from Pixels. In K. Chaudhuri & R. Salakhutdinov (Eds.), *Proc. 36th international conference on machine learning, icml'18* (pp. 2555–2565). PMLR. http://proceedings.mlr.press/v97/hafner19a/hafner19a.pdf

Hafner, D., Lillicrap, T. P., Norouzi, M., & Ba, J. (2021). Mastering Atari with Discrete World Models. *Proc. 9th International Conference on Learning Representations, ICLR'21*. https://openreview.net/forum?id=0oabwyZbOu

Hairer, M. (2016). *Convergence of Markov Processes* (tech. rep.). Mathematics Department, University of Warwick.

Han, M., Tian, Y., Zhang, L., Wang, J., & Pan, W. (2021). Reinforcement learning control of constrained dynamic systems with uniformly ultimate boundedness stability guarantee. *Automatica*, *129*, 109689. https://doi.org/10.1016/J.AUTOMATICA.2021.109689

Hasani, R., Lechner, M., Amini, A., Rus, D., & Grosu, R. (2021). Liquid Time-constant Networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, *35*(9 SE - AAAI Technical Track on Machine Learning II), 7657–7666. https://ojs.aaai.org/index.php/AAAI/article/view/16936

Hausknecht, M., & Stone, P. (2015). Deep Recurrent Q-Learning for Partially Observable MDPs. *AAAI Fall Symposium Series*, 1–9.

Heger, M. (1994). Consideration of Risk in Reinforcement Learning. In W. W. Cohen & H. Haym (Eds.), *Proc. 11th international conference on machine learning, icml'94* (pp. 105–111). Elsevier. https://doi.org/10.1016/B978-1-55860-335-6.50021-0

Henderson, P., Hu, J., Romoff, J., Brunskill, E., Jurafsky, D., & Pineau, J. (2020a). Towards the Systematic Reporting of the Energy and Carbon Footprints of Machine Learning. *Journal of Machine Learning Research*, *21*, 1–43. http://jmlr.org/papers/v21/20-312.html.

Henderson, P., Hu, J., Romoff, J., Brunskill, E., Jurafsky, D., & Pineau, J. (2020b). Towards the systematic reporting of the energy and carbon footprints of machine learning. *Journal of Machine Learning Research*, *21*(248), 1–43.

Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., & Silver, D. (2018). Rainbow: Combining Improvements in

Deep Reinforcement Learning. *Proc. 32nd Conference on Artificial Intelligence, AAAI'18*, 3215–3222. https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/download/17204/16680

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., & Lerchner, A. (2017). beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. *Proc. 5th International Conference on Learning Representations, ICLR'17*. https://openreview.net/forum?id=Sy2fzU9gl

Higgins, I., Pal, A., Rusu, A., Matthey, L., Burgess, C., Pritzel, A., Botvinick, M., Blundell, C., & Lerchner, A. (2017). DARLA: Improving Zero-Shot Transfer in Reinforcement Learning. In D. Precup & Y. W. Teh (Eds.), *Proceedings of the 34th international conference on machine learning* (pp. 1480–1490). PMLR. http://proceedings.mlr.press/v70/higgins17a.html

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8). https://doi.org/10.1162/neco.1997.9.8.1735

Hu, H., & Wang, Q. (2020). Implementation on benchmark of SC2LE environment with advantage actor - Critic method. *2020 International Conference on Unmanned Aircraft Systems, ICUAS 2020*, 362–366. https://doi.org/10.1109/ICUAS48674.2020.9214032

Hu, X., Liu, T., Qi, X., & Barth, M. (2019). Reinforcement Learning for Hybrid and Plug-In Hybrid Electric Vehicle Energy Management: Recent Advances and Prospects. *IEEE Industrial Electronics Magazine*, *13*(3), 16–25. https://doi.org/10.1109/MIE.2019.2913015

Huang, S., Ontanon, S., Bamford, C., & Grela, L. (2021). Gym-MicroRTS: Toward Affordable Full Game Real-time Strategy Games Research with Deep Reinforcement Learning. *Proc. 3rd IEEE Conference on Games*, 19. https://arxiv.org/abs/2105.13807v3

Huang, S., Su, H., Zhu, J., & Chen, T. (2020). SVQN: Sequential Variational Soft Q-Learning Networks. *International Conference on Learning Representations*. https://openreview.net/forum?id=r1xPh2VtPB

Izmailov, P., Podoprikhin, D., Garipov, T., Vetrov, D., & Wilson, A. G. (2018). Averaging Weights Leads to Wider Optima and Better Generalization. In A. G. R. Silva & A. Globerson (Eds.), *34th conference on uncertainty in artificial intelligence 2018* (pp. 876–885). Association For Uncertainty in Artificial Intelligence. http://arxiv.org/abs/1803.05407

Jaakkola, T., Jordan, M., & Singh, S. (1994). On the Convergence of Stochastic Iterative Dynamic Programming Algorithms. *Neural Computation*, *6*(6), 1185–1201. https://doi.org/10.1162/neco.1994.6.6.1185

Jaidee, U., & Muñoz-Avila, H. (2012). CLASSQ-L: A Q-Learning Algorithm for Adversarial Real-Time Strategy Games. *Proceedings, The Eighth AAAI Conference on*

Artificial Intelligence and Interactive Digital Entertainment, 8–13. http://www.aaai.org/ocs/index.php/AIIDE/AIIDE12/paper/viewFile/5515/5734

Janner, M., Fu, J., Zhang, M., & Levine, S. (2019). When to Trust Your Model: Model-Based Policy Optimization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. Alché-Buc, E. Fox, & R. Garnett (Eds.), *Proc. 33rd conference on neural information processing systems (neurips)* (pp. 12519–12530). Curran Associates, Inc. http://papers.nips.cc/paper/9416-when-to-trust-your-model-model-based-policy-optimization.pdf

Johnson, M., Hofmann, K., Hutton, T., & Bignell, D. (2016). The malmo platform for artificial intelligence experimentation. *IJCAI International Joint Conference on Artificial Intelligence*, *2016-Janua*.

Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., & Saul, L. K. (1999). Introduction to variational methods for graphical models. *Machine Learning*, *37*(2), 183–233. https://doi.org/10.1023/A:1007665907178

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*. https://doi.org/10.1.1.68.466

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Journal of Fluids Engineering, Transactions of the ASME*, *82*(1). https://doi.org/10.1115/1.3662552

Kearns, M., & Singh, S. (2002). Near-Optimal Reinforcement Learning in Polynomial Time. *Machine Learning 2002 49:2*, *49*(2), 209–232. https://doi.org/10.1023/A:1017984413808

Kempka, M., Wydmuch, M., Runc, G., Toczek, J., & Jaskowski, W. (2016). ViZDoom: A Doom-based AI research platform for visual reinforcement learning. *IEEE Conference on Computatonal Intelligence and Games, CIG*, *0*. https://doi.org/10.1109/CIG.2016.7860433

Kendall, A., & Gal, Y. (2017). What uncertainties do we need in Bayesian deep learning for computer vision? *Advances in Neural Information Processing Systems*, *2017-Decem*, 5580–5590. http://papers.nips.cc/paper/7141-what-uncertainties-do-we-need-in-bayesian-deep-learning-for-computer-vision

Kingma, D. P., & Ba, J. L. (2015). Adam: A Method for Stochastic Optimization. *Proceedings of the 3rd International Conference on Learning Representations*. https://doi.org/10.1145/1830483.1830503

Kingma, D. P., & Welling, M. (2013). Auto-Encoding Variational Bayes. *Proceedings of the 2nd International Conference on Learning Representations*. https://doi.org/10.1051/0004-6361/201527329

Kingma, D. P., & Welling, M. (2019). An introduction to variational autoencoders. https://doi.org/10.1561/2200000056

Kiumarsi, B., Vamvoudakis, K. G., Modares, H., & Lewis, F. L. (2018). Optimal and Autonomous Control Using Reinforcement Learning: A Survey. *IEEE Transactions on Neural Networks and Learning Systems*, *29*(6), 2042–2062. https://doi.org/10.1109/TNNLS.2017.2773458

Kobayashi, T. (2019). Student-t policy in reinforcement learning to acquire global optimum of robot control. *Applied Intelligence 2019 49:12*, *49*(12), 4335–4347. https://doi.org/10.1007/S10489-019-01510-8

Koenig, S., & Simmons, R. G. (1996). The Effect of Representation and Knowledge on Goal-Directed Exploration with Reinforcement-Learning Algorithms. *Machine Learning*, *22*(1/2/3), 227–250. https://doi.org/10.1023/A:1018068507504

Kurutach, T., Clavera, I., Duan, Y., Tamar, A., & Abbeel, P. (2018). Model-Ensemble Trust-Region Policy Optimization. *6th International Conference on Learning Representations*. https://openreview.net/forum?id=SJJinbWRZ

Kusy, M., & Zajdel, R. (2014). Probabilistic neural network training procedure based on Q(0)-learning algorithm in medical data classification. *Applied Intelligence 2014 41:3*, *41*(3), 837–854. https://doi.org/10.1007/S10489-014-0562-9

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings, IEEE*, *86*(11), 2278–2323. https://doi.org/10.1109/5.726791

Lee, J. (2020). *Introduction: The Development and Application of AI Technology* (1st ed.). Springer Singapore. https://doi.org/10.1007/978-981-15-2144-7_1

Legg, S., & Hutter, M. (2007). Universal Intelligence: A Definition of Machine Intelligence. *Minds and Machines 2007 17:4*, *17*(4), 391–444. https://doi.org/10.1007/S11023-007-9079-X

Levine, S., Finn, C., Darrell, T., & Abbeel, P. (2016a). End-to-End Training of Deep Visuomotor Policies. *Journal of Machine Learning Research*, *17*(1), 1334–1373.

Levine, S., Finn, C., Darrell, T., & Abbeel, P. (2016b). End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, *17*(1), 1334–1373. http://www.jmlr.org/papers/volume17/15-522/15-522.pdf

Liang, X., Wang, Q., Feng, Y., Liu, Z., & Huang, J. (2018). VMAV-C: A Deep Attention-based Reinforcement Learning Algorithm for Model-based Control. *ArXiv e-prints*.

Lin, Z., Gehring, J., Khalidov, V., & Synnaeve, G. (2017). STARDATA: A StarCraft AI Research Dataset. *Proceedings, The Thirteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. http://arxiv.org/abs/1708.02139

Littman, M. L., Cassandra, A. R., & Kaelbling, L. P. (1995). Learning policies for partially observable environments: Scaling up. *Machine Learning Proceedings 1995*, 362–370. https://doi.org/10.1016/B978-1-55860-377-6.50052-9

Loshchilov, I., & Hutter, F. (2019). Decoupled Weight Decay Regularization. *Proc. 7th International Conference on Learning Representations, ICLR'19*, 1–19. https://openreview.net/forum?id=Bkg6RiCqY7

Lu, Y. (2019). Artificial intelligence: a survey on evolution, models, applications and future trends. *https://doi.org/10.1080/23270012.2019.1570365*, *6*(1), 1–29. https://doi.org/10.1080/23270012.2019.1570365

Luo, Y., Xu, H., Li, Y., Tian, Y., Darrell, T., & Ma, T. (2018). Algorithmic Framework for Model-based Reinforcement Learning with Theoretical Guarantees. *Proceedings, 8th International Conference on Learning Representations (ICLR)*. https://openreview.net/forum?id=BJe1E2R5KX

Mahmud, M., Kaiser, M. S., Hussain, A., & Vassanelli, S. (2018). Applications of Deep Learning and Reinforcement Learning to Biological Data. *IEEE Transactions on Neural Networks and Learning Systems*, *29*(6), 2063–2079. https://doi.org/10.1109/TNNLS.2018.2790388

Makhzani, A., Shlens, J., Jaitly, N., Goodfellow, I., & Frey, B. (2015). Adversarial Autoencoders. http://arxiv.org/abs/1511.05644

Matignon, L., Laurent, G. J., & Le Fort-Piat, N. (2006). Reward function and initial values: Better choices for accelerated goal-directed reinforcement learning. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *4131 LNCS*, 840–849. https://doi.org/10.1007/11840817_87

McGee, L. A., & Schmidt, S. F. (1985). Discovery of the Kalman Filter as a Practical Tool for Aerospace and Industry. *NASA Technical Memorandum*, (November).

Mendel, O., & Bergström, J. (2019). SIMD Optimizations of Software Rendering in 2D Video Games.

Mihatsch, O., & Neuneier, R. (2002). Risk-sensitive reinforcement learning. *Machine learning*, *49*(2), 267–290.

Mileff, P., & Dudra, J. (2012). Efficient 2D software rendering. *Production Systems and Information Engineering*, *6*(2), 55–66.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., & Kavukcuoglu, K. (2016). Asynchronous Methods for Deep Reinforcement Learning. In M. F. Balcan & K. Q. Weinberger (Eds.), *Proceedings of the 33rd international conference on machine learning* (pp. 1928–1937). PMLR. http://proceedings.mlr.press/v48/mniha16.html

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533. https://doi.org/10.1038/nature14236

Moerland, T. M., Broekens, J., & Jonker, C. M. (2020). Model-based Reinforcement Learning: A Survey. *arxiv preprint arXiv:2006.16712*. https://arxiv.org/abs/2006.16712

Moldovan, T. M., & Abbeel, P. (2012). Safe Exploration in Markov Decision Processes. *Proceedings of the 29th International Conference on Machine Learning*.

Monahan, G. E. (1982). State of the Art—A Survey of Partially Observable Markov Decision Processes: Theory, Models, and Algorithms. *Management Science*, *28*(1), 1–16. https://doi.org/10.1287/MNSC.28.1.1

Nair, V., & Hinton, G. E. (2010). Rectified Linear Units Improve Restricted Boltzmann Machines. *Proceedings of the 27 th International Conference on Machine Learning*, 8.

Ontanon, S. (2013). The combinatorial multi-armed bandit problem and its application to real-time strategy games. *Proceedings, The Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 58–64. http://www.aaai.org/ocs/index.php/AIIDE/AIIDE13/paper/viewPaper/7377

Ontanon, S., Mishra, K., Sugandh, N., & Ram, A. (2008). Learning from demonstration and case-based planning for real-time strategy games. In *Studies in fuzziness and soft computing* (pp. 293–310). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-77465-5_15

Ontanon, S., Synnaeve, G., Uriarte, A., Richoux, F., Churchill, D., & Preuss, M. (2013). A survey of real-time strategy game AI research and competition in starcraft. *IEEE Transactions on Computational Intelligence and AI in Games*, *5*(4), 293–311. https://doi.org/10.1109/TCIAIG.2013.2286295

Ozair, S., Li, Y., Razavi, A., Antonoglou, I., van den Oord, A., & Vinyals, O. (2021). Vector Quantized Models for Planning. *Proc.. 38th International Conference on Machine Learning, PMLR 139, 2021*, 1–15. http://arxiv.org/abs/2106.04615

Pang, Z.-J., Liu, R.-Z., Meng, Z.-Y., Zhang, Y., Yu, Y., & Lu, T. (2019). On Reinforcement Learning for Full-Length Game of StarCraft. *Proceedings of the AAAI Conference on Artificial Intelligence*, *33*(01), 4691–4698. https://doi.org/10.1609/AAAI.V33I01.33014691

Pathak, D., Agrawal, P., Efros, A. A., & Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. *Proc. 34th International Conference on Machine Learning, ICML'17*, *70*, 2778–2787. https://dl.acm.org/doi/10.5555/3305890.3305968

Peng, P., Wen, Y., Yang, Y., Yuan, Q., Tang, Z., Long, H., & Wang, J. (2017). Multiagent Bidirectionally-Coordinated Nets: Emergence of Human-level Coordination in Learning to Play StarCraft Combat Games. *arxiv preprint arXiv:1703.10069*. https://doi.org/10.1007/11575726_13

Peres, R. S., Jia, X., Lee, J., Sun, K., Colombo, A. W., & Barata, J. (2020). Industrial Artificial Intelligence in Industry 4.0 -Systematic Review, Challenges and Outlook. *IEEE Access*. https://doi.org/10.1109/ACCESS.2020.3042874

Polydoros, A. S., & Nalpantidis, L. (2017). Survey of Model-Based Reinforcement Learning: Applications on Robotics. *Journal of Intelligent & Robotic Systems 2017 86:2*, *86*(2), 153–173. https://doi.org/10.1007/S10846-017-0468-Y

Ponsen, M., Lee-Urban, S., Muñoz-Avila, H., Aha, D., & Molineaux, M. (2005). Stratagus: An open-source game engine for research in real-time strategy games. *Reasoning, Representation, and Learning in Computer Games*, (Code 5515), 78. https://pdfs.semanticscholar.org/d005/87160d3b37c70f238ebd92c71454479e829e.pdf

Pu, Y., Gan, Z., Henao, R., Yuan, X., Li, C., Stevens, A., & Carin, L. (2016). Variational Autoencoder for Deep Learning of Images, Labels and Captions. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, & G. R. (Eds.), *Advances in neural information processing systems* (pp. 2352–2360). Curran Associates, Inc. http://papers.nips.cc/paper/6528-variational-autoencoder-for-deep-learning-of-images-labels-and-captions.pdf

Puiutta, E., & Veith, E. M. (2020). Explainable Reinforcement Learning: A Survey. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *12279 LNCS*, 77–95. https://doi.org/10.1007/978-3-030-57321-8_5

Puterman, M. L. (1990). Chapter 8 Markov decision processes. *Handbooks in Operations Research and Management Science*, *2*(100), 331–434. https://doi.org/10.1016/S0927-0507(05)80172-0

Razavi, A., van den Oord, A., Poole, B., & Vinyals, O. (2019). Preventing Posterior Collapse with delta-VAEs. *Proc. 7th International Conference on Learning Representations, ICLR'19*, 1–24. https://openreview.net/forum?id=BJe0Gn0cY7

Razavi, A., van den Oord, A., & Vinyals, O. (2019). Generating Diverse High-Fidelity Images with VQ-VAE-2. In H. Wallach, H. Larochelle, A. Beygelzimer, F. Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems 32* (pp. 14837–14847). Curran Associates, Inc. http://papers.nips.cc/paper/9625-generating-diverse-high-fidelity-images-with-vq-vae-2

Rodriguez, A., Parr, R., & Koller, D. (2000). Reinforcement Learning Using Approximate Belief States. In S. Solla, T. Leen, & K. Müller (Eds.), *Advances in neural information processing systems* (pp. 1036–1042). MIT Press. https://proceedings.neurips.cc/paper/1999/file/158fc2ddd52ec2cf54d3c161f2dd6517-Paper.pdf

Romoff, J., Henderson, P., Piche, A., Francois-Lavet, V., & Pineau, J. (2018). Reward Estimation for Variance Reduction in Deep Reinforcement Learning. In A. Billard, A. Dragan, J. Peters, & J. Morimoto (Eds.), *Proceedings of the 2nd conference* *on robot learning* (pp. 674–699). PMLR. https://proceedings.mlr.press/v87/romoff18a.html

Roodbergen, K. J., & Vis, I. F. A. (2009). A survey of literature on automated storage and retrieval systems. *European Journal of Operational Research*. https://doi.org/10.1016/j.ejor.2008.01.038

Rybkin, O., Daniilidis, K., & Levine, S. (2021). Simple and Effective VAE Training with Calibrated Decoders. *Proceedings of the 38th International Conference on Machine Learning*.

Santos, M. S., & Rust, J. (2004). Convergence Properties of Policy Iteration. *SIAM Journal on Control and Optimization*, *42*(6), 2094–2115. https://doi.org/10.1137/S0363012902399824

Saunders, W., Sastry, G., Stuhlmüller, A., & Evans, O. (2018). Trial without Error: Towards Safe Reinforcement Learning via Human Intervention. *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, 2067–2069.

Schmidhuber, J. (2010). Formal theory of creativity, fun, and intrinsic motivation (1990-2010). *IEEE Transactions on Autonomous Mental Development*, *2*(3), 230–247. https://doi.org/10.1109/TAMD.2010.2056368

Schön, T. B., Wills, A., & Ninness, B. (2011). System identification of nonlinear state-space models. *Automatica*, *47*(1), 39–49. https://doi.org/10.1016/J.AUTOMATICA.2010.10.013

Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015). Trust Region Policy Optimization. In F. Bach & D. Blei (Eds.), *Proceedings of the 32nd international conference on machine learning* (pp. 1889–1897). PMLR. http://proceedings.mlr.press/v37/schulman15.html

Schulman, J., Moritz, P., Levine, S., Jordan, M. I., & Abbeel, P. (2016). High-dimensional continuous control using generalized advantage estimation. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 1–14.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. *arxiv preprint arXiv:1707.06347*. http://arxiv.org/abs/1707.06347

Seetharaman, P., Wichern, G., Pardo, B., & Roux, J. L. (2020). Autoclip: Adaptive gradient clipping for source separation networks. *IEEE International Workshop on Machine Learning for Signal Processing, MLSP*, *2020-Septe*, 1–6. https://doi.org/10.1109/MLSP49062.2020.9231926

Sethy, H., Patel, A., & Padmanabhan, V. (2015). Real Time Strategy Games: A Reinforcement Learning Approach. *Procedia Computer Science*, *54*, 257–264. https://doi.org/10.1016/J.PROCS.2015.06.030

Sharma, J., Andersen, P.-A., Granmo, O.-C., & Goodwin, M. (2020). Deep Q-Learning With Q-Matrix Transfer Learning for Novel Fire Evacuation Environment. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*. https://doi.org/10.1109/tsmc.2020.2967936

Shen, Y., Tobia, M. J., Sommer, T., & Obermayer, K. (2013). Risk-sensitive Reinforcement Learning. *Neural Computation*, *26*(7), 1298–1328. https://doi.org/10.1162/NECO_a_00600

Shleyfman, A., Komenda, A., & Domshlak, C. (2014). On combinatorial actions and CMABs with linear side information. *Frontiers in Artificial Intelligence and Applications*, *263*, 825–830. https://doi.org/10.3233/978-1-61499-419-0-825

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, *529*(7587), 484–489. https://doi.org/10.1038/nature16961

Silver, D., Singh, S., Precup, D., & Sutton, R. S. (2021). Reward is enough. *Artificial Intelligence*, *299*, 103535. https://doi.org/10.1016/J.ARTINT.2021.103535

Smirnov, Y., Koenig, S., Veloso, M. M., & Simmons, R. G. (1996). Efficient goal-directed exploration. *Proceedings of the National Conference on Artificial Intelligence*, *1*.

Smith, L. N. (2015). Cyclical Learning Rates for Training Neural Networks. http://arxiv.org/abs/1506.01186

Starke, S., Zhao, Y., Zinno, F., & Komura, T. (2021). Neural Animation Layering for Synthesizing Martial Arts Movements. *ACM Transactions on Graphics*, *40*(4). https://doi.org/10.1145/3450626.3459881

Sutton, R. S. (1990). Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming. *Machine Learning Proceedings 1990*, 216–224. https://doi.org/10.1016/B978-1-55860-141-3.50030-4

Sutton, R. S. (2019). The Bitter Lesson. Retrieved June 29, 2019, from http://www.incompleteideas.net/IncIdeas/BitterLesson.html

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (2nd ed.). A Bradford Book. https://dl.acm.org/doi/book/10.5555/3312046

Sutton, R. S., Precup, D., & Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, *112*(1-2), 181–211.

Synnaeve, G., Nardelli, N., Auvolat, A., Chintala, S., Lacroix, T., Lin, Z., Richoux, F., & Usunier, N. (2016). TorchCraft: a Library for Machine Learning Research on Real-Time Strategy Games. *arxiv preprint arXiv:1611.00625*. http://arxiv.org/abs/1611.00625

Tang, Y., & Kucukelbir, A. (2017). Variational Deep Q Network. *Advances in Neural Information Processing Systems 30*. http://arxiv.org/abs/1711.11225

Thompson, N. C., Greenewald, K., Lee, K., & Manso, G. F. (2020). The Computational Limits of Deep Learning. https://arxiv.org/abs/2007.05558v1

Tian, Y., Gong, Q., Shang, W., Wu, Y., & Zitnick, C. L. (2017). ELF: An Extensive, Lightweight and Flexible Research Platform for Real-time Strategy Games. *Advances in Neural Information Processing Systems*, 2656–2666. http://arxiv.org/abs/1707.01067

Tijsma, A., Drugan, M., & Wiering, M. (2017). Comparing exploration strategies for Q-learning in random stochastic mazes. *2016 IEEE Symposium Series on Computational Intelligence, SSCI 2016*. https://doi.org/10.1109/SSCI.2016.7849366

Turchetta, M., Kolobov, A., Shah, S., Krause, A., & Agarwal, A. (2020). Safe Reinforcement Learning via Curriculum Induction. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, & H. Lin (Eds.), *Advances in neural information processing systems* (pp. 12151–12162). Curran Associates, Inc. https://proceedings.neurips.cc/paper/2020/file/8df6a65941e4c9da40a4fb899de65c55-Paper.pdf

Van Den Oord, A., Vinyals, O., & Kavukcuoglu, K. (2017). Neural discrete representation learning. *Advances in Neural Information Processing Systems*, *2017-Decem*, 1–11.

Van Otterlo, M., & Wiering, M. (2012). Reinforcement learning and markov decision processes. *Adaptation, Learning, and Optimization*, *12*, 3–42. https://doi.org/10.1007/978-3-642-27645-3_1

van Hasselt, H., Guez, A., & Silver, D. (2015). Deep Reinforcement Learning with Double Q-learning. *Proceedings, The Thirtieth AAAI Conference on Artificial Intelligence*, 13. http://arxiv.org/abs/1509.06461

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., . . . Silver, D. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature 2019 575:7782*, *575*(7782), 350–354. https://doi.org/10.1038/s41586-019-1724-z

Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., Makhzani, A., Küttler, H., Agapiou, J., Schrittwieser, J., Quan, J., Gaffney, S., Petersen, S., Simonyan, K., Schaul, T., van Hasselt, H., Silver, D., Lillicrap, T., Calderone, K., . . . Tsing, R. (2017). StarCraft II: A New Challenge for Reinforcement Learning. *Proceedings, The Thirteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 86–92. https://doi.org/https://deepmind.com/documents/110/sc2le.pdf

Walsh, T. J., Goschin, S., & Littman, M. L. (2010). Integrating Sample-Based Planning and Model-Based Reinforcement Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, *24*(1), 612–617. https://ojs.aaai.org/index.php/AAAI/article/view/7689

Wang, J., Gou, L., Shen, H. W., & Yang, H. (2019). DQNViz: A Visual Analytics Approach to Understand Deep Q-Networks. *IEEE Transactions on Visualization and Computer Graphics*, *25*(1), 288–298. https://doi.org/10.1109/TVCG.2018.2864504

Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning 1992 8:3*, *8*(3), 279–292. https://doi.org/10.1007/BF00992698

Wolpert, D. H., Kolchinsky, A., & Owen, J. A. (2019). A space–time tradeoff for implementing a function with master equation dynamics. *Nature Communications 2019 10:1*, *10*(1), 1–9. https://doi.org/10.1038/s41467-019-09542-x

Xiang, X., & Foo, S. (2021). Recent Advances in Deep Reinforcement Learning Applications for Solving Partially Observable Markov Decision Processes (POMDP) Problems: Part 1—Fundamentals and Applications in Games, Robotics and Natural Language Processing. *Machine Learning and Knowledge Extraction 2021, Vol. 3, Pages 554-581*, *3*(3), 554–581. https://doi.org/10.3390/MAKE3030029

Xiao, T., & Kesineni, G. (2016). *Generative Adversarial Networks for Model Based Reinforcement Learning with Tree Search* (tech. rep.). University of California. Berkeley. http://tedxiao.me/pdf/gans%7B%5C_%7Ddrl.pdf

Yang, J., & Qiu, W. (2005a). A measure of risk and a decision-making model based on expected utility and entropy. *European Journal of Operational Research*, 792–799. https://doi.org/10.1016/j.ejor.2004.01.031

Yang, J., & Qiu, W. (2005b). A measure of risk and a decision-making model based on expected utility and entropy. *European Journal of Operational Research*, *164*(3), 792–799. https://doi.org/10.1016/J.EJOR.2004.01.031

Younes, H. L. S., & Simmons, R. G. (2004). Solving Generalized Semi-Markov Decision Processes using Continuous Phase-Type Distributions. *Proceedings, The Ninth AAAI Conference on Artificial Intelligence*. www.aaai.org

Zahavy, T., Ben-Zrihem, N., & Mannor, S. (2016). Graying the black box: Understanding DQNs. In M. F. Balcan & K. Q. Weinberger (Eds.), *Proceedings of the 33rd international conference on machine learning* (pp. 1899–1908). PMLR. https://proceedings.mlr.press/v48/zahavy16.html

Zha, Z., Wang, B., & Tang, X. (2021). Evaluate, explain, and explore the state more exactly: an improved Actor-Critic algorithm for complex environment. *Neural Computing and Applications 2021*, 1–12. https://doi.org/10.1007/S00521-020-05663-3

Zhang, C., Patras, P., & Haddadi, H. (2019). Deep Learning in Mobile and Wireless Networking: A Survey. *IEEE Communications Surveys & Tutorials*, *21*(3), 2224–2287. https://doi.org/10.1109/COMST.2019.2904897

Zhang, C., Butepage, J., Kjellstrom, H., & Mandt, S. (2019). Advances in Variational Inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *41*(8), 2008–2026. https://doi.org/10.1109/TPAMI.2018.2889774

Zheng, Y., Tan, H., Tang, B., Zhou, H., Jiang, Z., Zheng, Y., Tan, H., Tang, B., & Zhou, H. (2017). Variational Deep Embedding: A Generative Approach to Clustering. *International Joint Conference on Artificial Intelligence 17*, 8. http://arxiv.org/abs/1611.05148

PART II

Paper A

Title:	FlashRL: A Reinforcement Learning Platform for Flash Games
Authors:	Andersen, Per-Arne et al.
Affiliation:	Department of ICT, University of Agder, Grimstad Norway
Conference:	NIK-2017 - Norwegian Information Conference 2017
Year:	2017

A

FlashRL: A Reinforcement Learning Platform for Flash Games

Per-Arne Andersen
Department of ICT
University of Agder
Grimstad, Norway
per.andersen@uia.no

Morten Goodwin
Department of ICT
University of Agder
Grimstad, Norway
morten.goodwin@uia.no

Ole-Christoffer Granmo
Department of ICT
University of Agder
Grimstad, Norway
ole.granmon@uia.no

Abstract

Reinforcement Learning (RL) is a research area that has blossomed tremendously in recent years and has shown remarkable potential in among others successfully playing computer games. However, there only exists a few game platforms that provide diversity in tasks and state-space needed to advance RL algorithms. The existing platforms offer RL access to Atari- and a few web-based games, but no platform fully expose access to Flash games. This is unfortunate because applying RL to Flash games have potential to push the research of RL algorithms.

This paper introduces the Flash Reinforcement Learning platform (FlashRL) which attempts to fill this gap by providing an environment for thousands of Flash games on a novel platform for Flash automation. It opens up easy experimentation with RL algorithms for Flash games, which has previously been challenging. The platform shows excellent performance with as little as 5% CPU utilization on consumer hardware. It shows promising results for novel reinforcement learning algorithms.

Norwegian ICT conference for research and education, Oslo, Norway
Submitted 28. Aug, 2017, Accepted 20. Oct 2017

1 Introduction

There are several challenges related to developing algorithms that can interact with human-level performance in real-world environments, such as computer games. Researchers often use toy experiments when working with *Reinforcement Learning* (RL), because it is easier, cheaper and consumes less time to orchestrate. With several applications for RL in daily life, it has become an essential field of research [11, 3]. However, existing learning platforms for games have major limitations such as few game environments and little environment control.

OpenAI is a non-profit company that is currently one of the leading researchers of RL. OpenAI *Universe* is a software platform that has several game environments aimed at artificial research. The problem with this software is that individual developers are not directly permitted to supplement new environments to the repository, and there is little documentation on how to contribute to new environments. *FlashRL* changes this with our proposed architecture as the control is given back to each researcher.

Adobe Flash is a multimedia software platform used for the production of applications and animation. The Flash run-time was recently declared deprecated by Adobe, and by 2020, no longer supported. Flash is still frequently used in web applications, and there are several thousand games created for this platform. Several browsers have removed support for Flash, making it impossible to access the mentioned game environments. Games have proven to be an excellent area of machine learning benchmarking, due to size and diversity of its state-space. It is therefore essential to preserve Flash as an environment for reinforcement learning.

Automating Flash applications is a relatively untouched area. The technology has been succeeded by several better options for web development, for example, HTML5. This makes it hard for algorithms to control Flash environments programmatically. There are already reinforcement learning platforms that support Flash games as part of their game library, but these use browsers to execute the Flash run-time.

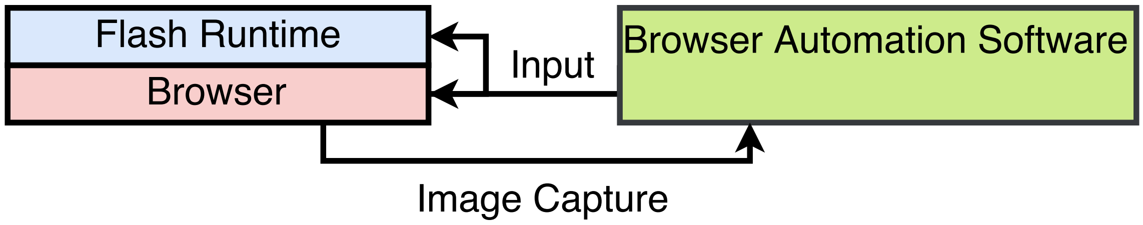


Figure 1: Interacting with Flash through browser automating

Figure 1 illustrates how interaction with the Flash environment would typically be carried out through browser automation software such as *Selenium*. *Selenium* can automate most modern browsers. It does not directly support Flash automation, but can easily be used for this purpose with minimal customisation [4]. With the loss of browser support, the difficulty of controlling Flash applications increases, and there is a significant risk that excellent game environments for reinforcement learning are lost.

FlashRL is unique for reinforcement learning as it allows researchers to use any desired Flash environment. It gives full control of the game environment and is not based on running Flash applications in the browser.

FlashRL is targeted research in reinforcement learning, but can also be used in other machine learning algorithms. It supports all kinds of Flash applications but is primarily used for agent-based gameplay. Several thousand game environments are included in the first release of the software[1]. Multitask 2 is a Flash game that is excellent for reinforcement learning as it requires the agent to perform several tasks simultaneously. We show in this paper that our learning platform can be used to train novel reinforcement algorithms without any customisation.

In Section 2, we discuss related work for existing learning platforms in machine learning. We also argue why web browsers are no longer viable as Flash run-time. Section 3 briefly outline what reinforcement learning is and explains how Q-Learning works. Section 4 outlines the proposed platform and thoroughly describe its underlying architecture. In Section 5 we show initial results of utilizing the proposed learning platform for reinforcement learning. At Section 6 summarises the work and argue why the proposed learning platform is used for reinforcement learning research. Section 7 outlines a road-map for further development of the platform.

2 Related Work

With the increasing popularity in RL, there is a need for flexible learning platforms. Several learning platforms exist that can run a limited number of games, but no platform that features an open-source interface with possibility to run *any* Flash game.

Bellemare et al. provided in 2012 a learning platform *Arcade Learning Environment* (ALE) that enabled scientists to conduct edge research in general deep learning [1]. The package provided hundreds of Atari 2600 environments that in 2013 allowed Minh et al. to do a breakthrough with Deep Q-Learning and A3C. The platform has been a key component in several breakthroughs in RL research. [8, 9, 7]

In 2016, Brockman et al. from OpenAI released GYM which they referred to as *"a toolkit for developing and comparing reinforcement learning algorithms"* [2]. GYM provides various types of environments from following technologies [2]: Algorithmic tasks, Atari 2600, Board games, Box2d physics engine, MuJoCo physics engine, and Text-based en-

[1]Author of this paper takes no credit for any game environments

vironments. OpenAI also hosts a website where researchers can submit their performance for comparison between algorithms. GYM is open-source and encourages researchers to add support for their environments.

OpenAI recently released a new learning platform called *Universe*. This environment further adds support for environments running inside VNC. It also supports running Flash games and browser applications. However, despite OpenAI's open-source policy, they do not allow researchers to add new environments to the repository. This limits the possibilities of running any environment. Universe is, however, a significant learning platform as it also has support for desktop games like Grand Theft Auto IV, that allow for research in autonomous driving [6].

Selenium is a software for automating web browsers and is used primarily for unit-testing of web content. There were some efforts to create a version that allowed to interact with Flash content, but it was quickly abandoned. There is limited support for interacting with Flash, by selecting the DOM-Element in HTML and sending key-presses via Javascript. Several learning platforms utilize this method, but due to the deprecation of Flash in browsers, it is no longer a viable option.

3 Reinforcement Learning

Reinforcement learning can be considered hybrid between supervised and unsupervised learning. We implement what we call an agent that acts in our environment. This agent is placed in the unknown environment where it tries to maximize the environmental reward [12].

Markov Decision Process (MDP) is a mathematical method of modeling decision-making within an environment. We often use this technique when utilizing model-based RL algorithms. In *Q-Learning*, we do not try to model the MDP. Instead, we try to learn the optimal policy by estimating the action-value function $Q^*(s,a)$, yielding maximum expected reward in state s executing action a. The optimal policy can then be found by

$$\pi(s) = argmax_a Q^*(s,a) \tag{1}$$

This is derived from *Bellman's Equation*, because we can consider $U(s) = max_a Q(s,a)$, the utility function to be true. This gives us the ability to derive following update-rule equation from Bellman's work:

$$Q(s,a) \leftarrow Q(s,a) + \underbrace{\alpha}_{\text{LR}} \left(\underbrace{R(s)}_{\text{Reward}} + \underbrace{\gamma}_{\text{Discount}} \underbrace{max_{a'} Q(s^{'},a^{'})}_{\text{New Q}} - \underbrace{Q(s,a)}_{\text{Old Q}} \right) \tag{2}$$

This is an iterative process of propagating back the estimated Q-value for each discrete time-step in the environment. It is guaranteed to converge towards the optimal action-value function, $Q_i \rightarrow Q^*$ as i $\rightarrow \infty$ [12, 8]. At the most basic level, Q-Learning utilize a table for storing $(s, a, r, s^{'})$ pairs. But we can instead use a non-linear function approximation in order to approximate $Q(s, a; \theta)$. θ describes tunable parameters for approximator. Artificial Neural Networks (ANN) are a popular function approximator, but training using ANN is relatively unstable.

4 Flash Reinforcement Learning (FlashRL)

The proposed platform is an interface that acts as a bridge between the *Gnash Flash player* and the reinforcement learning algorithms. *Flash Reinforcement Learning* (FlashRL) is a new platform that allows researchers to run algorithms on any Flash-based game efficiently.

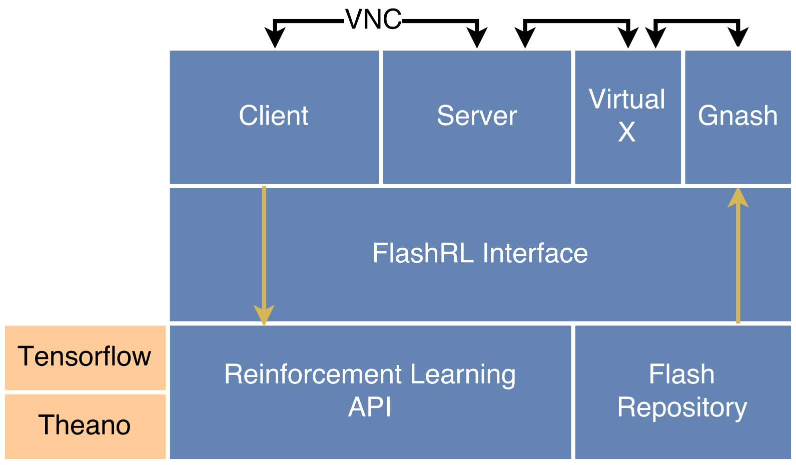


Figure 2: FlashRL Architecture Overview

The learning platform is developed primarily for the operating system Linux but is likely to run on Cygwin with few modifications. There are several key components that FlashRL uses to operate adequate, see Figure 2. It uses a Linux library called XVFB to create a virtual frame-buffer that is used for graphics rendering [5]. Inside this frame-buffer, a Flash game chosen by the researcher is executed by a third party flash player, for example, *Gnash*. A VNC server serves the XVFB frame-buffer and allows FlashRL to access it by utilizing a VNC Client. The VNC Client can then issue commands like keyboard presses and mouse movements. The VNC Client *pyVLC* was specially made for this learning platform. The code base originates from python-vnc-viewer [13]. The last component of FlashRL is the Reinforcement Learning API that allows the developer to access the input/output of the VNC client. This makes it easy to develop sequenced algorithms by using the API callbacks or manually by threading.

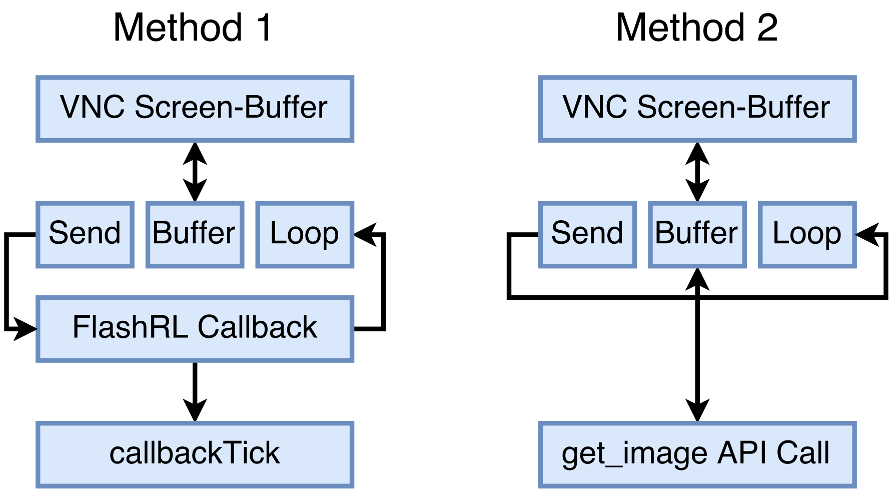


Figure 3: Frame-buffer Access Methods

Figure 3 illustrates two methods of accessing the frame-buffer from the Flash Game. Both approaches are sufficient to perform reinforcement learning, but each has its strength and weaknesses. Method 1, seen in Figure 3 allows the developer to get frames served at a fixed rate, for example, 60 frames per second. Method 2 does not restrict the frequency of how fast the frame-buffer is captured. This is preferable for developers that do not require images from fixed time-steps as it requires less processing power per frame. The framework was developed with deep learning in mind and is proven to work with Keras and Tensorflow.

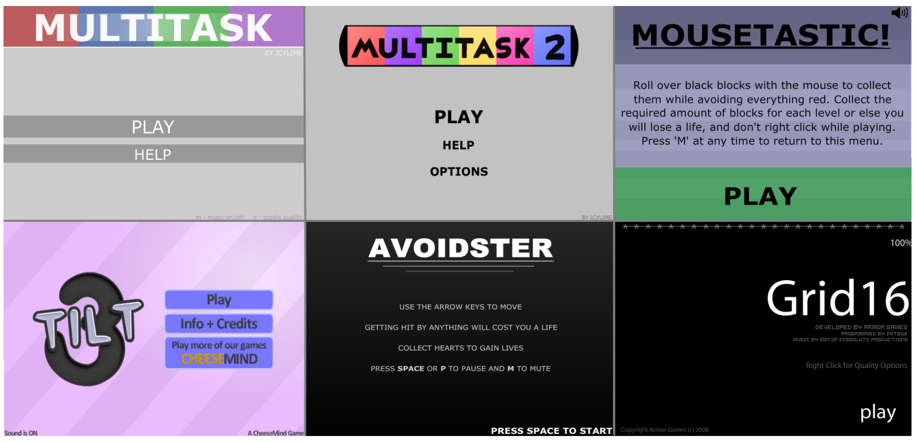

MULTITASK
PLAY
HELP
MULTITASK 2
PLAY
HELP
OPTIONS
MOUSETASTIC!
Roll over black blocks with the mouse to collect them while avoiding everything red. Collect the required amount of blocks for each level or else you will lose a life, and don't right click while playing. Press 'M' at any time to return to this menu.
PLAY
TILT
Play
Info + Credits
Play more of our games CHEESEMIND
Sound is ON
A CheeseMind Game
AVOIDSTER
USE THE ARROW KEYS TO MOVE
GETTING HIT BY ANYTHING WILL COST YOU A LIFE
COLLECT HEARTS TO GAIN LIVES
PRESS SPACE OR P TO PAUSE AND M TO MUTE
PRESS SPACE TO START
Grid16
100%
Right Click for Quality Options
play

Figure 4: Selected environments from the FlashRL game repository

Several thousand game environments are shipped with the initial version of FlashRL. These game environments were gathered from different sources on the web. FlashRL has a relatively small code-base and to preserve this size, all of the Flash games are hosted remotely. The quality varies, and some of the games are not tested or labeled. Most games are however tested and can be played without issues, see Figure 4.

5 Experiments

This section presents experiments of reinforcement learning algorithms applied in FlashRL. We use the game Multitask 2 [2] to test the learning platform. Multitask 2 was chosen because it challenges the algorithm to master four different mini-games simultaneously.

The experiments are grouped in two. The first experiment determines the hardware requirements of the platform and benchmarks the speed of critical operations. The second experiment is an implementation of standard Deep Q-Learning trained on raw state images from Multitask 2 to perform game actions. The latter is meant as a proof of concept that RL algorithms can be applied in FlashRL.

All experiments were conducted on Ubuntu Linux 17.04 x64 running Python 3.5.3. The machine has 64GB memory, Nvidia GeForce 1080TI, and Intel I7-7770k as hardware.

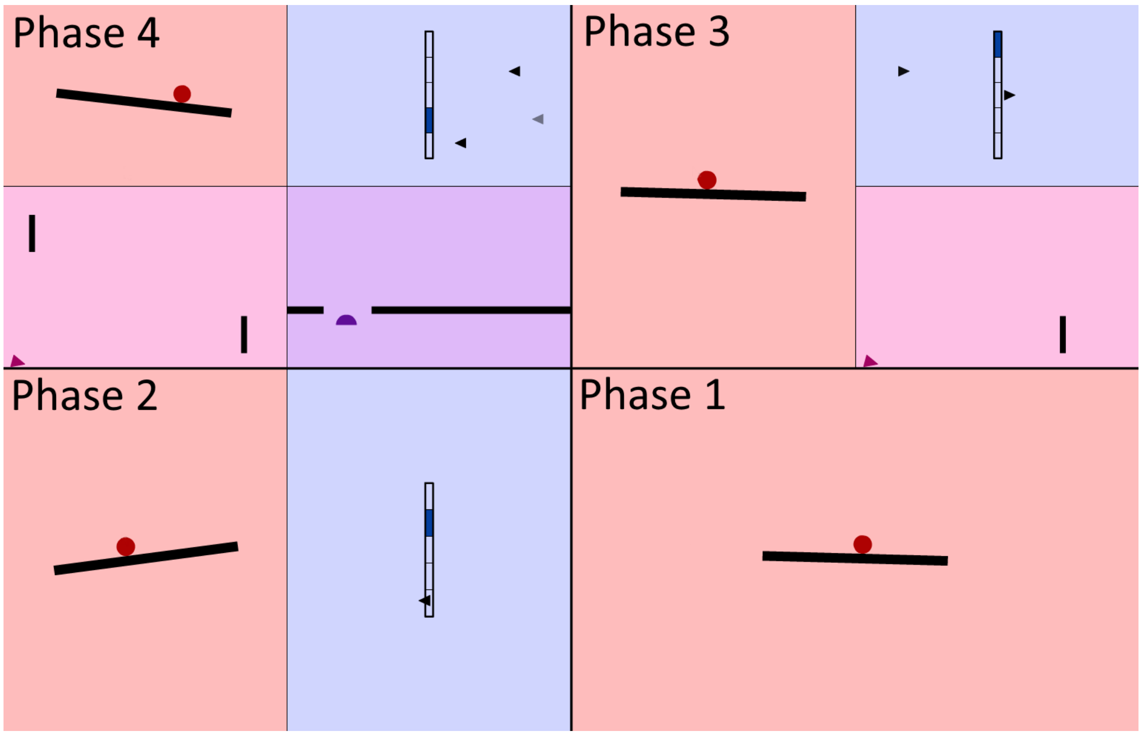


Figure 5: In-game footage of the game Multitask

[2]Multitask 2 - http://multitaskgames.com/multitask-2.html

5.1 Multitask 2

Figure 5 illustrates the game-play of Multitask 2. The game is split into four-game phases. The first phase (lower right corner in Figure 5) is a single paddle that the player must balance a ball on. In state two (lower left corner in Figure 5) , the player must control the second paddle to avoid arrows traveling towards it. The third phase (upper right corner in Figure 5) consist of an arrow with mechanics relatable to the game Flappy Bird [10]. In the final phase (upper left corner in Figure 5), the player must additionally jump over holes on the ground. For the player to succeed the game, he must control eight actions simultaneously. The score is calculated by adding a single point for each second survived in the game.

5.2 Experiment 1: Hardware Requirements

Recall from section 4 that there are two methods of accessing the frame-buffer. The first method (Method 1) is based on retrieving the frame-buffer at fixed time intervals. The second method (Method 2) does not have any interval restriction. This makes Method 2 faster because it does not require sleep between frames. This causes the framework to consume all available CPU, which is not always preferable.

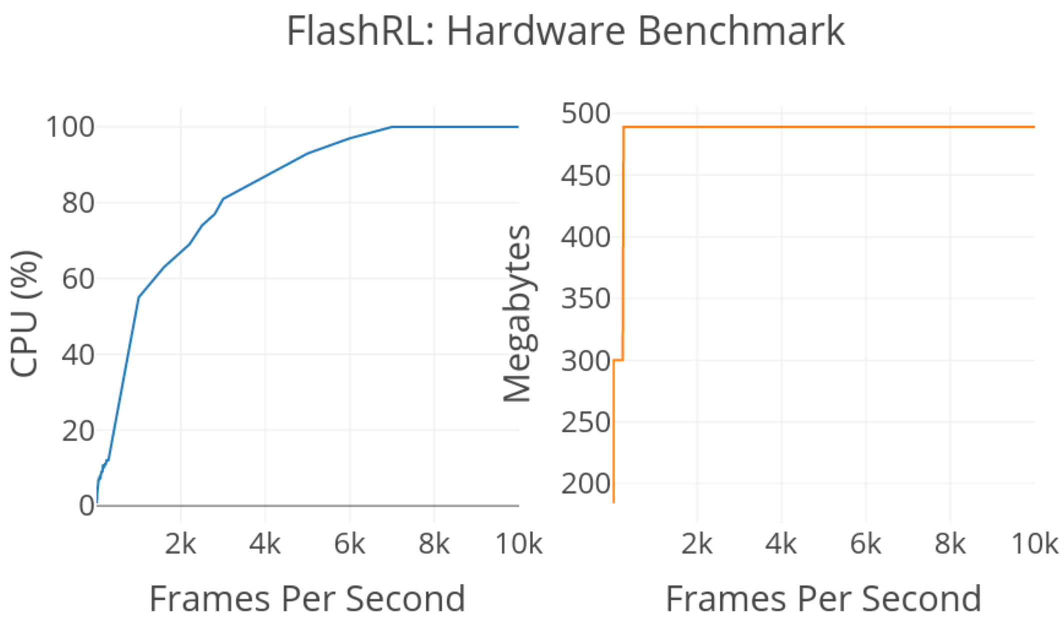


Figure 6: Hardware benchmark

We can see from Figure 6 that using Method 1 with the interval set to 30 fps uses approximately 5% of the CPU. Increasing the interval to 300 increases it to 13%. We gradually increased the interval until the CPU ran at maximum. A single I7-7700k can compute approximately 6300 fps images from the frame-buffer before struggling to keep up.

The GPU Did not recognize any load during these test because the Flash environment is software rendered. Memory consumed were between 200MB and 500MB depending on the speed. We believe that the reason for memory increase is that Python does not garbage collect old frame-buffer snapshots between iterations, and therefore gets an increased memory load.

5.3 Experiment 2: Reinforcement Learning

Deep Q-Network (DQN) is a novel algorithm architecture developed by Minh et al. at Google DeepMind. It combines Q-Learning estimating Q-Values from a neural network. [8]

In our tests we used Double Q-Learning from Hasselt et al. [14]. We also used Dueling from Wang et al. that increases the learning precision by using two estimators: state-value and action-advantage function [15]. We used a discount factor of 0.99, learning rate of 0.001 and mini-batch of 16. We used exploration/exploitation strategy with ϵ-greedy where it started at 0.9 and finished at 0.1. The ϵ annealing was set to 10 000 steps. This is a relatively low epsilon phase. But it seemed to work well in this environment.

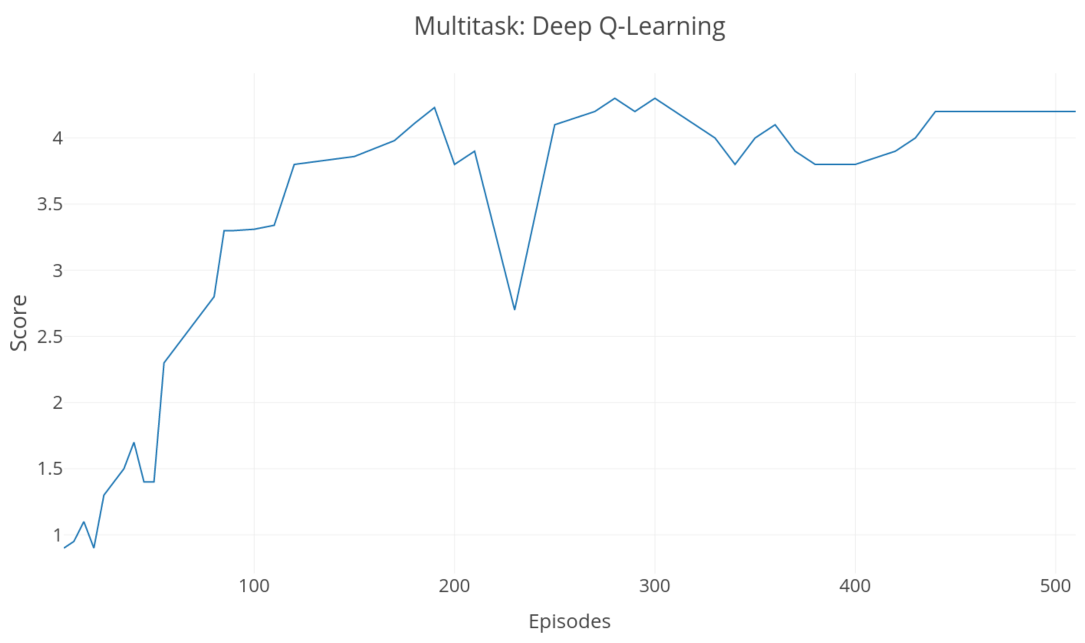


Figure 7: Deep Q-Learning Training

Figure 7 illustrates the training of DQN, where the x-axis represents episodes of the game and y-axis score before reaching the terminal state. The agent had troubles adapting to the third phase (see Section 5.1). Phase 3 is relatively hard to master because it requires the user balance the arrow in the air. At around 230 episodes we saw a drop in score. This is because the network seems to prioritize the first phase of the game. It reached the second phase a few times but was not able to successfully control the paddle for longer periods of time. This is why it stales at approximately 400 episodes. We believe that the network could have performed better with additional training time. It trained for a total of two days. Hopefully, it will be easier to train the network when FlashRL can speed-forward games, see section 7. The results are overall acceptable as we can see that FlashRL deliver quality states that a reinforcement learning agent can learn from.

6 Conclusion

FlashRL offers an easy-to-use architecture for performing RL in Flash-based games. It is demonstrated to work well for Multitask 2, one of the environments included. FlashRL fills the gap that emerged with the deprecation of Flash, Its main focus is RL, but can also be used for other machine learning genres. This paper shows that FlashRL can be used to train RL algorithms, in particular, Multitask 2. The work shows promising results and continuing to expand the game repository may provide new insights about RL in the future.

FlashRL will be kept alive as long as flash environments are an asset to the machine learning community. It is available to the public at `https://github.com/UIA-CAIR/FlashRL`, and can easily be adapted to every research requirement.

7 Future Work

Several improvements are planned for FlashRL. This paper outlined features of the initial version of the FlashRL, and it is by far sufficient for simple reinforcement learning research. As seen in section 5, a Deep Q-Learning based agent can successfully learn from the environment *Multitask* and gradually perform better.

7.1 Speed-forward Option

Learning algorithms often require several thousand episodes to gain expert knowledge of the environment. FlashRL is currently limited to the speed of which the game loop is executed (usually 30 fps in real-time). An important improvement would be to lift this restriction and allow algorithms to train at an accelerated rate. This would certainly improve training duration of feedback based algorithms.

7.2 Game Repository Analysis

The game repository features many unlabeled, unrated and untested games. Some games are potentially useless in a machine learning setting and require a review. The review phase is time-consuming, and authors of this paper did not have enough time to analyze each of the environments manually. The goal is to add labels and categorize all games in the repository gradually.

7.3 Website

A future goal is to allow execution of algorithms from a web interface and to add gamification aspects to the library. This would potentially create competition between researchers much like Kaggle and OpenAI Universe.

7.4 Cross-Platform Support

FlashRL is in the initial version, only supported in Python 3 on the Linux platform. The goal is to extend it so that it also can run without modifications on Microsoft Windows operating systems.

References

[1] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. In *IJCAI International Joint Conference on Artificial Intelligence*, volume 2015-Janua, pages 4148–4152, 2015.

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. jun 2016.

[3] Michael E. Grost, Trent Jaeger, Ming C. Leu, Dinesh K. Pai, Qiuming Zhu, A. Kusiak, M. Chen, F M. Brown, S S. Park, Ganapathy S. Kumar, P H. Cohen, B. Bidanda, O B. Arinze, Fatma Mili, Dahuan Shi, Patricia Zajko, and Ali Noui-Mehidi. Applications of Artificial Intelligence. In Birendra Prasad, S N. Dwivedi, and K B Irani, editors, *CAD/CAM Robotics and Factories of the Future: Volume II: Automation of Design, Analysis and Manufacturing*, pages 165–229. Springer Berlin Heidelberg, Berlin, Heidelberg, 1989.

[4] Guru99. Flash Testing with Selenium, 2017.

[5] Harold L Hunt and II Jon Turney. Cygwin/X Contributor's Guide, 2004.

[6] Yuxi Li. Deep Reinforcement Learning: An Overview. jan 2017.

[7] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. feb 2016.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari With Deep Reinforcement Learning. In *NIPS Deep Learning Workshop*. 2013.

[9] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[10] Matthew Piper. *How to Beat Flappy Bird: A Mixed-Integer Model Predictive Control Approach.* PhD thesis, The University of Texas at San Antonio, 2017.

[11] Stuart J Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*, volume 9. 1995.

[12] Richard S Sutton and Andrew G Barto. Reinforcement Learning: An Introduction. *IEEE Transactions on Neural Networks*, 9(5):1054–1054, 1998.

[13] Techtonik. python-vnc-viewer, 2015.

[14] Hado van Hasselt, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-learning. sep 2015.

[15] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling Network Architectures for Deep Reinforcement Learning. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48, pages 1995–2003, 2016.

Paper A: FlashRL: A Reinforcement Learning Platform for Flash Games

Paper B

Title:	Towards a Deep Reinforcement Learning Approach for Tower Line Wars
Authors:	Andersen, Per-Arne et al.
Affiliation:	Department of ICT, University of Agder, Grimstad Norway
Conference:	37th SGAI International Conference on Artificial Intelligence
Year:	2017

Towards a Deep Reinforcement Learning Approach for Tower Line Wars

Per-Arne Andersen
Department of ICT
University of Agder
Grimstad, Norway
per.andersen@uia.no

Morten Goodwin
Department of ICT
University of Agder
Grimstad, Norway
morten.goodwin@uia.no

Ole-Christoffer Granmo
Department of ICT
University of Agder
Grimstad, Norway
ole.granmon@uia.no

Abstract

There have been numerous breakthroughs with reinforcement learning in the recent years, perhaps most notably on Deep Reinforcement Learning successfully playing and winning relatively advanced computer games. There is undoubtedly an anticipation that Deep Reinforcement Learning will play a major role when the first AI masters the complicated game plays needed to beat a professional Real-Time Strategy game player. For this to be possible, there needs to be a game environment that targets and fosters AI research, and specifically Deep Reinforcement Learning. Some game environments already exist, however, these are either overly simplistic such as Atari 2600 or complex such as Starcraft II from Blizzard Entertainment.

We propose a game environment in between Atari 2600 and Starcraft II, particularly targeting Deep Reinforcement Learning algorithm research. The environment is a variant of Tower Line Wars from Warcraft III, Blizzard Entertainment. Further, as a proof of concept that the environment can harbor Deep Reinforcement algorithms, we propose and apply a Deep Q-Reinforcement architecture. The architecture simplifies the state space so that it is applicable to Q-learning, and in turn improves performance compared to current state-of-the-art methods. Our experiments show that the proposed architecture can

Thirty-seventh SGAI International Conference on Artificial Intelligence (AI-2017), Cambridge England
Submitted 07. Jul, 2017, Accepted 04. Sep 2017, Published 21. Nov, 2017

learn to play the environment well, and score 33% better than standard Deep Q-learning which in turn proves the usefulness of the game environment.

Keywords: Reinforcement Learning, Q-Learning, Deep Learning, Game Environment

1 Introduction

Despite many advances in AI for games, no universal reinforcement learning algorithm can be applied to Real-Time Strategy Games (RTS) without data manipulation or customization. This includes traditional games such as Warcraft III, Starcraft II, and Tower Line Wars. Reinforcement Learning (RL) has been applied to simpler games such as games for the Atari 2600 platform but has to the best of our knowledge not successfully been applied to RTS games. Further, existing game environments that target AI research are either overly simplistic such as Atari 2600 or complex such as Starcraft II.

Reinforcement Learning has had tremendous progress in recent years in learning to control agents from high-dimensional sensory inputs like vision. In simple environments, this has been proven to work well [1], but are still an issue for complex environments with large state and action spaces [2]. In games where the objective is easily observable, there is a short distance between action and reward which fuels the learning. This is because the consequence of any action is quickly observed, and then easily learned. When the objective is more complicated the game objectives still need to be mapped to the reward function, but it becomes far less trivial. For the Atari 2600 game Ms. Pac-Man this was solved through a hybrid reward architecture that transforms the objective to a low-dimensional representation [3]. Similarly, the OpenAI's bot is able to beat world's top professionals at 1v1 in DotA 2. It uses reinforcement learning while it plays against itself, learning to predict the opponent moves.

Real-Time Strategy Games, including Warcraft III, is a genre of games much more comparable to the complexity of real-world environments. It has a sparse state space with many different sensory inputs that any game playing algorithm must be able to master in order to perform well within the environment. Due to the complexity and because many action sequences are required to constitute a reward, standard reinforcement learning techniques including Q-learning are not able to master the games successfully.

This paper introduces a two-player version of the popular Tower Line Wars modification from the game Warcraft III. We refer to this variant as Deep Line Wars. Note that Tower Line Wars is not an RTS game, but has many similar elements such as time-delayed objectives, resource management, offensive, and defensive strategy planning. To prove that the environment is working we, inspired by recent advances from van Seijen et al. [3], apply a method of separating the abstract reward function of the environment into smaller rewards. This approach uses a Deep Q-Network using a Convolutional Neural

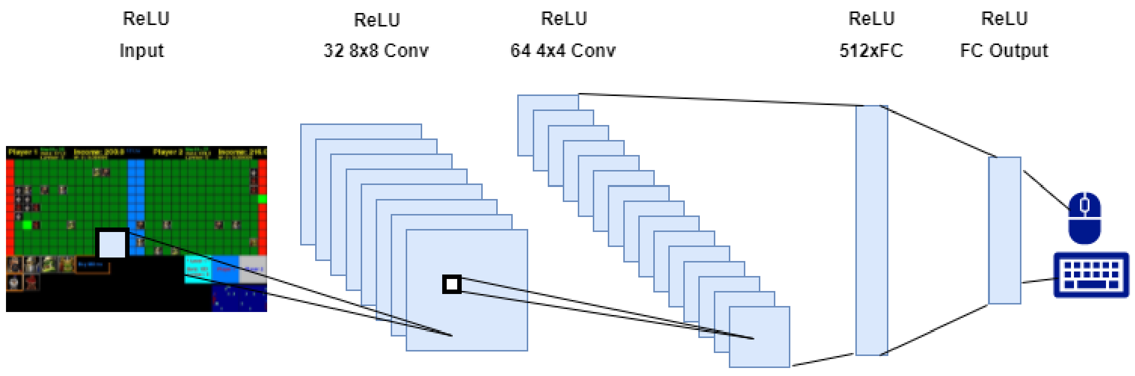


Figure 1: Deep Q-Learning architecture

Network to map actions to states and can play the game successfully and perform better than standard Deep Q-learning by 33%.

Rest of the paper is organized as follows: We first investigate recent discoveries in Deep RL in section 2. We then briefly outline how Q-Learning works and how we interpret Bellman's equation for utilizing Neural Networks as a function approximator in section 3. We present our contribution in section 4 and present a comparison of other game environments that are widely used in reinforcement learning. We introduce a variant of Deep Q-Learning in section 5 and present a comparison to other RL models used in state-of-the-art research. Finally we show results in section 6, define a roadmap of future work in section 7 and conclude our work in section 8

2 Related Work

There have been several breakthroughs related to reinforcement learning performance in recent years [4]. Q-Learning together with Deep Learning was a game-changing moment, and has had tremendous success in many single agent environments on the Atari 2600 platform [1]. Deep Q-Learning as proposed by Mnih et al. [1] as shown in Figure 1 used a neural network as a function approximator and outperformed human expertise in over half of the games [1].

Hasselt et al. proposed Double DQN, which reduced the overestimation of action values in the Deep Q-Network [5]. This led to improvements in some of the games on the Atari platform.

Wang et al. then proposed a dueling architecture of DQN which introduced estimation of the value function and advantage function [6]. These two functions were then combined to obtain the Q-Value. Dueling DQN were implemented with the previous work of van Hasselt et al. [6].

Harm van Seijen et al. recently published an algorithm called Hybrid Reward Architecture (HRA) which is a divide and conquer method where several agents estimate a reward and a Q-value for each state [3]. The algorithm performed above human expertise in Ms. Pac-Man, which is considered one of the hardest games in the Atari 2600 collection and is currently state-of-the-art in the reinforcement learning domain [3]. The drawback of this algorithm is that generalization of Minh et al. approach is lost due to a huge number of separate agents that have domain-specific sensory input.

There have been few attempts at using Deep Q-Learning on advanced simulators specifically made for machine-learning. It is probable that this is because there are very few environments created for this purpose.

3 Q-Learning

Reinforcement learning can be considered hybrid between supervised and unsupervised learning. We implement what we call an agent that acts in our environment. This agent is placed in the unknown environment where it tries to maximize the environmental reward [7].

Markov Decision Process (MDP) is a mathematical method of modeling decision-making within an environment. We often use this method when utilizing model-based RL algorithms. In Q-Learning, we do not try to model the MDP. Instead, we try to learn the optimal policy by estimating the action-value function $Q^*(s,a)$, yielding maximum expected reward in state s executing action a. The optimal policy can then be fosund by

$$\pi(s) = argmax_a Q^*(s,a) \tag{1}$$

This is derived from *Bellman's Equation*, because we can consider $U(s) = max_a Q(s,a)$, the Utility function to be true. This gives us the ability to derive following update-rule equation from Bellman's work:

$$Q(s,a) \leftarrow Q(s,a) + \underbrace{\alpha}_{\text{Learning Rate}} \left(\underbrace{R(s)}_{\text{Reward}} + \underbrace{\gamma}_{\text{Discount}} \underbrace{max_{a'} Q(s^{'},a^{'})}_{\text{New Estimate}} - \underbrace{Q(s,a)}_{\text{Old Estimate}} \right) \tag{2}$$

This is an iterative process of propagating back the estimated Q-value for each discrete time-step in the environment. It is guaranteed to converge towards the optimal action-value function, $Q_i \rightarrow Q^*$ as i $\rightarrow \infty$ [7, 1]. At the most basic level, Q-Learning utilize a table for storing $(s,a,r,s^{'})$ pairs. But we can instead use a non-linear function approximation in order to approximate $Q(s,a;\theta)$. θ describes tunable parameters for approximator.

Artificial Neural Networks (ANN) are a popular function approximator, but training using ANN is relatively unstable. We define the loss function as following.

$$L(\theta_i) = E\Big[(r + \gamma max_{a'} Q(s^{'}, a^{'}; \theta_i) - Q(s, a; \theta_i))^2\Big] \tag{3}$$

As we can see, this equation uses Bellman equation to calculate the loss for the gradient descent. To combat training instability, we use *Experience Replay*. This is a memory module which stores memories from experienced states and draws a uniform distribution of experiences to train the network [1]. This is what we call a *Deep Q-Network* and are as described in its most primitive form. See related work for recent advancements in DQN.

4 Deep Line Wars

For a player to play RTS games well, he typically needs to master high difficulty strategies. Most RTS strategies incorporate

- Build strategies,
- Economy management,
- Defense evaluation, and
- Offense evaluation.

These objectives are easy to master when separated but become hard to perfect when together. Starcraft II is one of the most popular RTS games, but due to its complexity, it is not expected that an AI-based system can beat this game anytime soon. At the very least, state-of-the-art Deep Q-Learning is not directly applicable. Blizzard entertainment and Google DeepMind has collaborated on an interface to the Starcraft II game. [8, 9]. Starcraft II is for many researchers considered the next big goal in AI research. Warcraft III is relatable to Starcraft II as they are the same genre and have near identical game mechanics.

Current state-of-the-art algorithms struggle to learn objectives in the state-space because the action-space is too abstract. [10]. State and action spaces define the range of possible configurations a game board can have. Existing DQN models use pixel data as input and objectively maps state to action [1]. This works when the game objective is closely linked to an action, such as controlling a paddle in Breakout, where the correct action is quickly rewarded, and a wrong action quickly punished. This is not possible in RTS games. If the objective is to win the game, an action will only be rewarded or punished after minutes or even hours of gameplay. Furthermore, gameplay would consist of thousands of actions and only combined will they result in a reward or punishment.

Game Property Chart

Game	Stochastic	Partial Observable	Simultaneous	Solved	Date
Tic Tac Toe	NO	NO	NO	YES	1970's
Connect Four	NO	NO	NO	YES	1970's
Chess	NO	NO	NO	YES	1996
GO (19x19)	NO	NO	NO	YES	2015
Backgammon	YES	NO	NO	YES	1979
Deep Line Wars	YES	**BOTH**	**YES**	**NO**	**NO**
Ms. Pac Man	NO	NO	YES	YES	2017
Starcraft II	**Yes**	**Yes**	**Yes**	NO	NO

Figure 2: Properties of selected game environments

Collected data in Figure 2 argues that games that have been solved by current state-of-the-art is usually non-stochastic and is fully observable. Also, current AI prefers environments which are not simultaneous, meaning they can be paused between each state transition. This makes sense because hardware still limits advances in AI.

By doing rough estimations of the state-space in-game environments from Figure 2, it is clear that state-of-the-art has done a big leap in recent years. With the most recent contribution being Ms. Pac-Man [3]. However, by computing the state-space of a regular Starcraft II map only taking unit compositions into account, the state space can be calculated to be $(128x128)^{400} = 16384^{400} = 10^{1685}$ [11].

The predicament is that the difference in complexity between Ms. Pac-Man and Starcraft II is tremendous. Figure 3 illustrates a relative and subjective comparison between state-complexity in relevant game environments. State-space complexity describes approximately how many different game configurations a game can have. It is based on map size, unit position, and unit actions. The comparison is a bit arbitrary because the games are complex in different manners. However, there is no doubt that the distance between Ms. Pac-Man, perhaps the most advanced computer game mastered so far, and Starcraft II is colossal. To advance AI solutions towards Starcraft II, we argue that there is a need for several new game environments that exceed the complexity of existing games and challenge researches on multi-agent issues closely related to Starcraft II [12]. We, therefore, introduce Deep Line Wars as a two-player variant of Tower Line Wars. Deep

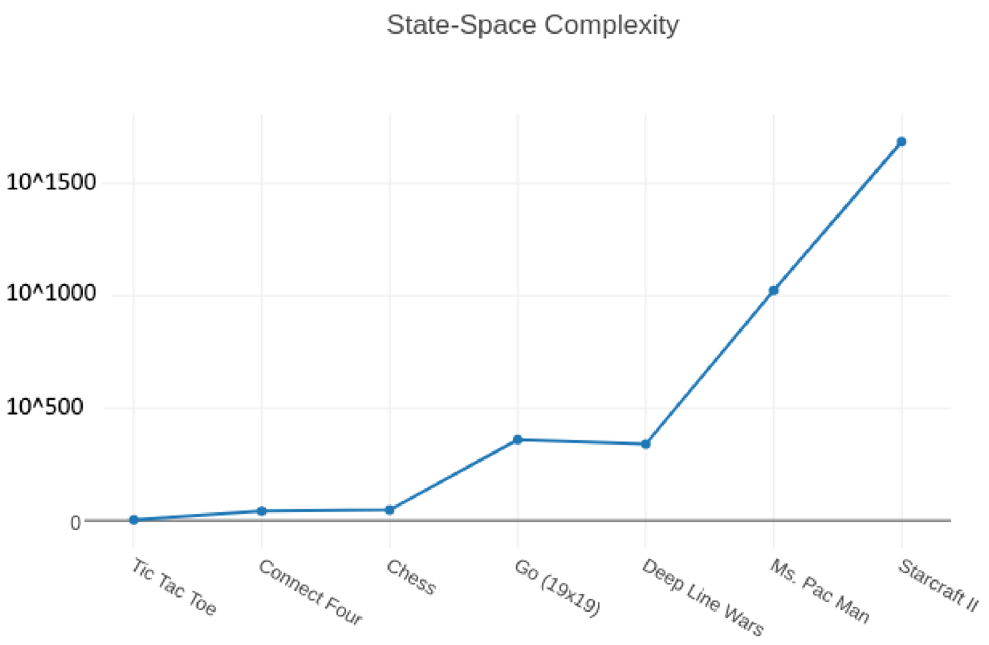


Figure 3: State-space complexity of selected game environments

Line Wars is a game simulator aimed at filling the gap between Atari 2600 and Starcraft II. It features the most important aspects of an RTS game.

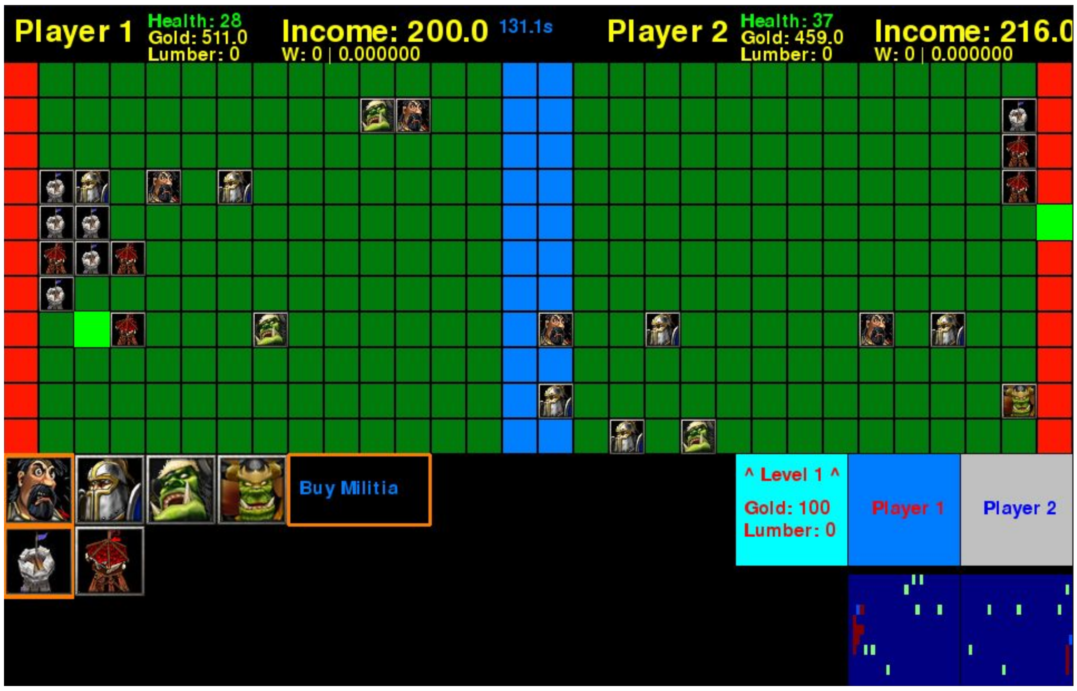


Figure 4: Graphical Interface of Deep Line Wars

The objective of this game is as seen in Figure 4 to invade the opposing player with units until all health is consumed. The opposing player's health is reduced for each friendly unit that enters the red area of the map. A unit spawns at a random location on the red line of

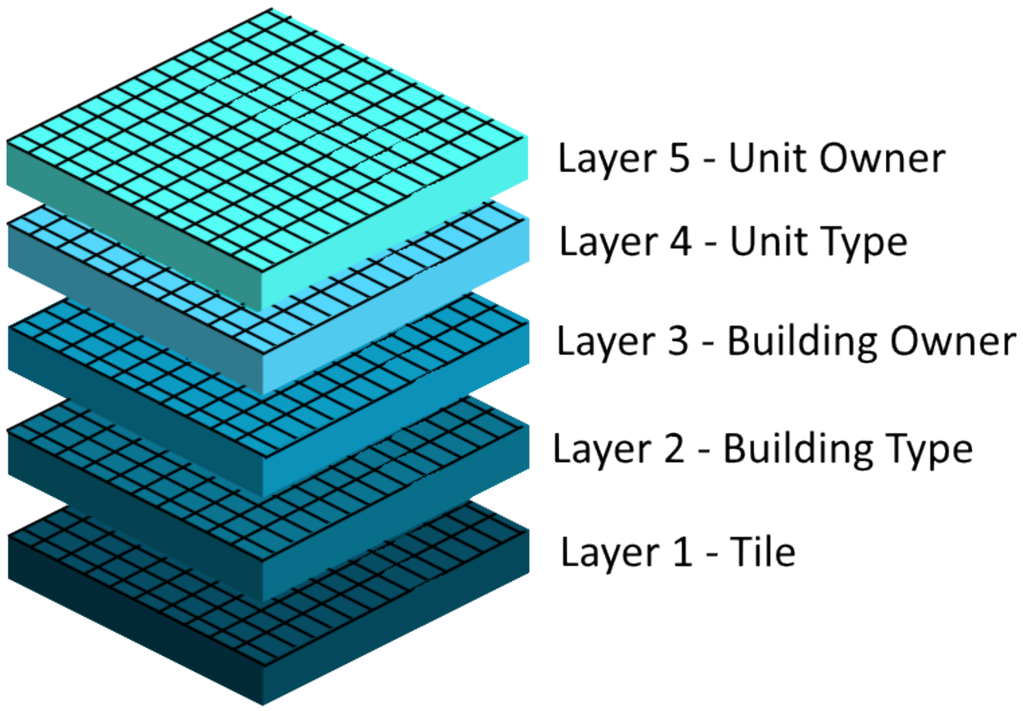


Figure 5: Game-state representation

the controlling player's side and automatically walks towards the enemy base. To protect your base against units, the player can build towers which shoot projectiles at enemy units. When an enemy unit dies, a fair percentage of the unit value is given to the player. When a player sends a unit, the income variable is increased by a defined percentage of the unit value. Players gold are increased at regular intervals determined in the configuration files. To master Deep Line Wars, the player must learn following skill-set:

- offensive strategies of spawning units,
- defending against the opposing player's invasions, and
- maintain a healthy balance between offensive and defensive in order to maximize income

and is guaranteed a victory if mastered better than the opposing player.

Because the game is specifically targeted towards machine learning, the game-state is defined as a multi-dimensional matrix. Figure 5 represents a 5x30x11 state-space that contains all relevant board information at current time-step. It is therefore easy to cherry-pick required state-information when using it in algorithms. Deep Line Wars also features possibilities of making an abstract representation of the state-space, seen in Figure 6. This is a heat-map that represent the state (Figure 5) as a lower-dimensional state-space. Heat-maps also allows the developer to remove noise that causes the model to diverge from the optimal policy, see Formula 3.

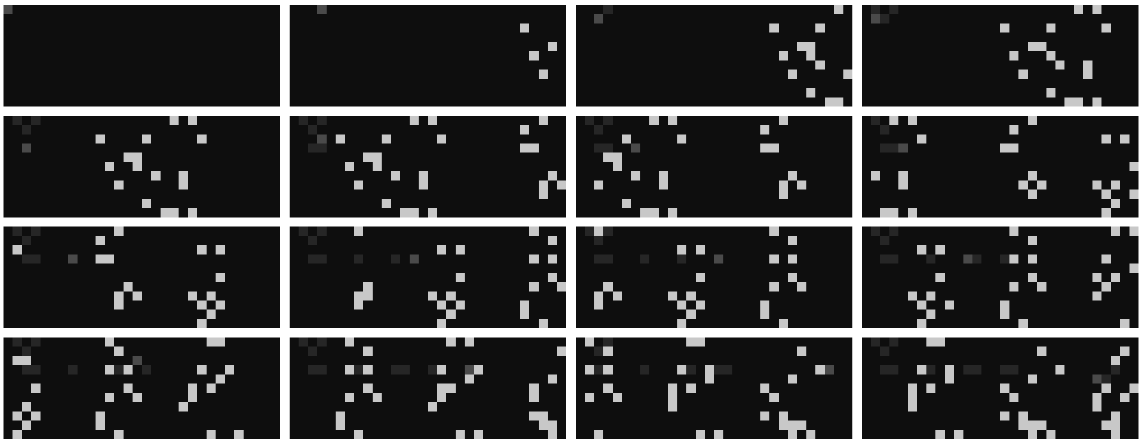


Figure 6: State abstraction using gray-scale heat-maps

We need to reduce the complexity of the state-space to speed up training. Using heat-maps made it possible to encode the five-dimensional state information into three dimensions. These dimensions are RGB values that we can find in imaging. Figure 6 show how the state is seen from the perspective of player 1 using gray-scale heatmaps. We define

- red pixels as friendly buildings,
- green pixels as enemy units, and
- teal pixels as the mouse cursor.

We also included an option to reduce the state-space to a one-dimensional matrix using gray-scale imaging. Each of the above features is then represented by a value between 0 and 1. We do this because Convolutional Neural Networks are computational demanding, and by reducing input dimensionality, we can speed up training. [1] We do not down-scale images because the environment is only 30x11 pixels large. The state cannot be described fully by these heat-maps as there are economics, health, and income that must be interpreted separately. This is solved by having a 1-dimensional vectorized representation of the data, that can be fed into the model.

5 DeepQRewardNetwork

The main contribution in this paper is the game environment presented in Section 4. A key element is to show that the game environment is working properly and we, therefore, introduce a learning algorithm trying to play the game. This is in no way meant as a perfect solver for Deep Line Wars, but rather as a proof of concept that learning algorithms can be applied in the Deep Line Wars environment. In our solution we consider the environment as a MDP having state set S, action set A, and a reward function set R.

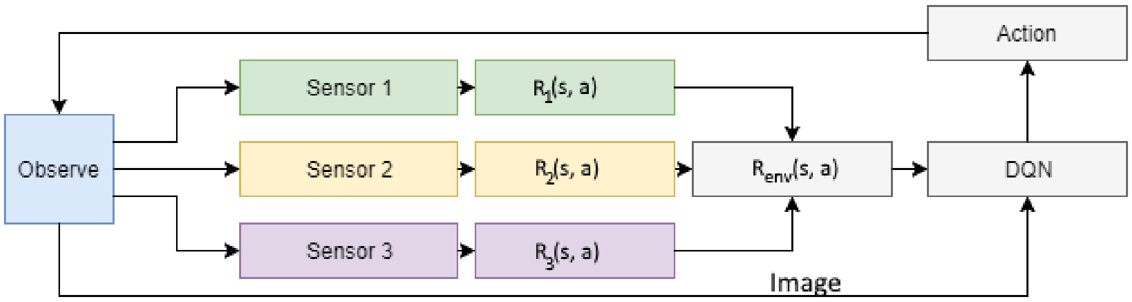


Figure 7: Separation of the reward function

Each of the weighted reward functions derives from a specific agent within the MDP and defines the absolute reward of the environment R_{env} with following equation:

$$R_{env}(s,a) = \sum_{i=1}^{n} w_i R_i(s,a) \tag{4}$$

Where $R_{env}(s,a)$ is the weighted sum w_i of reward function(s) $R_i(s,a)$. The proposed algorithm model is a method of dividing the ultimate problem into separate smaller problems which can be trivialized with certain kinds of generic algorithms.

When reward for the observed state is calculated, we calculate the Q-value of $Q(s,a)$ utilizing R_{env} by using a variant of DQN.

6 Experiments

We conducted experiments with several deep learning algorithms in order to benchmark current state-of-the-art put up against a multi-agent, multi-sensory environment. The experiments were conducted in Deep Line Wars, a multi-agent, multi-sensory environment. All algorithms were benchmarked with identical game parameters.

We tested *DeepQNetwork*, a state-of-the-art DQN from Mnih et al[1], *DeepQRewardNetwork*, rule-based, and random behaviour. Each of the algorithms was tested with several configurations, seen in Figure 8. We did not expect any of these algorithms to beat the rule-based challenge due to the difficulty of the AI. The extended execution graph algorithm (see Section 7) was not part of the test bed because it was not able to compete with any of the simpler DQN algorithms without guided mouse management.

Tests were done using Intel I7-4770k, 64GB RAM and NVIDIA Geforce GTX 1080TI. Each of the algorithms was trained/executed for 1500 episodes. Each episode is considered to be a game that either of the players wins, or the 600 seconds time limit is reached. DQN had a discount-factor of 0.99, learning rate of 0.001 and batch-size of 32.

Algorithm	Double Q-Learning	Prioritized Replay	Dueling DQN
Deep Q-Network	YES	YES/NO	NO
DeepQRewardNet...	NO	NO	NO
Random	N/A	N/A	N/A
Rule Based	N/A	N/A	N/A

Figure 8: Property matrix of tested algorithms

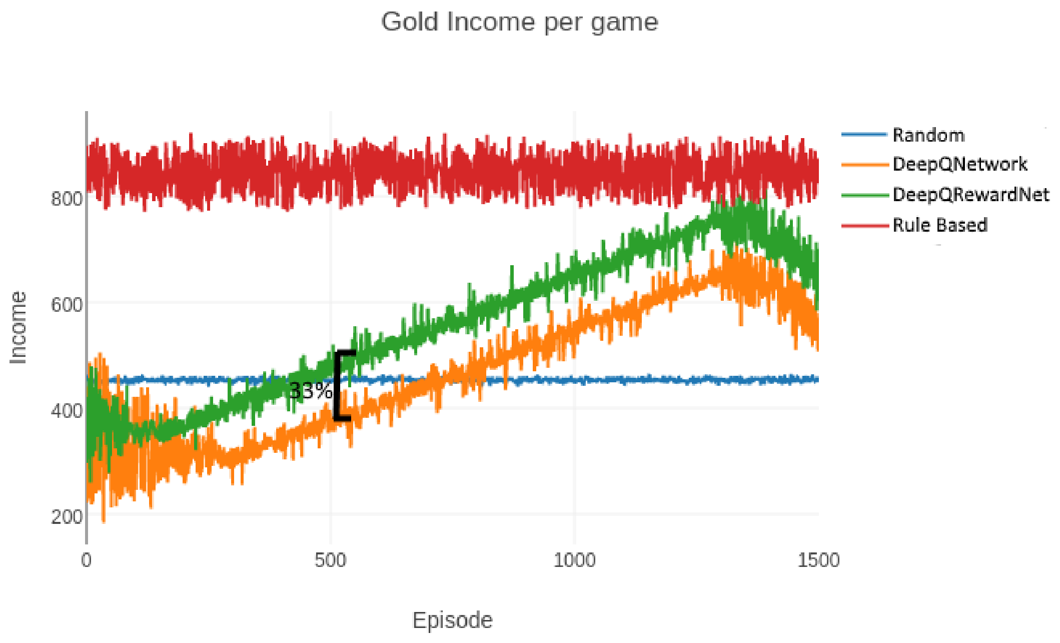


Figure 9: Income after each episode

Throughout the learning process, we can see that DeepQNetwork and DeepQReward-Network learn to perform resource management correctly. Figure 9 illustrates income throughout learning from 1500 episodes. The random player is presented as an aggregated average of 1500 games, but the remaining algorithms are only single instances. It is not practical to perform more than a single run of the Deep Learning algorithms because it takes several minutes per episode to finish which sums up to a huge learning time.

Figure 9 shows that the proposed algorithms outperform random behavior after relatively few episodes. DeepQRewardNetwork performs approximately 33% better than DeepQNetwork. We believe that this is because the reward function $R(s, a)$ is better defined and therefore easier to learn the optimal policy in a shorter period of time. These results show that DeepQRewardNetwork converges towards the optimal policy better, but as seen in Figure 9 diverges after approximately 1300 games. The reason for the divergence is that experience replay does not correctly batch important memories to the model. This causes the model to train on unimportant memories and diverges the model. This is considered a part of future work and is addressed more thoroughly in section 7. The rule-based algorithm can be regarded as an average player and can be compared to human level in this game environment.

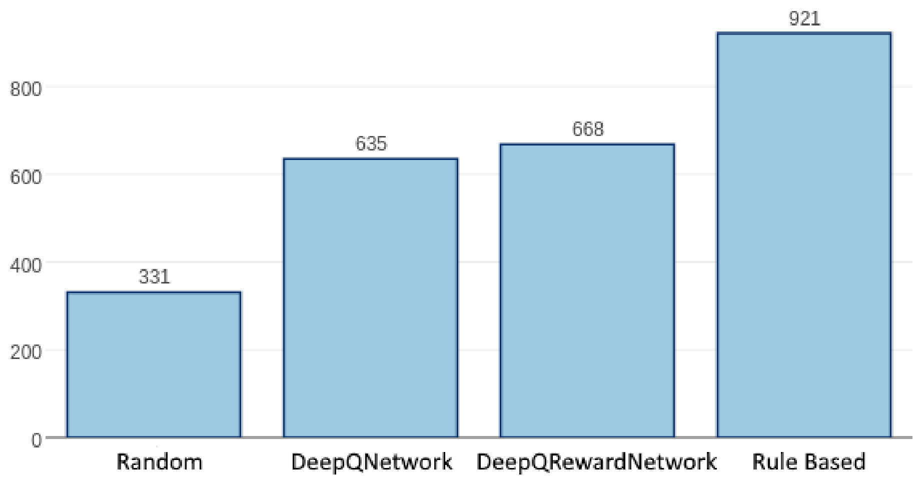


Figure 10: Victory distribution of tested algorithms

Figure 10 shows that DeepQNetwork and DeepQRewardNetwork have about 63-67% win ratio throughout the learning process. Compared to the rule-based AI it does not qualify to be near mastering the game, but we can see that it outperforms random behavior in the game environment.

7 Future Work

This paper introduced a new learning environment for reinforcement learning and applied state-of-the-art Deep-Q Learning to the problem. Some initial results showed progress towards an AI that could beat a rule-based AI. There are still several challenges that must be addressed for an unsupervised AI to learn complex environments like Line Tower Wars. Mouse input based games are difficult to map to an abstract state representation, because there are a huge number of sequenced mouse clicks that are required, to correctly act in the game. DQN cannot at current state handle long sequences of actions and must be guided in-order to succeed. Finding a solution to this problem without guiding is thought to be the biggest blocker for these types of environments, and will be the focus for future work.

DeepQNetwork and DeepQRewardNetwork had issues with divergence after approximately 1300 episodes. This is because our experience replay algorithm did not take into account that the majority of experiences are bad. It could not successfully prioritize the important memories. As future work, we propose to instead use prioritized experience replay from Schaul et al. [13].

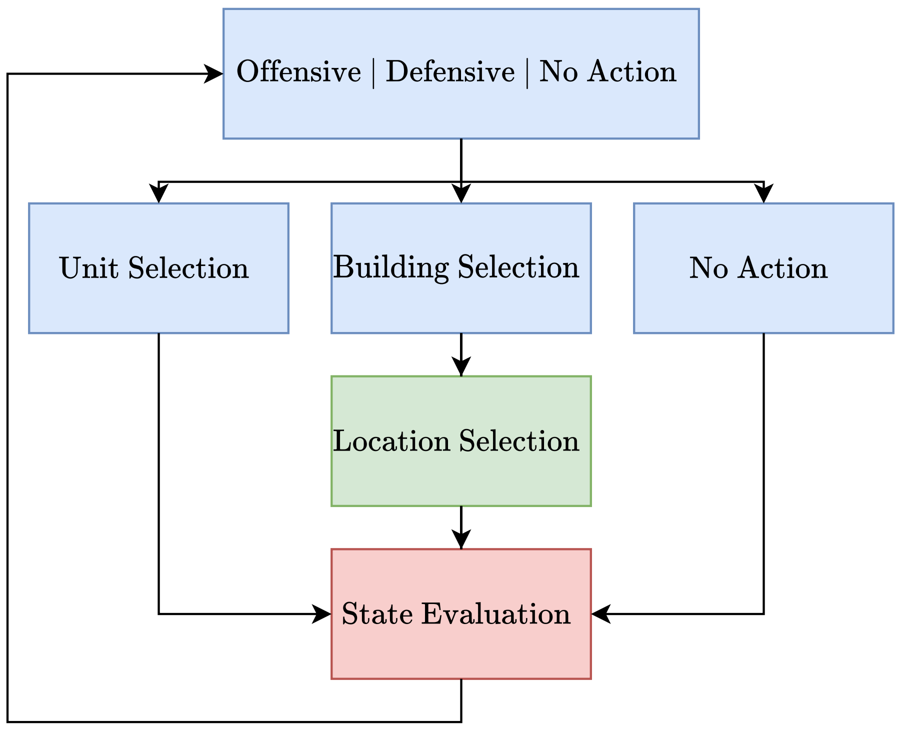


Figure 11: Divide & Conquer Execution graph

Figure 7 show that different sensors separate the reward from the environment to obtain a more precise reward bound to an action. In our research, we developed an algorithm that utilizes different models based on which state the player has. Figure 11 show the general idea, where the state is categorized into three different types *Offensive*, *Defensive*, and

No Action. This state is evaluated by a Convolutional Neural Network and outputs a one-hot vector that signal which state the player is currently in. Each of the blocks in Figure 11 then represents a form of state-modeling that is determined by the programmer. Our initial tests did not yield any promising results, but according to the Bellman equations, it is a qualified way of evaluating the state and successfully perform learning, on an iterative basis.

8 Conclusion

Deep Line Wars is a simple but yet advanced Real-Time (strategy) game simulator, which attempts to fill the gap between Atari 2600 and Starcraft II. DQN shows promising initial results but is far from perfect in current state-of-the-art. An attempt in making abstractions in the reward signal yielded some improved performance, but at the cost of a more generalized solution. Because of the enormous state-space, DQN cannot compete with simple rule-based algorithms. We believe that this is caused by specifically the mouse input which requires some understanding of the state to perform well. This also causes the algorithm to overestimate some actions, specifically the offensive actions, because the algorithm is not able to correctly build defensive without getting negative rewards. It is imperative that a solution of the mouse input actions are found before DQN can perform better. A potential approach could be using the StarCraft II API to get additional training data, including mouse sequences [14].

References

[1] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing atari with deep reinforcement learning. In: NIPS Deep Learning Workshop. (2013)

[2] Mirowski, P., Pascanu, R., Viola, F., Soyer, H., Ballard, A.J., Banino, A., Denil, M., Goroshin, R., Sifre, L., Kavukcuoglu, K., Kumaran, D., Hadsell, R.: Learning to navigate in complex environments. CoRR **abs/1611.03673** (2016)

[3] van Seijen, H., Fatemi, M., Romoff, J., Laroche, R., Barnes, T., Tsang, J.: Hybrid reward architecture for reinforcement learning. **abs/1706.04208** (2017)

[4] Gosavi, A.: Reinforcement learning: A tutorial survey and recent advances. INFORMS Journal on Computing **21**(2) (2009) 178–192

[5] van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. CoRR **abs/1509.06461** (2015)

[6] Wang, Z., de Freitas, N., Lanctot, M.: Dueling network architectures for deep reinforcement learning. CoRR **abs/1511.06581** (2015)

[7] Sutton, R.S., Barto, A.G.: Reinforcement Learning : An Introduction. MIT Press (1998)

[8] Traysent: Starcraft ii api – technical design. `https://us.battle.net/forums/en/sc2/topic/20751114921` (Nov)

[9] Vinyals, O.: Deepmind and blizzard to release starcraft ii as an ai research environment. `https://deepmind.com/blog/deepmind-and-blizzard-release-starcraft-ii-ai-research-environment/` (Nov 2016)

[10] Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. CoRR **abs/1509.02971** (2015)

[11] Uriarte, A., Ontañón, S.: Game-tree search over high-level game states in rts games (Oct 2014)

[12] Bellemare, M.G., Naddaf, Y., Veness, J., Bowling, M.: The arcade learning environment: An evaluation platform for general agents. CoRR **abs/1207.4708** (2012)

[13] Schaul, T., Quan, J., Antonoglou, I., Silver, D.: Prioritized experience replay. CoRR **abs/1511.05952** (2015)

[14] Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Sasha Vezhnevets, A., Yeo, M., Makhzani, A., Küttler, H., Agapiou, J., Schrittwieser, J., Quan, J., Gaffney, S., Petersen, S., Simonyan, K., Schaul, T., van Hasselt, H., Silver, D., Lillicrap, T., Calderone, K., Keet, P., Brunasso, A., Lawrence, D., Ekermo, A., Repp, J., Tsing, R.: StarCraft II: A New Challenge for Reinforcement Learning. ArXiv e-prints (August 2017)

Paper B: Towards a Deep Reinforcement Learning Approach for Tower Line Wars

Paper C

Title:	Deep RTS: A Game Environment for Deep Reinforcement Learning in Real-Time Strategy Games
Authors:	Andersen, Per-Arne et al.
Affiliation:	Department of ICT, University of Agder, Grimstad Norway
Conference:	IEEE Conference on Computational Intelligence and Games
Year:	2018, August

Deep RTS: A Game Environment for Deep Reinforcement Learning in Real-Time Strategy Games

Per-Arne Andersen
Department of ICT
University of Agder
Grimstad, Norway
per.andersen@uia.no

Morten Goodwin
Department of ICT
University of Agder
Grimstad, Norway
morten.goodwin@uia.no

Ole-Christoffer Granmo
Department of ICT
University of Agder
Grimstad, Norway
ole.granmon@uia.no

Abstract

Reinforcement learning (RL) is an area of research that has blossomed tremendously in recent years and has shown remarkable potential for artificial intelligence based opponents in computer games. This success is primarily due to the vast capabilities of convolutional neural networks, that can extract useful features from noisy and complex data. Games are excellent tools to test and push the boundaries of novel RL algorithms because they give valuable insight into how well an algorithm can perform in isolated environments without the real-life consequences.

Real-time strategy games (RTS) is a genre that has tremendous complexity and challenges the player in short and long-term planning. There is much research that focuses on applied RL in RTS games, and novel advances are therefore anticipated in the not too distant future. However, there are to date few environments for testing RTS AIs. Environments in the literature are often either overly simplistic, such as microRTS, or complex and without the possibility for accelerated learning on consumer hardware like StarCraft II.

This paper introduces the Deep RTS game environment for testing cutting-edge artificial intelligence algorithms for RTS games. Deep RTS is a high-performance RTS game made

IEEE Conference on Computational Intelligence and Games (CIG-2018), Maastricht, the Netherlands
Submitted 01. Mar, 2018, Accepted 30. Apr 2018, Published 15. Aug, 2018

specifically for artificial intelligence research. It supports accelerated learning, meaning that it can learn at a magnitude of 50 000 times faster compared to existing RTS games. Deep RTS has a flexible configuration, enabling research in several different RTS scenarios, including partially observable state-spaces and map complexity. We show that Deep RTS lives up to our promises by comparing its performance with microRTS, ELF, and StarCraft II on high-end consumer hardware. Using Deep RTS, we show that a Deep Q-Network agent beats random-play agents over 70% of the time. Deep RTS is publicly available at `https://github.com/cair/DeepRTS`.

Keywords: Real-Time Strategy Game, Deep Reinforcement Learning, Deep Q-Learning

1 Introduction

Despite many advances in Artificial Intelligence (AI) for games, no universal Reinforcement learning (RL) algorithm can be applied to complex game environments without extensive data manipulation or customization. This includes traditional Real-time strategy games (RTS) such as WarCraft III, StarCraft II, and Age of Empires. RL has recently been applied to simpler game environments such as those found in the Arcade Learning Environment [1](ALE) and board games [2] but has not successfully been applied to more advanced games. Further, existing game environments that target AI research are either overly simplistic such as ALE or complex such as StarCraft II.

RL has in recent years had tremendous progress in learning how to control agents from high-dimensional sensory inputs like images. In simple environments, this has been proven to work well [3], but are still an issue for complex environments with large state and action spaces [4]. The distinction between simple and complex tasks in RL often lies in how easy it is to design a reward model that encourages the algorithm to improve its policy without ending in local optima [5]. For simple tasks, the reward function can be described by only a few parameters, while in more demanding tasks, the algorithm struggles to determine what the reward signal is trying to accomplish [6]. For this reason, the reward function is in literature often a constant or single-valued variable for most time-steps, where only the final time-step determines a negative or positive reward [7–9]. In this paper we introduce Deep RTS, a new game environment targeted deep reinforcement learning (DRL) research. Deep RTS is an RTS simulator inspired by the famous StarCraft II video game by Blizzard Entertainment.

This paper is structured as follows. First, Section 2 and Section 3 thoroughly outlines previous work and central achievements using game environments for RL research. Next, Section 4 introduces the Deep RTS game environment. Section 5 presents the Deep RTS performance, a comparison between well-established game environments and Deep RTS, and experimental results using Deep Q-Network as an agent in Deep RTS. Subsequently, Section 6 concludes the contribution of this paper and outlines a roadmap for future work.

2 Related Game Environments

There exist several exciting game environments in the literature that focus on state-of-the-art research in AI algorithms. Few game environments target the RTS-genre. One the reason may be because these environments are by nature challenging to solve, and there are few ways to fit results with preprocessing tricks. It is, however, essential to include RTS as part of the active research of deep reinforcement learning algorithms as they feature long-term planning. This section outlines a thorough literature review of existing game platforms and environments and is summarized in Table 1.

Table 1: Selected game environments that is actively used in reinforcement learning research

Platform	RTS	Complex[1]	Year	Solved	Source
ALE	No	No	2012	Yes	[10]
Malmo Platform	No	Yes	2016	No	[11]
ViZDoom	No	Yes	2016	No	[12]
DeepMind Lab	No	Yes	2016	No	[13]
OpenAI Gym	No	No	2016	No	[14]
OpenAI Universe	No	Yes	2016	No	[15]
Stratagus	Yes	Yes	2005	No	[16]
microRTS	Yes	No	2013	No	[17]
TorchCraft	Yes	Yes	2016	No	[18]
ELF	Yes	Yes	2017	No	[19]
SC2LE	Yes	Yes	2017	No	[8]
Deep RTS	Yes	Yes	2018	No	-

2.1 Stratagus

Stratagus is an open source game engine that can be used to create RTS-themed games. Wargus, a clone of Warcraft II, and Stargus, a clone of StarCraft I are examples of games implemented in the Stratagus game engine. Stratagus is not an engine that targets machine learning explicitly, but several researchers have performed experiments in case-based reasoning [20, 21] and q-learning [22] using Wargus. Stratagus is still actively updated by contributions from the community.

[1] A Complex environment has an enormous state-space, with reward signals that are difficult to correlate to an action.

2.2 Arcade Learning Environment

Bellemare *et al.* provided in 2012 the arcade learning environment that enabled researchers to conduct cutting-edge research in general deep learning [10]. The package provided hundreds of Atari 2600 environments that in 2013 allowed Minh *et al.* to do a breakthrough using Deep Q-Learning and A3C. The platform has been a critical component in several advances in RL research. [1, 3, 23]

2.3 microRTS

microRTS is a simple RTS game, designed to conduct AI research. The idea behind microRTS is to strip away the computational heavy game logic to increase the performance and to enable researchers to test theoretical concepts quickly [17]. The microRTS game logic is deterministic, and include options for full and partially-observable state-spaces. The primary field of research in microRTS is game-tree search techniques such as variations of Monte-Carlo tree search and minimax [17, 24, 25].

2.4 TorchCraft

In 2016, a research group developed TorchCraft, a bridge that enables research in the game StarCraft. TorchCraft intends to provide the reinforcement learning community with a way to allow research on complex systems where only a fraction of the state-space is available [18]. In literature, TorchCraft has been used for deep learning research [26, 27]. There is also a dataset that provides data from over 65,000 StarCraft replays [28].

2.5 Malmo Platform

The Malmo project is a platform built atop of the popular game *Minecraft*. This game is set in a 3D environment where the object is to survive in a world of dangers. The paper *The Malmo Platform for Artificial Intelligence Experimentation* by Johnson *et al.* claims that the platform has all characteristics qualifying it to be a platform for general artificial intelligence research. [11]

2.6 ViZDoom

ViZDoom is a platform for research in visual reinforcement learning. With the paper *ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning* Kempka *et al.* illustrated that an RL agent could successfully learn to play the game *Doom*, a first-person shooter game, with behavior similar to humans. [29]

2.7 DeepMind Lab

With the paper *DeepMind Lab*, Beattie *et al.* released a platform for 3D navigation and puzzle solving tasks. The primary purpose of DeepMind Lab is to act as a platform for DRL research. [13]

2.8 OpenAI Gym

In 2016, Brockman *et al.* from OpenAI released GYM which they referred to as *"a toolkit for developing and comparing reinforcement learning algorithms"*. GYM provides various types of environments from following technologies: Algorithmic tasks, Atari 2600, Board games, Box2d physics engine, MuJoCo physics engine, and Text-based environments. OpenAI also hosts a website where researchers can submit their performance for comparison between algorithms. GYM is open-source and encourages researchers to add support for their environments. [14]

2.9 OpenAI Universe

OpenAI recently released a new learning platform called *Universe*. This environment further adds support for environments running inside VNC. It also supports running Flash games and browser applications. However, despite OpenAI's open-source policy, they do not allow researchers to add new environments to the repository. This limits the possibilities of running any environment. The OpenAI Universe is, however, a significant learning platform as it also has support for desktop games like Grand Theft Auto IV, which allow for research in autonomous driving [30].

2.10 ELF

The Extensive Lightweight Flexible (ELF) research platform was recently present at NIPS with the paper *ELF: An Extensive, Lightweight and Flexible Research Platform for Real-time Strategy Games*. This paper focuses on RTS game research and is the first platform officially targeting these types of games. [19]

2.11 StarCraft II Learning Environment

SC2LE (StarCraft II Learning Environment) is an API wrapper that facilitates access to the StarCraft II game-state using languages such as Python. The purpose is to enable reinforcement learning and machine learning algorithms to be used as AI for the game players. StarCraft II is a complex environment that requires short and long-term planning. It is difficult to observe a correlation between actions and rewards due to the imperfect state information and delayed rewards, making StarCraft II one of the hardest challenges so far in AI research.

3 Reinforcement Learning in Games

Although there are several open-source game environments suited for reinforcement learning, few of them are part of a success story. One of the reasons for this is that current state-of-the-art algorithms are seemingly unstable [30], and have difficulties to converge towards optimal policy in environments with multi-reward objectives [31]. This section exhibits the most significant achievements using reinforcement learning in games.

3.1 TD-Gammon

TD-Gammon is an algorithm capable of reaching an expert level of play in the board game *Backgammon* [7, 32]. The algorithm was developed by Gerald Tesauro in 1992 at IBM's Thomas J. Watson Research Center. TD-Gammon consists of a three-layer artificial neural network (ANN) and is trained using a reinforcement learning technique called *TD-Lambda*. TD-Lambda is a temporal difference learning algorithm invented by Richard S. Sutton [33]. The ANN iterates over all possible moves the player can perform and estimates the reward for that particular move. The action that yields the highest reward is then selected. TD-Gammon is the first algorithm to utilize self-play methods to improve the ANN parameters.

3.2 AlphaGO

In late 2015, *AlphaGO* became the first algorithm to win against a human professional Go player. AlphaGO is a reinforcement learning framework that uses Monte-Carlo tree search and two deep neural networks for value and policy estimation [9]. Value refers to the expected future reward from a state assuming that the agent plays perfectly. The policy network attempts to learn which action is best in any given board configuration. The earliest versions of AlphaGO used training data from previous games played by human professionals. In the most recent version, *AlphaGO Zero*, only self-play is used to train the AI [34]. In a recent update, AlphaGO was generalized to work for Chess and Shogi (Japanese Chess) only using 24 hours to reach a superhuman level of play [2].

3.3 DeepStack

DeepStack is an algorithm that can perform an expert level play in Texas Hold'em poker. This algorithm uses tree-search in conjunction with neural networks to perform sensible actions in the game [35]. DeepStack is a general-purpose algorithm that aims to solve problems with imperfect information. The DeepStack algorithm is open-source and available at `https://github.com/lifrordi/DeepStack-Leduc`.

3.4 Dota 2

DOTA 2 is a complex player versus player game where the player controls a hero unit. The game objective is to defeat the enemy heroes and destroy their base. In August 2017, OpenAI invented a reinforcement learning based AI that defeated professional players in one versus one games. The training was done by only using self-play, and the algorithm learned how to exploit game mechanics to perform well within the environment. DOTA 2 is used actively in research where the next goal is to train the AI to play in a team-game based environment.

4 The Deep RTS Learning Environment

There is a need for new RTS game environments targeting reinforcement learning research. Few game environments have a complexity suited for current state-of-the-art research, and there is a lack of flexibility the existing solutions.

The Deep RTS game environment enables research at different difficulty levels in planning, reasoning, and control. The inspiration behind this contribution is microRTS and StarCraft II, where the goal is to create an environment that features challenges between the two. The simplest configurations of Deep RTS are deterministic and non-durative. Actions in the non-durative configuration are directly applied to the environment within the next few game frames. This makes the correlation between action and reward easier to observe. The durative configuration complicates the state-space significantly because it then becomes a temporal problem that requires long-term planning. Deep RTS supports the OpenAI Gym abstraction through the Python API and is a promising tool for reinforcement learning research.

4.1 Game Objective

The objective of the Deep RTS challenge is to build a base consisting of a town-hall, and then strive to expand the base using gathered resources to gain the military upper hand. Military units are used to conduct attacks where the primary goal is to demolish the base of the opponent. Players start with a worker unit. The primary objective of the worker units is to expand the base offensive, defensive and to gather natural resources found throughout the game world. Buildings can further spawn additional units that strengthen the offensive capabilities of the player. For a player to reach the terminal state, all opponent units must be destroyed.

A regular RTS game can be represented in three stages: early-game, mid-game and late-game. Early-game is the gathering and base expansion stage. The mid-game focuses on the military and economic superiority, while the late-game stage is usually a deathmatch between the players until the game ends.

Table 2: An overview of available scenarios found in the Deep RTS game environment

Scenario Name	Description	Game Length	Map Size
10x10-2-FFA	2-Player game	600-900 ticks	10x10
15x15-2-FFA	2-Player game	900-1300 ticks	15x15
21x21-2-FFA	2-Player game	2000-3000 ticks	21x21
31x31-2-FFA	2-Player game	6000-9000 ticks	31x31
31x31-4-FFA	4-Player game	8000-11k ticks	31x31
31x31-6-FFA	6-Player game	15k-20k ticks	31x31
solo-score	Score Accumulation	1200 ticks	10x10
solo-resources	Resource Harvesting	600 ticks	10x10
solo-army	Army Accumulation	1200 ticks	10x10

Because Deep RTS targets a various range of reinforcement learning tasks, there are game scenarios such as resource gathering tasks, military tasks, and defensive tasks that narrows the complexity of a full RTS game. Table 2 shows nine scenarios currently implemented in the Deep RTS game environment. The first six scenarios are regular RTS games with the possibility of having 6 active players in a free-for-all setting. The *solo-score* scenario features an environment where the objective is to only generate as much score as possible in shortest amount of time. *solo-resources* is a game mode that focuses on resource gathering. The agent must find a balance between base expansion and resource gathering to optimally gather as many resources as possible. *solo-army* is a scenario where the primary goal is to expand the military forces quickly and launch an attack on an idle enemy. The Deep RTS game environment enables researchers to create custom scenarios via a flexible configuration interface.

4.2 Game Mechanics

The game mechanics of the Deep RTS are flexible and can be adjusted before a game starts. Table 3 shows a list of configurations currently available. An important design choice is to allow actions to affect the environment without any temporal delay. All actions are bound to a tick-timer that defaults to 10, that is, it takes 10 ticks for a unit to move one tile, 10 ticks for a unit to attack once, and 300 ticks to build buildings. The tick-timer also includes a multiplier that enables adjustments of how many ticks equals a second. For each iteration of the game-loop, the tick counter is incremented, and the tick-

Table 3: Central configuration flags for the Deep RTS game engine

Config Name	Type	Description
instant_town_hall	Bool	Spawn Town-Hall at game start.
instant_building	Bool	Non-durative Build Mode.
instant_walking	Bool	Non-durative Walk Mode.
harvest_forever	Bool	Harvest resources automatically.
auto_attack	Bool	Automatic retaliation when being attacked.
durative	Bool	Enable durative mode.

timers are evaluated. By using tick-timers, the game-state resembles how the StarCraft II game mechanics function while lowering the tick-timer value better resembles microRTS.

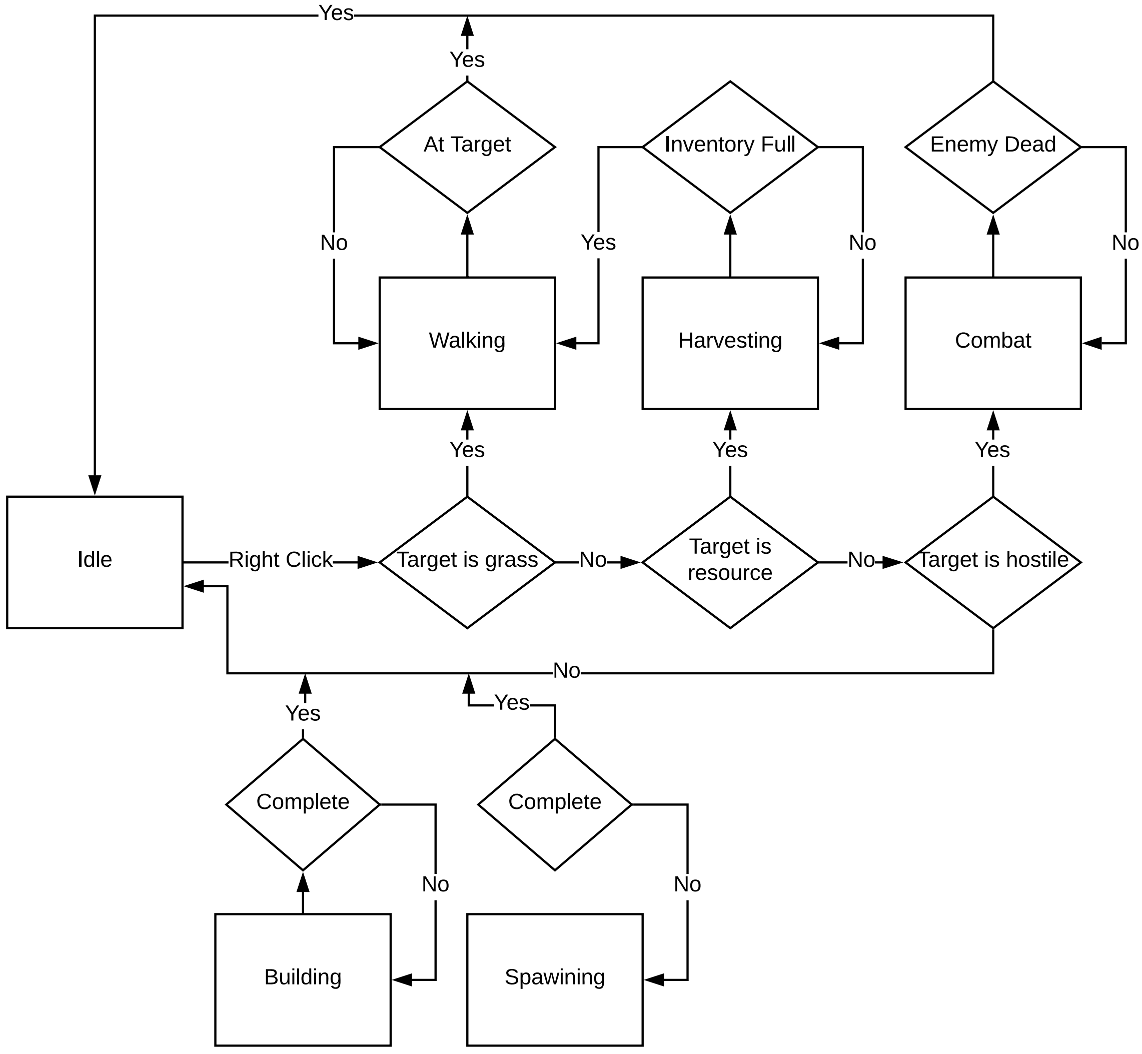


Figure 1: Unit state evaluation based on actions and current state

All game entities (Units and Buildings) have a state-machine that determine its current state. Figure 1 illustrates a portion of the logic that is evaluated through the state-machine. Entities start in the Spawning state transitioning to the Idle state when the entity spawn process is complete. The Idle state can be considered the default state of all entities and is only transitioned from when the player interacts with the entity. This implementation enables researchers to modify the state-transitions to produce alternative game logic.

Table 4: The available economic resources and limits available to players in Deep RTS

Player Resources					
Property	**Lumber**	**Gold**	**Oil**	**Food**	**Units**
Range	0 - 10^6	0 - 10^6	0 - 10^6	0 - 6000	0 - 2000

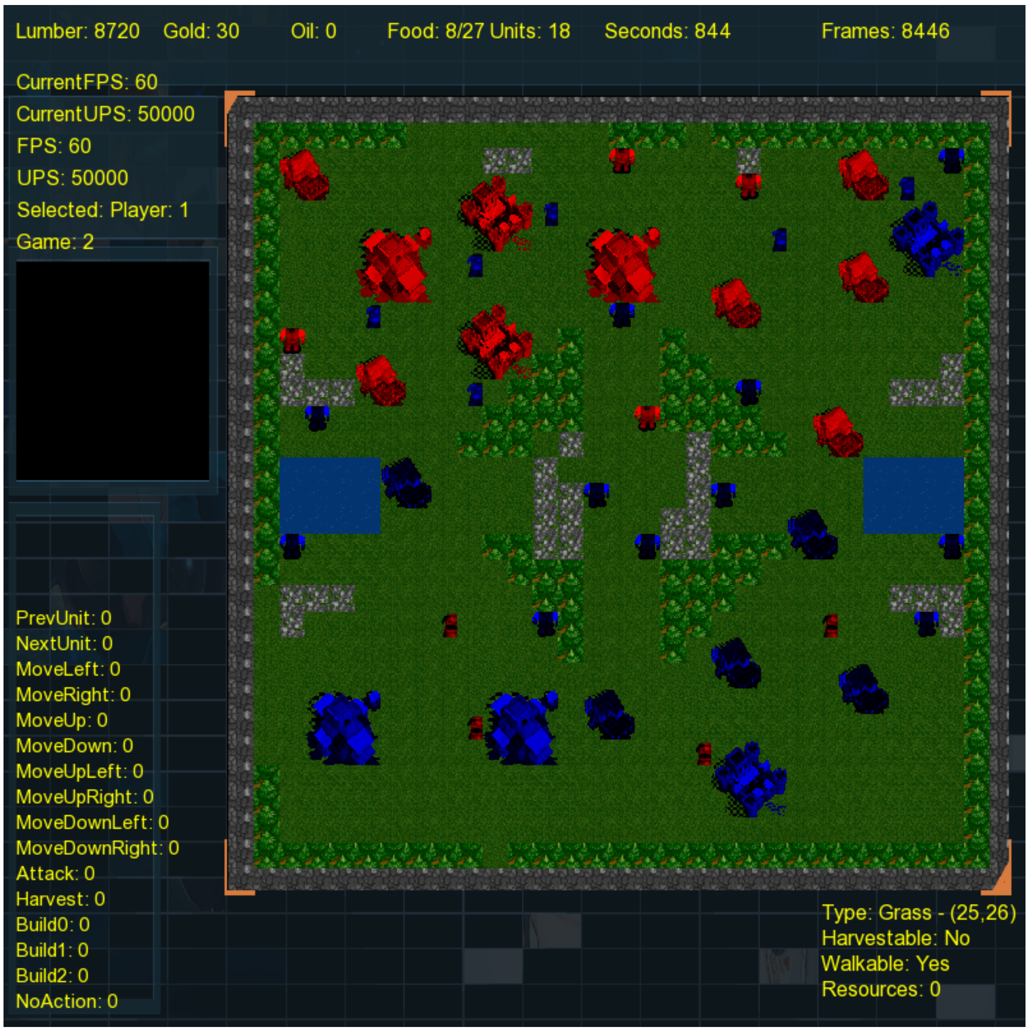
Table 4 shows the available resources and unit limits in the Deep RTS game environment. There are primarily three resources, gold, lumber, and oil that are available for workers to harvest. The value range is practically limited to the number of resources that exist on the game map. The food limit and the unit limit ensures that the player does not produce units excessively.

4.3 Graphics

The Deep RTS game engine features two graphical interface modes in addition to the headless mode that is used by default. The primary graphical interface relies on Python while the second is implemented in C++. The Python version is not interactive and can only render the raw game-state as an image. By using software rendering, the capture process of images is significantly faster because the copy between GPU and CPU is slow. The C++ implementation, seen in Figure 2 is fully interactive, enabling manual play of Deep RTS. Figure 3 shows how the raw game-state is represented as a 3-D matrix in headless mode. Deep learning methods often favor raw game-state data instead of image representation as sensory input. This is because raw data is often more concrete with clear patterns.

4.4 Action-space definition

The action-space of the Deep RTS game environment is separated into two abstract levels. The first level is actions that directly impact the environment, for instance, right-click, left-click, move-left, and select-unit. The next layer of abstraction is actions that combine actions from the previous layer, typically *select-unit* $\rightarrow$ *right-click* $\rightarrow$ *right-click* $\rightarrow$ *move-left*. The benefit of this abstraction is that algorithms can focus on specific areas within the game-state, and enable to build hierarchical models that each specialize in tasks (planning). The Deep RTS initially features 16 different actions in the first layer and 6 actions in the last abstraction layer, but it is trivial to add additional actions.



Lumber: 8720 Gold: 30 Oil: 0 Food: 8/27 Units: 18 Seconds: 844 Frames: 8446

CurrentFPS: 60
CurrentUPS: 50000
FPS: 60
UPS: 50000
Selected: Player: 1
Game: 2

PrevUnit: 0
NextUnit: 0
MoveLeft: 0
MoveRight: 0
MoveUp: 0
MoveDown: 0
MoveUpLeft: 0
MoveUpRight: 0
MoveDownLeft: 0
MoveDownRight: 0
Attack: 0
Harvest: 0
Build0: 0
Build1: 0
Build2: 0
NoAction: 0

Type: Grass - (25,26)
Harvestable: No
Walkable: Yes
Resources: 0

Figure 2: Overview of a battle in the fully-observable Deep RTS state-space using the C++ graphical user interface

4.5 Summary

This section presents some of the central parts what the Deep RTS game environment features for reinforcement learning research. It is designed to measure the performance of algorithms accurately having a standardized API through OpenAI Gym, which is widely used in the reinforcement learning community.

5 Experiments

5.1 Performance considerations in Deep RTS

The goal of Deep RTS is to simulate RTS scenarios with ultra high-performance accurately. The performance is measured by how fast the game engine updates the game-state, and how quickly the game-state can be represented as an image. Some experiments sug-

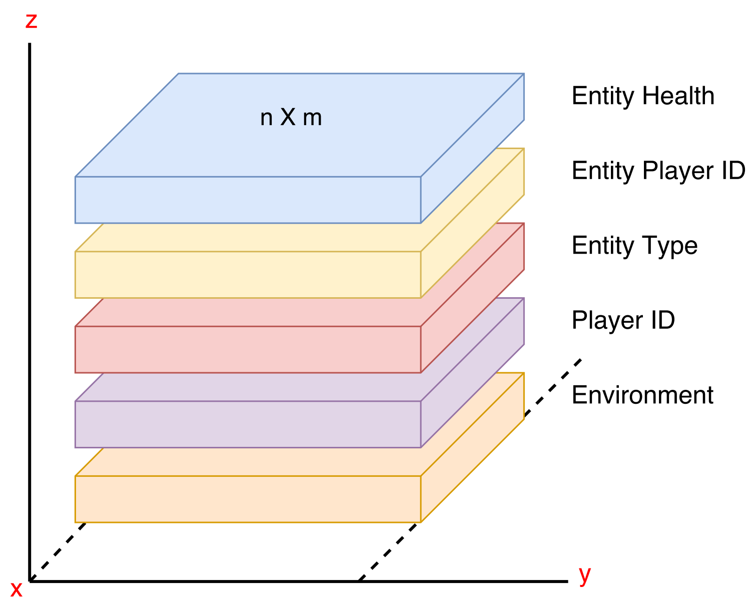


Figure 3: Illustration of how the raw state is represented using 3-D matrices

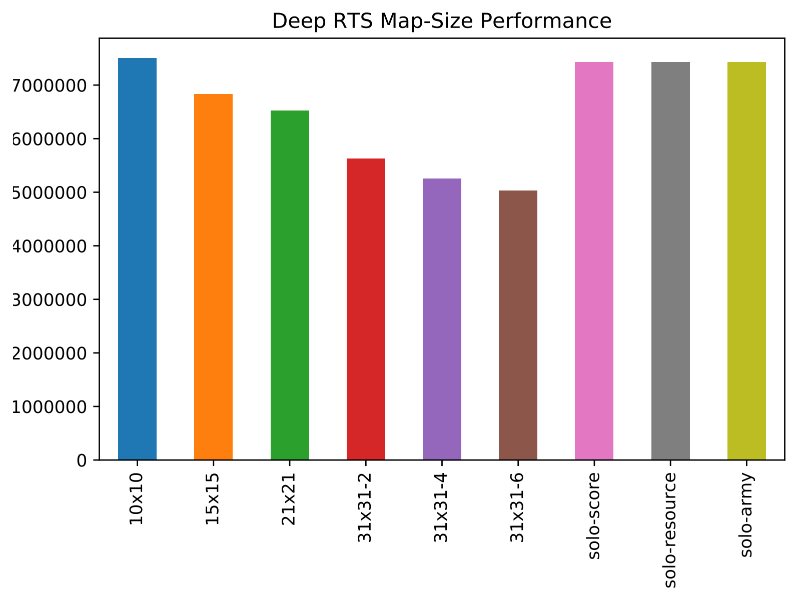
gest that it is beneficial to render game graphics on the CPU instead of the GPU. Because the GPU has a separate memory, there is a severe bottleneck when copying the screen buffer from the GPU to the CPU.

Figure 4a shows the correlation between the frame-rate and size of the game map. Observing the data, it is clear that the map-size has O(n) penalty to the frame-rate performance. It is vital to preserve this linearity, and optimally have the constant performance of O(1) per game update. Figure 4 extends this benchmark by testing the impact a unit has on the game performance, averaging 1 000 games for all map-sizes. The data indicates that entities have an exponential impact on the frame-rate performance. The reason for this is primarily the jump-point-search algorithm used for unit path-finding. The path-finding algorithm can be disabled using custom configurations.

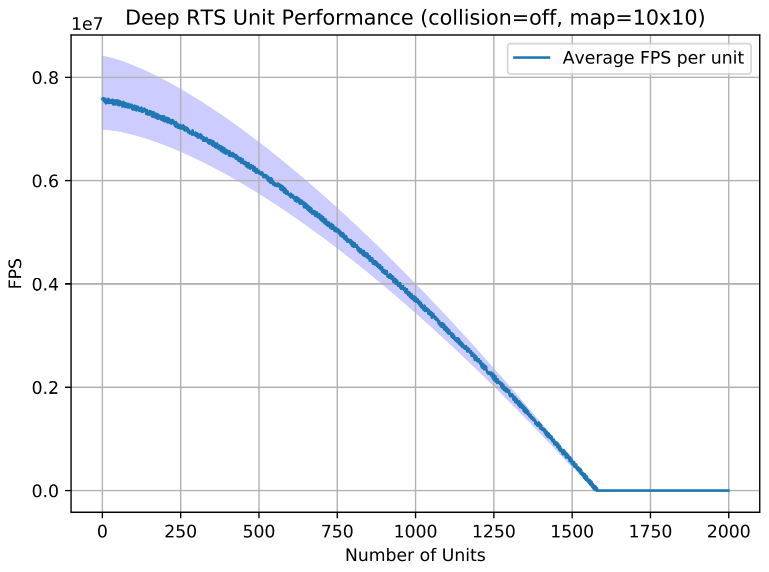
The Deep RTS game environment is high-performance, with few elements that significantly reduce the frame-rate performance. While some mechanics, namely path-finding is a significant portion of the update-loop it can be deactivated by configurations to optimize the performance further.

5.2 Comparing Deep RTS to existing learning environments

There is a substantial difference between the performance in games targeted research and those aimed towards gaming. Table 5 shows that the frame-rate difference ranges from 60 to 7 000 000 for selected environments. A high frame-rate is essential because some exploration algorithms often require a quick assessment of future states through forward-search. Table 5 shows that microRTS, ELF, and Deep RTS are superior in performance



(a) Correlation between FPS (Y-axis) and Map-sizes (X-axis)



(b) Correlation between FPS and Number of Units

Figure 4: FPS Performance in Deep RTS

compared to other game environments. Deep RTS is measured using the largest available map (Table 2) having a unit limit of 20 per player. This yields the performance of 24 000 updates-per-second. The Deep RTS game engine can also render the game state with up to 7 000 000 updates-per-second using the minimal configuration. This is a tremendous improvement on previous work and could enable algorithms with a limited time budget to do deeper tree-searches.

Table 5: Comparison of the FPS for selected environments. The Deep RTS benchmarks are performed using minimum and maximum configurations

Environment	Frame per second	Source
ALE	6,500	[10]
Malmo Platform	60-144	[11]
ViZDoom	8,300	[12]
DeepMind Lab	1,000	[13]
OpenAI Gym	60	[14]
OpenAI Universe	60	[15]
Stratagus	60-144	[16]
microRTS	11,500	[17]
TorchCraft	2,500	[18]
ELF	36,000	[19]
SC2LE	60-144	[8]
Deep RTS	24,000, 7,000,000	-

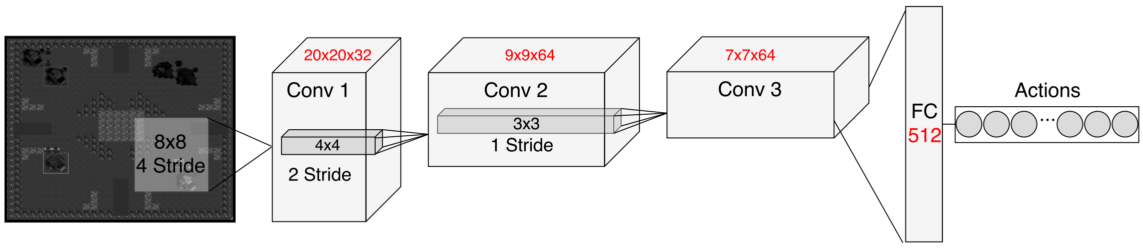


Figure 5: Overview of the Deep Q-Network architecture used in the experiments. Inspired by the work seen in [1]

5.3 Using Deep Q-Learning in Deep RTS

At the most basic level, Q-Learning utilizes a table for storing $(s, a, r, s^{'})$ pairs, where s is the states, a is the actions, r the rewards, and $s^{'}$ the next state. Instead, a non-linear function approximation can be used to approximate $Q(s, a; \theta)$. This is called **Deep-Q Learning**. θ describes the tunable parameters (weights) for the approximation function. Artificial neural networks are used as an approximation function for the Q-Table but at the cost of stability [3]. Using artificial neural networks is much like compression found in JPEG images. The compression is *lossy*, and some information is lost during the compression. Deep Q-Learning is thus unstable, since values may be incorrectly encoded during training [36].

This paper presents experimental results using the Deep Q-Learning architecture from [3, 37]. Figure 5 shows the network model, and figure 6 illustrates the averaged training loss of 100 agents. The agent uses gray-scale image game-state representations with an

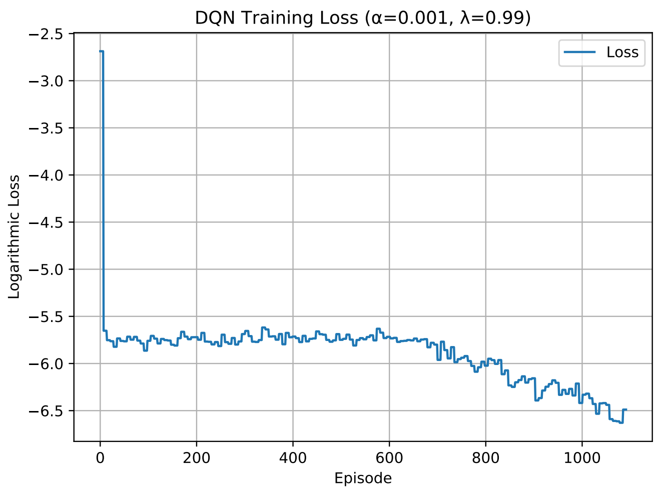


Figure 6: Training loss of the Deep Q-Network. Each episode consists of approximately 1 000 epochs.

additional convolutional layer to decrease the training time, but can also achieve comparable results after approximately 800 episodes of training with the exact architecture from [3][2]. The graph shows that the agent quickly learns the correlation between game-state, action and the reward function. The loss quickly stabilizes at a relatively low value, but it is likely that very small optimizations in the parameters have a significant impact on the agent's performance.

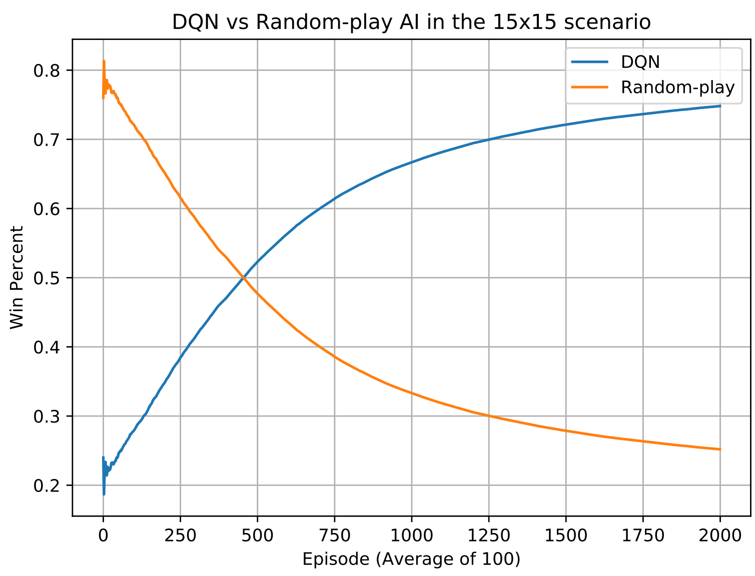
Figure 7a shows the win-rate against an AI with a random-play strategy. The agent quickly learns how to perform better than random behavior, and achieves 70 % win-rate at episode 1 250. Figure 7b illustrates the same agent playing against a rule-based strategy. The graph shows that the Deep Q-Network can achieve an average of 50 % win-rate over a 1 000 games. This strategy is considered an easy to moderate player, where its strategy is to expand the base towards the opponent and build a military force after approximately 600 seconds. Figure 2 shows how the rule-based player (blue) expands the base to gain the upper hand.

The experimental results presented in this paper show that the Deep RTS game environment can be used to train deep reinforcement learning algorithms. The Deep Q-Network does not achieve super-human expertise but performs similarly to a player of easy to moderate skill level, which is a good step towards a high-level AI.

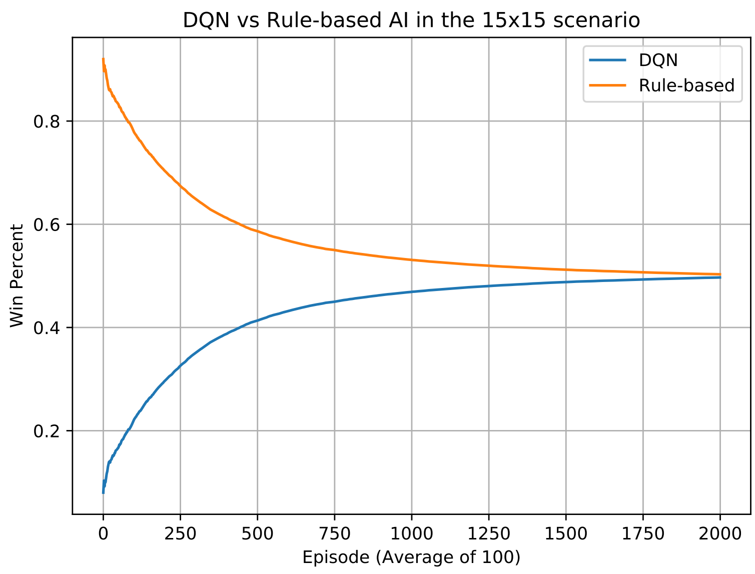
6 Conclusion and Future Work

This paper is a contribution towards the continuation of research into deep reinforcement learning for RTS games. The paper summarizes previous work and outlines the few but

[2]Each episode contains approximately 1 000 epochs of training with a batch size of 16



(a) DQN vs Random-play AI in the 15x15-2-FFA scenario



(b) DQN vs Rule-based AI in the 15x15-2-FFA scenario

Figure 7: Performance comparison of agents using Deep Q-Network, random-play, and rule-based strategies

essential success stories in reinforcement learning. The Deep RTS game environment is a high-performance RTS simulator that enables rapid research and testing of novel reinforcement learning techniques. It successfully fills the gap between the vital game simulator microRTS, and StarCraft II, which is the ultimate goal for reinforcement learning research for the RTS game genre.

The hope is that Deep RTS can bring insightful results to the complex problems of RTS [17] and that it can be a useful tool in future research.

Although the Deep RTS game environment is ready for use, several improvements can be applied to the environment. The following items are scheduled for implementation in the continuation of Deep RTS:

- Enable LUA developers to use Deep RTS through LUA bindings.
- Implement a generic interface for custom graphics rendering.
- Implement duplex WebSockets and ZeroMQ to enable any language to interact with Deep RTS
- Implement alternative path-finding algorithms to increase performance for some scenarios
- Add possibility for memory-based fog-of-war to better mimic StarCraft II

7 Acknowledgements

We would like to thank Santiago Ontanon at the University of Drexel for his excellent work on microRTS. microRTS has to us been a valuable tool for benchmarks in the reinforcement learning domain and continues to be the goto environment for research in tree-search algorithms.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing Atari with Deep Reinforcement Learning,” *arxiv preprint arXiv:1312.5602*, dec 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[2] W. Gaetz, S. K. Kessler, T. P. Roberts, J. I. Berman, T. J. Levy, M. Hsia, D. Humpl, E. S. Schwartz, S. Amaral, B. Chang, and L. S. Levin, “Massive cortical reorganization is reversible following bilateral transplants of the hands: evidence from the first successful bilateral pediatric hand transplant patient,” *Annals of Clinical and Translational Neurology*, vol. 5, no. 1, pp. 92–97, dec 2018. [Online]. Available: https://arxiv.org/abs/1712.01815

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, feb 2015. [Online]. Available: http://www.nature.com/articles/nature14236

[4] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, D. Kumaran, and R. Hadsell, "Learning to Navigate in Complex Environments," *arXiv preprint arXiv:1611.03673*, nov 2016. [Online]. Available: http://arxiv.org/abs/1611.03673

[5] M. A. Yasin, W. A. Al-Ashwal, A. M. Shire, S. A. Hamzah, and K. N. Ramli, "Tri-band planar inverted F-antenna (PIFA) for GSM bands and bluetooth applications," *ARPN Journal of Engineering and Applied Sciences*, vol. 10, no. 19, pp. 8740–8744, 2015.

[6] G. Konidaris and A. Barto, "Autonomous shaping," *Proceedings of the 23rd international conference on Machine learning - ICML '06*, pp. 489–496, 2006. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1143844.1143906

[7] G. Tesauro, "TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play," *Neural Computation*, vol. 6, no. 2, pp. 215–219, 1994. [Online]. Available: http://www.mitpressjournals.org/doi/10.1162/neco.1994.6.2.215

[8] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, J. Quan, S. Gaffney, S. Petersen, K. Simonyan, T. Schaul, H. van Hasselt, D. Silver, T. Lillicrap, K. Calderone, P. Keet, A. Brunasso, D. Lawrence, A. Ekermo, J. Repp, and R. Tsing, "StarCraft II: A New Challenge for Reinforcement Learning," *Proceedings, The Thirteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, pp. 86–92, 2017. [Online]. Available: http://arxiv.org/abs/1708.04782

[9] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[10] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *IJCAI International Joint Conference on Artificial Intelligence*, vol. 2015-Janua, pp. 4148–4152, 2015. [Online]. Available: http://arxiv.org/abs/1207.4708

[11] M. Johnson, K. Hofmann, T. Hutton, and D. Bignell, "The malmo platform for artificial intelligence experimentation," *IJCAI International Joint Conference on*

Artificial Intelligence, vol. 2016-Janua, pp. 4246–4247, 2016. [Online]. Available: https://www.ijcai.org/Proceedings/16/Papers/643.pdf

[12] E. Perot, M. Jaritz, M. Toromanoff, and R. D. Charette, "End-to-End Driving in a Realistic Racing Game with Deep Reinforcement Learning," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, vol. 2017-July, pp. 474–475, may 2017. [Online]. Available: http://arxiv.org/abs/1605.02097

[13] C. Beattie, J. Z. Leibo, D. Teplyashin, T. Ward, M. Wainwright, H. Küttler, A. Lefrancq, S. Green, V. Valdés, A. Sadik, J. Schrittwieser, K. Anderson, S. York, M. Cant, A. Cain, A. Bolton, S. Gaffney, H. King, D. Hassabis, S. Legg, and S. Petersen, "DeepMind Lab," *arXiv preprint arXiv:1612.03801*, dec 2016. [Online]. Available: http://arxiv.org/abs/1612.03801

[14] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *arXiv preprint arXiv:1606.01540*, jun 2016. [Online]. Available: http://arxiv.org/abs/1606.01540

[15] OpenAI, "OpenAI Universe," 2017. [Online]. Available: https://universe.openai.com/

[16] M. Ponsen, S. Lee-Urban, H. Muñoz-Avila, D. Aha, and M. Molineaux, "Stratagus: An open-source game engine for research in real-time strategy games," *Reasoning, Representation, and Learning in Computer Games*, no. Code 5515, p. 78, 2005. [Online]. Available: https://pdfs.semanticscholar.org/d005/87160d3b37c70f238ebd92c71454479e829e.pdf

[17] S. Ontanon, "The combinatorial multi-armed bandit problem and its application to real-time strategy games," in *Ninth Artificial Intelligence and Interactive Digital . . .*, 2013, pp. 58–64. [Online]. Available: http://www.aaai.org/ocs/index.php/AIIDE/AIIDE13/paper/viewPaper/7377

[18] G. Synnaeve, N. Nardelli, A. Auvolat, S. Chintala, T. Lacroix, Z. Lin, F. Richoux, and N. Usunier, "TorchCraft: a Library for Machine Learning Research on Real-Time Strategy Games," *arxiv preprint arXiv:1611.00625*, nov 2016. [Online]. Available: http://arxiv.org/abs/1611.00625

[19] Y. Tian, Q. Gong, W. Shang, Y. Wu, and C. L. Zitnick, "ELF: An Extensive, Lightweight and Flexible Research Platform for Real-time Strategy Games," *Advances in Neural Information Processing Systems*, pp. 2656–2666, jul 2017. [Online]. Available: http://arxiv.org/abs/1707.01067

[20] S. Ontañón, K. Mishra, N. Sugandh, and A. Ram, “Learning from demonstration and case-based planning for real-time strategy games,” in *Studies in Fuzziness and Soft Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, vol. 226, pp. 293–310. [Online]. Available: http://link.springer.com/10.1007/978-3-540-77465-5{_}15

[21] I. Fathy, M. Aref, O. Enayet, and A. Al-Ogail, “Intelligent online case-based planning agent model for real-time strategy games,” in *Proceedings of the 2010 10th International Conference on Intelligent Systems Design and Applications, ISDA’10*. IEEE, nov 2010, pp. 445–450. [Online]. Available: http://ieeexplore.ieee.org/document/5687225/

[22] U. Jaidee and H. Muñoz-Avila, “CLASSQ-L: A Q-Learning Algorithm for Adversarial Real-Time Strategy Games,” pp. 8–13, 2012. [Online]. Available: http://www.aaai.org/ocs/index.php/AIIDE/AIIDE12/paper/viewFile/5515/5734

[23] B. Lindström, I. Selbing, T. Molapour, and A. Olsson, “Racial Bias Shapes Social Reinforcement Learning,” *Psychological Science*, vol. 25, no. 3, pp. 711–719, feb 2014. [Online]. Available: http://arxiv.org/abs/1602.01783

[24] N. A. Barriga, M. Stanescu, and M. Buro, “Game Tree Search Based on Non-Deterministic Action Scripts in Real-Time Strategy Games,” *IEEE Transactions on Computational Intelligence and AI in Games*, pp. 1–1, 2017. [Online]. Available: http://ieeexplore.ieee.org/document/7954767/

[25] A. Shleyfman, A. Komenda, and C. Domshlak, “On combinatorial actions and CMABs with linear side information,” in *Frontiers in Artificial Intelligence and Applications*, vol. 263, 2014, pp. 825–830.

[26] D. Churchill, Z. Lin, and G. Synnaeve, “An Analysis of Model-Based Heuristic Search Techniques for StarCraft Combat Scenarios,” pp. 8–14, 22017. [Online]. Available: https://aaai.org/ocs/index.php/AIIDE/AIIDE17/paper/view/15916

[27] P. Peng, Y. Wen, Y. Yang, Q. Yuan, Z. Tang, H. Long, and J. Wang, “Multiagent Bidirectionally-Coordinated Nets: Emergence of Human-level Coordination in Learning to Play StarCraft Combat Games,” *arxiv preprint arXiv:1703.10069*, mar 2017. [Online]. Available: http://arxiv.org/abs/1703.10069

[28] Z. Lin, J. Gehring, V. Khalidov, and G. Synnaeve, “STARDATA: A StarCraft AI Research Dataset,” *Proceedings, The Thirteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, aug 2017. [Online]. Available: http://arxiv.org/abs/1708.02139

[29] E. Perot, M. Jaritz, M. Toromanoff, and R. D. Charette, "End-to-End Driving in a Realistic Racing Game with Deep Reinforcement Learning," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, vol. 2017-July, pp. 474–475, may 2017. [Online]. Available: https://arxiv.org/abs/1605.02097

[30] Y. Li, "Deep Reinforcement Learning: An Overview," *arXiv preprint arXiv:1701.07274*, pp. 1–30, 2017. [Online]. Available: http://arxiv.org/abs/1701.07274

[31] T. Mannucci, E. J. van Kampen, C. de Visser, and Q. Chu, "Safe Exploration Algorithms for Reinforcement Learning Controllers," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 16, pp. 1437–1480, 2017. [Online]. Available: http://www.jmlr.org/papers/volume16/garcia15a/garcia15a.pdf

[32] G. Tesauro, "Temporal difference learning and TD-Gammon," *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995. [Online]. Available: http://portal.acm.org/citation.cfm?doid=203330.203343

[33] R. S. Sutton and A. G. Barto, "Chapter 12: Introductions," *Acta Physiologica Scandinavica*, vol. 48, no. Mowrer 1960, pp. 57–63, 1960.

[34] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. Van Den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[35] M. Moravčík, M. Schmid, N. Burch, V. Lisý, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, and M. Bowling, "DeepStack: Expert-level artificial intelligence in heads-up no-limit poker," *Science*, vol. 356, no. 6337, pp. 508–513, jan 2017. [Online]. Available: http://arxiv.org/abs/1701.01724http://dx.doi.org/10.1126/science.aam6960

[36] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, vol. abs/1509.0, sep 2015. [Online]. Available: http://arxiv.org/abs/1509.02971

[37] W. Chen, M. Zhang, Y. Zhang, and X. Duan, "Exploiting meta features for dependency parsing and part-of-speech tagging," *Artificial Intelligence*, vol. 230, pp. 173–191, sep 2016. [Online]. Available: http://arxiv.org/abs/1509.06461

Paper C: Deep RTS: A Game Environment for Deep Reinforcement Learning in Real-Time Strategy Games

Paper D

Title:	The Dreaming Variational Autoencoder for Reinforcement Learning Environments
Authors:	Andersen, Per-Arne et al.
Affiliation:	Department of ICT, University of Agder, Grimstad Norway
Conference:	38th SGAI International Conference on Artificial Intelligence
Year:	2018, December

The Dreaming Variational Autoencoder for Reinforcement Learning Environments

Per-Arne Andersen
Department of ICT
University of Agder
Grimstad, Norway
per.andersen@uia.no

Morten Goodwin
Department of ICT
University of Agder
Grimstad, Norway
morten.goodwin@uia.no

Ole-Christoffer Granmo
Department of ICT
University of Agder
Grimstad, Norway
ole.granmon@uia.no

Abstract

Reinforcement learning has shown great potential in generalizing over raw sensory data using only a single neural network for value optimization. There are several challenges in the current state-of-the-art reinforcement learning algorithms that prevent them from converging towards the global optima. It is likely that the solution to these problems lies in short- and long-term planning, exploration and memory management for reinforcement learning algorithms. Games are often used to benchmark reinforcement learning algorithms as they provide a flexible, reproducible, and easy to control environment. Regardless, few games feature a state-space where results in exploration, memory, and planning are easily perceived. This paper presents *The Dreaming Variational Autoencoder* (DVAE), a neural network based generative modeling architecture for exploration in environments with sparse feedback. We further present Deep Maze, a novel and flexible maze engine that challenges DVAE in partial and fully-observable state-spaces, long-horizon tasks, and deterministic and stochastic problems. We show initial findings and encourage further work in reinforcement learning driven by generative exploration.

Thirty-eight SGAI International Conference on Artificial Intelligence (AI-2018), Cambridge England
Submitted 04. Jul, 2018, Accepted 10. Sep 2018, Published 16. Nov, 2018
Awarded the prize for student-paper in the technical stream of SGAI-2018

Keywords: Deep Reinforcement Learning, Environment Modeling, Neural Networks, Variational Autoencoder, Markov Decision Processes, Exploration, Artificial Experience-Replay

1 Introduction

Reinforcement learning (RL) is a field of research that has quickly become one of the most promising branches of machine learning algorithms to solve artificial general intelligence [2, 10, 12, 16]. There have been several breakthroughs in reinforcement learning in recent years for relatively simple environments [6, 14, 15, 21], but no algorithms are capable of human performance in situations where complex policies must be learned. Due to this, a number of open research questions remain in reinforcement learning. It is possible that many of the problems can be resolved with algorithms that adequately accounts for planning, exploration, and memory at different time-horizons.

In current state-of-the-art RL algorithms, long-horizon RL tasks are difficult to master because there is as of yet no optimal exploration algorithm that is capable of proper state-space pruning. Exploration strategies such as ϵ-greedy is widely used in RL, but cannot find an adequate exploration/exploitation balance without significant hyperparameter-tuning. Environment modeling is a promising exploration technique where the goal is for the model to imitate the behavior of the target environment. This limits the required interaction with the target environment, enabling nearly unlimited access to exploration without the cost of exhausting the target environment. In addition to environment-modeling, a balance between exploration and exploitation must be accounted for, and it is, therefore, essential for the environment model to receive feedback from the RL agent.

By combining the ideas of variational autoencoders with deep RL agents, we find that it is possible for agents to learn optimal policies using only generated training data samples. The approach is presented as the dreaming variational autoencoder. We also show a new learning environment, Deep Maze, that aims to bring a vast set of challenges for reinforcement learning algorithms and is the environment used for testing the DVAE algorithm.

This paper is organized as follows. Section 3 briefly introduces the reader to preliminaries. Section 4 proposes *The Dreaming Variational Autoencoder* for environment modeling to improve exploration in RL. Section 5 introduces the Deep Maze learning environment for exploration, planning and memory management research for reinforcement learning. Section 6 shows results in the Deep Line Wars environment and that RL agents can be trained to navigate through the deep maze environment using only artificial training data.

2 Related Work

In machine learning, the goal is to create an algorithm that is capable of constructing a model of some environment accurately. There is, however, little research in *game* environment modeling in the scale we propose in this paper. The primary focus of recent RL research has been on the value and policy aspect of RL algorithm, while less attention has been put into perfecting environment modeling methods.

In 2016, the work in [3] proposed a method of deducing the Markov Decision Process (MDP) by introducing an adaptive exploration signal (pseudo-reward), which was obtained using deep generative model. Their method was to compute the Jacobian of each state and used it as the pseudo-reward when using deep neural networks to learn the state-generalization.

Xiao et al. proposed in [22] the use of generative adversarial networks (GAN) for model-based reinforcement learning. The goal was to utilize GAN for learning dynamics of the environment in a short-horizon timespan and combine this with the strength of far-horizon value iteration RL algorithms. The GAN architecture proposed illustrated near authentic generated images giving comparable results to [14].

In [9] Higgins et al. proposed DARLA, an architecture for modeling the environment using β-VAE [8]. The trained model was used to extract the optimal policy of the environment using algorithms such as DQN [15], A3C [13], and Episodic Control [4]. DARLA is to the best of our knowledge, the first algorithm to properly introduce learning without access to the target environment during training.

Buesing et al. recently compared several methods of environment modeling, showing that it is far better to model the state-space then to utilize Monte-Carlo rollouts (RAR). The proposed architecture, state-space models (SSM) was significantly faster and produced acceptable results compared to auto-regressive (AR) methods. [5]

Ha and Schmidhuber proposed in [7] *World Models*, a novel architecture for training RL algorithms using variational autoencoders. This paper showed that agents could successfully learn the environment dynamics and use this as an exploration technique requiring no interaction with the target domain.

3 Background

We base our work on the well-established theory of reinforcement learning and formulate the problem as a MDP [20]. An MDP contains $(\mathcal{S}, \mathcal{A}, \mathcal{T}, r)$ pairs that define the environment as a model. The state-space, $\mathcal{S}$ represents all possible states while the action-space, $\mathcal{A}$ represents all available actions the agent can perform in the environment. $\mathcal{T}$ denotes

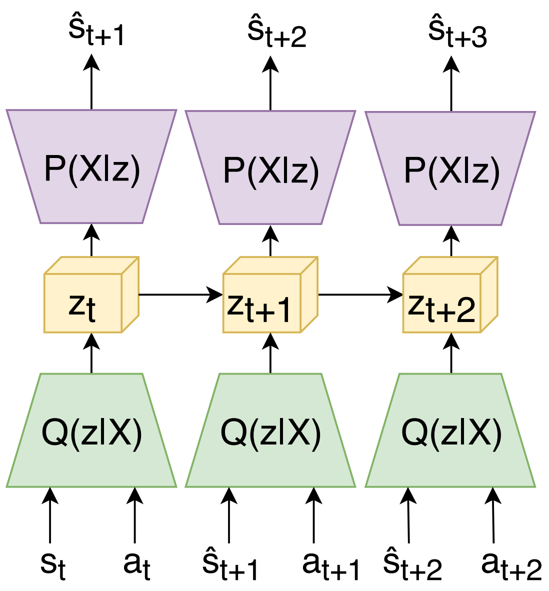


Figure 1: Illustration of the DVAE model. The model consumes state and action pairs, yielding the input encoded in latent-space. Latent-space can then be decoded to a probable future state. $\mathcal{Q}(z|X)$ is the encoder, z_t is latent-space, and $\mathcal{P}(X|z)$ is the decoder. DVAE can also use LSTM to better learn longer sequences in continuous state-spaces.

the transition function ($\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$), which is a mapping from state $s_t \in \mathcal{S}$ and action $a_t \in \mathcal{A}$ to the future state s_{t+1}. After each performed action, the environment dispatches a reward signal, $\mathcal{R} : \mathcal{S} \rightarrow r$.

We call a sequence of states and actions a *trajectory* denoted as
$\tau = (s_0, a_0, \dots, s_t, a_t)$ and the sequence is sampled through the use of a stochastic policy that predicts the optimal action in any state: $\pi_\theta(a_t|s_t)$, where π is the policy and θ are the parameters. The primary goal of the reinforcement learning is to *reinforce* good behavior. The algorithm should try to learn the policy that maximizes the total expected discounted reward given by, $\mathcal{J}(\pi) = \mathbb{E}_{(s_t,a_t)\sim p(\pi)}\left[\sum_{i=0}^{T} \gamma^i \mathcal{R}(s_i)\right]$ [15].

4 The Dreaming Variational Autoencoder

The Dreaming Variational Autoencoder (DVAE) is an end-to-end solution for generating probable future states $\hat{s}_{t+n}$ from an arbitrary state-space $\mathcal{S}$ using state-action pairs explored prior to s_{t+n} and a_{t+n}.

The DVAE algorithm, seen in Figure 1 works as follows. First, the agent collects experiences for utilizing experience-replay in the *Run-Agent* function. At this stage, the agent explores the state-space guided by a Gaussian distributed policy. The agent acts, observes, and stores the observations into the experience-replay buffer $\mathcal{D}$. After the agent reaches

Algorithm 1 The Dreaming Variational Autoencoder

```
 1: Initialize replay memory $\mathcal{D}$ and $\hat{\mathcal{D}}$ to capacity $\mathcal{N}$
 2: Initialize policy $\pi_\theta$
 3: function RUN-AGENT($\mathcal{T}$, $\mathcal{D}$)
 4:     for i = 0 to N_EPISODES do
 5:         Observe starting state, $s_0 \sim \mathcal{N}(0, 1)$
 6:         while $s_t$ not TERMINAL do
 7:             $a_t \leftarrow \pi_\theta(s_t = s)$
 8:             $s_{t+1}, r_t, terminal_t \leftarrow \mathcal{T}(s_t, a_t)$
 9:             Store experience into replay buffer $\mathcal{D}(s_t, a_t, r_t, s_{t+1}, terminal_t)$
10:             $s_t \leftarrow s_{t+1}$
11:         end while
12:     end for
13: end function
14: Initialize encoder $\mathcal{Q}(z|X)$
15: Initialize decoder $\mathcal{P}(X|z)$
16: Initialize DVAE model $\mathcal{T}_\theta = \mathcal{P}(X|\mathcal{Q}(z|X))$
17: function DVAE
18:     for $d_i$ in D do
19:         $s_t, a_t, r_t, s_{t+1} \leftarrow d_i$                    ▷ Expand replay buffer pair
20:         $X_t \leftarrow s_t, a_t$
21:         $z_t \leftarrow \mathcal{Q}(X_t)$                          ▷ Encode $X_t$ into latent-space
22:         $\hat{s}_{t+1} \leftarrow \mathcal{P}(z_t)$                ▷ Decode $z_t$ into probable future state
23:         Store experience into artificial replay buffer $\hat{\mathcal{D}}(\hat{s}_t, a_t, r_t, \hat{s}_{t+1}, terminal_t)$
24:         $\hat{s}_t = \hat{s}_{t+1}$
25:     end for
26:     return $\hat{\mathcal{D}}$
27: end function
```

terminal state, the DVAE algorithm encodes state-action pairs from the replay-buffer D into probable future states. This is stored in the replay-buffer for artificial future-states $\hat{D}$.

Table 1: DVAE algorithm for generating states using $\mathcal{T}_\theta$ versus the real transition function $\mathcal{T}$. First, a real state is collected from the replay-memory. DVAE can then produce new states from current the trajectory τ using the state-action pairs. θ represent the trainable model parameters.

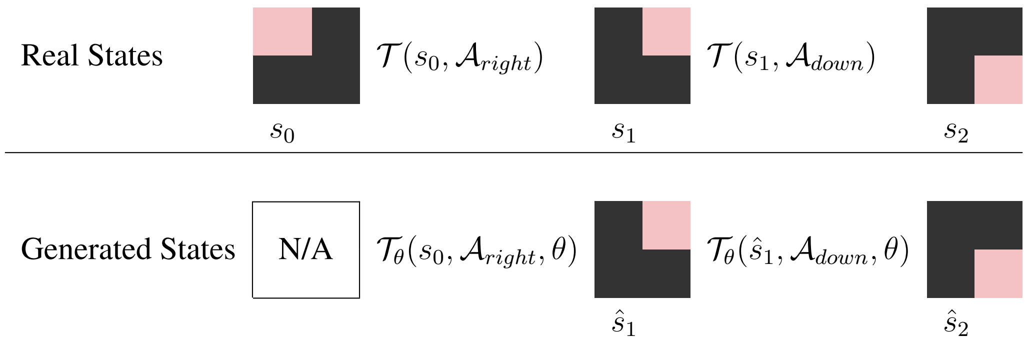
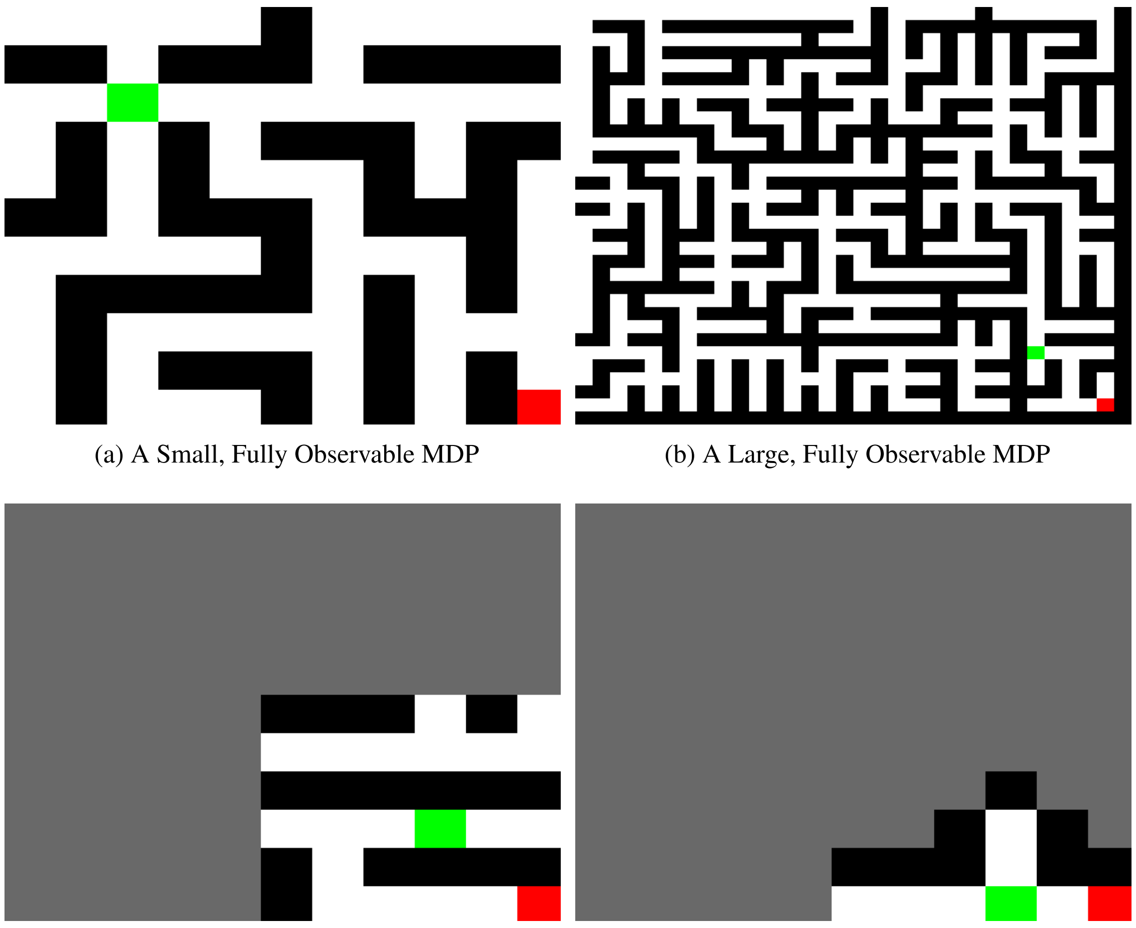

Real States
s_0
$\mathcal{T}(s_0, \mathcal{A}_{right})$
s_1
$\mathcal{T}(s_1, \mathcal{A}_{down})$
s_2
Generated States
N/A
$\mathcal{T}_\theta(s_0, \mathcal{A}_{right}, \theta)$
$\hat{s}_1$
$\mathcal{T}_\theta(\hat{s}_1, \mathcal{A}_{down}, \theta)$
$\hat{s}_2$

Table 1 illustrates how the algorithm can generate sequences of artificial trajectories using $\mathcal{T}_\theta = \mathcal{P}(X|\mathcal{Q}(z|X))$, where $z = \mathcal{Q}(z|X)$ is the encoder, and $\mathcal{T}_\theta = \mathcal{P}(X|z)$ is the decoder. With state s_0 and action $\mathcal{A}_{right}$ as input, the algorithm generates state $\hat{s}_1$ which in the table can be observed is similar to the real state s_1. With the next input, $\mathcal{A}_{down}$, the DVAE algorithm generates the next state $\hat{s}_2$ which again can be observed to be equal to s_2. Note that this is without ever observing state s_1. Hence, the DVAE algorithm needs to be initiated with a state, e.g. s_0, and actions follows. It then generates (dreams) next states,

The requirement is that the environment must be partially discovered so that the algorithm can learn to behave similarly to the target environment. To predict a trajectory of three timesteps, the algorithm does nesting to generate the whole sequence: $\tau = \hat{s}_1, a_1, \hat{s}_2, a_2, \hat{s}_3, a_3 = \mathcal{T}_\theta(\mathcal{T}_\theta(\mathcal{T}_\theta(s_0, \mathcal{A}_{rnd}), \mathcal{A}_{rnd}), \mathcal{A}_{rnd})$. The algorithm does this well in early on, but have difficulties with long sequences beyond eight in continuous environments.

5 Environments

The DVAE algorithm was tested on two game environments. The first environment is Deep Line Wars [1], a simplified Real-Time Strategy game. We introduce Deep Maze, a flexible environment with a wide range of challenges suited for reinforcement learning research.


(a) A Small, Fully Observable MDP

(b) A Large, Fully Observable MDP

(c) Partially Observable MDP having a vision distance of 3 tiles

(d) Partially Observable MDP having ray-traced vision

Figure 2: Overview of four distinct MDP scenarios using Deep Maze.

5.1 The Deep Maze Environment

The Deep Maze is a flexible learning environment for controlled research in exploration, planning, and memory for reinforcement learning algorithms. Maze solving is a well-known problem, and is used heavily throughout the RL literature [20], but is often limited to small and fully-observable scenarios. The Deep Maze environment extends the maze problem to over 540 unique scenarios including Partially-Observable Markov Decision Processes (POMDP). Figure 2 illustrates a small subset of the available environments for Deep Maze, ranging from small-scale MDP's to large-scale POMDP's. The Deep Maze further features custom game mechanics such as relocated exits and dynamically changing mazes.

The game engine is modularized and has an API that enables a flexible tool set for third-party scenarios. This extends the capabilities of Deep Maze to support nearly all possible scenario combination in the realm of maze solving.[1]

5.1.1 State Representation

RL agents depend on sensory input to evaluate and predict the best action at current timestep. Preprocessing of data is essential so that agents can extract features from the input. For this reason, Deep Maze has built-in state representation for RGB Images, Grayscale Images, and raw state matrices.

5.1.2 Scenario Setup

The Deep Maze learning environment ships with four scenario modes: (1) Normal, (2) POMDP, (3) Limited POMDP, and (4) Timed Limited POMDP.

The first mode exposes a seed-based randomly generated maze where the state-space is modeled as an MDP. The second mode narrows the state-space observation to a configurable area around the player. In addition to radius based vision, the POMDP mode also features ray-tracing vision that better mimic the sight of a physical agent. The third and fourth mode is intended for memory research where the agent must find the goal in a limited number of time-steps. In addition to this, the agent is presented with the solution but fades after a few initial time steps. The objective is the for the agent to remember the solution to find the goal. All scenario setups have a variable map-size ranging between 2×2 and 56×56 tiles.

[1]The Deep Maze is open-source and publicly available at `https://github.com/CAIR/deep-maze`.

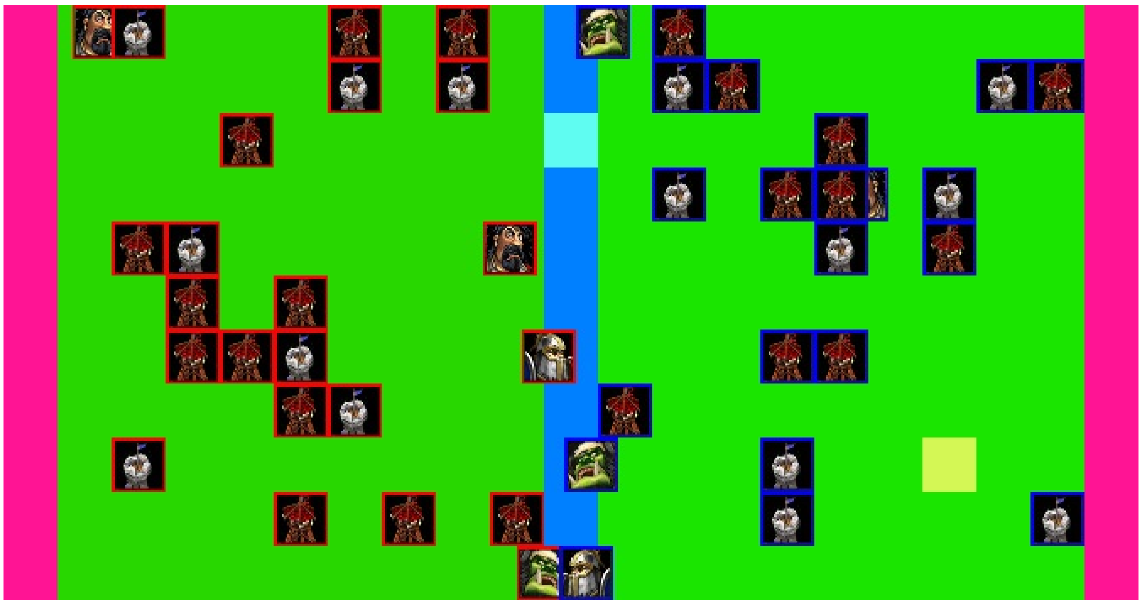


Figure 3: The Graphical User Interface of the Deep Line Wars environment.

5.2 The Deep Line Wars Environment

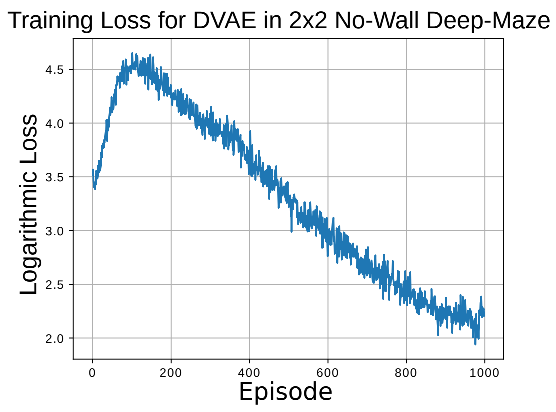
The Deep Line Wars environment was first introduced in [1]. Deep Line Wars is a real-time strategy environment that makes an extensive state-space reduction to enable swift research in reinforcement learning for RTS games.

The game objective of Deep Line Wars is to invade the enemy player with mercenary units until all health points are depleted, see Figure 3). For every friendly unit that enters the far edge of the enemy base, the enemy health pool is reduced by one. When a player purchases a mercenary unit, it spawns at a random location inside the edge area of the buyers base. Mercenary units automatically move towards the enemy base. To protect the base, players can construct towers that shoot projectiles at the opponents mercenaries. When a mercenary dies, a fair percentage of its gold value is awarded to the opponent. When a player sends a unit, the income is increased by a percentage of the units gold value. As a part of the income system, players gain gold at fixed intervals.

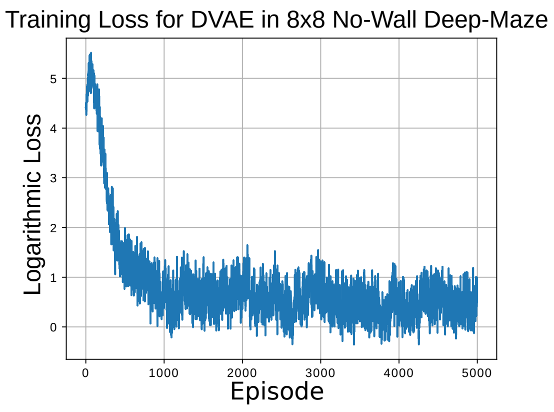
6 Experiments

6.1 Deep Maze Environment Modeling using DVAE

The DVAE algorithm must be able to generalize over many similar states to model a vast state-space. DVAE aims to learn the transition function, bringing the state from s_t to $s_{t+1} = \mathcal{T}(s_t, a_t)$. We use the deep maze environment because it provides simple rules,



(a) A Small, Fully Observable MDP



(b) A Large, Fully Observable MDP

Figure 4: The training loss for DVAE in the 2×2 No-Wall and 8×8 Deep Maze scenario. The experiment is run for a total of 1000 (5000 for 8×8) episodes. The algorithm only trains on 50% of the state-space to the model for the 2×2 environment while the whole state-space is trainable in the 8×8 environment.

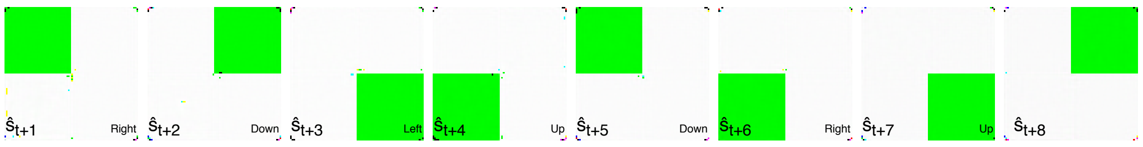


Figure 5: For the 2×2 scenario, only 50% of the environment is explored, leaving artifacts on states where the model is uncertain of the transition function. In more extensive examples, the player disappears, teleports or gets stuck in unexplored areas.

with a controllable state-space complexity. Also, we can omit the importance of reward for some scenarios.

We trained the DVAE model on two *No-Wall Deep Maze* scenarios of size 2×2 and 8×8. For the encoder and decoder, we used the same convolution architecture as proposed by [17] and trained for 5000 epochs for 8×8 and 1000 epochs for 2×2 respectively. For the encoding of actions and states, we concatenated the flattened state-space and action-space, having a fully-connected layer with ReLU activation before calculating the latent-space. We used the Adam optimizer [11] with a learning-rate of 1e-08 to update the parameters.

Figure 4 illustrates the loss of the DVAE algorithm in the No-Wall Deep Maze scenario. In the 2×2 scenario, DVAE is trained on only 50% of the state space, which results in noticeable graphic artifacts in the prediction of future states, see Figure 5. Because the 8×8 environment is fully visible, we see in Figure 6 that the artifacts are exponentially reduced.

Table 2: Results of the deep maze 11×11 and 21×21 environment, comparing DQN [15], TRPO [18], and PPO [19]. The optimal path yields performance of 100% while no solution yields 0%. Each of the algorithms ran 10000 episodes for both map-sizes. The last number represents at which episode the algorithm converged.

Algorithm	Avg Performance 11×11	Avg Performance 21×21
DQN-$\hat{\mathcal{D}}$	94.56% @ 9314	64.36% @ N/A
TRPO-$\hat{\mathcal{D}}$	96.32% @ 5320	78.91% @ 7401
PPO-$\hat{\mathcal{D}}$	98.71% @ 3151	89.33% @ 7195
DQN-$\mathcal{D}$	98.26% @ 4314	84.63% @ 8241
TRPO-$\mathcal{D}$	99.32% @ 3320	92.11% @ 4120
PPO-$\mathcal{D}$	99.35% @ 2453	96.41% @ 2904

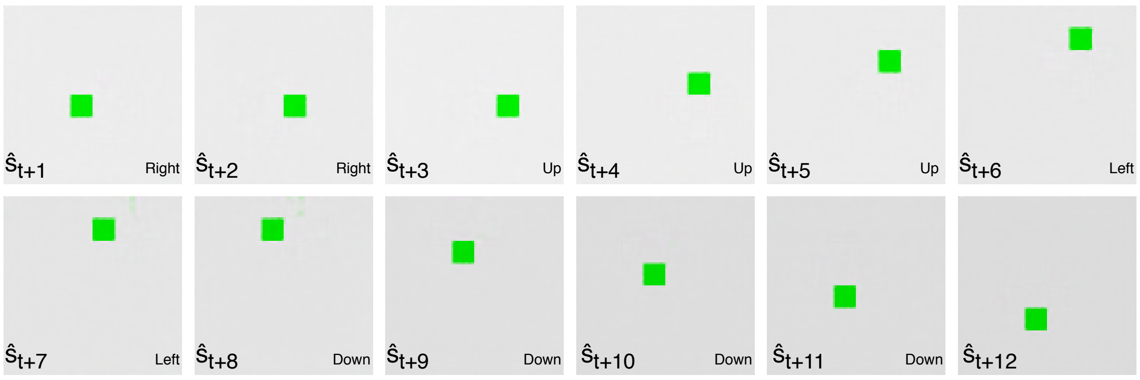


Figure 6: Results of 8×8 Deep Maze modeling using the DVAE algorithm. To simplify the environment, no reward signal is received per iteration. The left caption describes current state, s_t, while the right caption is the action performed to compute, $s_{t+1} = \mathcal{T}(s_t, a_t)$.

6.2 Using $\hat{\mathcal{D}}$ for RL Agents in Deep Maze

The goal of this experiment is to observe the performance of RL agents using the generated experience-replay $\hat{\mathcal{D}}$ from Figure 1 in Deep Maze environments of size 11×11 and 21×21. In Table 2, we compare the performance of DQN [14], TRPO [18], and PPO [19] using the DVAE generated $\hat{\mathcal{D}}$ to tune the parameters.

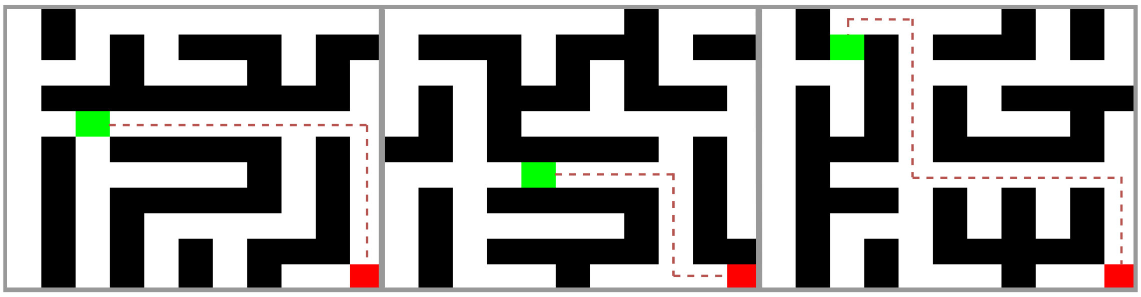


Figure 7: A typical deep maze of size 11×11. The lower-right square indicates the goal state, the dotted-line indicates the optimal path, while the final square represents the player's current position in the state-space. The controller agent is DQN, TRPO, and PPO (from left to right).

Figure 7 illustrates three maze variations of size 11×11, where the AI has learned the optimal path. We see that the best performing algorithm, PPO [19] beats DQN and TRPO using either $\hat{\mathcal{D}}$ or $\mathcal{D}$. The DQN-$\hat{\mathcal{D}}$ agent did not converge in the 21×21 environment, but it is likely that value-based algorithms could struggle with graphical artifacts generated from the DVAE algorithm. These artifacts significantly increase the state-space so that direct-policy algorithms could perform better.

6.3 Deep Line Wars Environment Modeling using DVAE

The DVAE algorithm works well in more complex environments, such as the Deep Line Wars game environment [1]. Here, we expand the DVAE algorithm with LSTM to improve the interpretation of animations, illustrated Figure 1.

Figure 8 illustrates the state quality during training of DVAE in a total of 6000 episodes (epochs). Both players draw actions from a Gaussian distributed policy. The algorithm understands that the player units can be located in any tiles after only 50 epochs, and at 1000 we observe the algorithm makes a more accurate statement of the probability of unit locations (i.e., some units have increased intensity). At the end of the training, the DVAE algorithm is to some degree capable of determining both towers, and unit locations at any given time-step during the game episode.

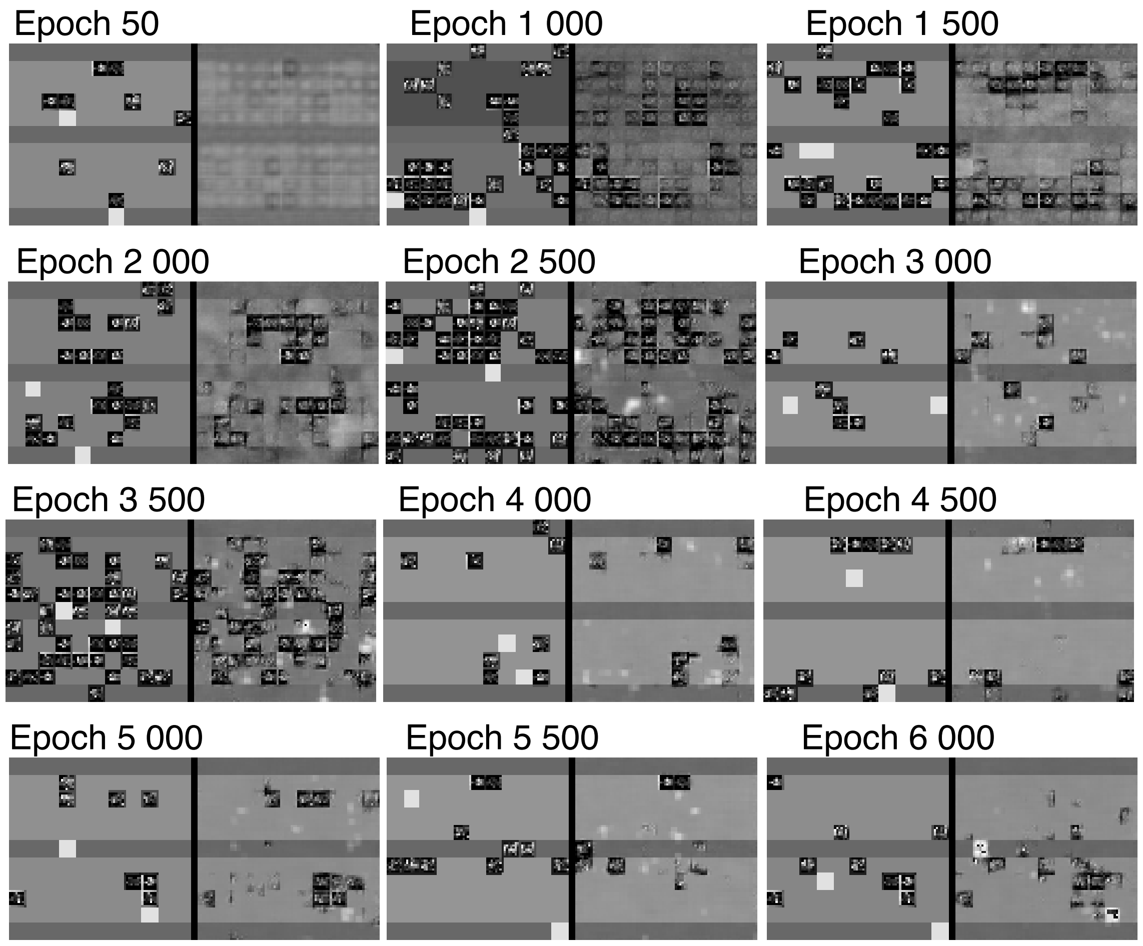


Figure 8: The DVAE algorithm applied to the Deep Line Wars environment. Each epoch illustrates the quality of generated states in the game, where the left image is real state s and the right image is the generated state $\hat{s}$.

7 Conclusion and Future Work

This paper introduces the *Dreaming Variational Autoencoder* (DVAE) as a neural network based generative modeling architecture to enable exploration in environments with sparse feedback. The DVAE shows promising results in modeling simple non-continuous environments. For continuous environments, such as Deep Line Wars, DVAE performs better using a recurrent neural network architecture (LSTM) while it is sufficient to use only a sequential feed-forward architecture to model non-continuous environments such as Chess, Go, and Deep Maze.

There are, however, several fundamental issues that limit DVAE from fully modeling environments. In some situations, exploration may be a costly act that makes it impossible to explore all parts of the environment in its entirety. DVAE cannot accurately predict the outcome of unexplored areas of the state-space, making the prediction blurry or false.

Reinforcement learning has many unresolved problems, and the hope is that the Deep Maze learning environment can be a useful tool for future research. For future work, we plan to expand the model to model the reward function $\hat{\mathcal{R}}$ using inverse reinforcement learning. DVAE is an ongoing research question, and the goal is that reinforcement learning algorithms could utilize this form of *dreaming* to reduce the need for exploration in real environments.

References

[1] Andersen, P.A., Goodwin, M., Granmo, O.C.: Towards a deep reinforcement learning approach for tower line wars. In: Bramer, M., Petridis, M. (eds.) Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). vol. 10630 LNAI, pp. 101–114 (2017)

[2] Arulkumaran, K., Deisenroth, M.P., Brundage, M., Bharath, A.A.: Deep reinforcement learning: A brief survey. IEEE Signal Processing Magazine **34**(6), 26–38 (2017)

[3] Bangaru, S.P., Suhas, J., Ravindran, B.: Exploration for Multi-task Reinforcement Learning with Deep Generative Models. arxiv preprint arXiv:1611.09894 (nov 2016)

[4] Blundell, C., Uria, B., Pritzel, A., Li, Y., Ruderman, A., Leibo, J.Z., Rae, J., Wierstra, D., Hassabis, D.: Model-Free Episodic Control. arxiv preprint arXiv:1606.04460 (jun 2016)

[5] Buesing, L., Weber, T., Racaniere, S., Eslami, S.M.A., Rezende, D., Reichert, D.P., Viola, F., Besse, F., Gregor, K., Hassabis, D., Wierstra, D.: Learning and Querying Fast Generative Models for Reinforcement Learning. arxiv preprint arXiv:1802.03006 (feb 2018)

[6] Chen, K.: Deep Reinforcement Learning for Flappy Bird. cs229.stanford.edu p. 6 (2015)

[7] Ha, D., Schmidhuber, J.: World Models. arxiv preprint arXiv:1803.10122 (mar 2018)

[8] Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., Lerchner, A.: beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. International Conference on Learning Representations (nov 2016)

[9] Higgins, I., Pal, A., Rusu, A., Matthey, L., Burgess, C., Pritzel, A., Botvinick, M., Blundell, C., Lerchner, A.: DARLA: Improving Zero-Shot Transfer in Reinforcement Learning. In: Precup, D., Teh, Y.W. (eds.) Proceedings of the 34th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 70, pp. 1480–1490. PMLR, International Convention Centre, Sydney, Australia (2017)

[10] Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement Learning: A Survey. Journal of Artificial Intelligence Research (apr 1996)

[11] Kingma, D.P., Ba, J.L.: Adam: A Method for Stochastic Optimization. Proceedings, International Conference on Learning Representations 2015 (2015)

[12] Li, Y.: Deep Reinforcement Learning: An Overview. arxiv preprint arXiv:1701.07274 (jan 2017)

[13] Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous Methods for Deep Reinforcement Learning. In: Balcan, M.F., Weinberger, K.Q. (eds.) Proceedings of The 33rd International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 48, pp. 1928–1937. PMLR, New York, New York, USA (2016)

[14] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing Atari with Deep Reinforcement Learning. Neural Information Processing Systems (dec 2013)

[15] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C.,

Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (feb 2015)

[16] Mousavi, S.S., Schukat, M., Howley, E.: Deep Reinforcement Learning: An Overview. In: Bi, Y., Kapoor, S., Bhatia, R. (eds.) Proceedings of SAI Intelligent Systems Conference (IntelliSys) 2016. pp. 426–440. Springer International Publishing, Cham (2018)

[17] Pu, Y., Gan, Z., Henao, R., Yuan, X., Li, C., Stevens, A., Carin, L.: Variational Autoencoder for Deep Learning of Images, Labels and Captions. In: Lee, D.D., Sugiyama, M., Luxburg, U.V., Guyon, I., R., G. (eds.) Advances in Neural Information Processing Systems. pp. 2352–2360. Curran Associates, Inc. (2016)

[18] Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P.: Trust Region Policy Optimization. In: Bach, F., Blei, D. (eds.) Proceedings of the 32nd International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 37, pp. 1889–1897. PMLR, Lille, France (2015)

[19] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal Policy Optimization Algorithms. arxiv preprint arXiv:1707.06347 (jul 2017)

[20] Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction, vol. 9. MIT Press (1998)

[21] Van Seijen, H., Fatemi, M., Romoff, J., Laroche, R., Barnes, T., Tsang, J.: Hybrid Reward Architecture for Reinforcement Learning. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30, pp. 5392–5402. Curran Associates, Inc. (2017)

[22] Xiao, T., Kesineni, G.: Generative Adversarial Networks for Model Based Reinforcement Learning with Tree Search. Tech. rep., University of California, Berkeley (2016)

Paper D: The Dreaming Variational Autoencoder for Reinforcement Learning Environments

Paper E

Title:	Towards Model-Based Reinforcement Learning for Industry-near Environments
Authors:	Andersen, Per-Arne et al.
Affiliation:	Department of ICT, University of Agder, Grimstad Norway
Conference:	39th SGAI International Conference on Artificial Intelligence
Year:	2019

Towards Model-based Reinforcement Learning for Industry-near Environments

Per-Arne Andersen
Department of ICT
University of Agder
Grimstad, Norway
per.andersen@uia.no

Morten Goodwin
Department of ICT
University of Agder
Grimstad, Norway
morten.goodwin@uia.no

Ole-Christoffer Granmo
Department of ICT
University of Agder
Grimstad, Norway
ole.granmon@uia.no

Abstract

Deep reinforcement learning has over the past few years shown great potential in learning near-optimal control in complex simulated environments with little visible information. Rainbow (Q-Learning) and PPO (Policy Optimisation) have shown outstanding performance in a variety of tasks, including Atari 2600, MuJoCo, and Roboschool test suite. While these algorithms are fundamentally different, both suffer from high variance, low sample efficiency, and hyperparameter sensitivity that in practice, make these algorithms a no-go for critical operations in the industry.

On the other hand, model-based reinforcement learning focuses on learning the transition dynamics between states in an environment. If these environment dynamics are adequately learned, a model-based approach is perhaps the most sample efficient method for learning agents to act in an environment optimally. The traits of model-based reinforcement are ideal for real-world environments where sampling is slow and for mission-critical operations. In the warehouse industry, there is an increasing motivation to minimise time and to maximise production. Currently, autonomous agents act suboptimally using handcrafted policies for significant portions of the state-space.

In this paper, we present The Dreaming Variational Autoencoder v2 (DVAE-2), a model-based reinforcement learning algorithm that increases sample efficiency, hence enable al-

Thirty-ninth SGAI International Conference on Artificial Intelligence (AI-2019), Cambridge England
Submitted 05. Jul, 2019, Accepted 13. Sep 2019, Published 19. Nov, 2019

gorithms with low sample efficiency function better in real-world environments. We introduce Deep Warehouse, a simulated environment for industry-near testing of autonomous agents in grid-based warehouses. Finally, we illustrate that DVAE-2 improves the sample efficiency for the Deep Warehouse compared to model-free methods.

Keywords: Deep Reinforcement Learning, Model-based Reinforcement Learning, Reinforcement Learning, Neural Networks, Variational Autoencoder, Markov Decision Processes, Exploration, Artificial Intelligence

1 Introduction

The goal of reinforcement learning is to maximise some notion of feedback through interaction with an environment [23]. The environment can be known, which makes this learning process trivial, or have hidden state information, which typically increases the complexity of learning significantly. In model-free reinforcement learning, actions are sampled from some policy that is optimised indirectly through direct policy search (Policy gradients), a state-value function (Q-learning), or a combination of these (Actor-Critic). There are many recent contributions to these algorithms that increase sample efficiency [8], reduce variance [10], and increase training stability [21].

It is challenging to deploy model-free methods in real-world environments because current state-of-the-art algorithms require millions of samples before any optimal policy is learned. Due to this, model-based reinforcement learning is an appealing approach because it has significantly better sample efficiency compared to the model-free methods [17]. The goal of model-based algorithms is to learn a predictive model of the real environment that is used to learn the controller of an agent. The downside of model-based reinforcement learning is that the predictive model may become inaccurate for longer time-horizons, or collapse entirely in areas of state-space that has not observed.

We propose a model-based reinforcement learning approach for industry-near systems where a predictive model is learned without direct interaction with the environment. We use Automated Storage and Retrieval Systems (ASRS) to benchmark our proposed algorithm. Learning a predictive model of the environment is isolated from the physical environment, which guarantees safety during training. If a predictive model is sufficiently trained, a model-free algorithm, such as DQN [19] can be trained off-line. Training can be done in a large-scale distributed setting, which significantly reduces the training time. When the model-free algorithm is trained sufficiently, it will be able to replace a sub-optimal expert-system with minimal effort.

The paper is organised as follows. Section 2 discusses the current state of the art in model-based reinforcement learning, and familiarise the reader of recent work in ASRS systems. Section 3 briefly outlines relevant background literature on reinforcement learning. Section 4 introduces the DVAE-2 algorithm and details the architecture thoroughly. Section 5 proposes the Deep Warehouse, a novel high-performance environment for industry-near testing of reinforcement learning algorithms. Section 6 presents our results using DVAE-2 in various environments, including complex environments such as Deep Warehouse, Deep RTS and Deep Line Wars. Finally, section 7 concludes our work and outlines a roadmap for our future work.

2 Literature Review

Reinforcement Learning is a maturing field in artificial intelligence, where a significant portion of the research is concerned with model-free approaches in virtual environments. Reinforcement learning methods in large-scale industry-near environments are virtually absent from the literature. The reason for this could be that (1) model-free methods do not give the sample efficiency required and that (2) there is little evidence that model-based approaches achieve reliable performance. In this section, we briefly discuss the previous work in ASRS systems and present promising results for model-based reinforcement learning.

2.1 Automated Storage and Retrieval Systems (ASRS)

There is to our knowledge no published work where reinforcement learning schemes are used to control taxi-agents in ASRS environments. The literature is focused on heuristic-based approaches, such as tree-search and traditional pathfinding algorithms. In [20], a detailed survey of the advancements in ASRS systems which categorise an ASRS system into five components; System Configuration, Storage Assignment, Batching, Sequencing, and Dwell-point. We adopt these categories in search of a reinforcement learning approach for ASRS systems

2.2 Model-based Reinforcement Learning

In model-based reinforcement learning, the goal is to learn state-transitions based on observations from the environment, the predictive model. If the predictive model is stable, with low variance and improves monotonically during training, it is, to some degree, possible to learn model-free agents to act optimally in environments that have never been observed directly.

Perhaps the most sophisticated algorithm for model-based reinforcement learning is the Model-based policy optimisation (MBPO) algorithm, proposed by Janner et al. [16] The authors empirically show that MBPO performs significantly better in continuous control tasks compared to previous methods. MBPO proves to be monotonically improving given that the following bounds hold:

$$\eta[\pi] \geq \hat{\eta}[\pi] - C$$

where $\eta[\pi]$ denotes the returns in the real environment under a policy whereas $\hat{\eta}[\pi]$ denotes the returns in the predicted model under policy π. Furthermore, the authors show that as long as they can improve the C, the performance will increase monotonically [16].

Gregor et al. proposed a scheme to train expressive generative models to learn belief-states of complex 3D environments with little prior knowledge. Their method was effective in predicting multiple steps into the future (overshooting) and significantly improve sample efficiency. In the experiments, the authors illustrated model-free policy training in several environments, including DeepMind Lab. However, the authors found it difficult to use their predictive model in model-free agents directly. [11]

Neural Differential Information Gain Optimisation (NDIGO) algorithm by Azar et al. is a self-supervised exploration model that learns a world model representation from noisy data. The primary features of NDIGO are its robustness to noise due to their method to cancel out negative loss and to give positive learning more value. The authors show in their maze environment that the model successfully converges towards an optimal world model even when introducing noise. The author claims that the algorithm outperforms previous state-of-the-art, being the Recurrent World Model from. [4]

The Dreaming Variational Autoencoder (DVAE) is an end-to-end solution for prediction the probable future state $p(\hat{s}_{t+1}|s_t, a_t$. The authors showed that the algorithm successfully predicted next state in non-continuous environments and could with some error predict future states in continuous state-space environments such as the Deep Line Wars environment. In the experiments, the authors used DQN, PPO, and TRPO using an artificial buffer to feed states to the algorithms. In all cases, the DVAE algorithm was able to create buffers that were accurate enough to learn a near-optimal policy. [3]

The algorithm VMAV-C is a combination of VAE and attention-based value function (AVF), and mixture density network recurrent neural network (MDN-RNN) from [12]. This modification to the original World Models algorithm improved performance in the Cart Pole environment. They used the on-policy algorithm PPO to learn the optimal policy from the latent representation of the state-space [18].

Deep Planning Network (PlaNet) is a model-based agent that interpret the pixels of a state to learn a predictive model of an environment. The environment dynamics are stored into latent-space, where the agent sample actions based on the learned representation. The proposed algorithm showed significantly better sample efficiency compared to model-free algorithms such as A3C [14].

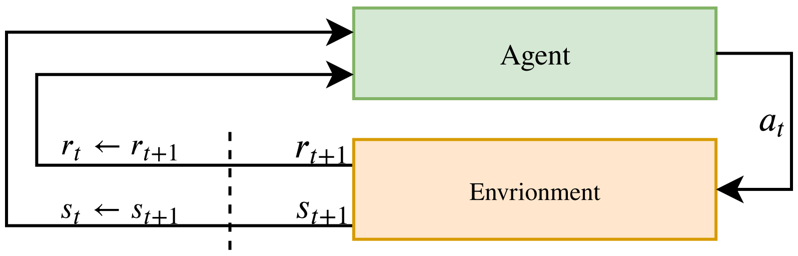


Figure 1: The agent-environment interaction in a Markov decision process [23]

In *Recurrent World Models Facilitate Policy Evolution*, a novel architecture for training RL algorithms using variational autoencoders. This paper showed that agents could successfully learn the environment dynamics and use this as an exploration technique requiring no interaction with the target domain. The architecture is mainly three components; vision, controller, and model, the vision model is a variational autoencoder that outputs a latent-space variable of an observation. The latent-space variable is processed in the model and is fed into the controller for action decisions. Their algorithms show state-of-the-art performance in self-supervised generative modelling for reinforcement learning agents. [12]

Chua et al. proposed *Probabilistic Ensembles with Trajectory Sampling* (PETS). The algorithm uses an ensemble of bootstrap neural networks to learn a dynamics model of the environment over future states. The algorithm then uses this model to predict the best action for future states. The authors show that the algorithm significantly lowers sampling requirements for environments such as half-cheetah compared to SAC and PPO. [9]

DARLA is an architecture for modelling the environment using β-VAE [15]. The trained model was used to learn the optimal policy of the environment using algorithms such as DQN [19], A3C, and Episodic Control [5]. DARLA is to the best of our knowledge, the first algorithm to introduce learning without access to the ground-truth environment during training.

3 Background

Markov decision processes (MDP's) are a mathematical framework commonly used to define reinforcement learning problems, as illustrated in Figure 1. In an MDP, we consider the tuple $(\mathcal{S}, \mathcal{A}, r, \mathcal{P}, \mathcal{P}_0, \gamma)$ [1]where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space available to the agent, $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the expected immediate reward function, P is the transition function which defines the probability $\mathcal{P}(s', s, a) = \mathcal{P}r(s'|s, a)$ and $\mathcal{P}_0$ is the probability for the initial state s_0.

[1]$\mathcal{S}$ and $\mathcal{A}$ is defined for discrete or continuous spaces. $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ where r is commonly referred to as $\mathcal{R}(s, s')$ in the literature.

The goal of a reinforcement learning agent is to encourage good behaviour and to discourage bad behaviour. Optimal behaviour is achieved when the agent finds a composition of parameters that maximise its performance, thus finds the optimal policyπ^*. Consider

$$\pi^* = \arg\max_{\pi\in\Pi} J(\pi), \tag{1}$$

where $J(\pi)$ is the objective function for maximising the expected discounted reward defined as

$$J(\pi) = \mathbb{E}_{s_0,a_0,s_1,\dots}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t) \mid \pi, s_0 \sim \mathcal{P}_0\right], \tag{2}$$

where $\gamma \in (0,1)$ is the discounting factor of future rewards. If $\gamma = 1$, all future state rewards are accounted for equally, while $\gamma = 0$, we are only concerned about the current state.

4 Learning policies using predictive models

The Dreaming Variational Autoencoder v2 (DVAE-2) is an architecture for learning a predictive model of arbitrary environments [3]. In this work, we aim to improve the first version of the DVAE for better performance in real-world environments. A common problem in model-based reinforcement learning is that it takes millions of samples to generalise well across sparse data. We aim to approve sample efficiency from the original DVAE and if possible, surpass the performance of model-free methods.

4.1 Motivation and Environment Safety

Figure 2 shows an abstract overview of DVAE-2 training in an environment. In real-world, industry-near environments, there is little room for interruptions. In model-free reinforcement learning, the agent interacts with the environment to learn its policy. Because this is not possible in many real-world environments, the DVAE-2 algorithm only observes during training. During training, the DVAE-2 algorithm learns how the transition function behaves and learns an estimated state-value function V that represent the value of being in that current state.

4.2 The Dreaming Variational Autoencoder v2

The original DVAE architecture had severe challenges with modelling of continuous state-spaces [3], and many algorithms were added to the model to improve performance across various environments including autoencoders, LSTMs, and fine-tuned variations of these. The DVAE-2 extends this with a split into three individual components; forming the **V**iew,

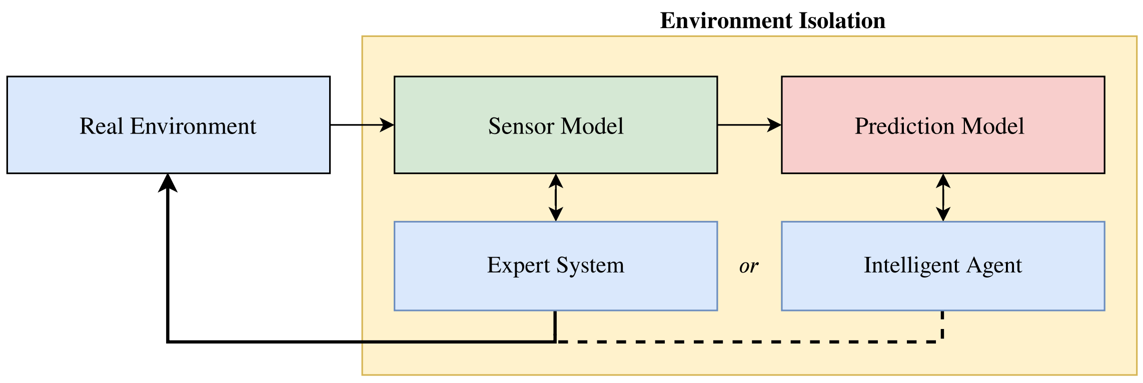


Figure 2: The proposed model isolates the intelligent agent from the mission-critical sensor model. The real environment projects onto a sensor model that the expert system uses to control taxis in a real environment. The predictive model observes the behaviour of the sensor model and the actions performed by the expert system. The predictive model is trained using error gradients, where the loss is the distance between the sensor model and the predictive model. When the error becomes sufficiently low, an intelligent agent can be trained using only data from the predictive model. Assuming that the intelligent agent converges to some performance threshold, it can be deployed as a drop-in replacement to the expert system.

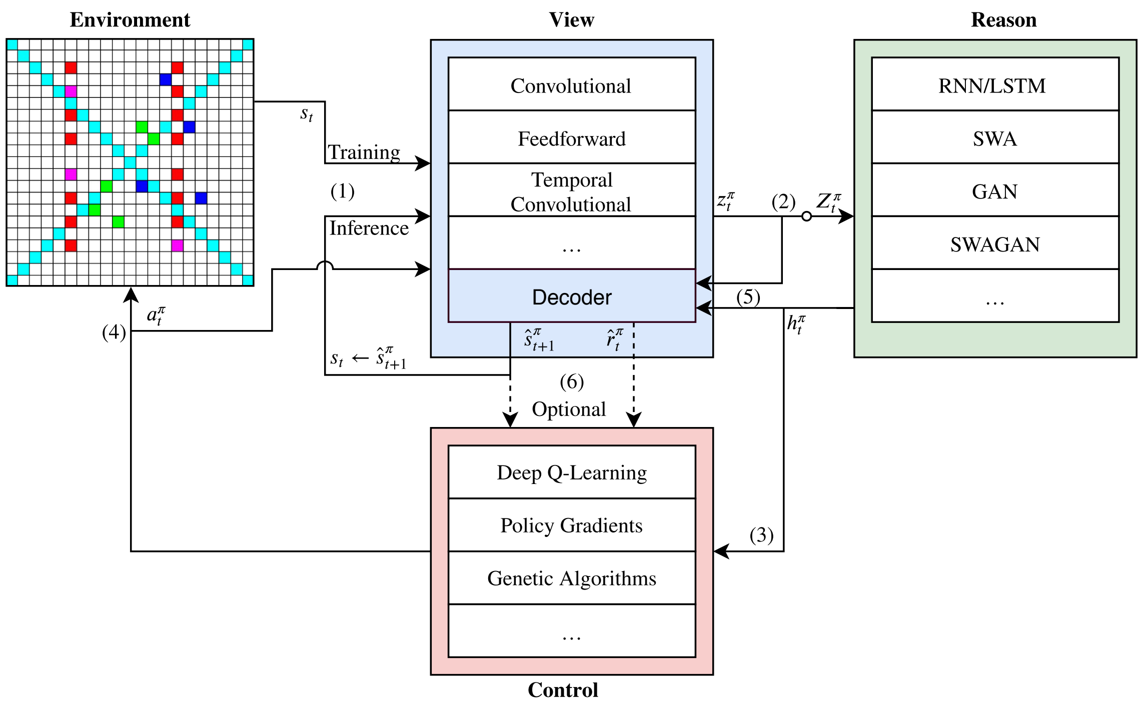


Figure 3: The component-based DVAE-2 architecture.

Reason and **C**ontrol (VRC) model. The VRC model embeds all improvements into a single model and learns which algorithms to use under certain conditions in an environment

Figure 3 shows an overview of the proposed VRC. (1) A state s_t is observed. During training, this observation stems from the real-environment while at inference time, from the predictive model. The observation is encoded in the **view** component (e.g. via AE or GAN) and outputs an embedding z at time t w.r.t policy π. (2) The **reason** component learns the time dynamics between state sequences. Encoded states are accumulated into a buffer $Z^{\pi}_{t} = \{z_{t-n} \ldots z_t\}^{\pi}$ and are then used to predict the hidden-state h^{π}_{t} w.r.t the encoded state sequence. The reason component typically consists of a model with RNN-like structure that generalises well on sequence data. (3) The hidden state is then used to evaluate an action using policy π, and (4) is sent to the environment and the view for the next iteration. (5) The decoder, prepares the hidden-state h^{π}_{t} and encoded state z^{π}_{t}, producing the succeeding state $\hat{s}^{\pi}_{t+1}$. The prediction is then used in the next iteration as current state s_t, which leads back to (1). As an optional mechanism, the controller can use the output from the decoder, instead of the hidden state information. This is beneficial when working with model-free algorithms such as deep q-networks [19].

4.3 Model selection

During technique selection in the components, we perform the following evaluation. An observation s_t is sent to the view component of DVAE-2. All of the view techniques are initially assumed to be uniformly qualified to encode and predict future states. For each iteration, the computed error is summarised as a score, and during inference, the technique with the lowest score is used[2]. We use the same method for determining the best reasoning algorithm in a specific environment.

4.4 Implementation

The implementation of the DVAE-2 algorithm with dynamic component selection enabled several significant improvements to over the previous DVAE model [3]. Notably, the k-step model rollout from [16] is implemented to stabilise training. We found that using shorter model-rollouts provided better control policies, but at the cost of higher sample efficiency. Also, by embedding time into the encoded state improved the model stability and prediction capabilities [13]. The DVAE-2 algorithm is defined as follows.

[2]In this setting, the lowest score is the technique with least accumulated error.

Algorithm 1 DVAE-2: Minimal Implementation

```
1:  Initialize policy $\pi_\theta(s_t|a_t)$, predictive model $p_\psi(\hat{s}_{t+1}, \hat{r}, h_t|s_t, a^\pi_t)$
2:  Let $Z = \{z^\pi_{t-n} \dots z^\pi_t\}$, a vector of encoded states
3:  Initialize encoder $ENC(z^\pi_t|s_t, a^\pi_t)$, temporal reasoner $TR(h^\pi_t|Z)$
4:  for N epochs do
5:      $\mathcal{D}_{env} \leftarrow$ Collect samples from $p_{env}$ under predefined policy $\pi$
6:      Train model $p_\psi$ on data batch $D_{env}$ via $MLE$[3]
7:  for M epochs do
8:      Sample initial state $s_0 \sim U(0, 1)$ from $\mathcal{D}_{env}$
9:      Construct $\{\mathcal{D}_{p_\psi}|t < k, TR(h^{\pi_\theta}_t|ENC(z_t|s_t, a_t)^{\pi_\theta}), s_t = s_0\}$
10:     Update policy $\pi_\theta$ using pairs of $(\hat{s}_t, a_t, \hat{r}_t, \hat{s}_{t+1})^{\pi_\theta}$
```

Algorithm 1 works as follows. (Line 1) We initialise the control policy and the predictive model (DVAE-2) parameters. (Line 2) The Z variable denotes a finite set of sequential view model (ENC) predictions that are used to capture time dependency between states in the reason model (TR). (Line 5) We collect samples from the real environment p_{env} under a predefined policy, such as an expert system, see Figure 2. (Line 6) The predictive model p_ψ is then trained using the collected data $\mathcal{D}_{env}$ via maximum likelihood estimation. In our case, we use mean squared error to measure the error distance $MSE(p_\psi||p_{env})$. When the DVAE-2 algorithm has trained sufficiently, the model-free algorithm will train for M epochs (Line 7) using the predictive model p_ψ instead of p_{env}. (Line 8) First, we sample the initial state s_0 uniformly from the real dataset $\mathcal{D}_{env}$. (Line 9) We then construct a prediction dataset $\mathcal{D}_{p_\psi}$ and predict future states using the control policy (i.e. sampling from the predictive model). (Line 10) The parameterised control policy is then optimised using $(\hat{s}_t, a_t, \hat{r}_t, \hat{s}_{t+1})^{\pi_\theta}$ pairs during rollouts.

5 The Deep Warehouse Environment

Training algorithms in real-world environments is known to have severe safety challenges during training and suffers from low sampling speeds [6]. It is therefore practical, to create a simulation of the real environment so that researches can quickly test algorithm variations with quick feedback on its performance.

This section presents the Deep Warehouse[4] environment for discrete and continuous action and state spaces. The environment has a wide range of configurations for time and

[3]We use the mean squared error (MSE) loss in our implementation.

[4]The deep warehouse environment is open-source and freely available at `https://github.com/cair/deep-warehouse`

agent behaviour, giving it tolerable performance in simulating proprietary automated storage and retrieval systems.

5.1 Motivation

In the context of warehousing, an Automated Storage and Retrieval System (ASRS) is a composition of computer programs working together to maximise the incoming and outcoming throughput of goods. There are many benefits of using an ASRS system, including high scalability, increased efficiency, reduced operating expenses, and operation safety. We consider a cube-based ASRS environment where each cell is stacked with item containers. On the surface of the cube, taxi-agents are collecting and delivering goods to delivery points placed throughout the surface. The taxi-agents are controlled by a computer program that reads sensory data from the taxi and determines the next action.

Although these systems are far better than manual labour warehousing, there is still significant improvement potential in current state-of-the-art. Most ASRS systems are manually crafted expert systems, which due to the high complexity of the multi-agent ASRS systems only performs sub-optimally. [20].

5.2 Implementation

Figure 4 illustrates the state-space in the deep warehouse environment. In a simple cube-based ASRS configuration, the environment consists of (B) passive and (C) active delivery-points, (D) pickup-points, and (F) taxis. Also, the simulator can model other configurations, including advanced cube and shelf-based automated storage and retrieval systems. In the deep warehouse environment, the goal is to store and retrieve goods from one location to another where each cell represents several layers of containers that a taxi can pick up. A taxi (F) receives feedback based on the time used on the task it performs. A taxi can move using a discrete or continuous controller. In discrete mode, the agent can increase and decrease thrust, and move in either direction, including the diagonals. For the continuous mode, all of these actions are floating point numbers between (off) 0 and (on) 1, giving a significantly harder action-space to learn. The simulator also features continuous mode for the state-space, where actions are performed asynchronously to the game loop. It is possible to create custom support modules for mechanisms such as task scheduling, agent controllers and fitness scoring.

A significant benefit of the deep warehouse is that it can accurately model real warehouse environments at high speed. The deep warehouse environment runs 1000 times faster on a single high-end processor core compared to real-world systems measured from the speed improvement by counting how many operations a taxi can do per second. The simulator can be distributed across many processing units to increase the performance further. In

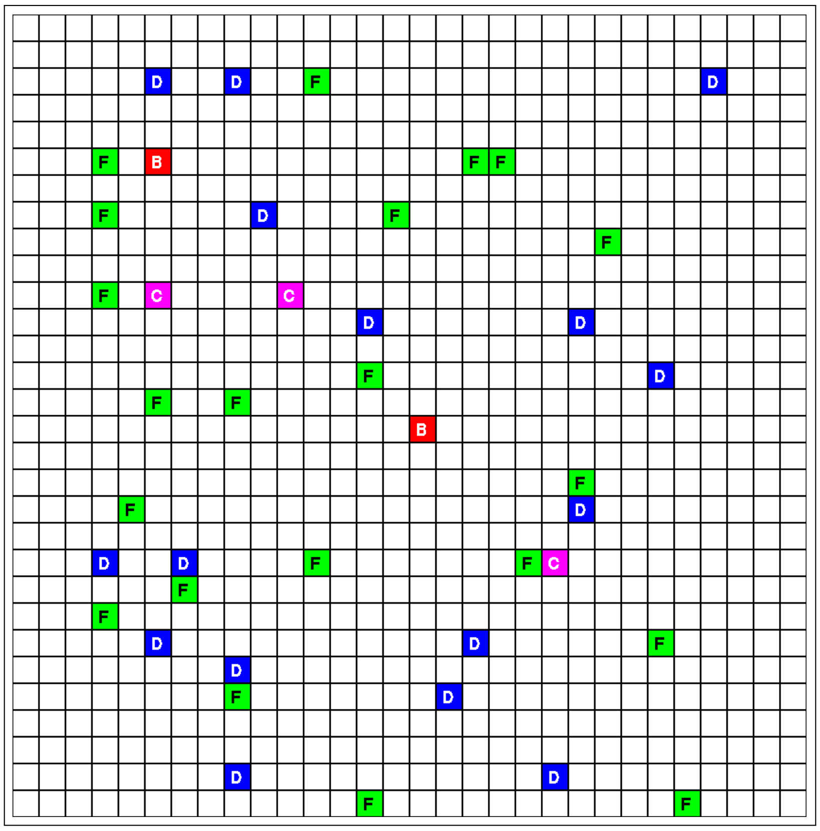


Figure 4: Illustration of the graphical interface in the deep-warehouse environment using cube-based ASRS configuration.

our benchmarks, the simulator was able to collect 1 million samples per second during the training of deep learning models using high-performance computing (HPC).

6 Experimental Results

In this section, we present our preliminary results of applied model-based reinforcement learning using DVAE-2. We aim to answer the following questions.

(1) Does the DVAE-2 algorithm improve sample efficiency compared to model-free methods? **(2)** How well do DVAE-2 perform versus model-free methods in the deep warehouse environment? **(3)** Which of DVAE-2 VRC components is preferred by the model?

6.1 The importance of compute

According to AI pioneer Richard S. Sutton “The biggest lesson that can be read from 70 years of AI research is that general methods that leverage computation are ultimately the most effective, and by a large margin.” [22]. It is therefore not surprising that compute is still the most decisive factor when training a large model, also for predictive models. DVAE-2 was initially trained using two NVIDIA 2080 RTX TI GPU cards that, if tuned

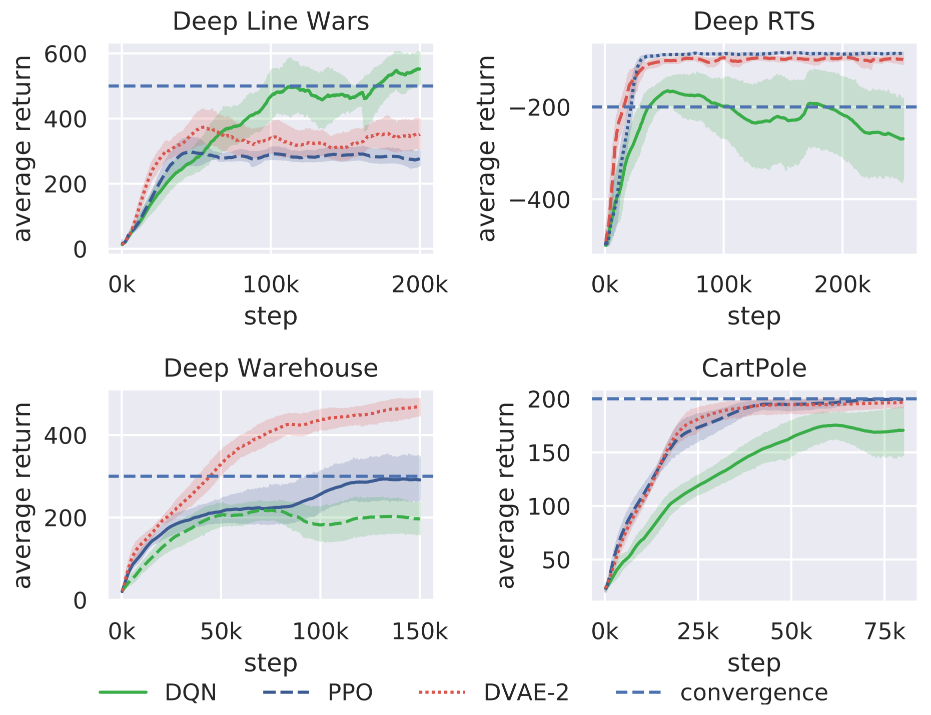


Figure 5: We compare DVAE-2 using two baseline algorithms, DQN and PPO. The solid curve illustrates the mean of 12 trials and shaded regions is the standard deviation between all trials. The x-axis shows the number of episodes performed and the y-axis shows the average return.

properly, can operate at approximately 26.9 TFLOPS. For simpler problems, such as grid-warehouses of size 5×5 and CartPole, the compute was enough to train the model in 5 minutes, but for larger environments, this time grew exponentially. To somewhat mitigate the computational issue for larger environments, we performed the experiments with approximately 1.25 PFLOPS of compute power. This led to significantly faster training speeds, and made large experiments feasible[5]

6.2 Results

Figure 5 shows that the average return value of DVAE-2 training four tasks, including Deep RTS [2], Deep Warehouse, Deep Line Wars [1] and CartPole [7].

Deep Warehouse: The environment is a contribution in this paper for industry-near testing of autonomous agents. The DVAE-2 algorithm outperforms both PPO and DQN in terms of sampling and performance during 150000 game steps. The score function is a counter of how many tasks the agent has performed during the episode. If the agent manages to collect and retrieve 300 packages, the agent has sufficient performance to beat

[5]We recognise large experiments to consist of environments where the agents require significant sampling to converge.

many handcrafted algorithms in ASRS systems. The environment is multi-agent, and in this experiment, we used a 30×30 grid with 20 taxis running the same policy.

Deep RTS is a flexible real-time strategy game (RTS) engine with multiple environments for unit control and resource management. In this experiment, we used the resource harvester environment where the goal is to harvest 500 wood resources before the time limit is up. The score is measured from -500 to 0, where 0 is the best score. For every wood harvested, the score increase with 1. We consider the task mastered if the agent has less than -200 score at the terminal state. DVAE-2 outperform the baseline algorithms in terms of sample efficiency but falls behind PPO in terms of score performance. [2]

Deep Line Wars: Surprisingly, the DQN policy outperforms the DVAE-2 and PPO policy in 11×11 discrete action-space environment. Because we used PPO as the policy for DVAE-2, we still see a marginal improvement over the same algorithm in a model-free setting yielding better performance and better sample efficiency. We found that DQN quickly learned the correct Q-values due to the small environment size. In future experiments, we would like to include larger map sizes that would increase the state-space significantly, hence making Q-values more challenging to learn. [1]

CartPole: As a simple baseline environment, we use CartPole from the OpenAI Gym environment suite [7]. The goal of this environment is to balance a pole on a moving cart using a discrete action-space of 2 actions. We found that DVAE-2 and PPO had similar performance, but DVAE-2 had marginally better sample efficiency after 25000 steps.

In terms of VRC, the algorithm tended to choose Convolutional + LSTM and Temporal Convolution and GAN for continuous control tasks (see Figure 1). It should be noted that PPO and DVAE-2 are presented with the same hyper-parameters, and are therefore directly comparable. We used PPO as our policy for DVAE-2, and we see that DVAE-2 is more sample efficient and performs equally good or better than model-free PPO in all tested scenarios.

7 Conclusion and Future Work

In this paper, we present DVAE-2, a novel model-based reinforcement learning algorithm for improved sample efficiency in environments where sampling is not available. We also present the deep warehouse environment for training reinforcement learning agents in industry-near ASRS systems. This section concludes our work and defines future work for DVAE-2..

Although the deep warehouse does not behave identical to a real-world system, it is adequate to determine the training time and performance. DVAE-2 is presented as a VRC model for training reinforcement learning algorithms with a learned model of the environment. The method is tested in the Deep warehouse several continuous game environments. Our algorithm reduces training time and depends less on data sampled from the real environment compared to model-free methods.

We find that a carefully tuned policy gradient algorithms can converge to near-optimal behaviour in simulated environments. Model-free algorithms are significantly harder to train in terms of sample efficiency and stability, but perform better if there is unlimited sampling available from the environment.

Our work shows promising results for reinforcement learning agents in ASRS. There are, however, open research questions that are essential for safe deployment in real-world systems. We wish to pursue the following questions to achieve safety deployment in real-world environments. **(1)** How do we ensure that the agent acts within defined safety boundaries? **(2)** How would the agent act if parts of the state-space changes to unseen data (i.e. a fire occurs, or a collision between agents.) **(3)** Can agents with a non-stationary policy function well in a multi-agent setting?

References

[1] Andersen, P.A., Goodwin, M., Granmo, O.C.: Towards a deep reinforcement learning approach for tower line wars. In: Bramer, M., Petridis, M. (eds.) Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). vol. 10630 LNAI, pp. 101–114 (2017). https://doi.org/10.1007/978-3-319-71078-5_8

[2] Andersen, P.A., Goodwin, M., Granmo, O.C.: Deep RTS: A Game Environment for Deep Reinforcement Learning in Real-Time Strategy Games. Proceedings of the IEEE International Conference on Computational Intelligence and Games (aug 2018), http://arxiv.org/abs/1808.05032

[3] Andersen, P.A., Goodwin, M., Granmo, O.C.: The Dreaming Variational Autoencoder for Reinforcement Learning Environments. In: Max Bramer, Petridis, M. (eds.) Artificial Intelligence, vol. 11311, pp. 143–155. Springer, Cham, xxxv edn. (dec 2018). https://doi.org/10.1007/978-3-030-04191-5_11, http://link.springer.com/10.1007/978-3-030-04191-5{_}11

[4] Azar, M.G., Piot, B., Pires, B.A., Grill, J.B., Altché, F., Munos, R.: World Discovery Models. arxiv preprint arXiv:1902.07685 (feb 2019), http://arxiv.org/abs/1902.07685

[5] Blundell, C., Uria, B., Pritzel, A., Li, Y., Ruderman, A., Leibo, J.Z., Rae, J., Wierstra, D., Hassabis, D.: Model-Free Episodic Control. arxiv preprint

arXiv:1606.04460 (jun 2016), http://arxiv.org/abs/1606.04460

[6] Botvinick, M., Ritter, S., Wang, J.X., Kurth-Nelson, Z., Blundell, C., Hassabis, D.: Reinforcement Learning, Fast and Slow. Trends in cognitive sciences **23**(5), 408–422 (may 2019). https://doi.org/10.1016/j.tics.2019.02.006, http://www.ncbi.nlm.nih.gov/pubmed/31003893

[7] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: OpenAI Gym. arxiv preprint arXiv:1606.01540 (jun 2016), http://arxiv.org/abs/1606.01540

[8] Buckman, J., Hafner, D., Tucker, G., Brevdo, E., Lee, H.: Sample-Efficient Reinforcement Learning with Stochastic Ensemble Value Expansion. Advances in Neural Information Processing Systems 32 pp. 8224–8234 (jul 2018), http://arxiv.org/abs/1807.01675

[9] Chua, K., Calandra, R., McAllister, R., Levine, S.: Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models. Advances in Neural Information Processing Systems 31 (may 2018), http://arxiv.org/abs/1805.12114

[10] Greensmith, E., Bartlett, P.L., Baxter, J.: Variance reduction techniques for gradient estimates in reinforcement learning. Journal of Machine Learning Research **5**(Nov), 1471–1530 (2004)

[11] Gregor, K., Rezende, D.J., Besse, F., Wu, Y., Merzic, H., van den Oord, A.: Shaping Belief States with Generative Environment Models for RL. arxiv preprint arXiv:1906.09237 (jun 2019), http://arxiv.org/abs/1906.09237

[12] Ha, D., Schmidhuber, J.: Recurrent World Models Facilitate Policy Evolution. Advances in Neural Information Processing Systems 31 (sep 2018), http://arxiv.org/abs/1809.01999

[13] Ha, D., Schmidhuber, J.: World Models. arxiv preprint arXiv:1803.10122 (mar 2018). https://doi.org/10.5281/zenodo.1207631, https://arxiv.org/abs/1803.10122

[14] Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., Davidson, J.: Learning Latent Dynamics for Planning from Pixels. Proceedings of the 36 th International Conference on Machine Learning (nov 2018), http://arxiv.org/abs/1811.04551

[15] Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., Lerchner, A.: beta-VAE: Learning Basic Visual Concepts with a Constrained

Variational Framework. International Conference on Learning Representations (nov 2016), https://openreview.net/forum?id=Sy2fzU9gl

[16] Janner, M., Fu, J., Zhang, M., Levine, S.: When to Trust Your Model: Model-Based Policy Optimization. arXiv preprint arXiv:1906.08253 (jun 2019), http://arxiv.org/abs/1906.08253

[17] Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement Learning: A Survey. Journal of Artificial Intelligence Research (apr 1996). https://doi.org/10.1.1.68.466, http://arxiv.org/abs/cs/9605103

[18] Liang, X., Wang, Q., Feng, Y., Liu, Z., Huang, J.: VMAV-C: A Deep Attention-based Reinforcement Learning Algorithm for Model-based Control. arxiv preprint arXiv:1812.09968 (dec 2018), http://arxiv.org/abs/1812.09968

[19] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing Atari with Deep Reinforcement Learning. Neural Information Processing Systems (dec 2013), http://arxiv.org/abs/1312.5602

[20] Roodbergen, K.J., Vis, I.F.A.: A survey of literature on automated storage and retrieval systems. European Journal of Operational Research (2009). https://doi.org/10.1016/j.ejor.2008.01.038

[21] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal Policy Optimization Algorithms. arxiv preprint arXiv:1707.06347 (jul 2017), http://arxiv.org/abs/1707.06347

[22] Sutton, R.S.: The Bitter Lesson (2019), http://www.incompleteideas.net/IncIdeas/BitterLesson.html

[23] Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT Press (2018)

Paper E: Towards Model-Based Reinforcement Learning for Industry-near Environments

Paper F

Title:	Increasing Sample Efficiency in Deep Reinforcement Learning using Generative Environment Modelling
Authors:	Andersen, Per-Arne et al.
Affiliation:	Department of ICT, University of Agder, Grimstad Norway
Journal:	Expert Systems. Impact factor in 2020: 2.587
Year:	2020, March

Increasing Sample Efficiency in Deep Reinforcement Learning using Generative Environment Modeling

Per-Arne Andersen
Department of ICT
University of Agder
Grimstad, Norway
per.andersen@uia.no

Morten Goodwin
Department of ICT
University of Agder
Grimstad, Norway
morten.goodwin@uia.no

Ole-Christoffer Granmo
Department of ICT
University of Agder
Grimstad, Norway
ole.granmon@uia.no

Abstract

Reinforcement learning is a broad scheme of learning algorithms that, in recent times, has shown astonishing performance in controlling agents in environments presented as Markov decision processes. There are several unsolved problems in current state-of-the-art that causes algorithms to learn suboptimal policies, or even diverge and collapse completely. Parts of the solution to address these issues may be related to short- and long-term planning, memory management, and exploration for reinforcement learning algorithms. Games are frequently used to benchmark reinforcement learning algorithms as they provide a flexible, reproducible, and easy to control environments. Regardless, few games feature the ability to perceive how the algorithm performs exploration, memorization, and planning. This paper presents *The Dreaming Variational Autoencoder with Stochastic Weight Averaging and Generative Adversarial Networks* (DVAE-SWAGAN), a neural network based generative modeling architecture for exploration in environments with sparse feedback. We present deep maze, a novel, and flexible maze game-engine that challenges DVAE-SWAGAN in partial and fully-observable state-spaces, long-horizon tasks,

Wiley: Journal of Expert Systems
Impact Factor: 2.587 (2020)
Submitted 16. May, 2019, Accepted 24. Jan 2020, Published 04. Oct, 2021

and deterministic and stochastic problems. We show results between different variants of the algorithm and encourage future work in reinforcement learning driven by generative exploration.

Keywords: Deep Reinforcement Learning; Environment Modeling; Neural Networks; Variational Autoencoder; Markov Decision Processes; Exploration; Artificial Experience-Replay, Generative Adversarial Networks, Model-Based RL, Generative Modeling

1 Introduction

Reinforcement learning (RL) is a field of research that has quickly become one of the most promising branches of machine learning algorithms to solve artificial general intelligence ([Kaelbling et al., 1996, Li, 2017, Arulkumaran et al., 2017], and [Mousavi et al., 2018]). There have been several breakthroughs in reinforcement learning research in recent years for environments such as the Atari Arcade ([Mnih et al., 2013, Mnih et al., 2015]), AlphaZero ([Silver et al., 2017]), OpenAI Five, and AlphaStar ([Arulkumaran et al., 2019]), but no algorithms are capable of human performance without extensive hardware requirements that are accessible to only a few institutions, such as Deep Mind and Open AI. Due to this, reinforcement learning still has several open research questions to address before the general public can deploy these algorithms successfully. It is possible that many of the issues in the current state of the art can be resolved with algorithms that adequately accounts for planning, exploration, and memory efficiency at different time-horizons.

In current state-of-the-art RL algorithms, long-horizon RL tasks are difficult to master because there is as of yet no optimal exploration algorithm that is capable of proper state-space pruning. Exploration strategies such as ϵ-greedy are widely used in RL, but cannot find an adequate exploration/exploitation balance without exhaustive hyperparameter-tuning. Environment modeling is a promising exploration method where the goal is to imitate the behavior of a target environment. By constructing an artificial model of the environment, the need to interact with the ground-truth environment is reduced significantly. This enables the RL-algorithm to explore large parts of the state space without the cost of exhausting the ground-truth environment. A balance between exploration and exploitation must be accounted for to learn the underlying dynamics of the environment. The algorithm must, therefore, select observations carefully to only learn essential features during the exploration phase.

By combining generative modeling with deep reinforcement learning, we find that it is possible for agents to learn optimal policies using only generated training data samples. The approach that we present is the dreaming variational autoencoder with two extensions, based on generative adversarial networks and stochastic weight averaging. We also present a new learning environment, deep maze, that aims to bring a vast set of challenges for reinforcement learning algorithms and is the environment used for testing the presented algorithms.

This paper is organized as follows. Section 2 surveys recent work in the field. Section 3 briefly introduces the reader to preliminaries. Section 4 proposes *The Dreaming Variational Autoencoder* for environment modeling to improve exploration in RL. Section 5 introduces the deep maze learning environment for exploration, planning and memory management research for reinforcement learning. Section 6 shows results in the deep line wars environment and that RL agents can be trained to navigate through the deep maze environment using only artificial training data. Finally, we summarize our findings and outlines future work in Section 7.

2 Literature Review

In machine learning, the goal is to create an algorithm that is capable of accurately representing a target environment. There is, however, little literature on generative modeling for game environments on the scale we propose in this paper. The primary focus of recent RL research has been through improvements in on and off policy algorithms, while less attention has been put into generative exploration. This section introduces a thorough literature review of reinforcement based generative modeling.

[Bangaru et al., 2016] proposed a method of deducing the Markov Decision Process (MDP) by introducing an adaptive exploration signal (pseudo-reward), which was obtained using deep generative model. Their approach was to compute the Jacobian of each state and used it as the pseudo-reward when using deep neural networks to learn the state-generalization.

[Xiao and Kesineni, 2016] proposed the use of generative adversarial networks (GAN) for model-based reinforcement learning. The goal was to utilize GAN for learning dynamics of the environment in a short-horizon timespan and combine this with the strength of far-horizon value iteration RL algorithms. The GAN architecture proposed illustrated near authentic generated images giving comparable results to [Mnih et al., 2013].

[Higgins et al., 2017] proposed DARLA, an architecture for modeling the environment using β-VAE ([Higgins et al., 2016]). The trained model was used to learn the optimal policy of the environment using algorithms such as Deep Q-Networks (DQN) ([Mnih et al., 2015]), Asynchronous Actor-Critic Agents (A3C) ([Mnih et al., 2016]), and Episodic Control ([Blundell et al., 2016]). DARLA is to the best of our knowledge, the first algorithm to introduce learning without access to the ground-truth environment during training.

Buesing et al. recently compared several methods of environment modeling, showing that it is far better to model the state-space then to utilize Monte-Carlo rollouts (RAR). The proposed architecture, state-space models (SSM) was significantly faster and produced acceptable results compared to auto-regressive (AR) methods. ([Buesing et al., 2018])

The algorithm VMAV-C is a combination of VAE and attention-based value function (AVF), and mixture density network recurrent neural network (MDN-RNN) from [Ha and Schmidhuber, 2018]. This modification to the original World Models algorithm improved performance in the Cart Pole environment. They used the on-policy algorithm PPO to learn the optimal policy from the latent representation of the state-space ([Liang et al., 2018]).

Deep Planning Network (PlaNet) is a model-based agent that interpret the pixels of a state to learn the dynamics of an environment. The environment dynamics are stored into latent-space, where the agent sample actions based on the learned representation. The proposed algorithm showed significantly better sample efficiency compared to algorithms such as A3C ([Hafner et al., 2018]).

[Chua et al., 2018] recently proposed Probabilistic Ensembles with Trajectory Sampling (PETS). The algorithm uses an ensemble of bootstrap neural networks to learn a dynamics model of the environment over future states. The algorithm then uses this model to predict the best action for future states. The authors show that the algorithm significantly lowers sampling requirements for environments such as half-cheetah compared to SAC and PPO.

Stochastic optimal control with latent representations (SOLAR) is an algorithm that learns the dynamics of the environment by exploiting the knowledge from a reinforcement learning policy. This enables the algorithm to learn local models which are used in policy learning for complex systems. SOLAR is built around a probabilistic graphical model (PGM) structure that allows efficient learning of the environment model. By exploiting the locality of the model, the authors show that the gradients give good direction for policy improvements during training. The algorithm was compared to model-free methods and showed significantly better performance and data efficient compared to algorithms such as LQR-FLM. ([Zhang et al., 2018])

[Ha and Schmidhuber, 2018] proposed in *Recurrent World Models Facilitate Policy Evolution*, a novel architecture for training RL algorithms using variational autoencoders. This paper showed that agents could successfully learn the environment dynamics and use this as an exploration technique requiring no interaction with the target domain. The architecture is mainly three components; vision, controller, and model, the vision model is a variational autoencoder that outputs a latent-space variable of an observation. The latent-space variable is processed in the model and is fed into the controller for action decisions. Their algorithms show state-of-the-art performance in self-supervised generative modeling for reinforcement learning agents.

One of the most recent advancements in generative modeling for reinforcement learning is the Neural Differential Information Gain Optimisation (NDIGO) algorithm by [Azar et al., 2019], a self-supervised exploration model that learns a world model representation from noisy data. The primary features of NDIGO are its robustness to noise due to their method to cancel out negative loss and to give positive learning more value. The authors show in their maze environment that the model successfully converges towards an optimal world model even when introducing noise. The author claims that the algorithm outperforms previous state-of-the-art, being the Recurrent World Model from.

3 Background

We base our work on the well-established theory of reinforcement learning originally formulated in [Sutton et al., 1999], defining the problem as a MDP as $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T} \rangle$ pairs. The state-space, $\mathcal{S}$ represents all possible states while the action-space, $\mathcal{A}$ represents all available actions the agent can perform in the environment. $\mathcal{R}$ is the reward function while $\mathcal{T}$ denotes the transition function ($\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$), which is a mapping from state $s_t \in \mathcal{S}$ and action $a_t \in \mathcal{A}$ to the future state s_{t+1}. After each performed action, the environment dispatches a reward signal, $\mathcal{R} : \mathcal{S} \rightarrow r$.

We call a sequence of states and actions a *trajectory* denoted as
$\tau = (s_0, a_0, \ldots, s_t, a_t)$ and the sequence is sampled through the use of a stochastic policy that predicts the optimal action in any state: $\pi_\theta(a_t|s_t)$, where π is the policy and θ are the parameters. The primary goal of the reinforcement learning is to *reinforce* good behavior. The algorithm should try to learn the policy that maximizes the total expected discounted reward given by, $\mathcal{J}(\pi) = \mathbb{E}_{(s_t,a_t)\sim p(\pi)} \left[\sum_{i=0}^{T} \gamma^i \mathcal{R}(s_i)\right]$ [Mnih et al., 2015].

Several algorithms try to address the problem of reinforcement learning, commonly divided into three categories; *Value Iteration, Policy Iteration and Actor-Critic Algorithms*, with variations of *on-policy*, and *off-policy* learning.

Autoencoders are commonly used in supervised learning to encode arbitrary input to a compact representation, and using a decoder to reconstruct the original data from the encoding. The purpose of autoencoders is to store redundant data into a densely packed vector form. In its simplest form, an autoencoder consists of feed-forward neural network where the input and output layer is of equal neuron capacity and the hidden layer smaller, used to compress the data. Such autoencoder can be defined as: $\phi : \mathcal{X} \rightarrow \mathcal{Z}$, $\psi : \mathcal{Z} \rightarrow \mathcal{X}$, where $\phi, \psi : \arg\min_{\phi,\psi} ||\mathcal{X} - (\phi \times \psi)\hat{\mathcal{X}}||^2$. In this notation, $\mathcal{X}$ defines the input, $\mathcal{Z}$, the encoding and $\hat{\mathcal{X}}$ as the reconstructed data. There are, however several issues with

Algorithm Branch	**Sample Efficient**	**Sequential Data**	**Generative Modeling**	**Labeled Data**
Reinforcement Learning	No	**Yes**	No	No
Variational Autoencoders	**Yes**	**Yes**	**Yes**	**Yes**
Generative Adversarial Networks	**Yes**	No	**Yes**	**Yes**
DVAE-GAN	**Yes**	**Yes**	**Yes**	**Yes**

Table 1: A comparison of features in general reinforcement learning, variational autoencoders, and generative adversarial networks. The purpose of this illustration is to show that the proposed algorithm uses generative modeling to inherit sample efficient generative modeling for sequential data for use in reinforcement learning algorithms.

autoencoders, as they are not generative, in the sense that they can transition between features in the input data. To remedy this, [Kingma and Welling, 2013] proposed the Variational Autoencoder (VAE) algorithm that enables interpolation between features in the latent space. The interpolation is done by introducing a vector of means μ and a vector of standard deviations σ. These vectors, along with the additional KL-Loss gives the algorithm the ability to learn variations in the input data, enabling the output to be vastly more diverse.

Our background for choosing the following branch of algorithms are described in Table 1. Algorithms based on reinforcement learning learns by hands-on interaction. Experience is attained by exploring the environment, but in some cases, the environment may be exhausted before the agent can learn to behave optimally. To remedy this, we introduce a model-based exploration approach using VAE to model the environment with sequential data as input. This increase the sampling efficiency of the RL algorithm by a significant amount. To increase the performance of the environment model, GAN is used to strengthen the quality of the environment-model, where the VAE and GAN continually reinforce each other's performance.

4 The Dreaming Variational Autoencoder

The Dreaming Variational Autoencoder (DVAE) is an end-to-end solution for generating probable future states $\hat{s}_{t+n}$ from an arbitrary state-space $\mathcal{S}$ using state-action pairs explored prior to s_{t+n} and a_{t+n}. Figure 1 illustrates the DVAE model, where Algorithm 1 works as follows. First, the agent collects experiences for utilizing experience-replay

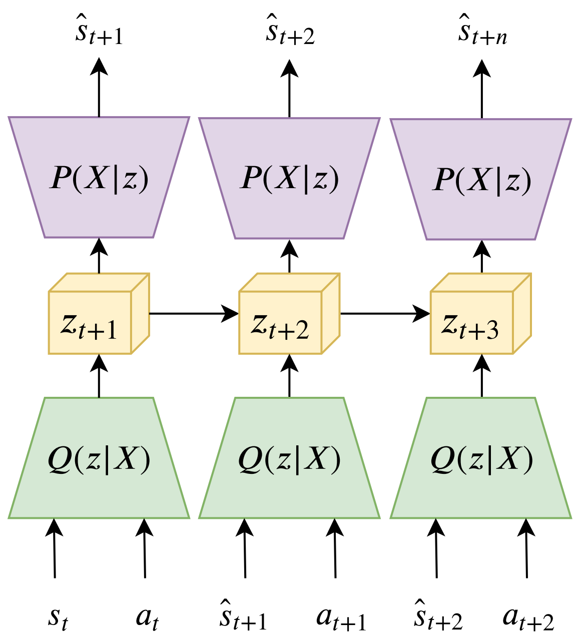


Figure 1: Illustration of the DVAE model. The model consumes state and action pairs, yielding the input encoded in latent-space. Latent-space can then be decoded to a probable future state. $\mathcal{Q}(z|X)$ is the encoder, z_t is latent-space, and $\mathcal{P}(X|z)$ is the decoder. DVAE can also use LSTM to better learn longer sequences in continuous state-spaces.

in the *Run-Agent* function. At this stage, the agent explores the state-space guided by a Gaussian distributed policy. The agent acts, observes, and stores the observations into the experience-replay buffer $\mathcal{D}$. After the agent reaches terminal state, the DVAE algorithm encodes state-action pairs from the replay-buffer D into probable future states. This is stored in the replay-buffer for artificial future-states $\hat{D}$.

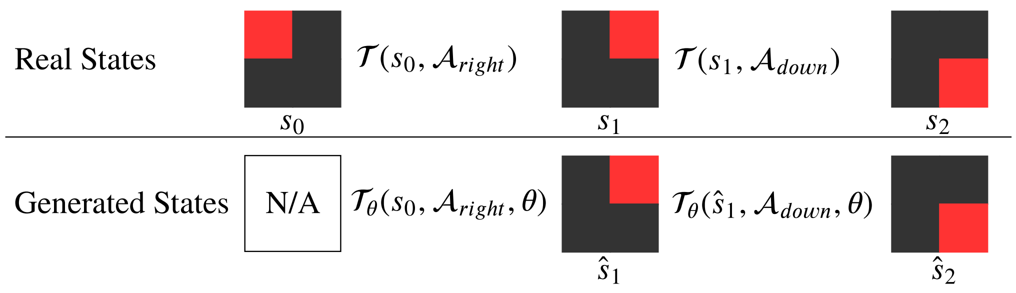

Real States
$\mathcal{T}(s_0, \mathcal{A}_{right})$
$\mathcal{T}(s_1, \mathcal{A}_{down})$
s_0
s_1
s_2
Generated States
N/A
$\mathcal{T}_\theta(s_0, \mathcal{A}_{right}, \theta)$
$\mathcal{T}_\theta(\hat{s}_1, \mathcal{A}_{down}, \theta)$
$\hat{s}_1$
$\hat{s}_2$

Figure 2: DVAE algorithm for generating states using $\mathcal{T}_\theta$ versus the real transition function $\mathcal{T}$. First, a real state is collected from the replay-memory. DVAE can then produce new states from current the trajectory τ using the state-action pairs. θ represent the trainable model parameters.

Table 2 illustrates how the algorithm can generate sequences of artificial trajectories using $\mathcal{T}_\theta = \mathcal{P}(X|\mathcal{Q}(z|X))$, where $z = \mathcal{Q}(z|X)$ is the encoder, and $\mathcal{T}_\theta = \mathcal{P}(X|z)$ is the decoder. With state s_0 and action $\mathcal{A}_{right}$ as input, the algorithm generates state $\hat{s}_1$ which in the

Algorithm 1 The Dreaming Variational Autoencoder

```
 1: Initialize replay memory $\mathcal{D}$ and $\hat{\mathcal{D}}$ to capacity $\mathcal{N}$
 2: Initialize policy $\pi_\theta$
 3: function Run-Agent($\mathcal{T}$, $\mathcal{D}$)
 4:     for i = 0 to N_EPOCHS do
 5:         Observe starting state, $s_0 \sim \mathcal{N}(0, 1)$
 6:         while $s_t$ not TERMINAL do
 7:             $a_t \leftarrow \pi_\theta(s_t = s)$
 8:             $s_{t+1}, r_t, terminal_t \leftarrow \mathcal{T}(s_t, a_t)$
 9:             Store experience into replay buffer $\mathcal{D}(s_t, a_t, r_t, s_{t+1}, terminal_t)$
10:             $s_t \leftarrow s_{t+1}$
11:         end while
12:     end for
13: end function
14: Initialize encoder $\mathcal{Q}(z|X)$
15: Initialize decoder $\mathcal{P}(X|z)$
16: Initialize DVAE model $\mathcal{T}_\theta = \mathcal{P}(X|\mathcal{Q}(z|X))$
17: function DVAE
18:     for $d_i$ in D do
19:         $s_t, a_t, r_t, s_{t+1} \leftarrow d_i$                    ▷ Expand replay buffer pair
20:         $X_t \leftarrow s_t, a_t$
21:         $z_t \leftarrow \mathcal{Q}(X_t)$                          ▷ Encode $X_t$ into latent-space
22:         $\hat{s}_{t+1} \leftarrow \mathcal{P}(z_t)$                 ▷ Decode $z_t$ into probable future state
23:         Store experience into artificial replay buffer $\hat{\mathcal{D}}(\hat{s}_t, a_t, r_t, \hat{s}_{t+1}, terminal_t)$
24:         $\hat{s}_t = \hat{s}_{t+1}$
25:     end for
26:     return $\hat{\mathcal{D}}$
27: end function
```

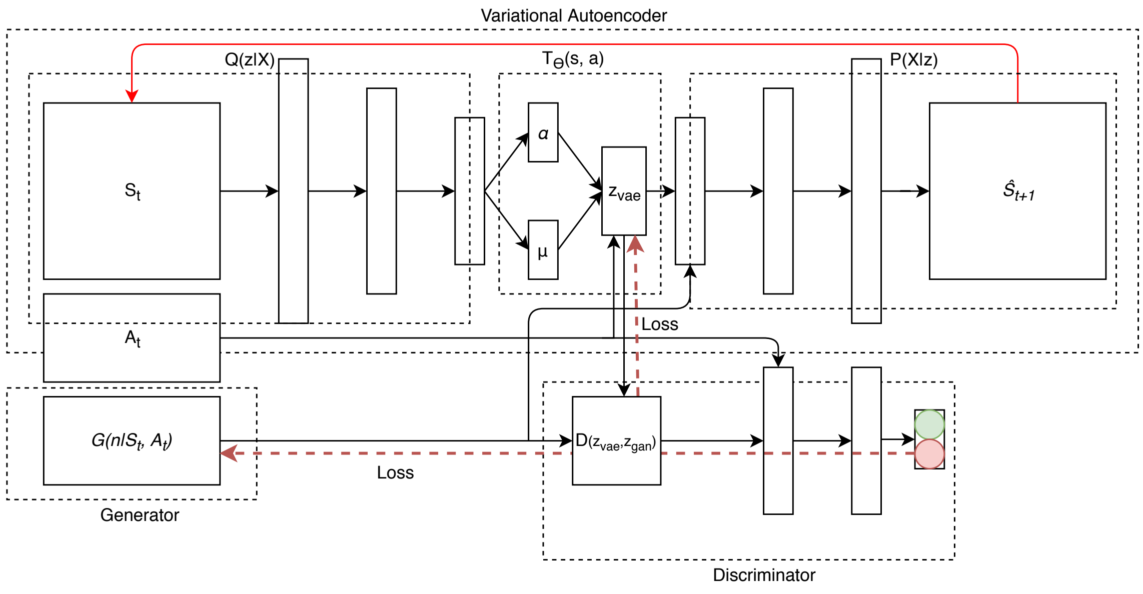


Figure 3: The proposed DVAE-GAN architecture. While the original VAE architecture persist, as described in Algorithm 1, a new generative adversarial networks component is added for increased generalization across the latent-space.

table can be observed is similar to the real state s_1. With the next input, $\mathcal{A}_{down}$, the DVAE algorithm generates the next state $\hat{s}_2$ which again can be observed to be equal to s_2. Note that this is without ever observing state s_1. Hence, the DVAE algorithm needs to be initiated with a state, e.g. s_0, and actions follows. It then generates (dreams) next states,

The requirement is that the environment must be partially discovered so that the algorithm can learn to behave similarly to the target environment. To predict a trajectory of three timesteps, the algorithm does nesting to generate the whole sequence: $\tau = \hat{s}_1, a_1, \hat{s}_2, a_2, \hat{s}_3, a_3 = \mathcal{T}_\theta(\mathcal{T}_\theta(\mathcal{T}_\theta(s_0, \mathcal{A}_{rnd}), \mathcal{A}_{rnd}), \mathcal{A}_{rnd})$. The algorithm does this well early on, but have difficulties with longer state sequences in continuous state-spaces.

Generative Adversarial Network Approach (DVAE-GAN) To combat the divergence behavior in continuous state-space, we extend the model to use generative adversarial networks. The most common cases of divergence are found where there are (1) complex state-transitions, (2) many state-transitions, and (3) sparse state-transitions. In general, we find these properties in continuous and stochastic environments. By using an adversarial approach, we see that DVAE-GAN better generalize for such problems and is far more stable due to increased diversity compared to DVAE (See table 2 for a detailed comparison).

Figure 3 illustrates the proposed DVAE-GAN extension to the original DVAE architecture. In this model, two new components; the *generator G* and *discriminator D* is introduced. This is an adversarial approach previously proposed by [Makhzani et al., 2015]. The generator $G(n|S_t, A_t)$ samples from a Gaussian distribution, while being conditioned on the current state and action, to predict the latent-space distribution z_{gan}. The discriminator $D(z_{vae}, z_{gan})$ is a neural network that tries to predict the validity of the input, in this case, if the latent-space variable is from the ground-truth distribution. A min-max game between the generator and the discriminator fuels learning where the generator minimizes its error towards the real latent-space and the discriminator learns to distinguish between the real and fake latent distribution. In the VAE model, the latent-space distribution is sampled from $z_{vae} = \alpha_t + (\mu_t \times N(0, 1))$ as proposed by [Kingma and Welling, 2013]. The discriminator should then evaluate z_{vae} to be genuine, and its parameters are updated according to the confidence of the prediction. To increase the stability of the VAE, we add the loss of the discriminator to the VAE loss, where we use the original loss function first proposed by [Kingma and Welling, 2013].

Stochastic Weight Averaging Approach (DVAE-SWAGAN) We introduced DVAE-GAN which tries to combat divergence for complex environments. The GAN architecture increases the diversity of the latent-space, but model collapse and instability become a problem. Table 2 outlines a quick overview of the benefits of each of the introduced generative models. VAE is known for its stability but has limited capabilities in its latent-space capacity. The quality of VAE is good but suffers from a severe blurring of the decoded latent-space. Compared to other algorithms, the VAE architecture does not generalize well to a diverse data set. The generative adversarial networks have gotten increased attention due to their diverse latent-space. The images of GANs are known for its sharpness, but they suffer from artifacts in the produced output. These networks are state-of-the-art in the sense that they generalize well to the data set. DVAE-GAN as proposed by [Makhzani et al., 2015] have few artifacts, which is an improvement from the regular GAN architecture. There is, however, high variance in the model which makes it unstable beyond what is seen in vanilla VAE and GAN. To improve the model instability, we introduce Stochastic weight averaging (SWA), first proposed by [Izmailov et al., 2018]. DVAE-SWAGAN is a combination of VAE, SWA, and GAN to significantly reduce the variance of the predictions. The algorithm is in principle the averaged ensemble of DVAE-GAN along the trajectory of SGD. We use a cyclical learning rate ([Smith, 2015]) and average the weights each training iteration creating the DVAE-SWAGAN model, seen in Figure 4.

Table 2: Feature overview of VAE, GAN, DVAE-GAN, and DVAE-SWAGAN. The highlighted text outlines the benefits of the model.

Model	**Features**	**Image Quality**	**Data Generalization**
VAE	• Converge to local minimum • **Stable**	• **Few Artifacts** • Blurry	• Low
GAN	• **Converge to saddle points** • Unstable	• Artifacts • **Sharp**	• Medium
DVAE-GAN	• Converge to saddle points • Unstable	• **Few Artifacts** • Sharp	• Medium
DVAE-SWAGAN	• Converge to saddle points • **Stable**	• **Few Artifacts** • Sharp	• **High**

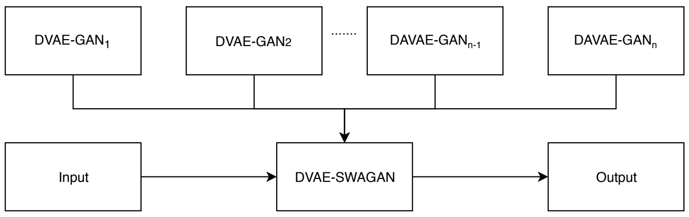
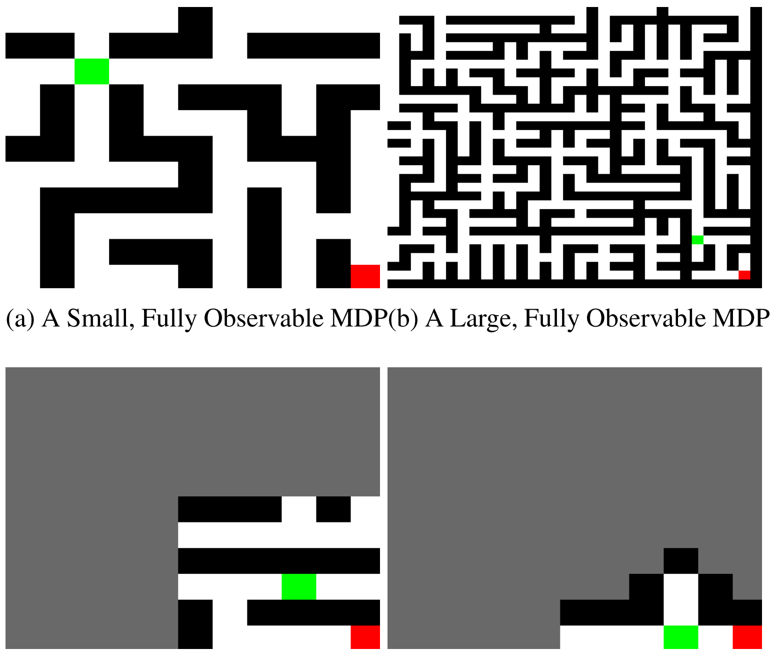


Figure 4: By using ensembles of DVAE-GAN models, the training is significantly more stable and prediction across sparse states yield better results for sequential input.


(a) A Small, Fully Observable MDP(b) A Large, Fully Observable MDP

(c) Partially Observable MDP having a vision distance of 3 tiles (d) Partially Observable MDP having ray-traced vision

Figure 5: Overview of four distinct MDP scenarios using deep maze.

5 Environments

The DVAE algorithm was tested in two environments. The first environment is the deep line wars that were introduced by [Andersen et al., 2017], a simplified Real-Time Strategy game. We present deep maze, a flexible environment with a wide range of challenges suited for reinforcement learning research. The deep line wars environments feature a continuous environment where complex strategies must be planned. The deep maze environment provides simpler rules and a non-continuous action and state-space that in-comparison is far simpler then deep line wars.

The Deep Maze Environment The deep maze is a flexible learning environment for controlled research in exploration, planning, and memory for reinforcement learning algorithms. Maze solving is a well-known problem, and is used heavily throughout the RL literature [Sutton et al., 1999], but is often limited to small and fully-observable scenarios. The deep maze environment extends the maze problem to over 540 unique scenarios including Partially-Observable Markov Decision Processes (POMDP). Figure 5 illustrates a small subset of the available environments in the deep maze, ranging from small-scale MDP's to large-scale POMDP's. The deep maze further features custom game mechanics such as relocated exits and dynamically changing mazes. RL agents depend on sensory input to evaluate and predict the best action at the current timestep. Preprocessing of data is essential so that agents can extract features from the input data. For this reason,

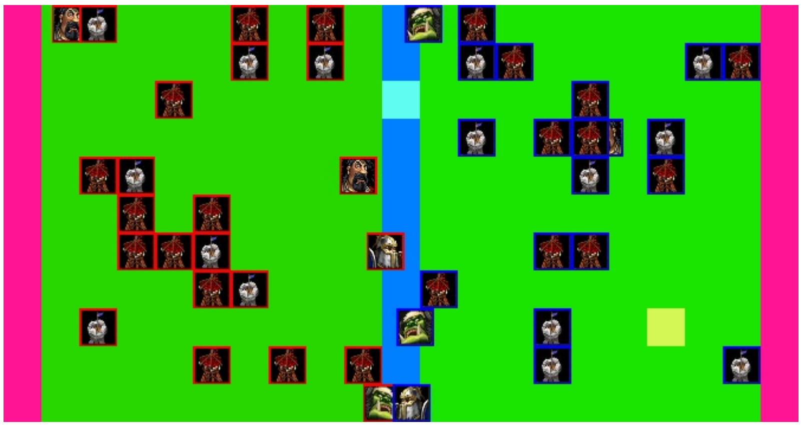


Figure 6: The Graphical User Interface of the deep line wars environment.

deep maze has built-in state representation for imaging and raw state matrices. The game engine is modularized and has an SDK that enables the development of third-party scenarios. This extends the capabilities of deep maze to support nearly all possible scenario combination in the realm of maze solving.[1]

The deep maze learning environment presents the following scenarios. (1) Normal, (2) POMDP, (3) Limited POMDP, and (4) Timed Limited POMDP. The first mode exposes a seed-based randomly generated maze where the state-space an MDP. The second mode narrows the state-space observation to a configurable area around the player. In addition to radius based vision, the POMDP mode also features ray-tracing vision that better mimic the sight of a physical agent. The third and fourth mode is intended for memory research where the agent must find the goal in a limited number of time-steps. In addition to this, the agent is presented with the solution but fades after a few initial time steps. The objective is for the agent to remember the solution to find the goal. All scenario setups have a variable map-size ranging between 2×2 and 56×56 tiles.

5.1 The Deep Line Wars Environment

The deep line wars environment was originally introduced in [Andersen et al., 2017]. Deep line wars is a real-time strategy environment that makes an extensive state-space reduction to enable swift research in reinforcement learning for RTS games.

The game objective of deep line wars is to invade the enemy player with mercenary units until all health points are depleted, see Figure 6. For every friendly unit that enters the far edge of the enemy base, the enemy health pool is reduced by one. When a player purchases a mercenary unit, it spawns at a random location inside the edge area of the buyers base. Mercenary units automatically move towards the enemy base. To protect the

[1]The deep maze is open-source and publicly available at `https://github.com/CAIR/deep-maze`.

base, players can construct towers that shoot projectiles at the opponent's mercenaries. When a mercenary dies, a fair percentage of its gold value is awarded to the opponent. When a player sends a unit, the income is increased by a portion of the units gold value. As a part of the income system, players gain gold at fixed intervals.

6 Experiments

We center our experiments around the DVAE, DVAE-GAN, DVAE-SWA, and DVAE-SWAGAN. The goal of the proposed extensions is to improve the model stability so that the DVAE algorithm can produce better quality output for continuous and sparse state-spaces. In this section, we show results of model-based reinforcement learning using DVAE in the deep-maze and deep-line wars environment. We show the performance of encoding raw pixel input to a compact representation and to decode this representation to probable future states.

6.1 The Dreaming Variational Autoencoder

Deep Maze For the deep maze environment, the algorithm must be able to generalize over many similar states to model a large state-space. DVAE aims to learn the transition function $\mathcal{T}(s_t, a_t)$, bringing the state from s_t to s_{t+1}. We use the deep maze environment because it provides simple rules, with a controllable state-space complexity. Also, we can omit the importance of reward for some scenarios.

We trained the DVAE model on two *No-Wall deep maze* scenarios of size 2×2 and 8×8. For the encoder and decoder, we used the same convolution architecture as proposed by [Pu et al., 2016] and trained for 5000 epochs[2] in the 8×8 scenario and 1000 epochs for 2×2 respectively. The reasoning behind different epoch lengths is because we expect simpler environments to converge faster. For the encoding of actions and states, we concatenate the flattened state-space and action-space, having a fully-connected layer with ELU activation before calculating the latent-space. We used the Adam optimizer [Kingma and Ba, 2015] with a learning-rate of 3e-05 to update the parameters. To calculate the loss we used the same loss function as proposed by [Kingma and Welling, 2013].

Figure 7 illustrates the loss during the training phase for the DVAE algorithm in the No-Wall Deep Maze scenario. In the 2×2 scenario, DVAE is trained on only 50% of the state space, which results in noticeable graphics artifacts in the prediction of future states, seen in Figure 8. In the 8×8 environment, the algorithm is allowed to train on all possible states, and we observe in Figure 9 that there is significantly better image quality throughout the sampling process.

[2]We regard the term epoch as an episode, which is frequently in literature.

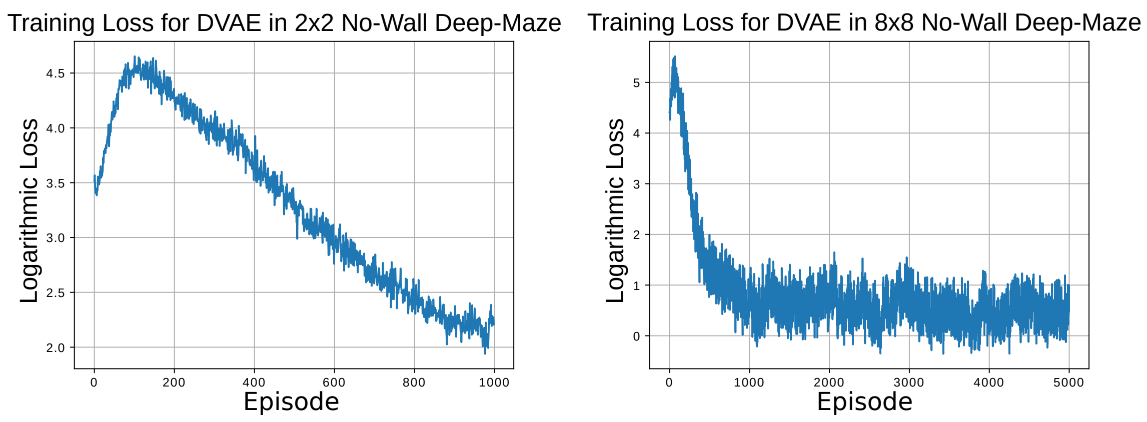


Figure 7: The training loss for DVAE in the 2×2 No-Wall and 8×8 deep maze scenario. The experiment is run for a total of 1000 (5000 for 8×8) epochs. The algorithm only trains on 50% of the state-space to the model for the 2×2 environment while the whole state-space is trainable in the 8×8 environment.

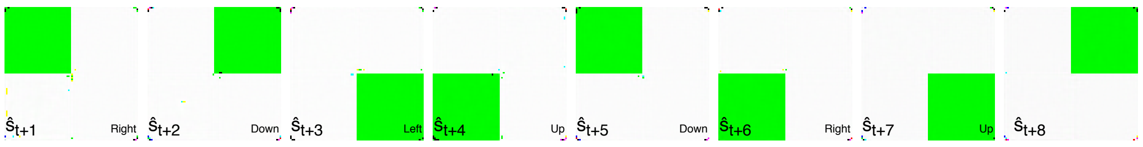


Figure 8: For the 2×2 scenario, only 50% of the environment is explored, leaving artifacts on states where the model is uncertain of the transition function. In more extensive examples, the player disappears, teleports or gets stuck in unexplored areas.

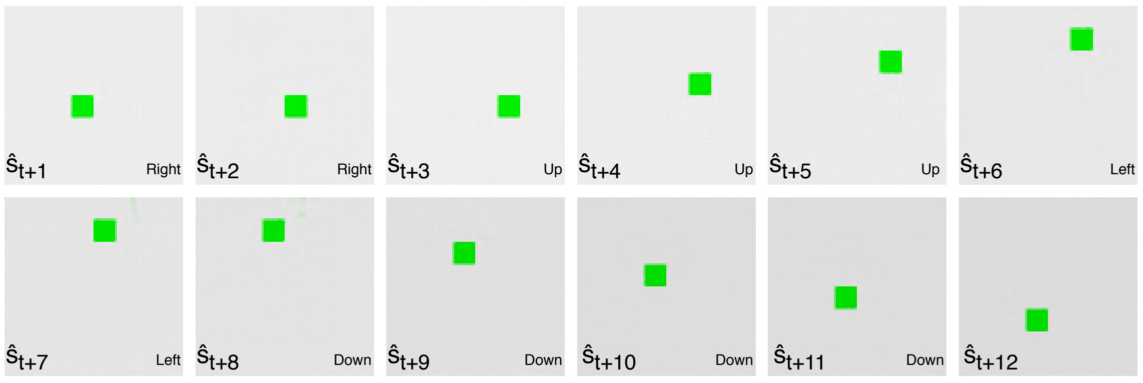


Figure 9: Results of 8×8 Deep Maze modeling using the DVAE algorithm. To simplify the environment, no reward signal is received per iteration. The left caption describes current state, s_t, while the right caption is the action performed to compute, $s_{t+1} = \mathcal{T}(s_t, a_t)$.

Table 3: Results of the deep maze 11×11 and 21×21 environment, comparing DQN [Mnih et al., 2015], TRPO [Schulman et al., 2015], and PPO [Schulman et al., 2017]. The optimal path yields performance of 100% while no solution yields 0%. Each of the algorithms ran 10000 epochs for both map-sizes. Converged Epoch represents at which epoch the algorithm converged during training.

Algorithm	11×11	Avg Perf.	Converged Epoch	21×21	Avg Perf.	Converged Epoch
DQN-$\hat{\mathcal{D}}$		94.56%	9314		64.36%	N/A
TRPO-$\hat{\mathcal{D}}$		96.32%	5320		78.91%	7401
PPO-$\hat{\mathcal{D}}$		98.71%	3151		89.33%	7195
DQN-$\mathcal{D}$		98.26%	4314		84.63%	8241
TRPO-$\mathcal{D}$		99.32%	3320		92.11%	4120
PPO-$\mathcal{D}$		99.35%	2453		96.41%	2904

Training RL-Agents using $\hat{\mathcal{D}}$ replay-buffer The goal of this experiment is to observe the performance of RL-agents using the generated experience-replay $\hat{\mathcal{D}}$ from Algorithm 1 in Deep Maze environments of size 11×11 and 21×21. In Table 3, we compare the performance of DQN [Mnih et al., 2013], TRPO [Schulman et al., 2015], and PPO [Schulman et al., 2017] using the DVAE generated $\hat{\mathcal{D}}$ to tune the parameters. Because TRPO and PPO are on-policy algorithms, the generated states must be generated on-the-fly so that the algorithm remains on-policy.

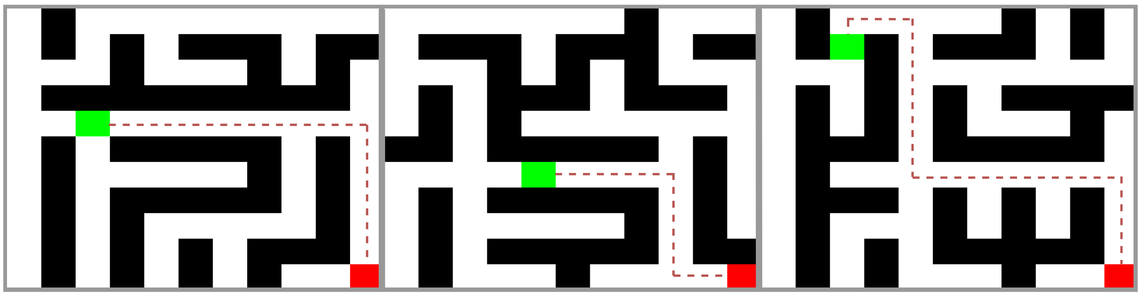


Figure 10: A typical deep maze of size 11×11. The lower-right square indicates the goal state, the dotted-line is a retrace of the predicted optimal path for that maze, while the final square represents the player's current position in the state-space. The controller agent is DQN, TRPO, and PPO (from left to right).

Figure 10 illustrates three maze variations of size 11×11, where the agent has learned the optimal path. We see that the best performing algorithm, PPO [Schulman et al., 2017] beats DQN and TRPO using either $\hat{\mathcal{D}}$ or $\mathcal{D}$. The DQN-$\hat{\mathcal{D}}$ agent did not converge in the 21×21 environment, but it is likely that value-based algorithms could struggle to map inaccurate states with graphical artifacts generated from the DVAE algorithm. These artifacts significantly increase the state-space significantly, but empirical data suggest that on-policy algorithms perform better on noisy state-spaces.

Deep Line Wars The DVAE algorithm works well in more complex environments, such as the deep line wars game environment [Andersen et al., 2017]. Here, we expand the DVAE algorithm with LSTM to improve the capability of generating time-bound data, such as animations seen in Figure 1.

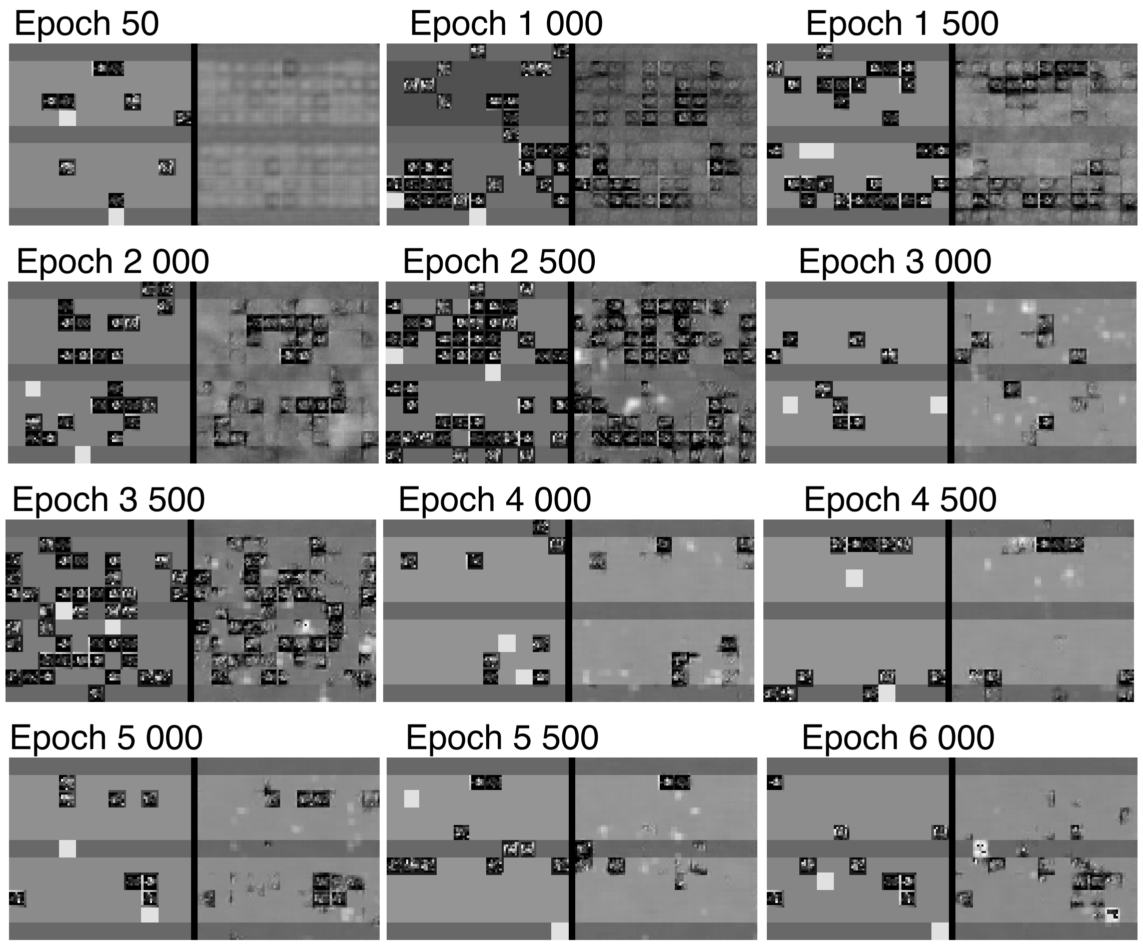
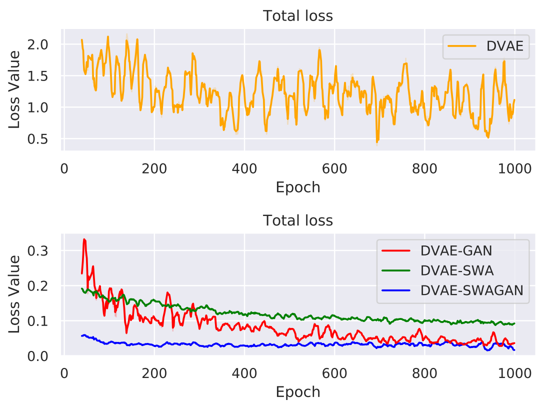


Figure 11: The DVAE algorithm applied to the deep line wars environment. Each epoch illustrates the quality of generated states in the game, where the left image is real state s, and the right image is the generated state $\hat{s}$.

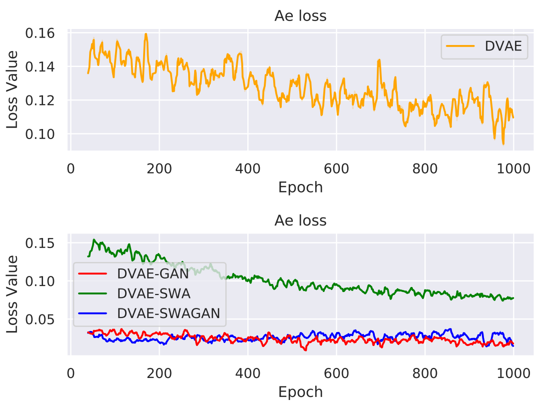
Figure 11 illustrates the state quality during training of DVAE in a total of 6000 epochs. Both players draw actions from a Gaussian distributed policy. The algorithm understands that the player units can be located in any tiles after only 50 epochs, and at 1000 epochs we observe the algorithm makes significantly better predictions of the probability of unit locations (i.e., some units show more densely in the output state). At the end of the training, the DVAE algorithm is to some degree capable of determining both towers, and unit locations at any given time-step during the game epoch.

6.2 Extending The Dreaming Variational Autoencoder

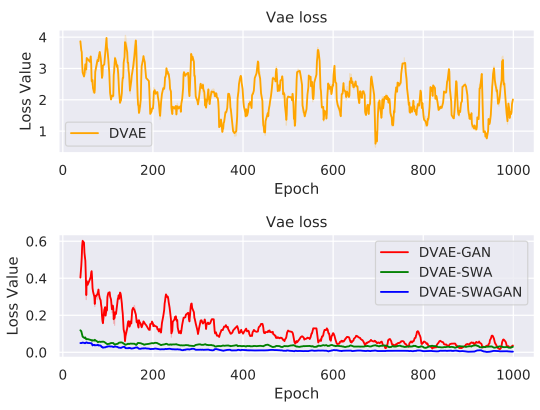
The goal of DVAE-SWA, DVAE-GAN and DVAE-SWAGAN is to perform better in continuous state-spaces, such as the deep line wars environment. The experiments were performed using a map size of 11×11 sampling actions from a PPO policy.



(a) Total Loss for the tested algorithms.



(b) Autoencoder loss (L2) for the algorithms.



(c) Variational Autoencoder Loss (KL) for the tested algorithms.

Figure 12: Training of DVAE compared to the DVAE-SWA, DVAE-GAN and DVAE-SWAGAN extension.

Figure 12 shows the training loss of the algorithms DVAE, DVAE-SWA, DVAE-GAN, and DVAE-SWAGAN for 1000 epochs (x-axis), where the y-axis describes the loss value. The new architectures perform significantly better than DVAE across all loss components of the architecture. The consequence of lower loss is better image quality (autoencoder loss), and better transitions (variational loss). For these experiments, we tried to model the deep line wars environment using RGB input. Figure 13 illustrates the resulting images

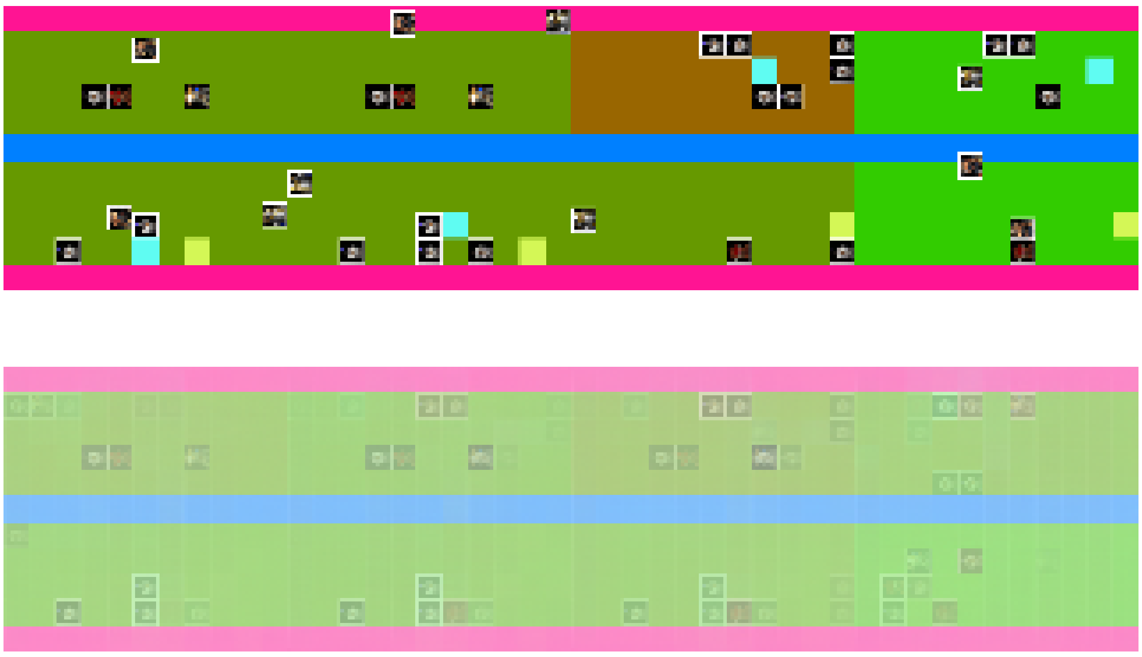


Figure 13: The first row represents the ground truth future state, while the second row is the predicted future state. The DVAE-SWAGAN algorithm sampled states after training for 1000 epochs, see Figure 12. Notice that the quality is notably better compared to Figure 11. We found that states were generalized too much producing some inaccurate predictions. Despite this inaccuracy, the RL agents learned a policy capable of beating random agents. We believe this is because similar states often represent similar value functions.

for each of the algorithms. Here, we see that DVAE-SWAGAN perform significantly best, in terms of quality and accuracy.

7 Conclusion and Future Work

This paper introduces *The Dreaming Variational Autoencoder* along with its extensions DVAE-SWA, DVAE-GAN and DVAE-SWAGAN as a neural network based generative modeling architecture to enable exploration in environments with sparse reward.

The DVAE algorithm successfully generates authentic world models in non-continuous state-spaces where the dynamics of the environment is simple. It works well for small environments but is limited when the state-sequence become too large. The algorithm performs marginally better when using LSTM for sequence prediction, but is a significant performance drop due to the increased model complexity. For most environments, such as deep line wars and deep maze, it is sufficient to run DVAE using only fully-connected nodes.

The DVAE-SWAGAN improves the original model significantly and enables the algorithm to imitate environment models with continuous state-space better. DVAE-SWAGAN performs better in all environments including deep line wars and most environments found in the GYM reinforcement learning environment.

There are, however, several fundamental issues that limit DVAE, and DVAE-SWAGAN from fully modeling environments. In some situations, exploration may be a costly act that makes it impossible to explore all parts of the environment in its entirety. The algorithms cannot accurately predict the outcome of unexplored areas of the state-space, making the prediction blurry or incorrect. To combat this, the model should be improved further to include some sense of logic, and understanding of the environment dynamics. In current state-of-the-art, this frequently introduced as domain knowledge that is manually crafted by the programmer, but the hope is that future research will find a method for self-supervised domain knowledge modeling.

Reinforcement learning has many unresolved problems, and the hope is that the deep maze and the deep line wars learning environment can be a useful tool for future research. For future work, we plan to introduce an inverse reinforcement learning component to learn the reward function $\hat{\mathcal{R}}$. We also plan to explore non-parametric variants. DVAE and environment modeling is an ongoing research question, and the goal is that reinforcement learning algorithms could utilize this form of *dreaming* to make the algorithm far more sample efficient.

References

[Andersen et al., 2017] Andersen, P.-A., Goodwin, M., and Granmo, O.-C. (2017). Towards a deep reinforcement learning approach for tower line wars. In Bramer, M. and Petridis, M., editors, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 10630 LNAI, pages 101–114.

[Arulkumaran et al., 2019] Arulkumaran, K., Cully, A., and Togelius, J. (2019). AlphaStar: An Evolutionary Computation Perspective. Technical report.

[Arulkumaran et al., 2017] Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38.

[Azar et al., 2019] Azar, M. G., Piot, B., Pires, B. A., Grill, J.-B., Altché, F., and Munos, R. (2019). World Discovery Models. *arxiv preprint arXiv:1902.07685*.

[Bangaru et al., 2016] Bangaru, S. P., Suhas, J., and Ravindran, B. (2016). Exploration for Multi-task Reinforcement Learning with Deep Generative Models. *arxiv preprint arXiv:1611.09894*.

[Blundell et al., 2016] Blundell, C., Uria, B., Pritzel, A., Li, Y., Ruderman, A., Leibo, J. Z., Rae, J., Wierstra, D., and Hassabis, D. (2016). Model-Free Episodic Control. *arxiv preprint arXiv:1606.04460*.

[Buesing et al., 2018] Buesing, L., Weber, T., Racaniere, S., Eslami, S. M. A., Rezende, D., Reichert, D. P., Viola, F., Besse, F., Gregor, K., Hassabis, D., and Wierstra, D. (2018). Learning and Querying Fast Generative Models for Reinforcement Learning. *arxiv preprint arXiv:1802.03006*.

[Chua et al., 2018] Chua, K., Calandra, R., McAllister, R., and Levine, S. (2018). Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models. *Advances in Neural Information Processing Systems 31*.

[Ha and Schmidhuber, 2018] Ha, D. and Schmidhuber, J. (2018). Recurrent World Models Facilitate Policy Evolution. *Advances in Neural Information Processing Systems 31*.

[Hafner et al., 2018] Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. (2018). Learning Latent Dynamics for Planning from Pixels. *Proceedings of the 36 th International Conference on Machine Learning*.

[Higgins et al., 2016] Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A. (2016). beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. *International Conference on Learning Representations*.

[Higgins et al., 2017] Higgins, I., Pal, A., Rusu, A., Matthey, L., Burgess, C., Pritzel, A., Botvinick, M., Blundell, C., and Lerchner, A. (2017). DARLA: Improving Zero-Shot Transfer in Reinforcement Learning. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1480–1490, International Convention Centre, Sydney, Australia. PMLR.

[Izmailov et al., 2018] Izmailov, P., Podoprikhin, D., Garipov, T., Vetrov, D., and Wilson, A. G. (2018). Averaging Weights Leads to Wider Optima and Better Generalization.

[Kaelbling et al., 1996] Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*.

[Kingma and Ba, 2015] Kingma, D. P. and Ba, J. L. (2015). Adam: A Method for Stochastic Optimization. *Proceedings, International Conference on Learning Representations 2015.*

[Kingma and Welling, 2013] Kingma, D. P. and Welling, M. (2013). Auto-Encoding Variational Bayes. *arxiv preprint arXiv:1312.6114.*

[Li, 2017] Li, Y. (2017). Deep Reinforcement Learning: An Overview. *arxiv preprint arXiv:1701.07274.*

[Liang et al., 2018] Liang, X., Wang, Q., Feng, Y., Liu, Z., and Huang, J. (2018). VMAV-C: A Deep Attention-based Reinforcement Learning Algorithm for Model-based Control. *arxiv preprint arXiv:1812.09968.*

[Makhzani et al., 2015] Makhzani, A., Shlens, J., Jaitly, N., Goodfellow, I., and Frey, B. (2015). Adversarial Autoencoders.

[Mnih et al., 2016] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous Methods for Deep Reinforcement Learning. In Balcan, M. F. and Weinberger, K. Q., editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA. PMLR.

[Mnih et al., 2013] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. *Neural Information Processing Systems.*

[Mnih et al., 2015] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

[Mousavi et al., 2018] Mousavi, S. S., Schukat, M., and Howley, E. (2018). Deep Reinforcement Learning: An Overview. In Bi, Y., Kapoor, S., and Bhatia, R., editors, *Proceedings of SAI Intelligent Systems Conference (IntelliSys) 2016*, pages 426–440. Springer International Publishing, Cham.

[Pu et al., 2016] Pu, Y., Gan, Z., Henao, R., Yuan, X., Li, C., Stevens, A., and Carin, L. (2016). Variational Autoencoder for Deep Learning of Images, Labels and Captions. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and R., G., editors, *Advances in Neural Information Processing Systems*, pages 2352–2360. Curran Associates, Inc.

[Schulman et al., 2015] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust Region Policy Optimization. In Bach, F. and Blei, D., editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France. PMLR.

[Schulman et al., 2017] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal Policy Optimization Algorithms. *arxiv preprint arXiv:1707.06347*.

[Silver et al., 2017] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., Van Den Driessche, G., Graepel, T., and Hassabis, D. (2017). Mastering the game of Go without human knowledge. *Nature*.

[Smith, 2015] Smith, L. N. (2015). Cyclical Learning Rates for Training Neural Networks.

[Sutton et al., 1999] Sutton, R. S., Precup, D., and Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. Technical report.

[Xiao and Kesineni, 2016] Xiao, T. and Kesineni, G. (2016). Generative Adversarial Networks for Model Based Reinforcement Learning with Tree Search. Technical report, University of California, Berkeley.

[Zhang et al., 2018] Zhang, C., Patras, P., and Haddadi, H. (2018). Deep Learning in Mobile and Wireless Networking: A Survey. *IEEE Communications Surveys & Tutorials*.

Author Biography

Per-Arne Andersen is a Ph.D candidate in Artificial Intelligence at the University of Agder, Norway where he also completed his B.Sc. and M.Sc in 2015 and 2017, respectively. He has expertise in several domains in computer science, including security, software development and machine learning. His primary field of research is reinforcement learning in environments with incomplete information, including real-time games and industry-near systems. Toward the future, Per-Arne hopes to have an active role in the scientific community for reinforcement learning and to contribute towards the understanding of artificial general intelligence.

Morten Goodwin received the B.Sc. and M.Sc. degrees from University of Agder, Norway, in 2003 and 2005, respectively, and the Ph.D. degree from Aalborg University Department of Computer Science, Denmark, in 2011, on applying machine learning algorithms on eGovernment indicators which are difficult to measure automatically. He is an Associate Professor with the Department of ICT, University of Agder, deputy director for Centre for Artificial Intelligence Research, coordinator for the International Master's Programme in Artificial Intelligence, a public speaker, and an active researcher. His main interests include swarm intelligence, deep learning, and adaptive learning in the fields of medicine, games and chatbots. He has more than 70 peer reviews scientific publications, and has supervised more than 110 student projects including Master's and Ph.D. theses.

Prof. Ole-Christoffer Granmo is director and founder of the Centre for Artificial Intelligence Research (CAIR) at the University of Agder, Norway. He obtained his master's degree in 1999 and the PhD degree in 2004, both from the University of Oslo, Norway. Granmo develops theory and algorithms for systems that explore, experiment and learn in complex real-world environments. His research interests include artificial intelligence, machine learning, learning automata, bandit algorithms, deep reinforcement learning, Bayesian reasoning, and computational linguistics. Within these areas of research, Dr. Granmo has written more than 125 refereed journal and conference publications. He is co-founder of the Norwegian Artificial Intelligence Consortium (NORA). Apart from his academic endeavors, Granmo is also co-founder of the company Anzyz Technologies AS

Paper F: Increasing Sample Efficiency in Deep Reinforcement Learning using Generative Environment Modelling

Paper G

Title:	Towards Safe Reinforcement-learning in Industrial Grid-warehousing
Authors:	Andersen, Per-Arne et al.
Affiliation:	Department of ICT, University of Agder, Grimstad Norway
Journal:	Information Sciences. Impact Factor in 2020: 6.795
Year:	2020

Towards Safe Reinforcement-Learning in Industrial Grid-Warehousing

Per-Arne Andersen
Department of ICT
University of Agder
Grimstad, Norway
per.andersen@uia.no

Morten Goodwin
Department of ICT
University of Agder
Grimstad, Norway
morten.goodwin@uia.no

Ole-Christoffer Granmo
Department of ICT
University of Agder
Grimstad, Norway
ole.granmon@uia.no

Abstract

Reinforcement learning has shown to be profoundly successful at learning optimal policies for simulated environments using distributed training with extensive compute capacity. Model-free reinforcement learning uses the notion of trial and error, where the error is a vital part of learning the agent to behave optimally. In mission-critical, real-world environments, there is little tolerance for failure and can cause damaging effects on humans and equipment. In these environments, current state-of-the-art reinforcement learning is not sufficient to learn optimal control policies safely.

On the other hand, model-based reinforcement learning tries to encode environment transition dynamics into a predictive model. The transition dynamics describes the mapping from one state to another, conditioned on an action. If this model is accurate enough, the predictive model is sufficient to train agents for optimal behavior in real environments.

The paper presents the *Dreaming Variational Autoencoder* (DVAE) for learning good policies safely with a significantly lower risk of catastrophes occurring during training. The algorithm combines variational autoencoders, risk-directed exploration, and curiosity to train deep-q networks inside *"dream"* states. We introduce a novel environment,

Elsevier: Journal of Information Sciences
Impact Factor: 6.795 (2020)
Submitted 27. Aug, 2019, Accepted 23. Apr 2020, Published 05. Aug, 2020

ASRS-Lab, for research in the safe learning of autonomous vehicles in grid-based warehousing. The work shows that the proposed algorithm has better sample efficiency with similar performance to novel model-free deep reinforcement learning algorithms while maintaining safety during training.

Keywords: Deep Reinforcement Learning, Model-based Reinforcement Learning, Reinforcement Learning, Neural Networks, Variational Autoencoder, Markov Decision Processes, Exploration, Artificial Intelligence, Safe Reinforcement Learning

1 Introduction

Reinforcement learning has recently demonstrated a high potential to learn efficient strategies in environments where there are noisy or incomplete data [1]. We find these achievements in many domains, such as robotics [2], wireless networking [3], and game-playing [4]. The common denominator between these domains is that they can be computer-simulated with significant resemblance to real-world environments. For this reason, reinforcement learning algorithms train at accelerated rates without the risk of compromising the safety of real-world systems [5].

The goal of reinforcement learning algorithms is to learn a policy (or behavior) that stimulates optimal actions based on sensory input and feedback from an environment. A policy is a parameterized model that is constructed in (exact) tabular form or using an (approximation) neural network with algorithms such as gradient descent [6]. The algorithm performs an iterative process of (sampling) exploration, exploitation, and (learning) policy updates that moves the policy in the direction of the desired behavior. Exploration is commonly performed using a separate policy, such as a (random sampling) Gaussian distribution. It is crucial that the algorithm balance exploration and exploitation with schemes such as ϵ-greedy [7] so that the policy updated towards a generalization of the whole environment.

The problems of guaranteed safety during reinforcement learning are many. (1) It requires a tremendous amount of sampling to learn a good policy [8]. (2) Stable and safe policies are challenging to achieve in non-deterministic and even deterministic, for fast-changing environments [9]. (3) Conventional model-free exploration methods are not safe in mission-critical environments. (4) Reinforcement learning methods depend on negative feedback to avoid catastrophic states and should be avoided for mission-critical systems [10]. Most reinforcement learning techniques are not designed for safe learning, and therefore, few solutions exist for mission-critical real-world environments [11].

Automated Storage and Retrieval Systems (ASRS) are a modern method of performing warehouse logistics where the system is partially or fully automated [12]. In industry, including ASRS, it is common to rely on complex expert systems to perform tasks such as control, storage, retrieval, and scheduling. If-else statements and traditional pathfinding algorithms drive these tasks. The benefit of expert systems is that it is trivial to model operative safety bounds that limit the system from entering catastrophic states. The downside is that expert systems do not adapt to changes automatically, and requires extensive testing if the environment is modified [13]. While it may be possible and perhaps trivial to construct safe routines with an expert system, it is inconceivable to expect optimal behavior due to the complexity of most real-world environments [14]. Reinforcement learning is perhaps the most promising approach to solve these problems because it can generalize well across many domains [4], and is designed to work in noisy environments with partial state-space visibility [15].

We propose *The Dreaming Variational Autoencoder* (DVAE), an algorithm for safer learning in real-world environments. DVAE is an improved version of previous work in [16] that emphasize on more reliable learning in mission-critical environments. The algorithm tries to address the problems (1, 2, 3, 4) that concern safe and sample efficient reinforcement learning. The algorithm does not require direct access to the real-world environment or prior knowledge to learn a stable policy. It is, however, possible to inject prior knowledge of catastrophic states to strengthen safer learning.

The key contributions of this paper are summarized as follows:

- DVAE, a predictive model for n-state predictions,
- safety constraints using a constrained MDP scheme,
- safe exploration through risk-directed exploration and curiosity,
- ASRS-Lab for industry near testing of the proposed approach,
- and analysis of empirical results.

The organization of the paper follows. Section 2 outlines progress in the field, including automated storage and retrieval systems, model-based reinforcement learning, and safe reinforcement learning. Section 3 presents the theoretical background of the proposed algorithm. Section 4 details the DVAE algorithm thoroughly and discuss the convergence guarantee for the algorithm. Section 5 introduces *The ASRS-Lab*, an industry-near learning environment that simulates real-world ASRS systems. The results are presented in Section 6 and show that the algorithm act safer than model-free reinforcement learning. The paper is finally summarized in Section 7 and proposes future work in the field of safe reinforcement learning.

2 Related Work

Recently, advancements in reinforcement learning have more frequent and included substantial performance improvements in numerous domains [17]. Many aspects play a role, but notwithstanding increased publicity, which attracts new institutions to work with reinforcement learning. This section presents work that relates to reinforcement learning in industry-near environments and research that attempts to address safety in these domains.

2.1 Reinforcement Learning

Reinforcement learning is applied previously in industry-near environments, and perhaps the most widespread application is autonomous vehicles. The proposed method in this paper uses an auxiliary policy to label data for supervised training. With only 12 hours of labeled data, [18] illustrates learning performant policies using a direct perception approach with convolutional neural networks. This approach is much like a variational autoencoder that simplifies the perception of the world significantly. This simplifaction of the input significantly speeds up inference, which enables the system to issue control commands more frequently. Many other significant contributions in autonomous vehicle control directly relate to control in ASRS environments, such as [19, 20, 21].

2.2 Model-based RL

In model-based reinforcement learning, the goal is to learn state-transitions based on observations from the environment, the predictive model. If the predictive model is stable, with low variance and improves monotonically during training, it is, to some degree, possible to learn model-free agents to act optimally in environments that have never been observed directly.

Perhaps the most sophisticated algorithm for model-based reinforcement learning is the Model-based policy optimization (MBPO) algorithm, proposed by Janner et al. [22] The authors empirically show that MBPO performs significantly better in continuous control tasks compared to previous methods. MBPO proves to be monotonically improving, given that the following bounds hold:

$$\eta[\pi] \geq \hat{\eta}[\pi] - C$$

where $\eta[\pi]$ denotes the returns in the real environment under a policy whereas $\hat{\eta}[\pi]$ denotes the returns in the predicted model under policy π. Furthermore, the authors show that as long as they can improve the C, the performance will increase monotonically [22].

Gregor et al. proposed a scheme to train expressive generative models to learn belief-states of complex 3D environments with little prior knowledge. Their method was effective in predicting multiple steps into the future (overshooting) and significantly improve sample efficiency. In the experiments, the authors illustrated model-free policy training in several environments, including DeepMind Lab. However, the authors found it difficult to use their predictive model in model-free agents directly. [23]

Neural Differential Information Gain Optimization (NDIGO) algorithm by Azar et al. is a self-supervised exploration model that learns a world model representation from noisy data. The primary features of NDIGO are its robustness to noise due to their method to cancel out negative loss and to give positive learning more value. The authors show in their maze environment that the model successfully converges towards an optimal world model even when introducing noise. The author claims that the algorithm outperforms the state-of-the-art, such as Recurrent World Models [24].

The Dreaming Variational Autoencoder (DVAE) is an end-to-end solution for predicting the probable future state $p(\hat{s}_{t+1}|s_t, a_t)$. The authors showed that the algorithm successfully predicted the next state in non-continuous environments and could, with some error, predict future states in continuous state-space environments such as the Deep Line Wars environment. In the experiments, the authors used DQN, PPO, and TRPO using an artificial buffer to feed states to the algorithms. In all cases, the DVAE algorithm was able to create buffers that were accurate enough to learn a near-optimal policy. [16]

The algorithm VMAV-C is a combination of VAE and attention-based value function (AVF), and mixture density network recurrent neural network (MDN-RNN) from [25]. This modification to the original World Models algorithm improved performance in the Cart Pole environment. They used the on-policy algorithm PPO to learn the optimal policy from the latent representation of the state-space [26].

Deep Planning Network (PlaNet) is a model-based agent that interprets the pixels of a state to learn a predictive model of an environment. The environment dynamics are stored into latent-space, where the agent sample actions based on the learned representation. The proposed algorithm showed significantly better sample efficiency compared to model-free algorithms such as A3C [27].

In *Recurrent World Models Facilitate Policy Evolution*, a novel architecture for training RL algorithms using variational autoencoders. This paper showed that agents could successfully learn the environment dynamics and use this as an exploration technique requiring no interaction with the target domain. The architecture is mainly three components; vision, controller, and model, the vision model is a variational autoencoder that outputs a latent-space variable of an observation. The latent-space variable is processed in the model and feeds into the controller for action decisions. Their algorithms show state-of-the-art performance in self-supervised generative modeling for reinforcement learning agents. [25]

Chua et al. proposed *Probabilistic Ensembles with Trajectory Sampling* (PETS). The algorithm uses an ensemble of bootstrap neural networks to learn a dynamics model of the environment over future states. The algorithm then uses this model to predict the best action for future states. The authors show that the algorithm significantly lowers sampling requirements for environments such as half-cheetah compared to SAC and PPO. [28]

DARLA is an architecture for modeling the environment using β-VAE [29]. The trained model was used to learn the optimal policy of the environment using algorithms such as DQN [4], A3C, and Episodic Control [30]. DARLA is, to the best of our knowledge, the first algorithm to introduce learning without access to the ground-truth environment during training.

For further details on model-based RL, we refer the reader to [31].

2.3 Safe Reinforcement Learning

A majority of established systems in the industry, an expert system already acts as the controller for the environment. In real-world environments, the need for safe and stable learning is critical so that existing routines are not interrupted.

Similar to the proposed algorithm, [32] assumes a predictive model that learns the dynamics of the environment. The authors propose that the policy should be limited to a safe-zone, called the **R**egion **O**f **A**ttraction (ROA). Everything within the bounds of the ROA is considered “safe states” that the policy can visit, and during training, the ROA gradually expands by carefully exploring unknown states. The algorithm shrinks the ROA to ensure stability if the feedback indicates movement towards catastrophic states.

The proposed algorithm encodes the observations as latent embeddings using a variational autoencoder (VAE) similar to the View model in [25]. In the world model approach, the authors define three components. The (VAE) *view* encodes observations to a compact latent embedding. The *model* (MDM-RNN)[1] is the predictive model used to learn the (predictive model) world model. Finally, the (C) *controller* is a general framework that enables model-free algorithms to interact with the world model.

3 Background

The optimization problem is modeled as **M**arkov **D**ecision **P**rocesses (MDP). The MDP consists of the tuple (S, A, R, P, γ) where S is the set of possible states, A is the set of possible actions, $R\colon S \times A \times S \to \mathbb{R}$ is the reward function, $P\colon S \times A \times S \to [0, 1]$ is the transition probability function (where $P(s'|s, a)$ denotes the probability of transitioning

[1]Mixture Density Network combined with a Recurrent Neural Networks.

to next state s' given that the agent takes action a in state s), and $\gamma \in [0, 1]$ is the discount factor for future rewards.

A policy (π) is a parameterized model that maps together (input) observations and (output) actions to form behavior. **The goal** of the reinforcement learning agent is to select actions in a way that maximizes future rewards [6].

$$G_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} \tag{1}$$

Equation 1 denotes the discounted cumulative future rewards, often referred to as the *discounted return* in literature [6]. We assume that if the policy adjusts its parameters to find actions towards maximizing the return, the policy will ultimately converge optimally.

$$\pi^* = \arg\max_{\pi} V^{\pi}(s) \quad \forall s \in S \tag{2}$$

Equation 2 denotes the optimal policy and is a policy that yields the highest attainable state-value $V^{\pi}(s)$ for all states while under the control of the policy π [4]. The state-value function is denoted

$$V^{\pi}(s) = \mathbb{E}_{\pi}[G_t | S_t = s] = \mathbb{E}_{\pi}[R_{t+1} + \gamma V^{\pi}(S_{t+1}) | S_t = s], \tag{3}$$

and quantifies how good it is for an agent to be in a particular state. Furthermore, the state-action function indicates how good it is for the agent to take any possible action being in state s [4], where

$$V^*(s) = \max_{a \in A} Q^*(s, a) \quad \forall s \in S \tag{4}$$

describes the relationship between the state-value and state-action function. As long as the agent selects actions that maximize the Q-values, the state-value is also optimal [6]. Therefore,

$$\pi^* = \arg\max_{a \in A} Q^*(s) \quad \forall s \in S \tag{5}$$

the optimal policy is found at the point where the agent always makes actions that maximize the Q-value.

Traditional RL learns the optimal policy according to an *optimization criterion*. This optimization criterion varies with different algorithms but is commonly implemented to minimize time or to maximize reward. The *return maximization criterion* is frequently used in Q-Learning, where

$$Q^{\pi}(s_t, a_t) \leftarrow Q^{\pi}(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_{a \in A} Q^{\pi}(s_{t+1}, a) - Q^{\pi}(s_t, a_t)]^2 \tag{6}$$

[2]The equation illustrates the Q-Learning algorithm without any extensions and without deep learning considerations.

backpropagates the Q-estimate of the following state to the former state.

It becomes evident that there is **no safety guarantee** in the traditional view of reinforcement learning [32]. The primary focus is for the agent to find the policy that maximizes some feedback signal, and through dynamic programming, monte-carlo methods, or temporal-difference, find a way to learn by trial and error. For mission-critical environments, reinforcement learning is insufficient, and therefore, we seek a method to learn good policies while reducing the number of catastrophic states.

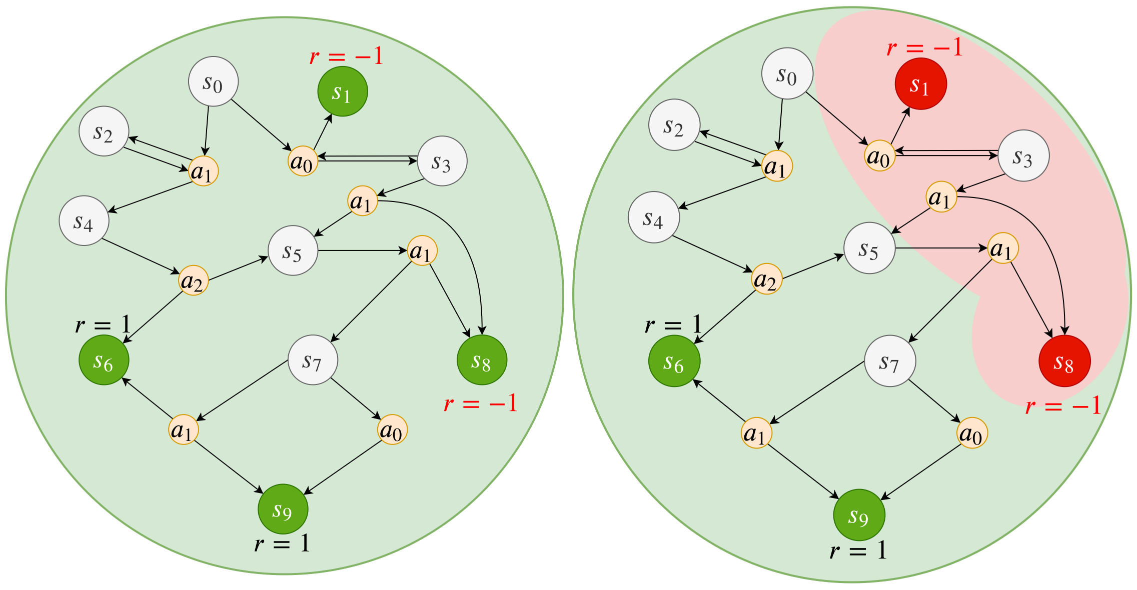


Figure 1: Illustration of an MDP where actions are made according to a policy π. The colors are in the view of the policy where green is *safety*, red is danger, gray is non-terminal states, and orange is actions in a stochastic environment. On the left, the policy follows traditional RL optimization where trial and error occurs in order to map recognize bad states. On the right, the policy has some notion of danger (red area) for actions leading to states with negative feedback.

Figure 1 illustrates a stochastic MDP in the view of a traditional RL agent (left) and an agent that is safety-aware (right). The MDP considers state-space $S = \{s_0 \dots s_9\}$ and an action-space $A = \{a_0 \dots a_2\}$ controlled with the probabilistic policy $\pi(a|s)$, with the probability of transitioning to the next state $P(s'|s, a)$ (stochastic transition). The traditional model-free RL agent must explore to learn a policy that would keep a distance from catastrophic states. This means that the agent would eventually take action a_0 in state s_0 and enter state s_1, which leads to a catastrophic outcome. The motivation for a safer learning system becomes evident, and the idea is to find a method to define good (green) and bad (red) state-space regions before the agent starts exploration.

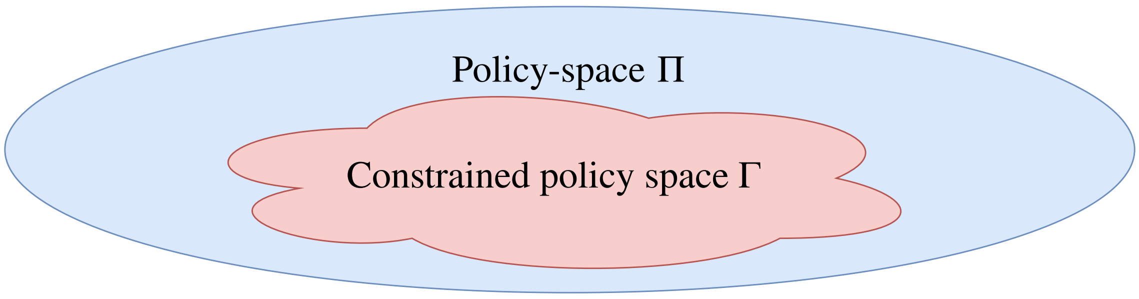


Figure 2: The policy-space (blue) Π and the subset of policies (red) $\Gamma \subseteq \Pi$, where each policy $\pi \in \Gamma$ must satisfy the constraints $c_i \in C$.

3.1 Safe Policy Selection

Risk is a function that indicates the danger of making an action under the policy $\pi(a|s)$ [33]. It is founded on the uncertainty associated with future events and is inevitable since the consequences of actions are unknown at the time when an action is made[34]. There are numerous definitions of the term risk, namely *Risk-Sensitive Criterion* [35], *Worst Case Criterion* [36], and *Constrained Criterion* [37]. A policy that disregards risk evaluation is *risk-neutral*, and the learning objective is to maximize the expectation of the return,

$$\max_{\pi \in \Pi} \mathbb{E}_\pi(G) = \max_{\pi \in \Pi} \mathbb{E}_\pi(\sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}) \tag{7}$$

where it becomes apparent that Equation 7 is the same objective as Equation 1. This gives motivation for modification of the objective function so that the policy is *risk-aware* when maximizing the return.

The **Constrained Criterion** is an appealing approach as it extends the standard MDP framework described as the tuple (S, A, R, P, γ, C), where C is a set of constraints applied to the policy. The goal of the constraint set is to, with high probability, eliminate policies such as the *unsafe* example in Figure 1, similar to the work in [32][3]. The general form of the constrained criterion is defined,

$$\max_{\pi \in \Pi} \mathbb{E}_\pi(G) \;\; \textit{subject to} \;\; c_i \in C, c_i = \{h_i \gtreqless \alpha_i\} \tag{8}$$

where c_i is the i_{th} constraint in the set C that must be satisfied by the policy π. Additionally, h_i is a function related to the return G that is an upper or lower bound to the threshold value α_i. Consider all constraints satisfied, then the policy-space is reduced to a subset $\Gamma \subseteq \Pi$, and the policy exists only within this subset $\pi \in \Gamma$. The idea is that the constraints lead to a significantly smaller policy-space, where it is more likely that a safe solution is found, seen in Figure 2. Given that the algorithm selects policies only from the

[3]Equation 3 leading up to Algorithm 1 use constraints similar to the proposed approach here.

safe subset Γ, the objective function can be written as

$$\max_{\pi \in \Gamma} E_{\pi}(G) \tag{9}$$

which is the standard notion of expected return from Equation 1, but with respect to the subset of safe policies Γ.

Constraint Selection is a delicate user-defined process, which largely depends on a specific problem [35]. It is possible to form constraints from any metric originating from the MDP. Our approach attempts to use a general approach for safe policy updates across various domains. In the proposed algorithm, only a single constraint is formed using the error (uncertainty) of a predictive model [38]. The α parameter acts as a threshold for how much risk we allow when evaluating a policy. Higher the value, the constraint is more restrictive, and for lower values, more permissive [39].

3.2 Safe Exploration

Policies with a constrained criterion do not guarantee safety in the short term because it is challenging to choose parameters within the subset of safe policies Γ initially. Therefore, we also consider **safer exploration** as a means to guide the agent towards making safe actions in the short term.

Risk-directed Exploration uses a notion risk to determine in which direction the agent should explore. We refer the reader to [40] for an in-depth definition. There are several ways to define risk, such as keeping below a variance threshold [41], but our approach uses *normalized expected return with weighted sum entropy* [40, 42].

The risk of taking action in a particular state is given by

$$Risk(s,a) = \mathcal{R}(s,a) = wH(s,a) - (1-w)\frac{\mathbb{E}[G]}{\max\limits_{a \in A} |\mathbb{E}[G]|}, \tag{10}$$

where H is the policy entropy, and the second term describes how good the action is for the particular state. The weight $w \in [0 \ldots 1]$ determines the balance between entropy and return, where higher values of w indicate *more risk* due to less deterministic behavior. The utility function is then updated as follows

$$Utility_{risk}(s,a) = U_{ri}(s,a) = \alpha \mathcal{R}(s,a) + (1-\alpha)\pi(a|s). \tag{11}$$

where $\alpha \in [0 \ldots 1]$ controls the risk-awareness of the agent. At $\alpha = 0$, the agent does not perform risk-directed exploration but considers safety more as $\alpha \to 1$. The risk function $\mathcal{R}(s,a)$ outputs a vector describing the risk of each action in the action space. The risk vector adds to the probabilities and state-action values for the current state ($\pi(a|s)$). The updated utility function $U_{ri}(s,a)$ ensures that sampling is performed in favor of safe and conservative (less exploratory) actions, depending on the weight parameter w, and risk aversion parameter α [40].

3.3 Safe predictive model

In model-based reinforcement learning, the goal is to efficiently learn a predictive model that accurately learns the environment dynamics to predict future states given the current state and action[27]. During the learning of a predictive model, explorative agents are frequently used. However, in a real-world environment where catastrophic states exist, it is little room for errors. However, these environments are often eligible for the deployment of expert systems. Therefore, it is possible to collect observations, with a sub-optimal agent for a user-specified amount of time. The collected observations significantly increase the accuracy of the predictive model, which enables the use of the concept of *curiosity* [43] to create constraints to increase safety. Curiosity-driven exploration is composed of two rewards, extrinsic (the environment) and intrinsic (curiosity) reward, where the agent is encouraged to enter unexplored states. For this work, we **negate** this effect and encourage the agent to stay in states where the predictive model has low uncertainty. For each evaluation using the predictive model, we can calculate the error, which is the difference between the predicted state and the actual state (the state observed by the agent). For predicted states with high error, the model knows little about the consequences of doing the action, indicating that the action will lead to a catastrophic state. Curiosity is the mean squared error of the predicted future state features $\hat{\mathcal{M}}(\hat{s_{t+1}}|s_t, a_t)$ and the ground truth future state $\mathcal{M}(s_{t+1}|s_t, a_t; P)$ where

$$C_u(\mathcal{M}, \hat{\mathcal{M}}) = \frac{1}{2}||\hat{\mathcal{M}}(\hat{s_{t+1}}|s_t, a_t) - \mathcal{M}(s_{t+1}|s_t, a_t; P)||_2^2 \tag{12}$$

defines the curiosity vector. In curiosity-driven exploration, the goal is to pursue states that maximize curiosity, but for safe exploration, we aim to minimize C_u for actions with high uncertainty. In our approach, the weighted curiosity vector adds to the action probability distribution such that

$$U(s, a) = U_{ri+Cu}(s, a) = U_{ri} + \alpha C_u \tag{13}$$

where α is the risk-aversion parameter previously defined in Equation 11.

The updated utility is then compatible with Q-Learning updates using neural network function approximator with weight θ ad the Q-Network. The network is trained by sequentially minimizing the loss function $L_i(\theta_i)$ where i denotes the iteration, such that,

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim p(\cdot)}\bigg[(y_i - Q(s, a; \theta_i))^2\bigg], \tag{14}$$

where $y_i = \mathbb{E}_{s'\sim\mathcal{E}}[U + \gamma \max_{a'} Q(s', a'; \theta_{i-1})|s, a]$ is the target for iteration i and $p(s, a)$ is the behavior distribution[44]. Finally, the standard differentiated loss, w.r.t to the weights

θ denoted,

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim p(\cdot);s'\sim\mathcal{E}}\left[\left(U + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i)\right)\nabla_{\theta_i} Q(s,a;\theta_i)\right], \quad (15)$$

where U is the modified reward from Equation 13.

4 Safe Dreaming

This paper aims to increase the safety of agents that act in environments with catastrophic state outcomes. The *Dreaming Variational Autoencoder* (DVAE) is a model-based reinforcement learning approach for safe and efficient learning. A predictive model learns the dynamics of the environment and acts as a safety precaution as the agent learn fully offline from the real environment. Additionally, the algorithm models the problem as a constrained MDP with a combination of risk-directed exploration and negated curiosity. The algorithm performs learning three steps, *predictive model learning*, *RL training*, and *transition to the real environment*, and describes as follows.

A predictive model learns the transition dynamics of the real environment. To learn these transitions, the model gathers experience through observation of an expert system. An expert system quite regularly in use already; hence, minimal effort is required to train the predictive model. Also, the expert system already makes safe decisions because of hand-crafted features but often operate at a sub-optimal performance. Therefore, a reinforcement learning algorithm is well suited for decision making in industry-near environments, as it can improve performance and safety with the learning guidance of a predictive model.

Training model-free RL algorithms using the learned predictive model is safe and efficient in terms of sampling efficiency. Deep Q-Learning from [44] is well suited because it converges with off-policy data. A combination of the DVAE algorithm and model-free reinforcement learning ensures that learning is performed safely without the risk of entering catastrophic states or cause damage to the real-world environment.

Training duration depends on the problem, and should, therefore, rely on some mechanic to determine the learning stopping criteria. As the algorithm learns an optimal policy for the predictive model, it gradually transitions to make actions in the real environment based on the rate it enters catastrophic states. At such a time, when the agent interacts directly with the real environment, it is possible to enter catastrophic states. The algorithm adds a negated curiosity bonus to reduce the exploration of state-space regions with high uncertainty. This way, the fully deployed algorithm will behave cautiously when the movement towards novel states appears, or if the environment is changed dynamically.

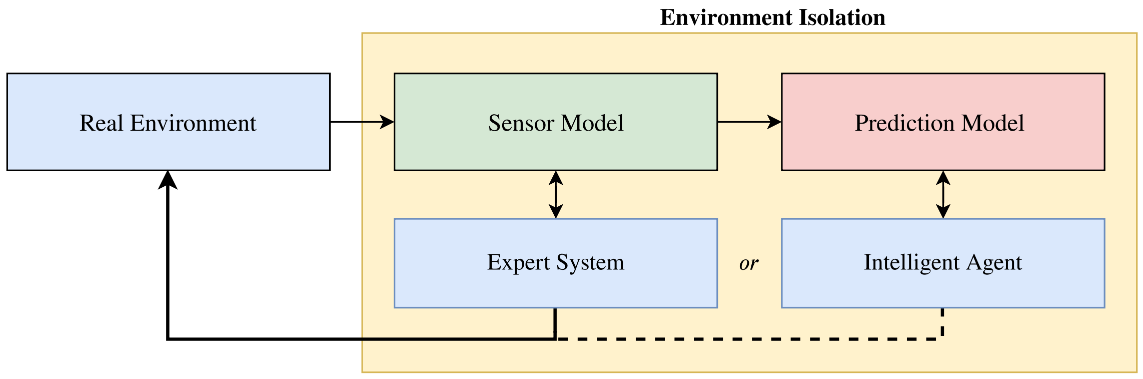


Figure 3: **Isolation of the real environment.** The general idea of DVAE is to isolate the agent training to reduce the risk of catastrophic behavior in the real environment. The predictive model observes the sensors of the real environment and estimates its transition function. The intelligent agent uses the predictive model to train in an offline setting, without the risk of making mistakes in the real world. After training, the algorithm is deployed to the real environment, with significantly less chance of entering catastrophic states.

The training procedure illustrated in Figure 3 works as follows. (1) The predictive model observes and learns the real environment using a sensor model. The same sensor model is the interface that the expert system uses for making actions. (2) The intelligent agent (i.e., a reinforcement learning agent) interact with the predictive model and improve its policy. (3) At the point where the intelligent agent is sufficiently trained, it can replace existing expert systems with comparable performance. (4) If desirable, the intelligent agent can train further in the real-world environment.

The execution graph of DVAE is shown in Figure 4 and works as follows. The policy $\pi(a|s)$ predicts the best action a for the observed state s. The first action is sent to the real environment to produce initial state $s_t = s_0$. The initial state s_t and initial action a_t is processed by the predictive model M and outputs predicted future state $\hat{s}_{t+1}$ and reward $\hat{r}_{t+1}$. The reward is used for policy updates during training and the state for further action prediction. The policy predicts a $\hat{s}_{t+1}$ and is sent to the predictive model, now to predict the two-step predicted state $\hat{s}_{t+2}$. The procedure continues until the algorithm meets the stopping criteria.

A detailed illustration of the DVAE-architecture is shown in Figure 5, including the encoder, decoder, policy, and environment. Initially, the interaction is between the predictive model M and the policy $\pi(s|a)$. The encoder takes a state s_t and predicted action a_t as input and outputs the embedding z_t. An embedding is a compression of the input and leads to faster training and better performance. Considering that the replay-buffer RB holds millions of embedding, the memory footprint is significantly reduced. The

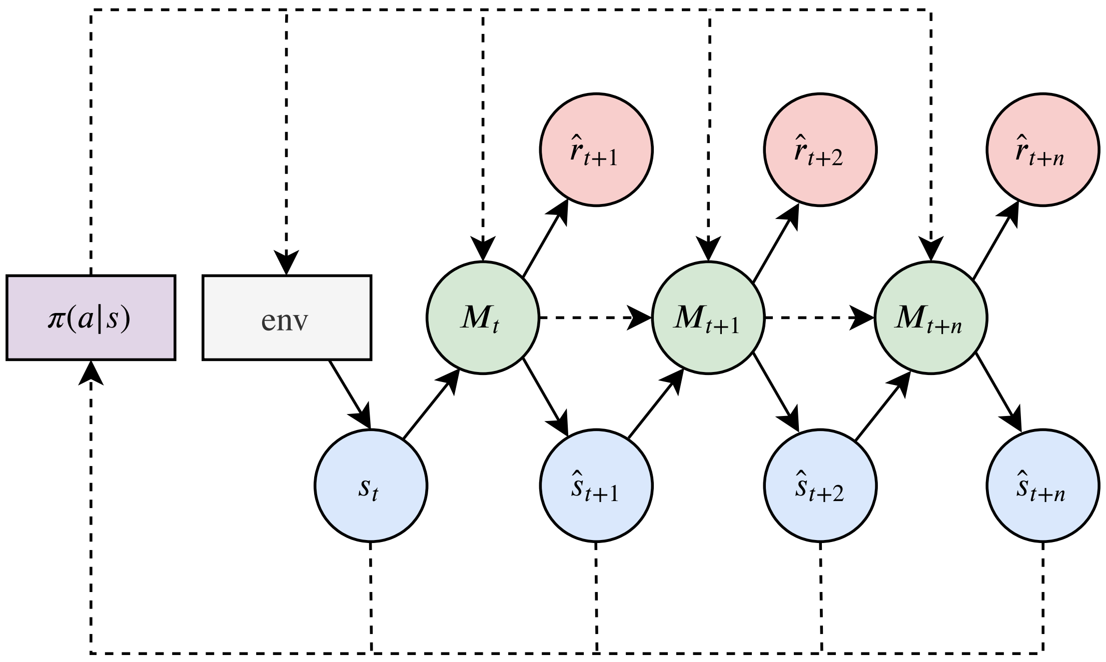


Figure 4: **Prediction of future states.** For each time-step, the agent observes a state from the environment or the predictive model. The agent makes an action that results in a transition to the next state with the corresponding reward. M denotes the predictive model where $\hat{s}$ and $\hat{r}$ is the predicted state and reward.

replay-buffer generates sequential batches of embeddings that are input to the t-encoder enc_t.

The s-encoder is responsible for transforming raw input data into a meaningful and compact feature embedding. DVAE uses a variational autoencoder primarily for this task, but other methods are also applicable, such as generative adversarial networks (GAN). Depending on the environment and the input data, it is possible to visualize the embedding $z_x \in \mathbb{Z}$ by manually altering its values. In Figure 7, we illustrate this with a (green) agent in an empty grid-world. The embedding layer consists of two neurons where the first and second neuron learns the vertical and horizontal location, respectively.

The t-encoder learns the time dependency between states, or in MDP terms, the transition function $T\colon S \times A \rightarrow S$. The t-encoder model computes the future state embedding z^{π}_{t+1} based on a batch of previous embeddings from the view $Z^{\pi}_t = \{z_{t-n} \ldots z_t\}^{\pi}$. The π denotes the policy which DVAE operates under. In DVAE, *long short-term memory* (LSTM) performed best when learning the future state embedding.

The control policy $\pi(s|a)$ is responsible for interaction with the environment and the predictive model (s-encoder and t-encoder). The control is the primary model for making actions that are safe and progress the learning in the right direction. In DVAE, we consider Deep Q-Networks (DQN) using a **constrained optimization criterion** and for exploration, **risk-directed exploration**, and **negated curiosity**. The negated curiosity act as the **constrained criterion** for the MDP. The input to the algorithm is a raw-state, commonly a high-dimensional data structure that is difficult to interpret spatial information. The benefit of the DVAE architecture is that the t-model finds an embedding that can represent the state with the order of magnitudes less complexity. The DVAE algorithm also enables initial training fully offline in a *dream* version of the real environment.

Algorithm 1 DVAE with Deep Q-Learning

```
1: Initialize policy $\pi_\theta(s_t|a_t)$
2: Initialize predictive model $M_\psi(\hat{s}', \hat{r}'|s, a_\pi)$
3:     Initialize encoder $enc(z_t|s_t, a_t)$
4:     Initialize replay-buffer $RB(Z_t|\{z_t \dots z_{t+n}\})$
5:     Initialize t-encoder $enc_t(h_t|Z_t)$,
                 $\hat{s}$-decoder $dec_s(\hat{s}_t|h_t)$,
                 $\hat{r}$-decoder $dec_r(\hat{r}_t|h_t)$
                                              ▷ Training of the predictive model
6: while predictive model needs training; episode = 1, E do
7:     Make decisions using predefined expert system policy
8:     Store transition $(s_t, a_t, r_t, s_{t+1})$ in buffer $D$
9:     Train predictive model $M_\psi$ on data batch $d \subseteq D$ using $MLE$ loss
                     ▷ Training of the Deep Q-Network (or similar RL algorithm)
10: for episode = 1, E do
11:     Sample initial state $s_0$ from $D$
12:     Predict action using policy $\pi(a|s;\theta)$
13:     Predict future state using the predictive model $M_\psi$ where,
14:         Encode input state and action to embedding $enc(z_t|s_t, a_t)$
15:         Store $z_t$ in $RB$ and form sequential subset of n-elements $Z_t \subseteq RB$
16:         Encode sequence of embeddings w.r.t time $enc_t(h_t|Z_t)$
17:         Decode future state and reward $enc_{s+r}(\hat{s}', \hat{r}'|h_t)$
18:     Update policy $\pi_\theta$ with pairs of $(\hat{s}_t, a_t, \hat{r}_t, \hat{s}_{t+1})^{\pi_\theta}$ according to Equation 15
```

The definition of the DVAE algorithm, seen in Algorithm 1, has the following procedure. **(Line 1-5)** The π_θ, predictive model M_ψ with corresponding encoder $enc(z_t|s_t, a_t)$, replay-buffer $RB(Z_t|\{z_t \dots z_{t+n}\})$ t-encoder $enc_t(h_t|Z_t)$, $\hat{s}$-decoder $dec_s(\hat{s}_t|h_t)$ where each component is a function approximator, is initialized with random weights.

(Line 6) starts the training procedure of the predictive model for E episodes. **(Line 7-8)** A expert system algorithm makes decisions and is recorded into buffer D. This step is primarily for observation of the environment to learn the sensor model transition dynamics.**(Line 9)** initiates training of the predictive model using mini-batch stochastic gradient descent with MLE-loss.

(Line 10) initiates a for loop of E episodes to train the reinforcement learning algorithm using the learned predictive model from the procedure at line numbers 6-9. **Line 11-12** is similar to the standard reinforcement learning loop, but instead of taking actions in the real environment, the decision is sent to the predictive model M_ψ. **(Line 13)** the predictive model outputs the estimated future state from the agent decision with the following steps. **(Line 14)** encode current state s_t and action a_t into the embedding z_t. **(Line 15)** The embedding is stored in the replay buffer and is retrieved in batches of n-elements to form Z_t. **(Line 16)** encode the sequential batch of embeddings to capture transition dynamics between states, yielding h_t. **(Line 17)** The $dec_{\hat{s}+\hat{r}}$ decoder outputs the predicted future state and the corresponding reward. Finally, using the predicted values, the reinforcement learning policy is updated using Equation 15 **(Line 18)**, similar to [44]. The process is repeated separately for the predictive model and the reinforcement learning algorithm until reaching an acceptable convergence threshold.

4.1 Exploration and policy update constraints

There are significant improvements to the exploration and policy update for finding safe policies in DVAE. During sampling, the policy the uses *risk-directed exploration* bonus [40]. This is added to the probability distribution over actions before sampling is performed, as described in Section 3. The **policy updates** are constrained to a set of criteria defined as follows. During the learning of the predictive model, feedback is received from the real-world environment. All actions are bound to some feedback even though only 1 of these are received depending on which action the agent performed. In our model, we assume that all actions that were not chosen by the agent are considered unsafe. This way, the algorithm gradually maps the unsafe policy space, as illustrated in Figure 1. It is important to note that this mapping does not influence the choices of the agent when learning the predictive model. When the agent revisits a state, the agent may select another action, and this will label the state safe. Depending on how much the expert system behavior is observered, the better understanding the predictive model gets, as well as the state-risk mapping of the state-space.

4.2 Analysis of convergence guarantees

The Dreaming Variational Autoencoder combines several approaches that previous work has shown to have convergence properties. The algorithm model the problem as an MDP, which is proven to have convergence properties in several works [46, 47, 48, 49]. The markov property is especially interesting, and the proof is detailed well in [50]. The

DVAE algorithm use *constrained MDP* and is proved to have convergence properties for the discounted case used in this work [51].

Tabular Q-Learning is known to converge as time goes towards T, but deep learning variants, specifically neural network estimated Deep Q-Networks has primarily empirical success. There are efforts such as [52] that prove theoretical convergence for simplified DQN, but, no proof for the general case. In regards to using neural network estimators for the predictive model, the proposed approach is based primarily on empirical observations. DVAE uses a similar approach to [38, 25, 27] where the predictive model encoder constructs a variational bound on the data log-likelihood:

$$\begin{aligned} lnM_d(s_1 : T) &\triangleq ln \int \prod_{t=1}^{T} M(s_t|s_{t-d}) M(s_t|s_t^o) ds_{1:T} \\ &\geq \sum_{t=1}^{T} \Bigg(\underbrace{\mathbb{E}_{q(s_t|s_t^o))}[lnM(s_t^o|s_t)]}_{\text{reconstruction}} \\ &\underbrace{- \mathop{\mathbb{E}}_{M(s_{t-1}|s_{t-d})q(s_{t-d}||s^o \leq t-d)} [KL[q(s_t|s_t^o \leq t) \| M(s_t|s_{t-1})]] \Bigg)}_{\text{multi-step prediction}} \end{aligned} \tag{16}$$

where s^o denote unprocessed states. We refer the reader to [27] for the derivation. The curiosity bonus used in the proposed algorithm is shown to work well empirically, but there is, to the best of our knowledge, no proof of convergence. Through trial and error, the proposed algorithm converges empirically, but theoretical convergence remains future work.

5 The ASRS Lab

Safety during learning in RL has been a less prevalent priority in recent years compared to improve the performance of existing non-safe algorithms. We argue that this may be due to the high cost of physical systems to experiment on and that the RL research community primarily uses games as a benchmark tool, which naturally encourages to maximize the agent performance by trial and error. In this section, we propose the **A**utomated **S**torage and **R**etrieval **S**ystem **L**ab (ASRS-Lab), a flexible and industry-near environment for reinforcement learning research. The ASRS-Lab environment focuses on how RL algorithms can learn good policies with minimal negative feedback and enable auxiliary policies to create a predictive model that can be used for safe offline training of reinforcement learning algorithms.

5.1 Motivation

It is well known that the training of algorithms in real-world environments is complicated for several quintessential reasons, which causes non-deterministic side-effects. **First**, in real-world environments, there is no option to accelerate the sampling speed to increase training speed since the training speed depends on real-world time. **Second**, reinforcement learning builds on trial and error, which is not applicable for many mission-critical systems as an error can have catastrophic consequences. **Third**, in real-world environments, there are additional uncertainty factors that can alter the state-space. Most RL algorithms can adapt sufficiently to slight changes, but with the risk of policy collapse for drastic changes. **Fourth**, a system which in simulation is deterministic will because of the side-effects mentioned above, in a real-world environment becomes stochastic. All of these factors cause challenges to guarantee **safety** during training in real-world environments.

5.2 Implementation

With safe reinforcement learning in mind, the ASRS-Lab is implemented with flexible options for state, action, and reward-representations. There are many categories of ASRS systems in the real-world, and to build an environment flexible enough to accommodate all requirements for all systems was unfeasible. However, the ASRS-Lab could successfully reconstruct shuttle-based, aisle-based, and grid-based warehouses. For this paper, we consider the grid-based architecture.

Figure 8 illustrates the observable state-space from a two-dimensional point of view. In a simple cube-based ASRS system, the environment consists of (B) passive and (C) active delivery-points, (D) pickup-points, and (F) taxis. **The goal** of the environment is to find a positive terminal state using minimal time with a limited set of actions. One episode of the environment is defined as follows. The (taxi) agent starts at an arbitrary position on the plane. At the same time, the agent receives a *retrieve order* from the ASRS *scheduling system*. This order describes a target location for goods to be retrieved. The agent must now reach the target location in minimal time using its controls. Considering that there are many other agents on the plane, the control task is challenging to learn because each action has a significant risk of collision with other agents, as well as the outer bounds of the grid system. When an agent enters a target position, it is rewarded and is assigned a *delivery task* from the scheduling system. The agent must now move to the designated location described by the delivery task. When the agent reaches its destination, a large reward is given.

A taxi can move using a **discrete** or a **continuous controller**. In the discrete mode, the agent can increase and decrease thrust and move in either direction, including the diagonals. For the continuous mode, all of these actions are floating-point numbers between (off) 0 and (on) 1, giving a significantly harder action-space to learn. The simulator also features a continuous mode for the state-space, where actions are performed asynchronously to the game loop. The environment supports custom modules for mechanisms such as the scheduling system, agent controllers, and fitness scoring.

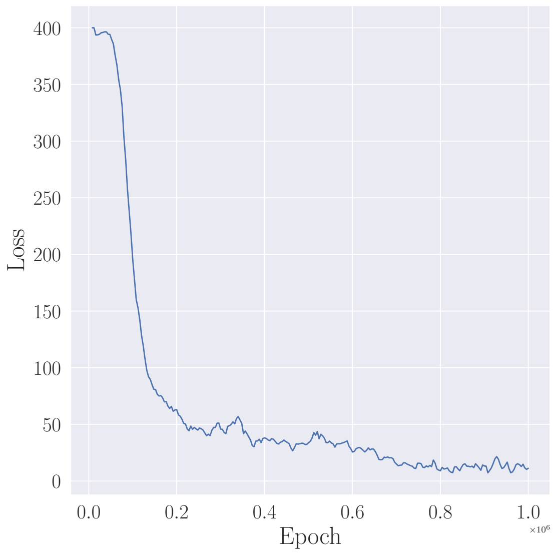
5.3 Benefits

A notable benefit of the ASRS-Lab is that it can accurately model real-world warehouse environments at high speed. The ASRS-Lab environment runs an order of magnitudes faster on a single high-end processing unit compared to real-world systems. The performance is measured by comparing the number of actions a taxi performs in the real environment versus the virtual environment. The environment can be distributed on many processing units to increase the performance further. In our benchmarks, the simulator was able to collect 1 million samples per second during the training of deep learning models using high-performance computing (HPC).

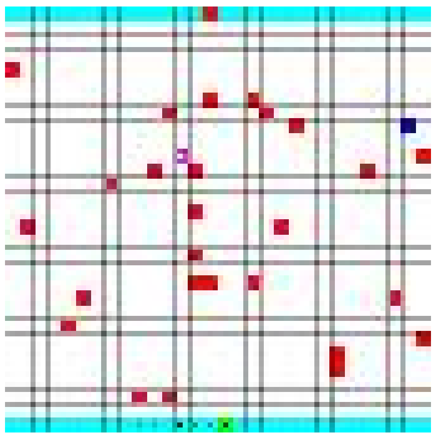
6 Results

In state-of-the-art model-free reinforcement learning algorithms, it is common to perform a (random) gaussian-based exploration method to map the return to states. These algorithms are excellent at finding an average point on the optimization plane that generalizes well across multiple domains. The issue, however, is that there are no guarantees that the learned policy avoids catastrophic states. In this section, we show that DVAE is capable of learning an accurate predictive model for model-free algorithms and learn good policies while behaving safely during exploration. We apply the proposed constrained criterion to the policy updates and use risk-directed exploration to enforce safer actions as described in Section 3. DVAE is integrated with **Deep Q-Network** (DQN) [53] and compared against DQN (Rainbow) and **Proximal Policy Optimization** (PPO). The algorithm tests across various environments, including popular Atari 2600 games [54], Deep RTS [55], Deep Line Wars [56], and the industry-near environment, ASRS-Lab [31].

6.1 Predictive Model



(a) **Average predictive model M loss.**



(b) **Decoder output after training.** The decoder output of the predictive model after 5 hours of training without speed acceleration.

Figure 9

The prediction model's objective is to learn environment dynamics and features so that it can accurately mimic the environment behavior. Figure 9a illustrates the average loss for all tested environments. The predictive trains using specially crafted **expert systems** that perform well in each of the tested environments. The trend is for the loss to start high, and quickly reduce to only minor weight adjustments during training. These minor weight adjustments play a significant role in learning accurate embeddings, as illustrated by Figure 9b

A way to measure the accuracy of the predictive model is to investigate the cumulative prediction error. Figure 10 illustrates this cumulative prediction error for all tested environments. The experiments show that the prediction error tends towards exponential growth when the predictive model makes predictions for longer time horizons. As seen in Table 1, the predictive model has an error of 284 (the decoded state is 284×284) for the Deep RTS environment at predictions done for 100 timesteps in the future. This means that every pixel in the predicted state is incorrect, and hence, difficult to use for training model-free algorithms.

It is sensible to limit the prediction horizon for environments that are too advanced or difficult to extract the dynamics from . The downside of limiting the prediction horizon is that the algorithm is not able to train fully offline. However, the algorithm reduces the volume of **real** training data needed to converge model-free approaches by magnitudes successfully.

Table 1: **Exponential cumulative prediction error.** Depending on the environment, the cumulative prediction error increase exponentially for all environments. The table shows that the exponential growth are consistently less extreme for simple environments. The numbers in the header presents the state n-th in the future.

Environment	10	25	50	75	100
ASRS-Lab-11x11	0.34	2.08	7.47	14.91	25.10
ASRS-Lab-21x21	0.36	1.91	7.53	16.62	28.75
ASRS-Lab-41x41	0.43	2.98	10.50	25.06	43.02
Acrobot-v1	0.42	2.04	9.15	19.56	34.86
BeamRider-ramNoFrameskip-v4	1.77	9.33	33.99	75.94	135.49
Breakout-ramNoFrameskip-v4	1.52	7.20	28.84	67.37	110.48
CartPole-v0	0.31	1.52	6.13	13.62	22.58
CartPole-v1	0.35	1.50	6.66	14.53	26.01
DeepLineWars	0.54	3.81	13.73	29.41	50.58
DeepRTS-1v1	2.72	16.29	65.98	143.02	255.99
DeepRTS-GoldCollect	0.69	4.55	19.75	46.17	78.00
MountainCar-v0	0.65	3.66	15.31	30.73	49.95
Qbert-ramNoFrameskip-v4	0.77	4.20	18.23	38.69	63.38
SpaceInvaders-ramNoFrameskip-v4	1.04	4.98	21.96	53.11	89.28

The predictive model successfully learns several environments sufficiently, including ASRS-Lab-21x21, CartPole, and Deep Line Wars. It is likely that tuning the α, w, and learning-rate would improve accuracy for other environments, but parameters remain problem-specific and must be **carefully tuned**.

6.2 Agent Failure Rate

The failure rate is measured by counting the number of negative rewards the agent receives during an episode while training. The environment has a negative reward for catastrophic states and positive on the contrary. Recall that the algorithm should interpret the MDP with constraints and label catastrophic states accordingly, see Figure 1.

Figure 11 illustrates the failure-rate for DVAE with three hyperparameter configurations, $\alpha = 0.99, w = 0.01$, $\alpha = 0.7, w = 0.3$, $\alpha = 0.5, w = 0.5$. Recall that higher α and lower w values account for safe-aware behavior. Safer configuration of DVAE clearly impacts the rate by which the algorithm makes mistakes.

The algorithm does not always learn good policies, such as in the DeepRTS environment. The reason is perhaps that the reward function does not represent the goal, and further investigations discovered that this is the case for DeepRTS. For the DeepRTSGold environment DVAE outperformed PPO and DQN significantly

DVAE increases safety significantly for the majority of the environments tested in this paper. The results from Figure 11 shows a consistent decrease in failures when increasing the safety-awareness sensitivity using the α and w hyperparameter. The benefits of having high safety-awareness increase action safety, but at the cost of **slower convergence** or local minima problems.

6.3 Agent performance

DVAE has comparable performance to DQN and PPO in terms of accumulating reward during training. Figure 12 shows the performance **after** the DVAE algorithm is pre-trained on the predictive model. We perform these tests on DeepLineWars, DeepRTSGold, Acrobot, CartPole, and ARS-Lab-41x41. The figure clearly shows that DVAEsuccessfully trains the algorithm to a sufficient behavior level, and can improve further when training on the real environment.

DVAE is not always stable when training in complex environments, as seen in the DeepLineWars plot. Out of 100 trials, the DVAEconfiguration using $\alpha = 0.7$ and $\alpha = 0.5$ diverged, and hence, was stopped before reaching 1 million timesteps.

The pretraining was done using a horizon of 40 frames for 2 million timesteps. In practice, this only results in 50000 timesteps in the real environment, resulting in magnitudes lower risk of failures.

Sensitivity to hyperparameters is, however, a significant issue that limits the algorithm from functioning well throughout all tests, without extensive tuning.

7 Conclusion and Future Work

The Dreaming Variational Autoencoder increases safety during the training of reinforcement learning agents. Section 6 shows that,

1. Agents have significantly lower failure-rate when pretraining using the predictive model,
2. has similar performance, in terms of accumulative reward, to state-of-the-art algorithms, including DQN and PPO, and
3. can predict longer time-horizons with usable training quality, however, the prediction error grows exponentially.

Despite that DVAE is less stable than model-free approaches and is, which is the biggest challenge of using a dream model for safety-critical tasks, the sample efficiency is significantly improved. As is common in many other models, the proposed algorithm requires significant hyperparameter tuning to function well, and it could be difficult to find general parameters that work across many environments. However, we found that $\alpha = 0.99, w = 0.01$, $\alpha = 0.7, w = 0.3$, and $\alpha = 0.5, w = 0.5$ to perform best during the experiments.

The most considerable achievement is that **DVAE improves sample efficiency** significantly when using the predictive model when pretraining the agent. The algorithm can predict future-state sequences of up to 100 frames with an accuracy sufficient for pretraining. This reduces the need for interacting with the real environment and hence defeats the potential risk of entering catastrophic states.

Continued research of this work is dedicated to better combine proximal policy optimization with the presented methods for safe reinforcement learning. In the DVAE t-model, we would like to investigate if temporal convolutional networks [58] could further improve the performance of learning the predictive model. Also, we hope to experiment with the recent *vector quantized variational autoencoder* [59] for more accurate latent space (embedding) encoding. While this paper contributes new findings in safe reinforcement learning, it is still room for improvement, in which we hope to contribute more in the future.

References

[1] R. Fox, A. Pakman, N. Tishby, Taming the Noise in Reinforcement Learning via Soft Updates, 32nd Conference on Uncertainty in Artificial Intelligence (2015). arXiv:1512.08562.

[2] S. Levine, C. Finn, T. Darrell, P. Abbeel, End-to-End Training of Deep Visuomotor Policies, Journal of Machine Learning Research 17 (apr 2015). arXiv:1504.00702.
URL http://arxiv.org/abs/1504.00702

[3] C. Zhang, P. Patras, H. Haddadi, Deep Learning in Mobile and Wireless Networking: A Survey, IEEE Communications Surveys & Tutorials (mar 2018). arXiv:1803.04311.
URL http://arxiv.org/abs/1803.04311

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing Atari with Deep Reinforcement Learning, Neural Information Processing Systems (dec 2013). arXiv:1312.5602.
URL http://arxiv.org/abs/1312.5602

[5] N. Fulton, A. Platzer, Safe reinforcement learning via formal methods: Toward safe control through proof and learning, in: The Thirty-Second AAAI Conference on Artificial Intelligence, 2018, p. 8.

[6] R. S. Sutton, A. G. Barto, Reinforcement learning: An introduction, MIT Press, 2018.

[7] M. Tokic, Adaptive ϵ-greedy exploration in reinforcement learning based on value differences, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (2010). doi:10.1007/978-3-642-16111-7_23.

[8] D. Steckelmacher, H. Plisnier, D. M. Roijers, A. Nowé, Sample-Efficient Model-Free Reinforcement Learning with Off-Policy Critics, CEUR Workshop Proceedings 2491 (mar 2019). arXiv:1903.04193.
URL http://arxiv.org/abs/1903.04193

[9] S. Padakandla, P. K. J, S. Bhatnagar, Reinforcement Learning in Non-Stationary Environments, arxiv preprint arXiv:1905.03970 (2019) 17arXiv:1905.03970.
URL http://arxiv.org/abs/1905.03970

[10] M. Alshiekh, R. Bloem, R. Udiger Ehlers, B. Konighofer, S. Niekum, U. Topcu, Safe Reinforcement Learning via Shielding, Thirty-Second AAAI Conference on Artificial Intelligence (2017) 23.
URL www.aaai.org

[11] J. Garcia, F. Fernandez, Safe Exploration of State and Action Spaces in Reinforcement Learning, Journal of Artificial Intelligence Research 45 (2014) 515–564. arXiv:1402.0560, doi:10.1613/jair.3761.
URL http://arxiv.org/abs/1402.0560http://dx.doi.org/10.1613/jair.3761

[12] S. T. Hackman, M. J. Rosenblatt, J. M. Olin, Allocating items to an automated storage and retrieval system, IIE Transactions (Institute of Industrial Engineers) 22 (1) (1990) 7–14. doi:10.1080/07408179008964152.

[13] S. Wiriyacoonkasem, A. C. Esterline, Adaptive learning expert system, in: Conference Proceedings - IEEE SOUTHEASTCON, IEEE, 2000, pp. 445–448. doi:10.1109/secon.2000.845609.

[14] S. P. Leo Kumar, Knowledge-based expert system in manufacturing planning: state-of-the-art review (2019). doi:10.1080/00207543.2018.1424372.

[15] Y. Wang, H. He, X. Tan, Robust Reinforcement Learning in POMDPs with Incomplete and Noisy Observations, arxiv preprint arXiv:1902.05795 (2019). arXiv:1902.05795.

[16] P.-A. Andersen, M. Goodwin, O.-C. Granmo, The Dreaming Variational Autoencoder for Reinforcement Learning Environments, in: Max Bramer, M. Petridis (Eds.), Artificial Intelligence, xxxv Edition, Vol. 11311, Springer, Cham, 2018, pp. 143–155. doi:10.1007/978-3-030-04191-5_11.
URL http://link.springer.com/10.1007/978-3-030-04191-5{_}11

[17] K. Arulkumaran, M. P. Deisenroth, M. Brundage, A. A. Bharath, Deep reinforcement learning: A brief survey, IEEE Signal Processing Magazine 34 (6) (2017) 26–38. arXiv:1708.05866, doi:10.1109/MSP.2017.2743240.
URL https://ieeexplore.ieee.org/document/8103164/

[18] T. Xiao, G. Kesineni, Generative Adversarial Networks for Model Based Reinforcement Learning with Tree Search, Tech. rep., University of California, Berkeley (2016).
URL http://tedxiao.me/pdf/gans{_}drl.pdf

[19] E. Santana, G. Hotz, Learning a Driving Simulator, arxiv preprint arXiv:1608.01230 (2016). arXiv:1608.01230.

[20] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, K. Zieba, End to End Learning for Self-Driving Cars, arxiv preprint arXiv:1604.07316 (2016). arXiv:1604.07316.

[21] S. Shalev-Shwartz, S. Shammah, A. Shashua, On a Formal Model of Safe and Scalable Self-driving Cars, arxiv preprint arXiv:1708.06374 (2017). arXiv:1708.06374.

[22] M. Janner, J. Fu, M. Zhang, S. Levine, When to Trust Your Model: Model-Based Policy Optimization, arXiv preprint arXiv:1906.08253 (jun 2019). arXiv:1906.08253.
URL http://arxiv.org/abs/1906.08253

[23] K. Gregor, D. J. Rezende, F. Besse, Y. Wu, H. Merzic, A. van den Oord, Shaping Belief States with Generative Environment Models for RL, arxiv preprint arXiv:1906.09237 (jun 2019). arXiv:1906.09237.
URL http://arxiv.org/abs/1906.09237

[24] M. G. Azar, B. Piot, B. A. Pires, J.-B. Grill, F. Altché, R. Munos, World Discovery Models, arxiv preprint arXiv:1902.07685 (feb 2019). arXiv:1902.07685.
URL http://arxiv.org/abs/1902.07685

[25] D. Ha, J. Schmidhuber, Recurrent World Models Facilitate Policy Evolution, Advances in Neural Information Processing Systems 31 (sep 2018). arXiv:1809.01999.
URL http://arxiv.org/abs/1809.01999

[26] X. Liang, Q. Wang, Y. Feng, Z. Liu, J. Huang, VMAV-C: A Deep Attention-based Reinforcement Learning Algorithm for Model-based Control, arxiv preprint arXiv:1812.09968 (dec 2018). arXiv:1812.09968.
URL http://arxiv.org/abs/1812.09968

[27] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, J. Davidson, Learning Latent Dynamics for Planning from Pixels, Proceedings of the 36 th International Conference on Machine Learning (nov 2018). arXiv:1811.04551.
URL http://arxiv.org/abs/1811.04551

[28] K. Chua, R. Calandra, R. McAllister, S. Levine, Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models, Advances in Neural Infor-

mation Processing Systems 31 (may 2018). arXiv:1805.12114.
URL http://arxiv.org/abs/1805.12114

[29] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, A. Lerchner, beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework, International Conference on Learning Representations (nov 2016).
URL https://openreview.net/forum?id=Sy2fzU9gl

[30] C. Blundell, B. Uria, A. Pritzel, Y. Li, A. Ruderman, J. Z. Leibo, J. Rae, D. Wierstra, D. Hassabis, Model-Free Episodic Control, arxiv preprint arXiv:1606.04460 (jun 2016). arXiv:1606.04460.
URL http://arxiv.org/abs/1606.04460

[31] P.-A. Andersen, M. Goodwin, O.-C. Granmo, Towards Model-based Reinforcement Learning for Industry-near Environments, arxiv preprint arXiv:1907.11971 (jul 2019). arXiv:1907.11971.
URL http://arxiv.org/abs/1907.11971

[32] F. Berkenkamp, M. Turchetta, A. P. Schoellig, A. Krause, Safe Model-based Reinforcement Learning with Stability Guarantees, Proceedings of Neural Information Processing Systems (2017). arXiv:1705.08551.

[33] M. Heger, Consideration of Risk in Reinforcement Learning, Proceedings of the Eleventh International Conference (1994).

[34] Y. Shen, M. J. Tobia, T. Sommer, K. Obermayer, Risk-sensitive Reinforcement Learning, Neural Computation 26 (7) (2013) 1298–1328. arXiv:1311.2097, doi:10.1162/NECO_a_00600.
URL http://arxiv.org/abs/1311.2097http://dx.doi.org/10.1162/NECO{_}a{_}00600

[35] P. Geibel, F. Wysotzki, Risk-Sensitive Reinforcement Learning Applied to Control under Constraints, Journal of Artificial Intelligence Research 24 (2005) 81–108.

[36] C. Gaskett, Reinforcement learning under circumstances beyond its control, Proceedings of the international conference on computational intelligence for modelling control and automation (2003).
URL http://www.his.atr.co.jp/cgaskett/

[37] T. M. Moldovan, P. Abbeel, Safe Exploration in Markov Decision Processes, Proceedings of the 29th International Conference on Machine Learning (2012). arXiv:1205.4810.

[38] D. Ha, J. Schmidhuber, World Models, arxiv preprint arXiv:1803.10122 (mar 2018). arXiv:1803.10122, doi:10.5281/zenodo.1207631.
URL https://arxiv.org/abs/1803.10122

[39] Y. Chow, M. Ghavamzadeh, L. Janson, M. Pavone, Risk-constrained reinforcement learning with percentile risk criteria, Journal of Machine Learning Research (2018). arXiv:1512.01629.

[40] L. L. Edith, C. Melanie, P. Doina, R. Bohdana, Risk-directed Exploration in Reinforcement Learning, IJCAI 2005 Workshop on Planning and Learning in A Priori Unknown or Dynamic Domains (2005).

[41] N. Dilokthanakul, M. Shanahan, Deep Reinforcement Learning with Risk-Seeking Exploration, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 10994 LNAI, Springer Verlag, 2018, pp. 201–211. doi:10.1007/978-3-319-97628-0_17.

[42] J. Yang, W. Qiu, A measure of risk and a decision-making model based on expected utility and entropy, in: European Journal of Operational Research, 2005, pp. 792–799. doi:10.1016/j.ejor.2004.01.031.

[43] D. Pathak, P. Agrawal, A. A. Efros, T. Darrell, Curiosity-driven exploration by self-supervised prediction, in: 34th International Conference on Machine Learning, ICML 2017, 2017, p. 12. arXiv:1705.05363.

[44] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529–533.
URL http://dx.doi.org/10.1038/nature14236

[45] R. S. Sutton, The Bitter Lesson (2019).
URL http://www.incompleteideas.net/IncIdeas/BitterLesson.html

[46] E. A. Feinberg, M. E. Lewis, On the Convergence of Optimal Actions for Markov Decision Processes and the Optimality of (s,S) Inventory Policies, Naval Research Logistics 65 (8) (2015) 619–637. arXiv:1507.05125.
URL http://arxiv.org/abs/1507.05125

[47] S. Haddad, B. Monmege, Reachability in MDPs: Refining convergence of value iteration, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 8762 (2014) 125–137. doi:10.1007/978-3-319-11439-2_10.

[48] E. A. Feinberg, P. O. Kasyanov, M. Z. Zgurovsky, Convergence of value iterations for total-cost MDPs and POMDPs with general state and action sets, in: IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL), Institute of Electrical and Electronics Engineers Inc., 2014, p. 8. doi:10.1109/ADPRL.2014.7010613.

[49] M. S. Santos, J. Rust, S. J. Control Optim, CONVERGENCE PROPERTIES OF POLICY ITERATION *, Society for Industrial and Applied Mathematics 42 (6) (2004) 2094–2115. doi:10.1137/S0363012902399824.
URL http://www.siam.org/journals/sicon/42-6/39982.html

[50] M. Hairer, Convergence of Markov Processes, Tech. rep., Mathematics Department, University of Warwick (2016).

[51] E. Altman, Constrained Markov decision processes, 2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT) (1999).

[52] J. Fan, Z. Wang, Y. Xie, Z. Yang, A Theoretical Analysis of Deep Q-Learning, arxiv preprint arXiv:1901.00137 (2019) 65arXiv:1901.00137.
URL http://arxiv.org/abs/1901.00137

[53] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, D. Silver, Rainbow: Combining Improvements in Deep Reinforcement Learning, Proceedings, The Thirty-Second AAAI Conference on Artificial Intelligence (oct 2018). arXiv:1710.02298.
URL http://arxiv.org/abs/1710.02298

[54] M. G. Bellemare, Y. Naddaf, J. Veness, M. Bowling, The arcade learning environment: An evaluation platform for general agents, IJCAI International Joint Conference on Artificial Intelligence 2015-Janua (2015) 4148–4152. arXiv:1207.4708, doi:10.1613/jair.3912.
URL http://arxiv.org/abs/1207.4708

[55] P.-A. Andersen, M. Goodwin, O.-C. Granmo, Deep RTS: A Game Environment for Deep Reinforcement Learning in Real-Time Strategy Games, Proceedings of the IEEE International Conference on Computational Intelligence and Games (aug

2018). arXiv:1808.05032.
URL http://arxiv.org/abs/1808.05032

[56] P.-A. Andersen, M. Goodwin, O.-C. Granmo, Towards a deep reinforcement learning approach for tower line wars, in: M. Bramer, M. Petridis (Eds.), Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 10630 LNAI, 2017, pp. 101–114. arXiv:1712.06180, doi:10.1007/978-3-319-71078-5_8.

[57] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal Policy Optimization Algorithms, arxiv preprint arXiv:1707.06347 (jul 2017). arXiv:1707.06347.
URL http://arxiv.org/abs/1707.06347

[58] S. Bai, J. Z. Kolter, V. Koltun, An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling, arxiv preprint arXiv:1803.01271 (2018). arXiv:1803.01271.

[59] A. Razavi, A. van den Oord, O. Vinyals, Generating Diverse High-Fidelity Images with VQ-VAE-2, arxiv preprint arXiv:1906.00446 (2019). arXiv:1906.00446.

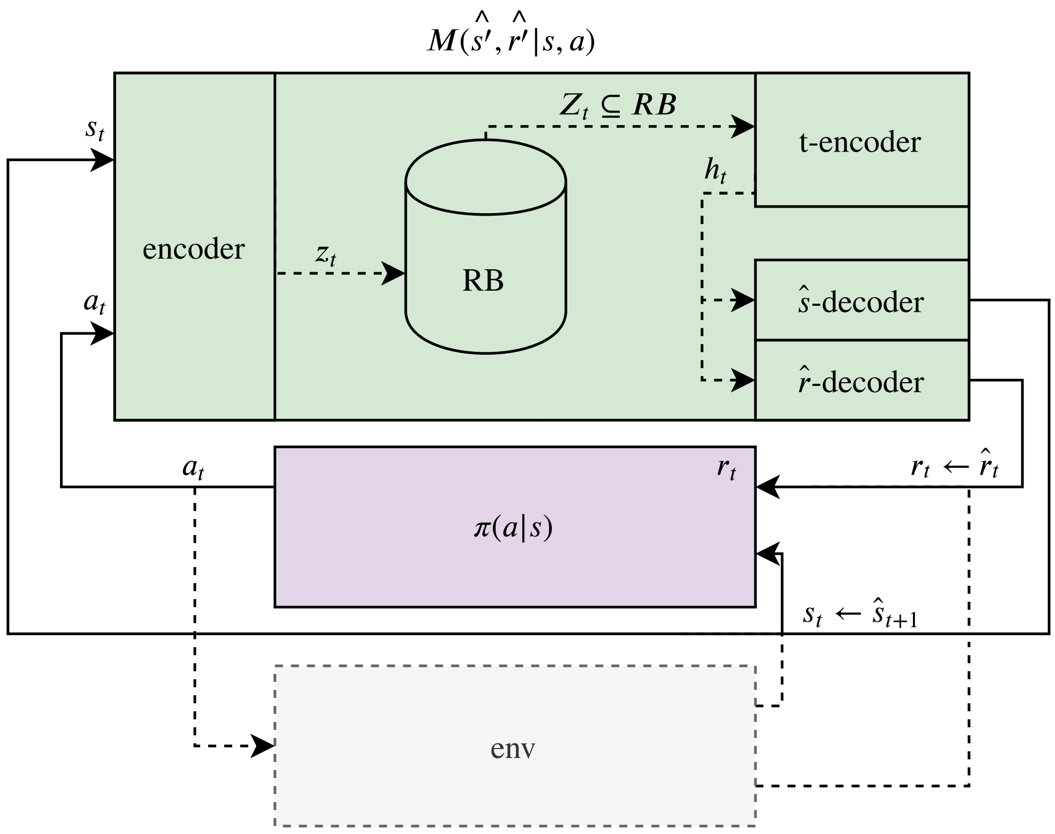


Figure 5: **Architecture overview of DVAE.** The architecture of the proposed predictive model. The predictive model M takes action a and state s as input to the encoder. The input to the encoder is transformed into the embedding z_t and is stored in the replay-buffer RB. The t-encoder (temporal-encoder) retrieve $Z_t \subseteq RB$ (size determined by hyperparameter) to learn the transition dynamics h_t w.r.t time. The $\hat{s} - decoder$ and $\hat{r} - decoder$ decodes h_t into a predicted future state and reward, which feeds into the policy for decision making and training. The dotted lines illustrate the standard reinforcement learning interaction between agent and environment [45], which the algorithm uses after training.

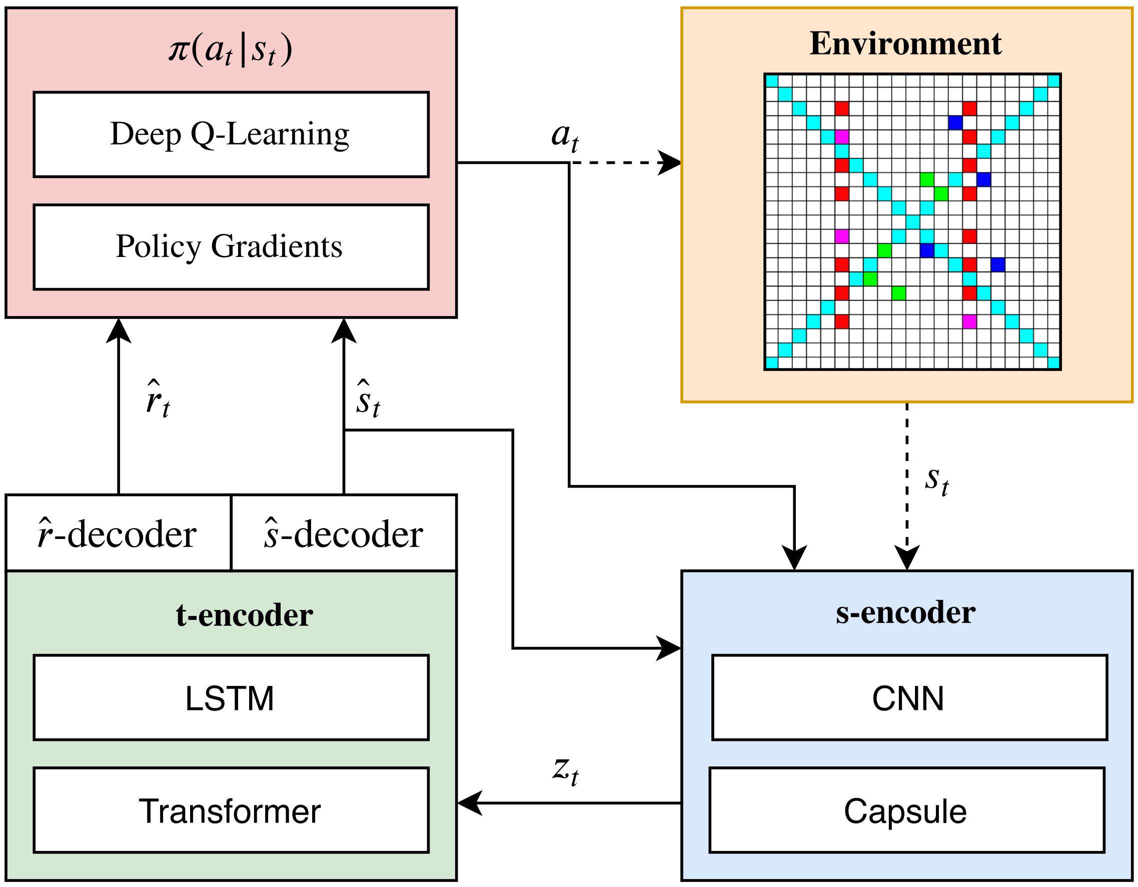
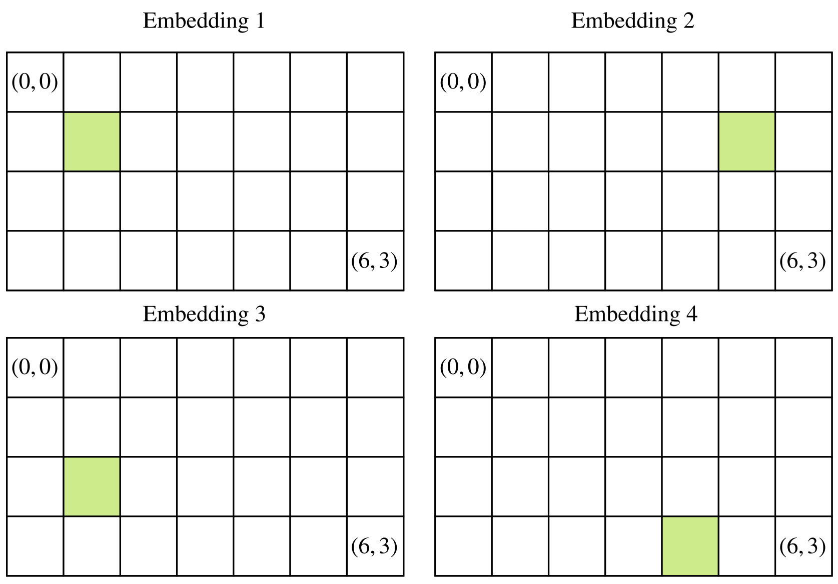


Figure 6: **Detailed architecture for DVAE.** The algorithm is modular so that compatible algorithms and schemes are usable and problem independent.



Embedding	z_0	z_1
1	0.285	0.285
2	0.857	0.285
3	0.285	0.750
4	0.714	1.000

Figure 7: **Learned features of the encoder**. The table illustrates a two-parameter embedding of a grid-world environment where the learned embedding refers to the location of the player. The idea is that the state information, a $n \times m$ grid is compressed significantly and can be retrieved by decoding the embedding.

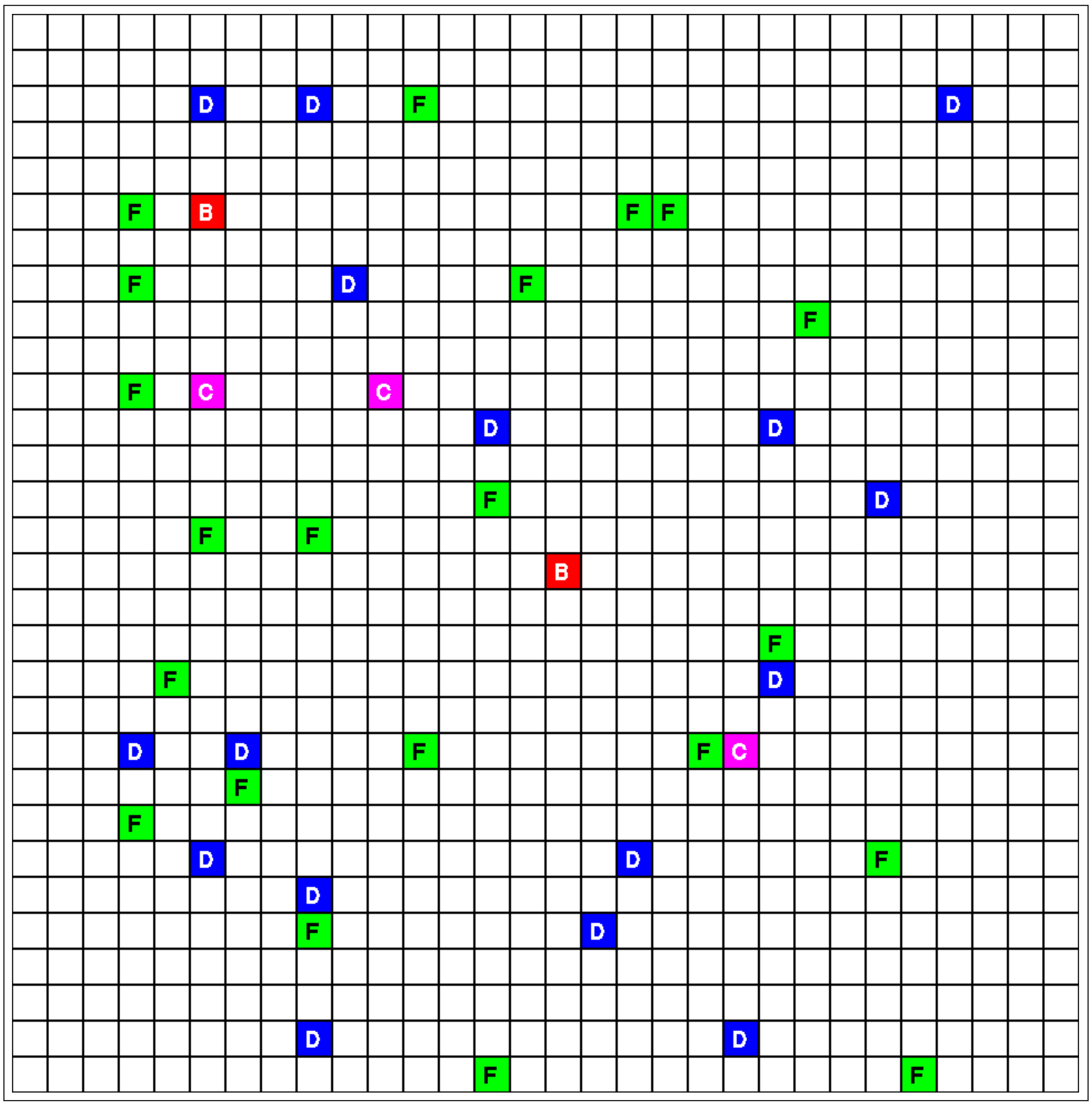


Figure 8: Visual observation of the ASRS-Lab environment using cube-based ASRS configuration.

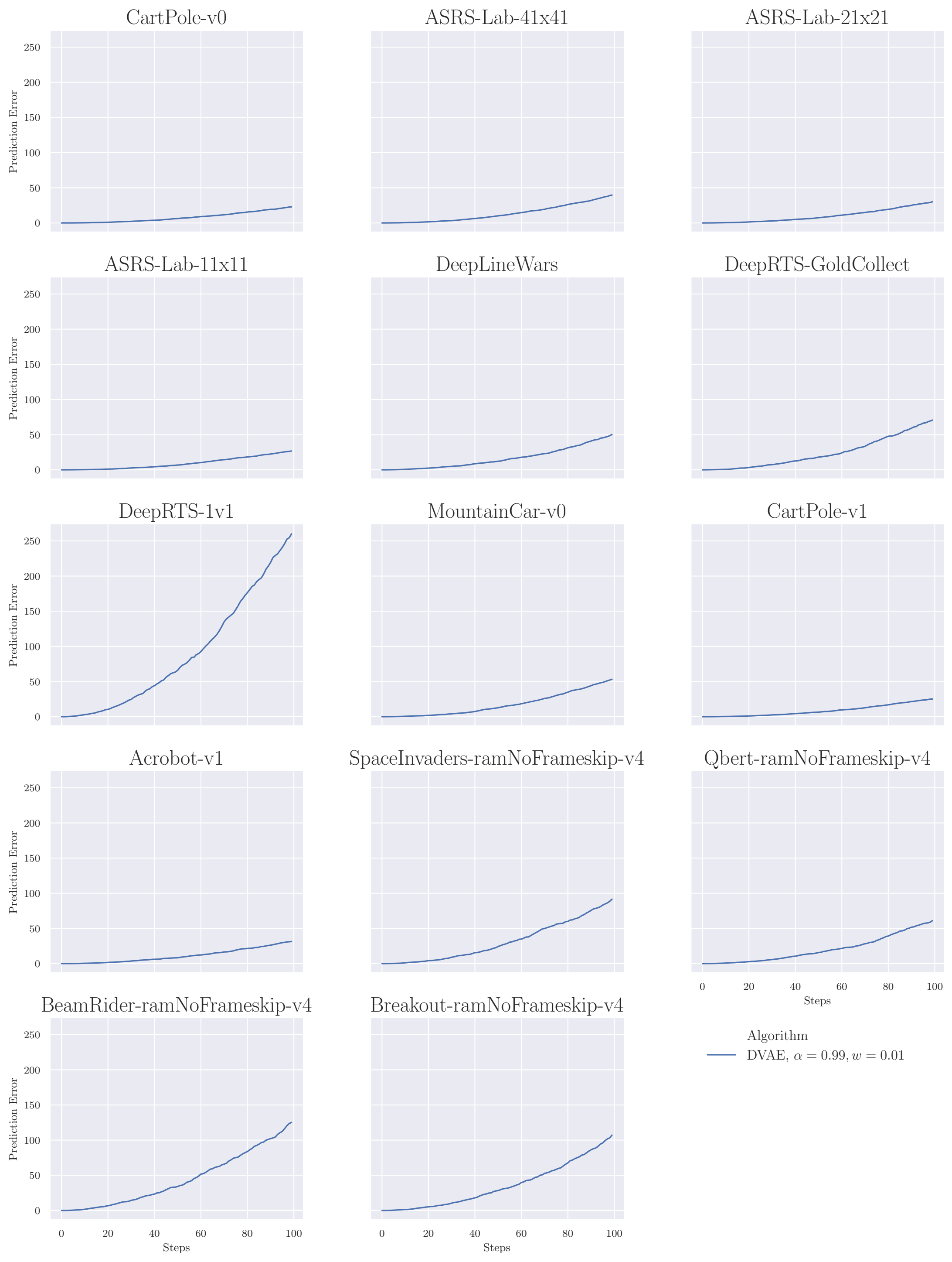


Figure 10: **Cumulative Prediction Error** The y-axis shows the pixel error where each whole number represent a 2-dimensional error. For example, an error of 32 means that 32×32 pixels has incorrect values. The x-axis is how many predictions in the future is made without interaction with the real environment (how many states in the future has the algorithm "dreamed").

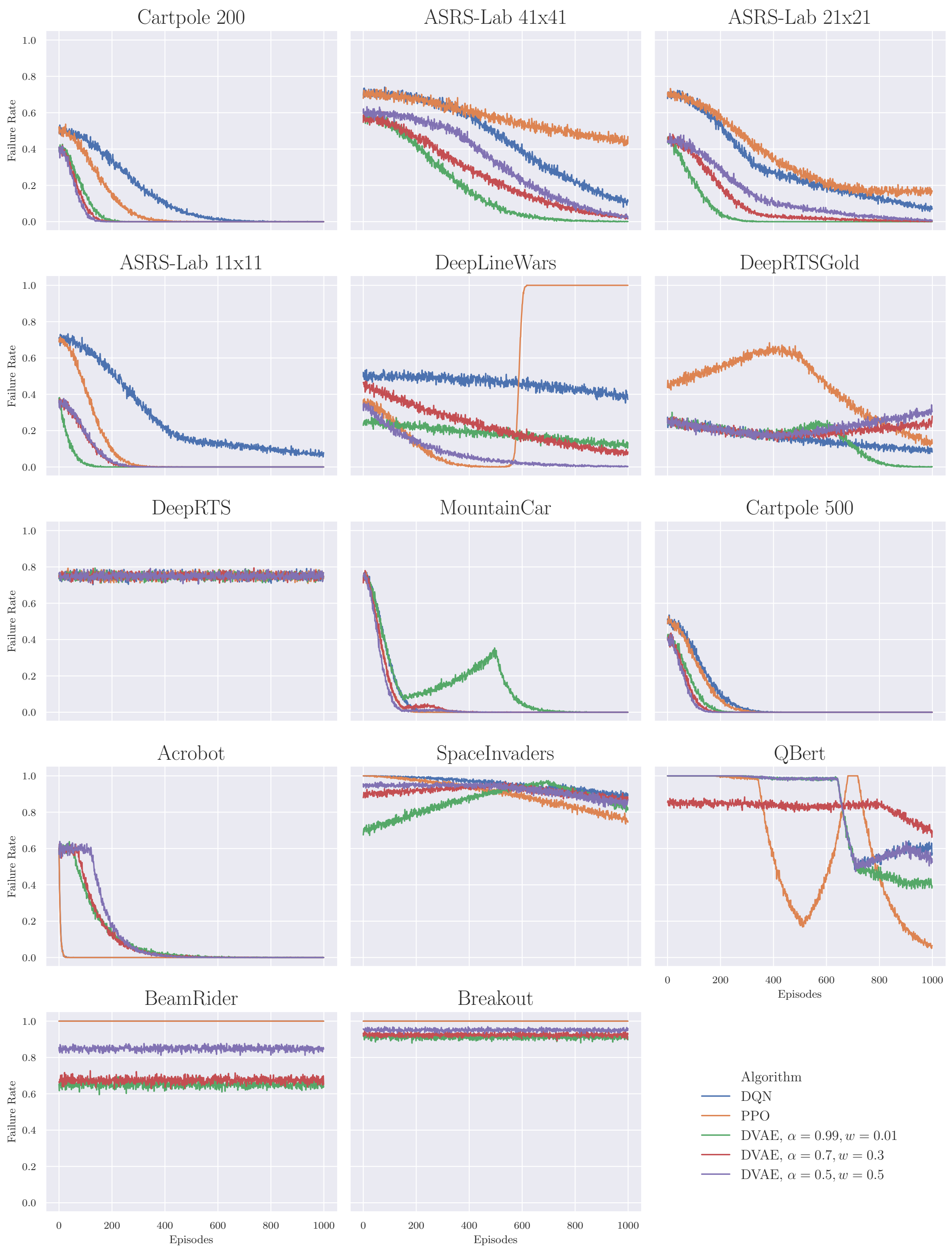


Figure 11: **Agent failure rate.** We evaluate the rate of which an agent fails during trials across various environments where the x-axis illustrates the episode number, and the y-axis the rate in percentage. Each environment is averaged over 100 trials for 1000 episodes. We compare three safety configurations of DVAE against DQN [44] and PPO [57]

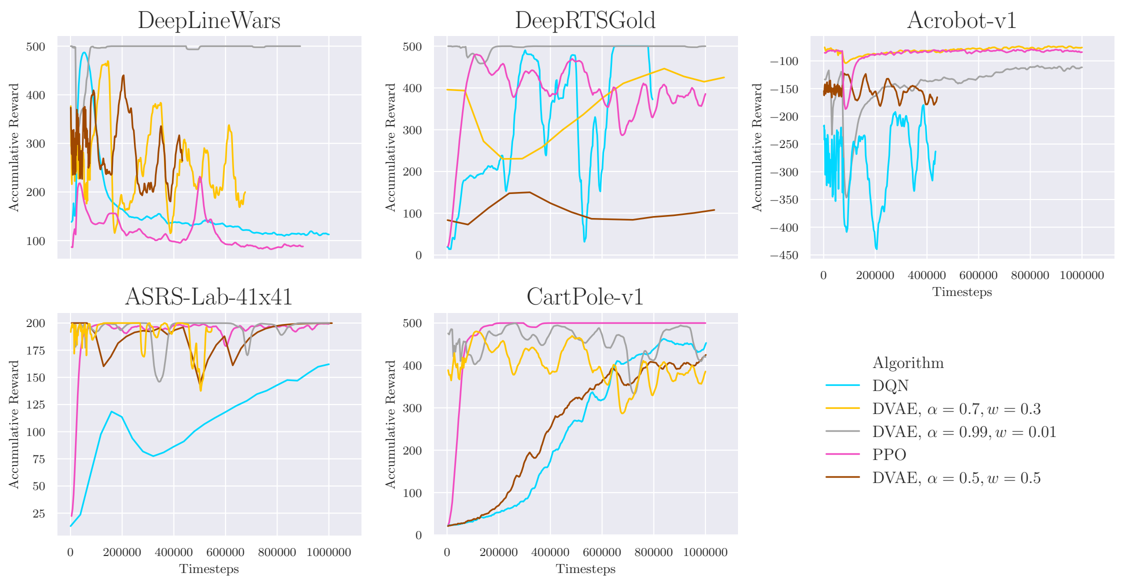


Figure 12: **Behavioral agent performance.** DVAE shows good performance when accumulating reward (y-axis) during training for 1 million timesteps (x-axis). The experiment was averaged across 100 runs and due to execution time, limited to only a subset of the environments

Paper G: Towards Safe Reinforcement-learning in Industrial Grid-warehousing

G

Paper H

Title:	CostNet: An End-to-End Framework for Goal-Directed Reinforcement Learning
Authors:	Andersen, Per-Arne et al.
Affiliation:	Department of ICT, University of Agder, Grimstad Norway
Conference:	40th SGAI International Conference on Artificial Intelligence
Year:	2020, December

CostNet: An End-to-End Framework for Goal-Directed Reinforcement Learning

Per-Arne Andersen
Department of ICT
University of Agder
Grimstad, Norway
per.andersen@uia.no

Morten Goodwin
Department of ICT
University of Agder
Grimstad, Norway
morten.goodwin@uia.no

Ole-Christoffer Granmo
Department of ICT
University of Agder
Grimstad, Norway
ole.granmon@uia.no

Abstract

Reinforcement Learning (RL) is a general framework concerned with an agent that seeks to maximize rewards in an environment. The learning typically happens through trial and error using explorative methods, such as ϵ-greedy. There are two approaches, model-based and model-free reinforcement learning, that show concrete results in several disciplines. Model-based RL learns a model of the environment for learning the policy while model-free approaches are fully explorative and exploitative without considering the underlying environment dynamics. Model-free RL works conceptually well in simulated environments, and empirical evidence suggests that trial and error lead to a near-optimal behavior with enough training. On the other hand, model-based RL aims to be sample efficient, and studies show that it requires far less training in the real environment for learning a good policy.

A significant challenge with RL is that it relies on a well-defined reward function to work well for complex environments and such a reward function is challenging to define. Goal-Directed RL is an alternative method that learns an intrinsic reward function with emphasis on a few explored trajectories that reveals the path to the goal state.

Fortieth SGAI International Conference on Artificial Intelligence (AI-2020), Cambridge England
Submitted 3. Jul, 2020, Accepted 1. Sep 2020, Published 08. Dec, 2020

This paper introduces a novel reinforcement learning algorithm for predicting the distance between two states in a Markov Decision Process. The learned distance function works as an intrinsic reward that fuels the agent's learning. Using the distance-metric as a reward, we show that the algorithm performs comparably to model-free RL while having significantly better sample-efficiently in several test environments.

Keywords: Reinforcement Learning, Markov Decision Processes, Neural Networks, Representation Learning, Goal-directed Reinforcement Learning

1 Introduction

Goal-directed reinforcement learning (GDRL) separates the learning into two phases, where phase one aims to solve the goal-directed exploration problem (GDE). To solve the GDE problem, the agent must determine at least one viable path from the initial state to the goal state. In phase two, the agent uses the learned path to find a near-optimal path. The two phases iterate until the agent policy is converged.

Reinforcement learning (RL) classifies into two categories of algorithms. Model-free RL learns a policy or a value-function by interaction with the environment and succeeds in various simulated areas, including video-games [19, 25], robotics [12, 15], and autonomous vehicles [7, 24], but comes at the cost of efficiency. Specifically, model-free approaches suffer from low sample efficiency and are a fundamental limitation for application in real-world physical systems.

On the other hand, Model-based reinforcement learning (MBRL) aims to learn a predictive model of the environment to increase sample efficiency. The agent samples from the learned predictive model, which reduces the required interaction with the environment. However, it is challenging to achieve good accuracy of the predictive model for many domains, specifically for high complexity environments. With high complexity comes high modeling error (model-bias) and it is perhaps the most common problem for unstable and collapsing policies in model-based RL. Recent work in model-based RL focuses primarily on learning high-dimensional and complex predictive models with graphics as part of the MDP. This complicates the model severely and limits long-horizon predictions as the prediction-error increases exponentially.

This paper address this issue with a combination of GDRL and MBRL by learning a predictive model and a distance model that describes the distance between two states. The learned predictive model abstracts the state-space to distance between state and goal, which reduce the state-complexity significantly. The learned distance is applied to the reward-function of Deep Q-Learning (DQN) [18] and accelerates the learning effectively. The proposed algorithm, CostNet, is an end-to-end solution for goal-directed reinforcement learning where the main contributions are summarized as follows.

1. CostNet for estimating the distance between arbitrary states and terminal states,
2. modified objective for DQN for efficient goal-directed reinforcement learning, and
3. the proposed method demonstrates excellent performance in simulated grid-like environments.

The paper is organized as follows. Section 2 details the preliminary work for the proposed method. Section 3 presents a detailed overview of related work. Section 4 introduces CostNet, a novel algorithm for cost-directed reinforcement learning. Section 5 thoroughly presents the results of the proposed approach, and Section 6 summarizes the work and propose future work in Goal-Directed Reinforcement learning.

2 Background

Model-based reinforcement learning builds a model of the environment to derive its behavioral policy. The underlying mechanism is a Markov Decision Process (MDP), which mathematically defines the synergy between state, reward, and actions as a tuple $M = (S, A, T, R)$, where $S = \{s_n, \ldots, s_{t+n}\}$ is a set of possible states and $A = \{a_n, \ldots, a_{t+n}\}$ is a set of possible actions. The state transition function $T : S \times A \times S \to [0, 1]$, which the predictive model tries to learn is a probability function such that $T_{a_t}(s_t, s_{t+1})$ is the probability that current state s_t transitions to s_{t+1} given that the agent choses action a_t. The reward function $R : S \times A \to \mathbb{R}$ where $R_{a_t}(s_t, s_{t+1})$ returns the immediate reward received on when taking action a in state s_t with transition to s_{t+1}. The policy takes the form $\pi = \{s_1, a_1, s_2, a_2, \ldots, s_n, a_n\}$ where $\pi(a|s)$ denotes chosen action given a state. Model-based reinforcement learning divides primarily into three categories: 1) Dyna-based, 2) Policy Search-based, and 3) Shooting-based algorithms in which this work concerns Dyna-based approaches. The Dyna algorithm from [26] trains in two steps. First, the algorithm collects experience from interaction with the environment using a policy from a model-free algorithm (i.e., Q-learning). This experience is part of learning an estimated model of the environment, also referred to as a predictive model. Second, the agent policy samples imagined data generated by the predictive model and update its parameters towards optimal behavior.

Autoencoders are commonly used in supervised learning to encode arbitrary input to a compact representation, and using a decoder to reconstruct the original data from the encoding. The purpose of autoencoders is to store redundant data into a densely packed vector form. In its simplest form, an autoencoder consists of a feed-forward neural network where the input and output layer is of equal neuron capacity and the hidden layer smaller, used to compress the data. The model consists of an encoder $Q(z|X)$, latent variable distribution $P(z)$, and decoder $P(\hat{X}|z)$. The input X is a vector that represents only a fraction of the ground truth. The objective is for the autoencoder to learn the distribution of all possible training samples, including data not in the training data, but nevertheless, part of the distribution $P(X)$. The final objective for the model is $\mathbb{E}[logP(X|z)] - D_{KL}[Q(z|X) \| P(z)]$, where the first term denotes the reconstruction loss, similar to standard autoencoders and the second term the distance between the estimated latent-space and the ground truth space. The ground truth latent-space is difficult to define, and therefore it is assumed to be a Gaussian, and hence, the learned distribution should also be a Gaussian.

3 Related Work

Pioneering work of the goal-directed viewpoint of reinforcement learning, uniformly suggests that pre-processing of the state-representation (i.e., model-based RL) and careful reward modeling is the preferred method to perform efficient GDRL. The following section introduces related work in GDRL and relevant model-based reinforcement learning methods[1].

3.1 Goal-Directed Reinforcement Learning

Earlier studies have contributed significantly to improve the abilities to solve reinforcement learning problems with a goal-directed approach. Perhaps the most well-known study of the Goal-Directed Reinforcement Learning problem begins with Koenig and Simmons [13]. Their approach splits the problem into two phases, known as Goal-directed exploration (GDEP) and knowledge exploitation. The study finds that the convergence of GDRL-based $\hat{Q}$- and Q-learning closely relates to the state representation and volume of prior knowledge. Furthermore, their work shows that computationally intractable problems are tractable with minor modifications to the state- representation.

Braga and Araújo apply GDRL in [5], using temporal-difference learning to collect prior knowledge and to create a reward and penalty surface explaining the environment dynamics. The map acts as an expert advisor for the TD algorithm and proves the policy performance. Their work shows that the concept of GDRL works well in grid-based environments and includes significantly better sample efficiency compared to Q-Learning.

In [17], the authors study the importance of reward function and initial Q-values for GDRL. The authors thoroughly studied the effect of different initial states of the Q-table and found it challenging to design a generic algorithm for initially setting optimal parameters. However, they found that initial values impact the performance and sample efficiency considerably. Furthermore, the author shows that adding a *goal bias leads to much faster*

[1]The reader is referred to [20] for an in-depth survey of MBRL-based methods.

learning and recommends an *adjustable continuous reward function*. More recently, Debnath et al. propose a hybrid approach, formalized as a GDRL problem, where the first phase optimizes a predictive model of the environment with samples from a model-free reinforcement learning policy. The second phase exploits the learned predictive model to improve the policy further, similar to [3]. The authors show that GDRL-based algorithms accelerate learning and improve sample efficiency considerably [6].

3.2 Model-Based Reinforcement Learning

The Model-Ensemble Trust-Region Policy Optimization (ME-TRPO), formally proposed by [14], is a Dyna-based algorithm for learning a predictive model. The ME-TRPO method uses an ensemble of neural networks to form the predictive model, which significantly reduces model-bias, increasing its generalization abilities. The ensemble individually trains using single-step L2 loss in a supervised setting. After training of the algorithm, the authors use Trust-Region Policy Optimization from [22] as the model-free approach. The work shows significantly faster convergence in several continuous control tasks.

The ME-TRPO method extends to Stochastic Lower Bound Optimization (SLBO) [16]. In comparison, SLBO modifies the single-step L2 loss to multi-step L2-norm loss to the train ensemble predictive model. The authors present a mathematical framework for the guaranteed monotonic improvement of the predictive model.

In [10], the authors analyze previous methods and their capability to generalize well for longer time horizons. Their analysis suggests that the performance is good for shorter time horizons, but exponentially decrease as uncertainty appears when predicting longer rollouts. The proposed algorithm is called Model-based Policy Optimization (MBPO) and balance a trade-off between sample efficiency and performance. The paper suggests a prediction horizon between and 1-15 states, up to 200 states. In conclusion, MBPO shows that model-based approaches can outperform state-of-the-art model-free reinforcement learning when tuning appropriately.

4 CostNet for Goal-Directed RL

CostNet is a combination of four disciplines in Deep Learning, 1) Goal Directed RL [13], 2) Model-Based RL [27], and 3) Variational Autoencoders [11] and forms a novel approach for learning the cost between states modeled after an MDP. The algorithm accumulates training data from using expert systems or random sampling. For systems where safety is a priority, it is advised to perform sampling according to manually defined risk constraints at the cost of increased sample complexity [3].

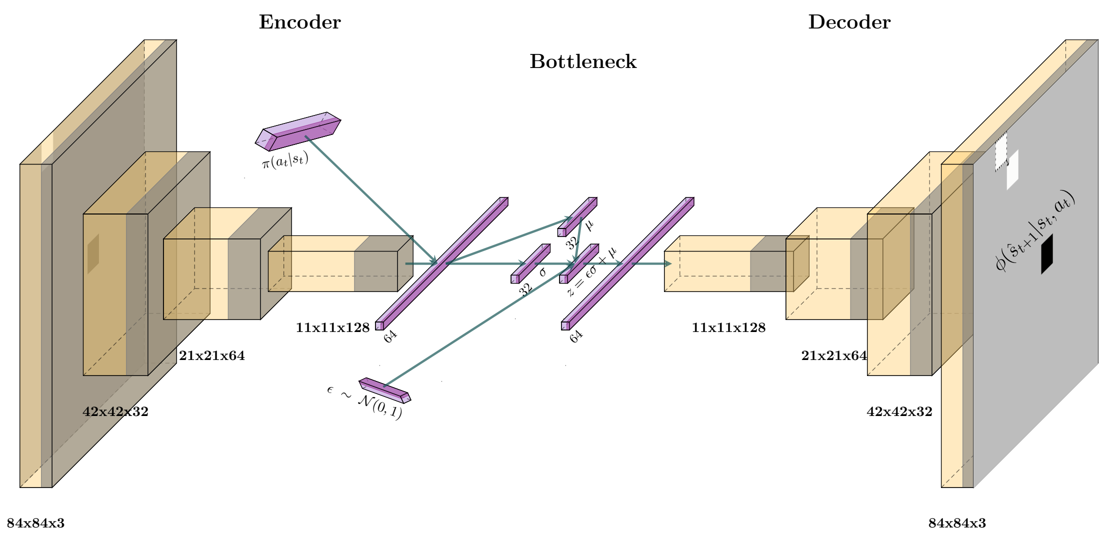


Figure 1: The encoder-latent-decoder architecture for learning a compact representation of states. The model is a convolutional variational autoencoder with three layers of convolutions before the latent-vector computation. The input is a state s_t. The latent-space, z forms from an estimated μ, and σ, mean and standard-deviation respectively, from a Gaussian. The $\epsilon \sim N$ denotes sampling using the reparametrization trick, as described in [11]. On the right-hand side, the estimated latent-variable z reconstructs into the future state $\hat{s}_{t+1}$

The initial phase of training revolves around training a predictive model of the environment. Recent work indicates that state-of-the-art models suffer from sever policy drift after a few predictions [2, 8, 10], and CostNet is no exception. Therefore, the problem is redefined to learning only the one-step prediction under a policy $\phi(\hat{s}_{t+1}|s_t, a_t)^\pi$, where ϕ denotes the predictive model. The predictive model is a variational autoencoder (VAE), where the goal is to map input (state) to latent-vectors that describes best possible describe the input. Figure 1 illustrates the proposed structure for the encoder-latent-decoder model for CostNet. The input is an image of an arbitrary state, and the hidden layers are convolutions with 32, 64, and 128 filters, a kernel size of 2, and a stride of 2 with ReLU activation, respectively. The latent-vector size is 64 neurons, but it is highly advised to fine-tune these hyper-parameters as the required embedding capacity varies on the state-complexity.

The model-based approach to encode states as is developed as a method to improve the performance of the feed-forward neural network. Figure 2 shows the proposed architecture for the CostNet algorithm and consists of two models with different objectives. The first model, CostNet$_\theta^0$, predicts which of the two states are closest to the goal, $state_A$, or $state_B$. The output is a vector that describes the probability of both $state_A$ and $state_B$ being closest to the goal. The second model, CostNet$_\theta^1$, predicts the absolute distance to a

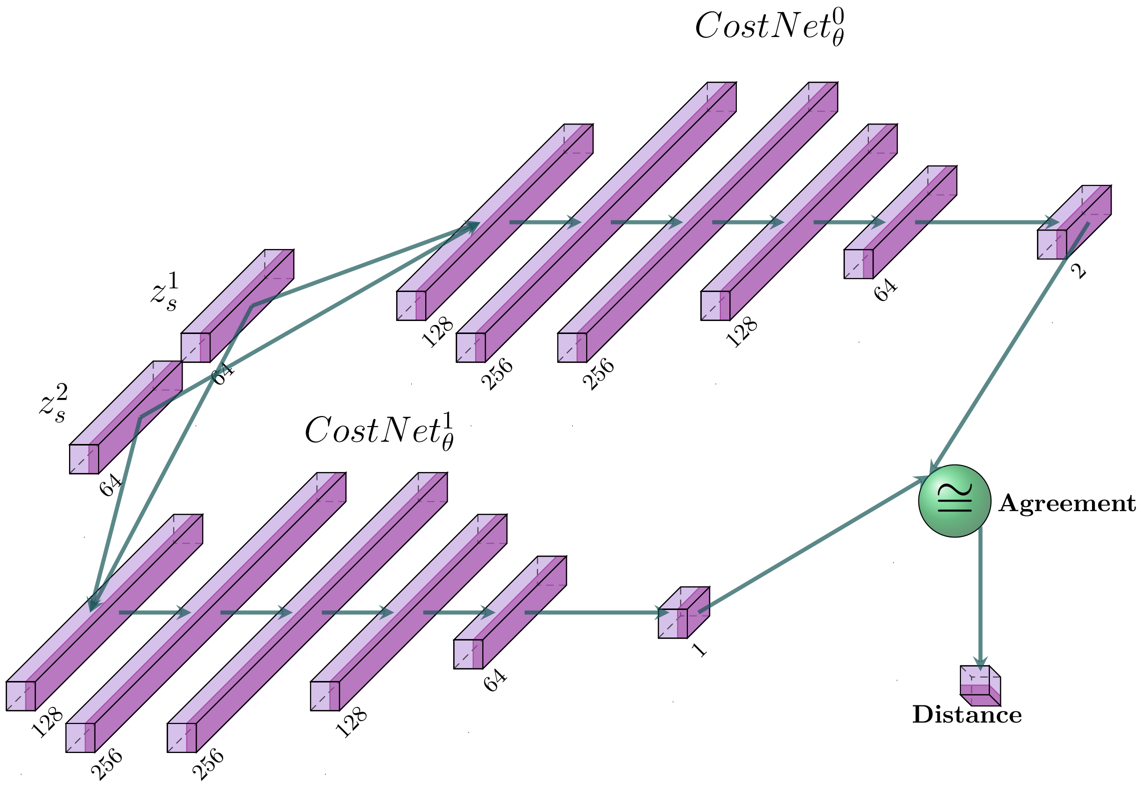


Figure 2: The proposed CostNet architecture. There are in-total two inputs, z_s^1 and z_s^2, that represent encoded states (see Figure 1. The inputs are sent through two streams (models), CostNet$_\theta^0$, and CostNet$_\theta^1$, and learns two separate objectives using MSE. During training, both networks must agree on the answer for gradients to contribute in a positive direction. When both networks predict the same state to be closest to the goal state, the training is completed. The hidden layers are regular fully-connected with ReLU activation. The output for CostNet$_\theta^0$ activates with softmax, and CostNet$_\theta^1$ with sigmoid activation.

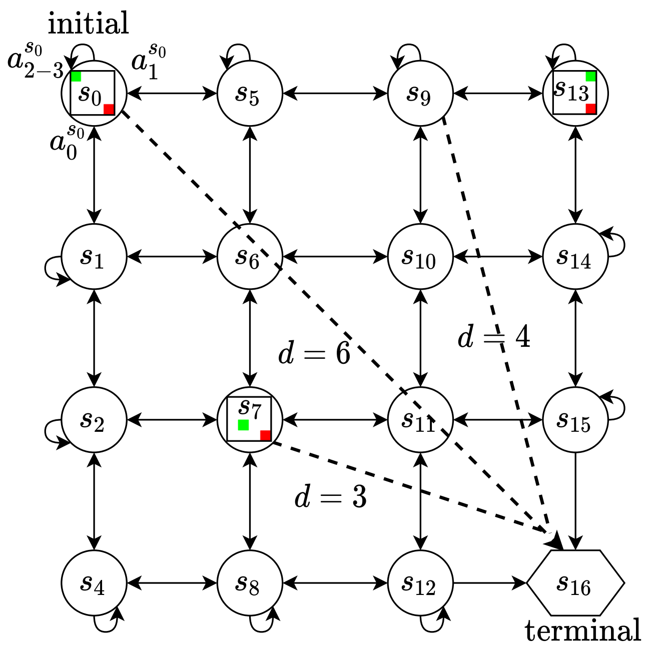


Figure 3: Illustration of the reformulation of the model-based MDP problem. State 0, 7, and 13 illustrate the usual state complexity for each node in the MDP graph; a pixel-based (full-state) representation. CostNet, on the other hand, simplifies the state nodes to only a single metric: **distance**. This simplifies the complexity of the state-space significantly, and for GDRL-based approaches is sufficient representation.

goal state as a real number between 0 and 1, where 0 is at the goal state, and 1 is at maximum possible distance. Both networks train using mean squared error (MSE) loss, where the labels stem from the experience-buffer and the distance label from a backtracking algorithm. The predictions are considered correct (reliable) when there is an agreement between both networks, i.e. that CostNet$_\theta^0$ correctly predicts which of $state_A$ or $state_B$ is closest, and CostNet$_\theta^1$ predict the actual distance.

To exemplify, consider the inputs z_s^1 ($state_A$) and z_s^2 ($state_B$) where z_s^1 is closest to the goal state. In this case, the first index in the vector from the CostNet$_\theta^0$ prediction should be the largest signal, and the predicted distance from CostNet$_\theta^1$ for z_s^1 should be less for the similar prediction z_s^1. If this is in place, we claim that the models are in agreement. When the agreement between the networks is consistent, the training is considered complete.

Figure 3 shows how the MDP is reduced to only focus on the distance to the goal-state. In regular MDP's the whole state information is represented at each node, illustrated by the inner square in state $s_0, s_7, \text{and} s_{13}$. However, in this work, the MDP nodes only try to model the distance from one node to an arbitrary goal node. The problem with this formulation is that 1) there may be many goal states, and 2) agent must visit a goal state at least once. Therefore, the goal-directed approach works best in environments with less stochasticity in terms location of the goal states.

Algorithm 1: CostNet with Deep Q-Learning

```
Result: Optimized policy $\pi$ given a set of states $S$, actions $A$)
1  Hyperparameters: Discount factor $\gamma \in [0 \dots 1]$, Learning-rate $\alpha \in [0 \dots 1]$, and
     Drift threshold $\psi \in [0 \dots 1]$
2  Assumptions: Experience-Replay (ER) from expert-system or random sampling, $\Omega$
3  while training do
4      Train predictive model $\phi(\hat{s}_{t+1}, z_t | s_t, s_{t+1}, a_t)$ from ER using objective
         $\mathbb{E}[logP(X|z)] - D_{KL}[Q(z|X) \| P(z)]$
5      if $\phi$ ¡ $\psi$ then
6          Train first supervised CostNet $C_\theta^1$ using $z_t, \hat{z}_{t+1}$ from $\phi(z_t|s_t)$ and
             $\phi(\hat{z}_{t+1}|\hat{s}_{t+1})$ with MSE loss.
7          Train second supervised CostNet $C_\theta^2$ using $z_t$ from $\phi(z_t|s_t)$ with MSE loss.
8          if $C_\theta^1 \cong C_\theta^2$ for most predictions (agreement) then
9              training = false
10         end
11     end
12 end
13 while training DQN do
14     Choose action $a$ based on $\epsilon$-greedy Execute $a$ at state $s$ and get $s_{t+1}, r$ Perform
         Q-Update:
15     $L_i(\theta_i) = \mathbb{E}_{s,a,r,s' \sim \rho(.)}\left[(y_i - Q(s,a;\theta_i))^2\right]$ where
         $y_i = \frac{r}{1-C_\theta^2} + \gamma \max_{a'} Q(s',a';\theta_{i-1})$
16 end
```

Predictive model. The algorithm is summarized in the following line-by-line procedure. (Line 1) Initialize hyper-parameters for γ, α, and ψ where the drift-threshold evaluates for n future predictions. When the algorithm is consistently below the threshold, training is complete. (Line 4) Train the predictive model using the ER-buffer using the objective from [11] where $Q(z|X)$ is the encoder to latent-space, $P(z)$ is the distribution of latent-space, and $P(X|z)$ is the decoder distribution. The object splits into two terms. The first term is the reconstruction loss (MSE), and the second term computes the KL distance between the predicted latent distribution $Q(z|X)$ and the assumed normal distributed space $P(z) \sim N(0,1)$. (Line 5) when the predicted model is below the drift-threshold ψ, the training concludes.

CostNet. (Line 6, Line 7) The CostNet$_\theta^0$ model trains with encoded states z_t and z_{t+1} as input and produces a vector that predicts a probability for each of the states being

Table 1: Hyper-parameters of CostNet algorithm

Parameter	**Value**
Learning Rate (DQN)	0.01
Discount Factor (DQN)	0.95
ER-Size (DQN)	5000
Optimizer	Adam
Optimizer Learning Rate	0.001
Drift-Threshold ψ	0.3

closest to the goal state. A second model, CostNet$^1_\theta$ predicts the distance for a single state, and when the predictions align consistently, the training concludes[2]. (Line 13) The training CostNet is complete, and regular model-free RL is performed, using DQN [18] respectively.

Model-free RL. (Line 14, Line 15) The agent performs regular sampling to accumulate ER for training. The training is performed similarly to [18], but modifies the reward signal to account for the distance from the goal state. When the distance signal is weak, the agent receives little reward and otherwise large rewards for states close to the goal state.

5 Results and Discussion

The CostNet algorithm is tested in four environments, CartPole-v1 from [4], DeepRTS GoldCollect [1][3], and DeepMaze StaticNoWalls [2][4]. The experiments compare CostNet to DQN [18], and PPO [23] for 1000000 timesteps during 100 experiments for statistical analysis of the results[5].

5.1 Results

The hyper-parameters for CostNet are shown in Table 1. Figure 4 shows the environments used in the experiments. The first environment is CartPole, a common benchmark for exploratory reinforcement learning research. The objective is to balance a pole on a cart for 500 timesteps at which the episodes end. The second environment is DeepRTS Gold-Collect, a simple environment where the goal is to accumulate as much gold as possible for 5 minutes. The optimal episodic reward for this environment is 1000. Finally, the DeepMaze StaticNoWalls environment is an 11×11 grid structure where the goal is

[2]CostNet$^x_\theta$ where θ is the parameters that are optimized for the model.

[3]The DeepRTS environment is available at: `https://github.com/cair/deep-rts`

[4]The DeepMaze environment is available at: `https://github.com/cair/deep-maze`

[5]The experiments are available here `https://github.com/cair/CostNet`

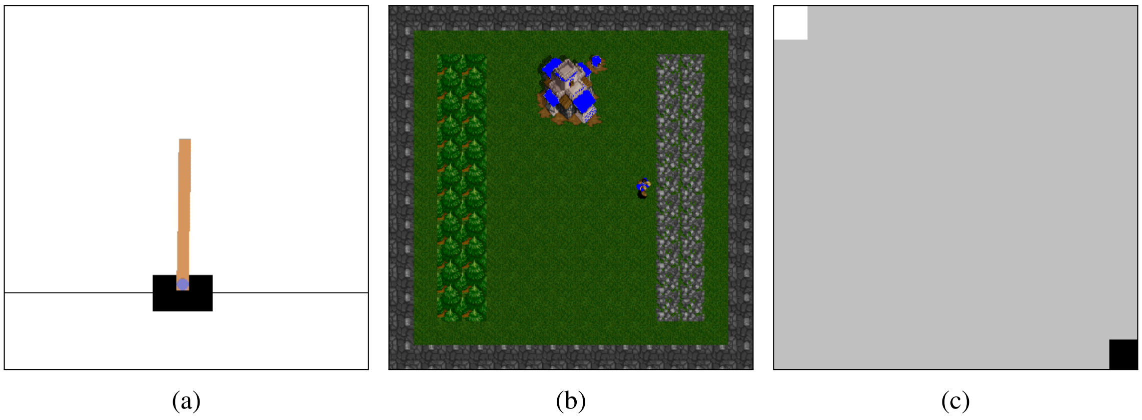


Figure 4: Illustration of the experiments. (a) Cart Pole. Goal is to balance the pole until terminal state occurs. (b) DeepRTS GoldCollect. Gather as much gold as possible in a time-frame of 5 minutes. (c) DeepMaze. The player (white) must enter the terminal state (black) in shortest possible time.

located at a fixed position. The reward for DeepMaze is the length of the maze because the agent and goal are located at opposite corners.

Figure 5 compares the performance of CostNet against two competing algorithms, DQN and PPO. Several parameters for the drift-threshold parameterψ is tested, but a value of 0.3 seems to be stable across several environments. The PPO algorithm uses the parameters defined in [23] and for DQN in [19]. CostNet shows significantly better performance across all environments in terms of variance, seen clearly in the DeepMaze environment results. The primary reason for this is that the algorithm starts with a relatively good idea of the underlying environment dynamics from learning the predictive model. Furthermore, in terms of raw performance, the CostNet agent starts at near-optimal performance in some environments, such as the DeepRTS Gold-Collect environment. There are still challenges to be investigated, such as preventing divergence if the policy is already doing good behavior. Another problem is that CostNet demands initial data from expert systems which, is not possible in all environments. Regardless of these challenges, the algorithm is a good leap in the right direction and clearly CostNetwith a modified DQN reward-function, significantly increases the agent's performance, especially in more complex environments such as Deep RTS.

5.2 Discussion

This paper's contribution shows that it is possible to learn distances between states in an MDP reliably and that the learned distance is useful for generic reward functions. The significance of applicability for CostNet spans across several disciplines. **Games** are perhaps the most obvious application for the algorithm as it is not always trivial to

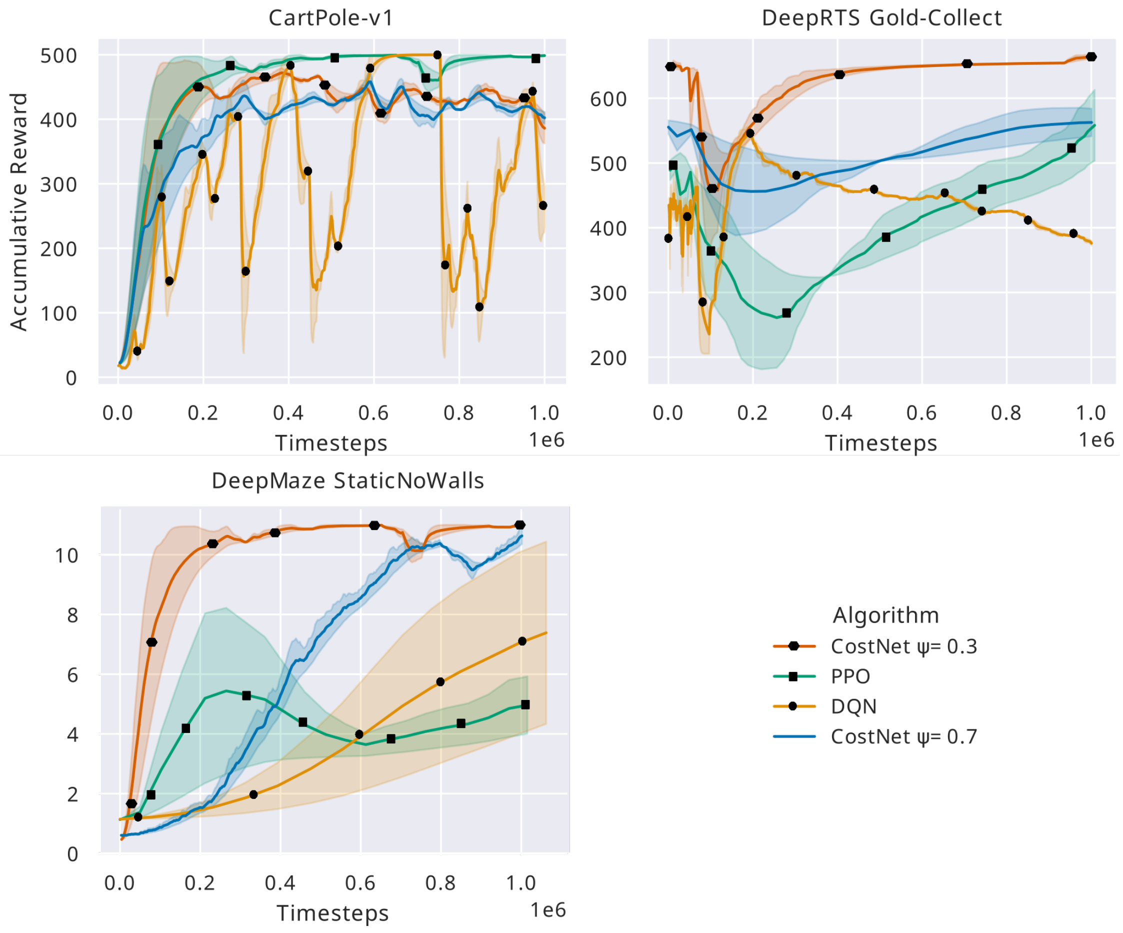


Figure 5: A comparison of PPO (square), DQN (circle), CostNet (unmarked and diamond) performance in CartPole, DeepRTS ,and Deep Maze environment. The y-axis shows the accumulated reward, and the x-axis is at which timestep. Every experiment runs for 100 episodes for 1 million timesteps. CostNet shows outstanding performance compared to fully model-free variants in two of three environments. In CartPole, PPO is superior, but CostNet closely follows. The experiments show that increasing the drift-threshold ψ also decreases performance and are an indication that CostNet impacts performance positively.

design reward functions generic enough to describe every state a complex MDP. While the proposed algorithm also suffers from generalization for multi-objective environments, it is still more accurate in learning reward functions compared to manually crafted functions. **Industry** the CostNet algorithm is applicable to the industry, especially in areas where the goal is stationary for all timesteps. One example is grid-warehousing, where agents operate on an A-to-B objective. However, upholding safety is a big concern when using RL-based algorithms, and therefore, GLDR-based approaches should be used with care.

6 Future Work and Conclusion

One question that merits future investigation is how to define optimal encode the state-space into latent-vectors. Using VAE is efficient, but still suffers from severe policy-shift for many-step future predictions. The proposed method is generic and should yield significant benefits from encoders that surpass VAE. Specifically, the VQ-VAE2 [21] shows promise, surpassing VAE in several disciplines. It would be interesting to see the effect VQ-VAE's discrete latent representation has on the overall performance when calculating state-to-state distances.

Another enticing direction for future work is analytical work for the CostNet architecture. The algorithm shows promising results empirically, which is often the case for deep reinforcement learning, but it remains future work to analytically prove the algorithm. Furthermore, in the extension of this work, the goal is to test the algorithm in many environments such as the MuJoCo, Atari Arcade, and DeepMind Lab environment to investigate its capabilities to generalize.

CostNet is a novel architecture for accelerating model-free reinforcement learning by combining goal-directed reinforcement learning and model-based reinforcement learning. The hybrid approach learns a predictive model, similar to [9], but learns a simpler model, CostNet, which captures only the distance between any given state and a terminal state. The algorithm outperforms DQN and PPO in several environments and shows outstanding stability during learning. Furthermore, CostNet shows promise for several disciplines, including games, industry, and autonomous driving. The hope is that future studies will lead to many more successes.

References

[1] Andersen, P., Goodwin, M., Granmo, O.: Deep RTS: A Game Environment for Deep Reinforcement Learning in Real-Time Strategy Games. In: 2018 IEEE Conference on Computational Intelligence and Games (CIG). pp. 1–8 (aug 2018). https://doi.org/10.1109/CIG.2018.8490409

[2] Andersen, P.A., Goodwin, M., Granmo, O.C.: The Dreaming Variational Autoencoder for Reinforcement Learning Environments. In: Max Bramer, Petridis, M. (eds.) Artificial Intelligence XXXV, vol. 11311, pp. 143–155. Springer, Cham, xxxv edn. (dec 2018). https://doi.org/10.1007/978-3-030-04191-5_11

[3] Andersen, P., Goodwin, M., Granmo, O.: Increasing sample efficiency in deep reinforcement learning using generative environment modelling. Expert Systems (mar 2020). https://doi.org/10.1111/exsy.12537

[4] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: OpenAI Gym. arxiv preprint arXiv:1606.01540 (jun 2016), http://arxiv.org/abs/1606.01540

[5] de S. Braga, A.P., Araújo, A.F.R.: Goal-Directed Reinforcement Learning Using Variable Learning Rate. pp. 131–140. Springer, Berlin, Heidelberg (1998). https://doi.org/10.1007/10692710_14

[6] Debnath, S., Sukhatme, G., Liu, L.: Accelerating Goal-Directed Reinforcement Learning by Model Characterization. In: IEEE International Conference on Intelligent Robots and Systems. pp. 8666–8673. Institute of Electrical and Electronics Engineers Inc. (dec 2018). https://doi.org/10.1109/IROS.2018.8593728

[7] Grigorescu, S., Trasnea, B., Cocias, T., Macesanu, G.: A survey of deep learning techniques for autonomous driving. Journal of Field Robotics **37**(3), 362–386 (apr 2020). https://doi.org/10.1002/rob.21918

[8] Ha, D., Schmidhuber, J.: Recurrent World Models Facilitate Policy Evolution. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31, pp. 2450–2462. Curran Associates, Inc., Montréal, CA (sep 2018), http://papers.nips.cc/paper/7512-recurrent-world-models-facilitate-policy-evolution.pdf

[9] Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., Davidson, J.: Learning Latent Dynamics for Planning from Pixels. In: Chaudhuri, K., Salakhutdinov, R. (eds.) Proc. 36th International Conference on Machine Learning, ICML'18. vol. 97, pp. 2555–2565. PMLR, Long Beach, CA, USA (jun 2019), http://proceedings.mlr.press/v97/hafner19a/hafner19a.pdf

[10] Janner, M., Fu, J., Zhang, M., Levine, S.: When to Trust Your Model: Model-Based Policy Optimization. In: Wallach, H., Larochelle, H., Beygelzimer, A., Alché-Buc, F., Fox, E., Garnett, R. (eds.) Proc. 33rd Conference on Neural Information Processing Systems (NeurIPS). pp. 12519–12530. Curran Associates, Inc., Vancouver,

BC, Canada (jun 2019), http://papers.nips.cc/paper/9416-when-to-trust-your-model-model-based-policy-optimization.pdf

[11] Kingma, D.P., Welling, M.: Auto-Encoding Variational Bayes. Proceedings of the 2nd International Conference on Learning Representations (dec 2013). https://doi.org/10.1051/0004-6361/201527329

[12] Kober, J., Bagnell, J.A., Peters, J.: Reinforcement learning in robotics: A survey. The International Journal of Robotics Research **32**(11), 1238–1274 (sep 2013). https://doi.org/10.1177/0278364913495721

[13] Koenig, S., Simmons, R.G.: The Effect of Representation and Knowledge on Goal-Directed Exploration with Reinforcement-Learning Algorithms. Machine Learning **22**(1/2/3), 227–250 (1996). https://doi.org/10.1023/A:1018068507504

[14] Kurutach, T., Clavera, I., Duan, Y., Tamar, A., Abbeel, P.: Model-Ensemble Trust-Region Policy Optimization. In: 6th International Conference on Learning Representations. Vancouver, BC, Canada (2018), https://openreview.net/forum?id=SJJinbWRZ

[15] Levine, S., Finn, C., Darrell, T., Abbeel, P.: End-to-end training of deep visuomotor policies. Journal of Machine Learning Research **17**(1), 1334–1373 (2016), http://www.jmlr.org/papers/volume17/15-522/15-522.pdf

[16] Luo, Y., Xu, H., Li, Y., Tian, Y., Darrell, T., Ma, T.: Algorithmic Framework for Model-based Reinforcement Learning with Theoretical Guarantees. Proceedings, 8th International Conference on Learning Representations (ICLR) (2018), https://openreview.net/forum?id=BJe1E2R5KX

[17] Matignon, L., Laurent, G.J., Le Fort-Piat, N.: Reward function and initial values: Better choices for accelerated goal-directed reinforcement learning. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). vol. 4131 LNCS, pp. 840–849. Springer Verlag (2006). https://doi.org/10.1007/11840817_87

[18] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing Atari with Deep Reinforcement Learning. Neural Information Processing Systems (dec 2013), http://arxiv.org/abs/1312.5602

[19] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis,

D.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (feb 2015). https://doi.org/10.1038/nature14236

[20] Polydoros, A.S., Nalpantidis, L.: Survey of Model-Based Reinforcement Learning: Applications on Robotics. Journal of Intelligent & Robotic Systems **86**(2), 153–173 (may 2017). https://doi.org/10.1007/s10846-017-0468-y

[21] Razavi, A., van den Oord, A., Vinyals, O.: Generating Diverse High-Fidelity Images with VQ-VAE-2. In: Wallach, H., Larochelle, H., Beygelzimer, A., Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32. pp. 14837–14847. Curran Associates, Inc., Vancouver, BC, Canada (2019), `http://papers.nips.cc/paper/9625-generating-diverse-high-fidelity-images-with-vq-vae-2`

[22] Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P.: Trust Region Policy Optimization. In: Bach, F., Blei, D. (eds.) Proceedings of the 32nd International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 37, pp. 1889–1897. PMLR, Lille, France (2015), `http://proceedings.mlr.press/v37/schulman15.html`

[23] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal Policy Optimization Algorithms. arxiv preprint arXiv:1707.06347 (jul 2017), `http://arxiv.org/abs/1707.06347`

[24] Shah, S., Dey, D., Lovett, C., Kapoor, A.: AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. pp. 621–635. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-67361-5_40

[25] Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D.: Mastering the game of Go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016). https://doi.org/10.1038/nature16961

[26] Sutton, R.S.: Dyna, an integrated architecture for learning, planning, and reacting. ACM SIGART Bulletin **2**(4), 160–163 (jul 1991). https://doi.org/10.1145/122344.122377

[27] Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. A Bradford Book, Cambridge, MA, USA, 2 edn. (2018), `https://dl.acm.org/doi/book/10.5555/3312046`

Paper H: CostNet: An End-to-End Framework for Goal-Directed Reinforcement Learning

H

Paper I

Title:	Safer Reinforcement Learning for Agents in Industrial Grid-Warehousing
Authors:	Andersen, Per-Arne et al.
Affiliation:	Department of ICT, University of Agder, Grimstad Norway
Conference:	6th International Conference on Machine Learning, Optimization, and Data Science (LOD)
Year:	2020, July

Safer Reinforcement Learning for Agents in Industrial Grid-Warehousing

Per-Arne Andersen
Department of ICT
University of Agder
Grimstad, Norway
per.andersen@uia.no

Morten Goodwin
Department of ICT
University of Agder
Grimstad, Norway
morten.goodwin@uia.no

Ole-Christoffer Granmo
Department of ICT
University of Agder
Grimstad, Norway
ole.granmon@uia.no

Abstract

In mission-critical, real-world environments, there is typically a low threshold for failure, which makes interaction with learning algorithms particularly challenging. Here, current state-of-the-art reinforcement learning algorithms struggle to learn optimal control policies safely. Loss of control follows, which could result in equipment breakages and even personal injuries.

On the other hand, a model-based reinforcement learning algorithm aims to encode environment transition dynamics into a predictive model. The transition dynamics define the mapping from one state to another, conditioned on an action. A sufficiently accurate predictive model should learn optimal behavior, also even in real environments.

The paper's heart is the introduction of the novel, Safer Dreaming Variational Autoencoder, which combines constrained criterion, external knowledge, and risk-directed exploration to learn good policies. Using model-based reinforcement learning, we show that the proposed method performs comparably to model-free algorithms safety constraints, with a substantially lower risk of entering catastrophic states.

Keywords: Deep Reinforcement Learning, Model-based Reinforcement Learning, Neural Networks, Variational Autoencoder, Markov Decision Processes, Exploration

The Sixth International Conference on Machine Learning, Optimization, and Data Science (LOD-2020)
Submitted 18. Jun, 2020, Accepted 24. Jun 2020, Published 07. Jan, 2021

1 Introduction

Reinforcement learning has recently demonstrated a high capacity to learn efficient strategies in environments where there are noisy or incomplete data [7]. We find achievements in many domains, such as robotics [12], wireless networking [20], and game-playing [13]. The common denominator between these domains is that they can be computer-simulated with significant resemblance to real-world environments, and therefore, let algorithms train at accelerated rates with strong safety guarantees.

The goal of reinforcement learning algorithms is to learn a behavioral policy that produces optimal actions based on sensory input and feedback from an environment. A policy is a parameterized model that is constructed in exact tabular form or using an approximation neural network with gradient descent [17]. The algorithm performs an iterative process of exploration (often through sampling) , exploitation, and learning policy updates that moves the policy in the direction of better behavior. Exploration is commonly performed using a separate policy, such as random sampling from a Gaussian distribution. It is crucial that the algorithm balance exploration and exploitation with schemes such as ϵ-greedy so that the policy learns with the best possible data distribution.

The challenges towards achieving safer reinforcement learning are many. The main limitations of current state-of-the-art in this context are (1) It requires a tremendous amount of sampling to learn a good policy. (2) Stable and safe policies are challenging to achieve in dynamic environments. (3) Model-free exploration methods are not safe in mission-critical environments. (4) Reinforcement learning methods depend on negative feedback through trial and error and is therefore not applicable to mission-critical systems. Most reinforcement learning techniques are not designed for safe learning, and therefore, few solutions exist for mission-critical real-world environments.

Automated Storage and Retrieval Systems (ASRS) is a modern method of performing warehouse logistics where the system is partially or fully automated. In industry, including ASRS, it is common to rely on complex expert systems to perform tasks such as control, storage, retrieval, and scheduling. If-else statements and traditional pathfinding algorithms drive these tasks. The benefit of expert systems is that it is trivial to model operative safety bounds that limit the system from entering catastrophic states. The downside is that expert systems do not adapt to changes automatically, and requires extensive testing if altered. While it may be possible and perhaps trivial to construct safe routines with an expert system, it is inconceivable to expect optimal behavior due to the complexity of the environment. Reinforcement learning is possibly the most promising approach to solve these problems because it can generalize well across many domains [13] and is designed to work in noisy environments with partial state-space visibility [18].

This paper presents *The Safer Dreaming Varational Autoencoder* (DVAE-S), an algorithm for safe reinforcement learning in real-world environments. DVAE-S is based on previous work in [2], where a model-based approach was proposed for training fully off-line without direct access to the real environment. However, the algorithm suffers from making catastrophic actions, a common problem in reinforcement learning and, therefore, does not satisfy the safety requirements. The following research question was raised.

How to ensure that the agent acts within defined safety boundaries?

DVAE-S address this question by eliminating catastrophic states from the state-space and utilizing external knowledge from previously hand-crafted algorithms

The rest of the paper is structured as follows. Section 2 discusses related work to the proposed algorithm. Section 3 introduces preliminaries. Section 4 presents the DVAE-S algorithm for safe reinforcement learning. Section 5, presents an approach for safer learning of ASRS agents using the DVAE-S algorithm. Finally, Section 6 concludes the work and proposes a roadmap for future work in safe reinforcement learning.

2 Related Work

This section address related work to applied reinforcement learning in industry-near environments and on improving safety in reinforcement learning. Advancements in reinforcement learning have recently been more frequent and with substantial performance improvements in various domains [4]. Trial and error is central in standard reinforcement learning and therefore, little attention is given to safety during training.

Reinforcement learning is previously applied to industry-near environments, and perhaps the most widespread application is autonomous vehicles. The proposed method in this paper uses an auxiliary policy to label data for supervised training. With only 12 hours of labeled data, [19] shows that it is possible to learn a performant policy using a direct perception approach with convolutional neural networks. This approach is much like a variational autoencoder that simplifies the perception of the world significantly. This simplifaction of the input significantly speeds up inference, which enables the system to issue control commands more frequently. Many other significant contributions in autonomous vehicle control directly relate to control in ASRS environments, such as [15].

2.1 Safe Reinforcement Learning

In the majority of established systems in the industry, an expert system acts as the controller for the environment. It is critical for safe and stable learning in real world-environments so that ongoing operations are not interrupted.

Similar to this work, [5] assumes a predictive model that learns the dynamics of the environment. The authors propose that the policy should be limited to a safe-zone, called the **R**egion **O**f **A**ttraction (ROA). Everything within the bounds of the ROA is “safe states” that the policy can visit, and during training, the ROA gradually expands by carefully exploring unknown states. The algorithm shrinks the ROA to ensure stability if the feedback signal indicates movement towards catastrophic states.

The proposed algorithm encodes observations as latent embeddings or vectors using a variational autoencoder (VAE), similar to the View model in [10]. The World Model approach defines three components. The *view* (VAE) encodes observations to a compact latent embedding. The *model* (MDM-RNN)[1] is the predictive model used to learn the world model. Finally, the *controller* (C) is an interoperability abstraction that enables model-free algorithms to interact with the world model.

3 Background

This work is modeled according to an **M**arkov **D**ecision **P**rocess (MDP) described formally by a tuple (S, A, T, R), where S is the state-space, A is the action-space, $T\colon S \times A \to S$ is the transition function and $R\colon S \times A \to \mathbb{R}$ is the reward function [17].

3.1 Safer Policy Updates

A policy π in reinforcement learning is a parametrized[2] model that maps (input) observations to (output) actions. The goal is to find an optimal policy $\pi^* = \arg\max_\pi V^\pi(s) \quad \forall s \in S$ where $\max_\pi V^\pi(s)$ denotes the highest possible state-value under policy π.

In traditional RL, the algorithm learns according to an *optimization criterion*. This optimization criterion varies with different algorithms but is commonly implemented to minimize time or to maximize reward. Any cost metric can be used and is defined by the algorithm designer. The *return maximization criterion* is one example where the agent seeks a policy that achieves the highest possible reward, where $R = \sum_{t=0}^{\infty} \gamma^t r_t$ denotes the *expected cumulative future discounted reward.* $\gamma \in [0, 1]$ is the *discount factor* that determines the importance of future rewards. The problem with the return maximization criterion is that it is **not sufficient** for environments with catastrophic states. We adopt the term *risk* from [11], which is a metric that explains the risk of doing actions using a policy π. There is several proposed criterion methods to improve safer policy updates, namely *Risk-Sensitive Criterion* [9], *Worst Case Criterion* [8], and *Constrained Criterion* [14].

[1]Mixture Density Network combined with a Recurrent Neural Networks.

[2]All models in this paper are neural network approximations.

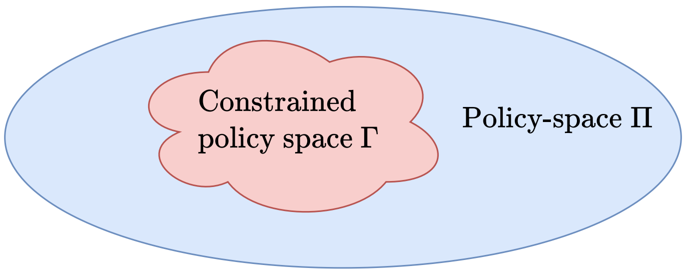


Figure 1: The policy-space (blue) Π and the subset of policies (red) $\Gamma \subseteq \Pi$, where each policy $\pi \in \Gamma$ must satisfy the constraints $c_i \in C$.

This paper use the **Constrained Criterion**, where the objective is modified to maximize the return, similar to [5]. The difference from the traditional objective function is that it introduces additional constraints to the policy objective function. These constraints act as a lower or upper bound for the maximization of the return. The general form of the constrained criterion is defined

$$\max_{\pi \in \Pi} E_\pi(R) \text{ subject to } \; c_i \in C, c_i = \{h_i \gtreqless \alpha_i\} \tag{1}$$

where c_i is the ith constraint in C that must be satisfied by the policy π, additionally $c_i = \{h_i \gtreqless \alpha_i\}$ must specify a function related to the return h_i with a threshold value α_i.

Figure 1 illustrates that a constrained policy-space eliminates a considerable portion of possible policies. The objective of the constraints is to maximize the number of eliminated states that lead to catastrophic outcome and minimize elimination of safe states. The objective function is denoted $\max_{\pi \in \Gamma} E_\pi(R)$ given that only updates from the constrained subset of policies Γ is used. This increases the safety of policy updates far more than the traditional return maximization methods. Depending on the application, the constraints can be tuned using the threshold value α. Higher the value, the constraint is more permissive, and for lower values, more restrictive.

3.2 Safe Exploration

Policies with a constrained criterion do not guarantee safety in the short term. **Risk-directed Exploration** is a good approach to improve short term safety and uses risk to determine in which direction the agent should explore. There are many possible ways to define a risk metrics, but this paper uses *normalized expected return with weighted sum entropy* [6].

$$Risk(s,a) = \mathcal{R}(s,a) = wH(X) - (1-w)\frac{\mathbb{E}[R(s,a,s')]}{\max_{a \in A} |\mathbb{E}[R(s,a,s')]|} \tag{2}$$

where the entropy is defined as

$$H(s,s) = \mathbb{E}\left[-\pi(a|s)\log\pi(a|s)\right]. \tag{3}$$

The resulting utility function denotes as follows:

$$Utility(s,a) = U(s,a) = p(1 - \mathcal{R}(s,a)) + (1-p)\pi(s,a). \quad (4)$$

The calculated risk is multiplied with the action probability distribution for states in the MDP. The updated utility function ensures that sampling is performed in favor of safe and conservative actions, depending on the weight parameter w, and interpolation parameter p [6]. The risk-directed exploration plays well with introducing external knowledge from existing expert systems. The proposed method works well with external knowledge originating from expert systems already running in an environment. Initial knowledge significantly boosts the accuracy of risk and therefore improves the safety and performance of the agent. The downside of using external knowledge is that the predictive model becomes biased and may not learn well when entering unvisited areas of the state-space. However, the benefit is that large portions of the state-space are quickly labeled, which provides better safety guarantees during learning.

4 Safer Dreaming Variational Autoencoder

This work extends the *Dreaming Varational Autoencoder* (DVAE) from [2, 3] to reduce the risk of entering catastrophic states in mission-critical systems. DVAE is a model-based reinforcement learning approach where the objective is two-fold.

1. The algorithm learns a predictive model of the environment. The goal of the predictive model is to capture the ($T\colon S \times A \rightarrow S$) dynamics between states and to learn the ($R\colon S \times A \rightarrow \mathbb{R}$) reward function.

2. The DVAE algorithm uses the learned predictive model to train model-free algorithms, where the interaction during training is primarily with the predictive model.

Figure 2 demonstrates the algorithm procedure in the following steps. (1) The predictive model observes and learns the real environment dynamics using a sensor model. The same sensor model is the interface that the expert system uses for making actions. (2) The intelligent agent (i.e., reinforcement learning agent) interacts with the predictive model and learns a policy. (3) When the intelligent agent is sufficiently trained, it replaces existing expert systems with comparable performance. (4) The intelligent agent can optionally train further in the real-world environment for improved performance, at the risk of making catastrophic actions.

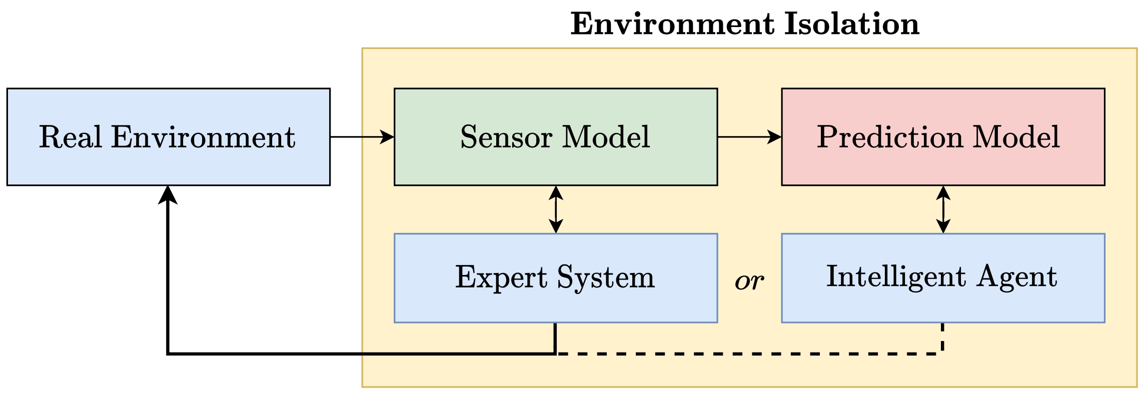


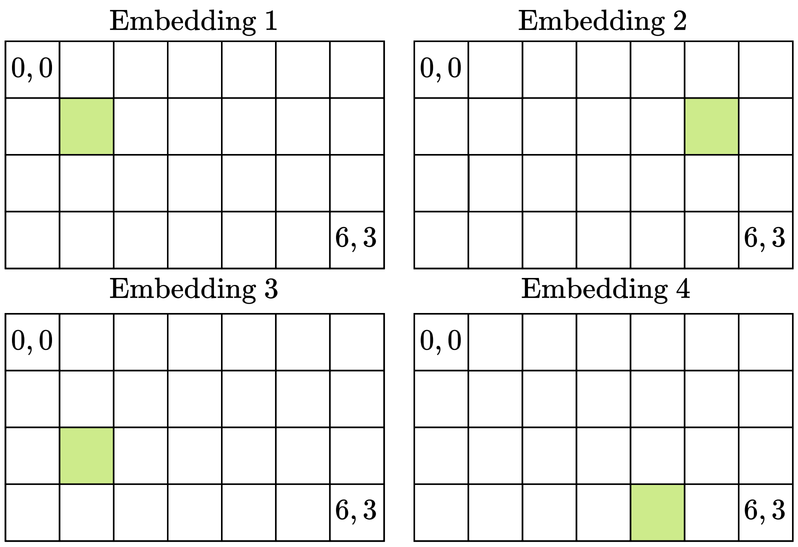
Figure 2: The predictive model of DVAE-S learns the sensor model behavior. The intelligent agent uses the predictive model to train entirely off-line, without the risk of making mistakes in the real environment. After training, the algorithm performs inference in the real environment, with significantly less chance of making actions that lead to catastrophic states.

4.1 Implementation

The DVAE-S algorithm is composed of the following three models: The **V**iew, **R**eason, and **C**ontrol (VRC).

The **view model** is responsible for transforming raw input data into a meaningful and compact feature embedding. DVAE-S uses a variational autoencoder primarily for this task, but there exist alternative methods, such as generative adversarial networks (GAN). It is possible to visualize the embedding $z_x \in \mathbb{Z}$ by manually altering input values. This is especially useful when predicting long horizon futures. Figure 3 illustrates this with a (green) agent in an empty grid-world. The embedding layer consists of two neurons where the first and second neuron learns the vertical and horizontal location, respectively. The **reason model** learns the time dependency between states, or in MDP terms, the transition function $T\colon S \times A \to S$. The reason model computes the future state embedding z^{π}_{t+1} based on a batch of previous embeddings from the view $Z^{\pi}_{t} = \{z_{t-n} \ldots z_t\}^{\pi}$. The policy π denotes the behavior that DVAE-S follows. DVAE-S uses the *long short-term memory* (LSTM) architecture as it was found best learn the transition between states.

The **control model** is responsible for interaction with the environment. The control is the primary model for performing actions that are safe and progress the learning in the right direction. DVAE-S uses primarily Deep Q-Networks (DQN) and Proximal Policy Optimization (PPO) with **constrained optimization criterion** and the **risk-directed exploration** method. The input to the algorithm is a raw-state, commonly a high-dimensional data structure that is difficult to interpret for humans. The benefit of the DVAE-S architecture is that the reason model finds an embedding that is describes states with significantly



Embedding	z_0	z_1
1	0.285	0.285
2	0.857	0.285
3	0.285	0.750
4	0.714	1.000

Figure 3: Four distinct (table) embedding value sets and its (grid-world) decoded output.

fewer dimensions. The DVAE-S algorithm also enables initial training entirely off-line in a *dream* version of the real environment.

Algorithm 1 The DVAE-S algorithm

```
1:  Initialize policy $\pi_\theta(s_t|a_t)$, predictive model $p_\psi(\hat{s}_{t+1}, \hat{r}, h_t|s_t, a_t^\pi)$
2:  Let $Z = \{z_{t-n}^\pi \ldots z_t^\pi\}$, a vector of encoded states
3:  Initialize encoder $ENC(z_t^\pi|s_t, a_t^\pi)$, temporal reasoner $TR(h_t^\pi|Z)$
4:  for N epochs do
5:      $\mathcal{D}_{env} \leftarrow$ Collect samples from $p_{env}$ under predefined policy $\pi$
6:      Train model $p_\psi$ on data batch $D_{env}$ via $MLE$
7:  for M epochs do
8:      Sample initial state $s_0 \sim U(0, 1)$ from $\mathcal{D}_{env}$
9:      Construct $\{\mathcal{D}_{p_\psi}|t < k, TR(h_t^{\pi_\theta}|ENC(z_t|s_t, a_t)^{\pi_\theta}, s_t = s_0)\}$
10:     Update policy $\pi_\theta$ using pairs of $(\hat{s}_t, a_t, \hat{r}_t, \hat{s}_{t+1})^{\pi_\theta}$ discard if $\pi_\theta \notin \Gamma$
```

Algorithm 1 works as follows. **(Line 1)** Initialize the control policy and the predictive model parameters. **(Line 2)** The Z variable denotes a finite set of sequential predictions from the view model (ENC) that captures time dependency between states from the reason model (TR). **(Line 5)** Collect samples from the real environment p_{env} under a predefined policy (i.e, expert system), see Figure 2. **(Line 6)** The predictive model p_ψ trains using the collected data $\mathcal{D}_{env}$. The loss is calculated using maximum likelihood estimation (MLE) or mean squared error $MSE(p_\psi||p_{env})$ until the loss is sufficiently low. **(Line 7)**Training of the control model now starts using the predictive model p_ψ in place of the real environment p_{env}. **(Line 8)** The first state s_0 is sampled from the real dataset $\mathcal{D}_{env}$ and the control policy makes action in the prediction model to create an artificial dataset of simulated states (sampling from the predictive model). **(Line 10)** The parametrized control policy is optimized using $(\hat{s}_t, a_t, \hat{r}_t, \hat{s}_{t+1})^{\pi_\theta \notin \Gamma}$ pairs under the constrained policy where "hat" denotes predicted values.

4.2 Exploration and policy update constraints

There are significant improvements to the exploration and policy update for finding safe policies in DVAE-S. During sampling, the control model uses the *risk-directed exploration* bonus [6]. This bonus is added to the probability distribution over actions before sampling is performed, as described in Section 3. The **policy updates** are constrained to a set of criterion defined as follows. During the learning of the predictive model, feedback is received from the real-world environment. The model assumes that all actions that were not chosen by the agent are considered unsafe. This way, the algorithm gradually maps the unsafe policy space. It is important to note that this mapping does not influence the choices of the agent when learning the predictive model. When the agent revisits a state, the agent may select another action, and this will label the state as safe.

5 Results

The content of this section summarizes experiments with the following contributions. The ASRS-Lab and Deep RTS is introduced as the test environment for the experiments. The proposed method is then applied to Deep Q-Networks [13] , which we refer to as DVAE-S-DQN, and Proximal Policy optimization, which we refer to as DVAE-S-PPO, [16] and tested in three environments, respectively, ASRS-Lab 11x11, ASRS-Lab 30x30, and DeepRTS Gold-Collect. Each experiment is averaged over 100 separate runs. Finally, the predictive model performance and safety performance is discussed with the resulting output from decoded embeddings.

The ASRS Lab[3] is a detailed simulator for research into autonomous agents in grid-warehousing [3]. The simulator's purpose is to create a framework that would efficiently represent a real-world environment without the risk of damaging costly equipment. Cost is perhaps the reason that safety in reinforcement learning is less prevalent than standard model-free approaches. There are many categories of ASRS systems in the real world, and the ASRS Lab environment supports a subset of existing setups. The ASRS-Lab features a shuttle-based, aisle-based, and grid-based warehouse simulation where this work focus on solving the grid-based architecture.

DeepRTS[4] is a lightweight game engine specifically designed for reinforcement learning research [1]. The motivation behind DeepRTS is to set up scenarios that this easily repro-

[3]https://github.com/cair/deep-warehouse
[4]https://github.com/cair/deep-rts

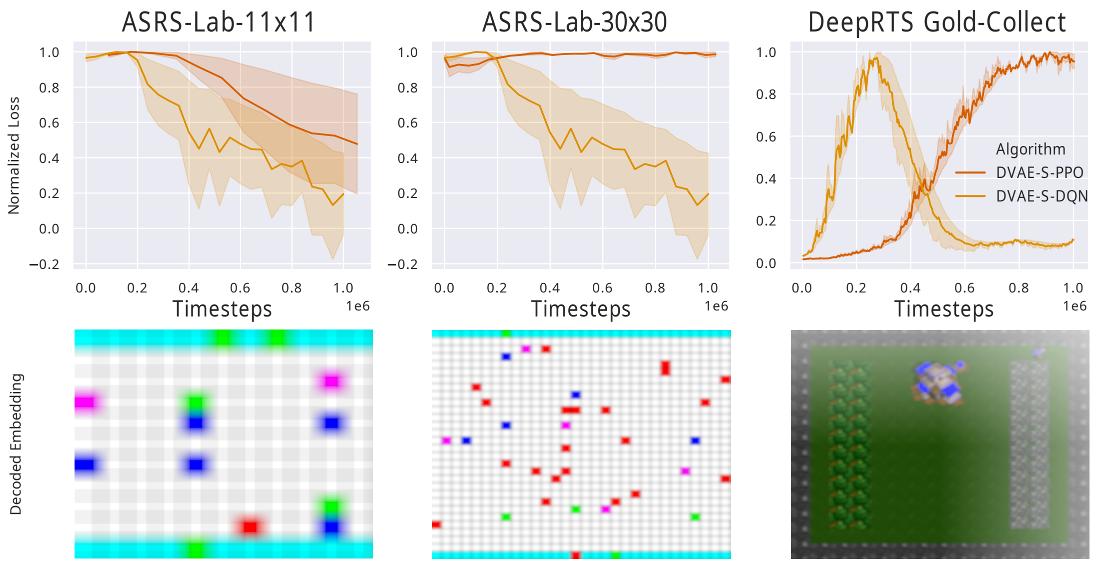


Figure 4: Training loss of the DVAE-S predictive model including the resulting decoded latent-space variables for the DQN variant. The DVAE-S-DQN move towards convergence while DVAE-S-PPO fails to improve. The bottom row illustrates the quality of predicted future states after applying median filtering.

ducible quickly. This paper focuses on solving the "DeepRTS Gold-Collect" environment where the goal is to accumulate as much gold as possible before a predefined timer runs out.

5.1 Predictive Model

The predictive model trains with a learning rate of 0.0003 using Adam optimizer. The observations originate from expert systems, a manhattan-distance agent in ASRS-Lab, and rule-based agent for DeepRTS Gold-Collect[5] Note that the experiments are performed after the safe policy pretraining. Figure 4 shows the training loss for the predictive model for each environment. The second row is the decoded embeddings when predicting future states using the DVAE-S-DQN model. The model achieves this embedding and decoding quality using a learning rate of 0.002, batch size of 16 (previous states) with the adam optimizer. For the DVAE-S-DQN model, the performance gradually decreases for ASRS environments with similar results. The loss in the Deep RTS environment increased in early training but converged after approximately 500000 timesteps. The DVAE-S-PPO method cannot reproduce the DQN variant's performance because of divergence during agent training.

[5]The agent in DeepRTS walks to the nearest gold deposit when the inventory is empty. When inventory is full, it returns the gold to the base.

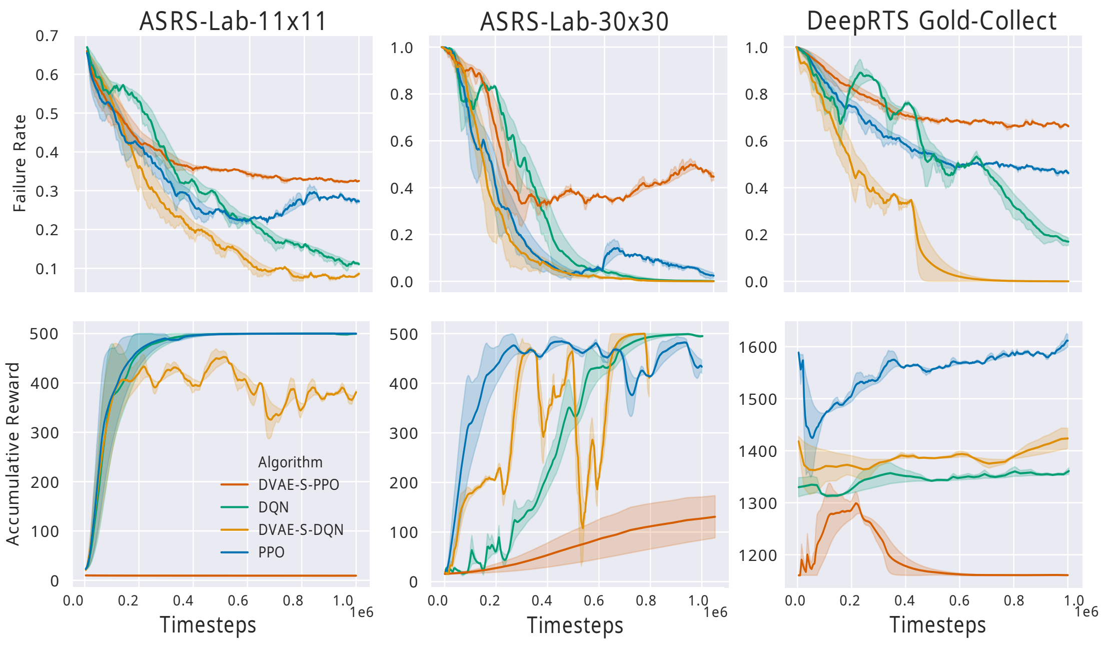


Figure 5: A comparative study of DQN and PPO using DVAE-S. Each plot shows results in an environment, respectively, ASRS-Lab-11x11, ASRS-Lab-30x30, and DeepRTS Gold-Collect. The topmost row shows the failure rate of agents, and the bottom row illustrates the accumulated reward during the experiment. All plots have a duration of 1 million frames. The plots clearly show that DVAE-S-DQN performs comparably to standard DQN while having a significantly lower failure rate. On the contrary, DVAE-S-PPO performs worse than vanilla PPO.

5.2 Failure Rate

The failure-rate experiment measures how often an agent makes actions that lead to catastrophic states in the environment where 1.0 consistently failure and 0 is error-free behavior. The accumulated reward is recorded simultaneously to verify that the agent learns a better behavior. The parameters for risk-directed exploration is w=0.7, and p=0.85, and $\alpha = 0.4$ for all constraints.

Figure 1 shows that DQN agents using the DVAE-S algorithm significantly reduces the frequency of catastrophic states during a training period of 1 000 000 timesteps. The expert systems form the baseline performance of an accumulated reward of 300 in the ASRS-Lab environments and 1300 for the DeepRTS environment. The DVAE-S-DQN performs comparably to vanilla DQN, but the empirical evidence suggests that using constrained criterions and risk-directed exploration reduces the failure rate significantly. The PPO variant did not perform better, and it remains future work to investigate why it poses a challenge to combine direct-policy search algorithms and DVAE-S.

Setting the low performance of DVAE-S-PPO aside, the DQN variant shows impressive results where it performs significantly better than vanilla DQN and above the in the DeepRTS Gold-Collect environment. The performance of DVAE-S-DQN is marginally lower in the ASRS-Lab environment but at the benefit of increased safety, which is the primary concern addressed in this work.

6 Conclusion and Future Work

The results in this paper show that the combination of DVAE, constrained criterion, and risk-directed exploration is a promising approach for improving safety while maintaining comparable behavioral performance. The empirical evidence suggests a strong relationship between control performance and predictive model performance, where good policies lead to good predictive models and the contrary for bad policies.

By using the constrained criterion for policy updates and risk-directed exploration, the DVAE-S architecture significantly improves the safety of learned policy in some algorithms. The DVAE-S-DQN algorithm performs comparably to agents without safety constraints but has a significantly lower risk of entering catastrophic states. The DVAE-S architecture is a novel approach for reinforcement learning in industry-near environments.

This work's continued research is dedicated to better combine proximal policy optimization with DVAE-S for safe reinforcement learning. Temporal convolutional networks look promising for improving the performance of the predictive model. We also hope to experiment with the novel vector quantized variational autoencoders for better embedding high-dimensional states. While this paper contributes new findings in safe reinforcement learning, it is still room for significant improvements, in which we hope to contribute more in the future.

References

[1] Andersen, P.-A., Goodwin, M., Granmo, O.-C.: Deep RTS: A Game Environment for Deep Reinforcement Learning in Real-Time Strategy Games. In: 2018 IEEE Conference on Computational Intelligence and Games (CIG). IEEE (2018). https://doi.org/10.1109/CIG.2018.8490409

[2] Andersen, P.-A., Goodwin, M., Granmo, O.-C.: The Dreaming Variational Autoencoder for Reinforcement Learning Environments. In: LNCS. pp. 143–155. Springer International Publishing (2018). https://doi.org/10.1007/978-3-030-04191-5_11

[3] Andersen, P.-A., Goodwin, M., Granmo, O.-C.: Towards Model-Based Reinforcement Learning for Industry-Near Environments. In: LNCS. pp. 36–49. Springer International Publishing (2019). https://doi.org/10.1007/978-3-030-34885-4_3.

[4] Arulkumaran, K., Deisenroth, M.P., Brundage, M., Bharath, A.A.: Deep Reinforcement Learning: A Brief Survey. IEEE Signal Processing Magazine. 34, 26–38 (2017). https://doi.org/10.1109/MSP.2017.2743240

[5] Berkenkamp, F., Turchetta, M., Schoellig, A.P., Krause, A.: Safe Model-based Reinforcement Learning with Stability Guarantees. Advances in Neural Information Processing Systems 30, pp. 908–918. Curran Associates, Inc., CA, USA (2017)

[6] Edith, L.L., Melanie, C., Doina, P., Bohdana, R.: Risk-directed Exploration in Reinforcement Learning. IJCAI (2005)

[7] Fox, R., Pakman, A., Tishby, N.: Taming the Noise in Reinforcement Learning via Soft Updates. In: Proc. 32nd Conference on Uncertainty in Artificial Intelligence. pp. 202–211. UAI'16, AUAI Press, Arlington, Virginia, USA (2016). https://doi.org/10.5555/3020948.3020970

[8] Gaskett, C.: Reinforcement learning under circumstances beyond its control. Proc. of the Int. conference on computational intelligence for modelling control and automation (2003), `http://www.his.atr.co.jp/cgaskett/`

[9] Geibel, P., Wysotzki, F.: Risk-Sensitive Reinforcement Learning Applied to Control under Constraints. JAIR **24**, 81–108 (jul 2005). https://doi.org/10.1613/jair.1666

[10] Ha, D., Schmidhuber, J.: Recurrent World Models Facilitate Policy Evolution. Advances in Neural Information Processing Systems 31, pp. 2450–2462. Cur-

ran Associates, Inc., Montréal, CA (sep 2018), https://arxiv.org/abs/1809.01999

[11] Heger, M.: Consideration of Risk in Reinforcement Learning. Proc. 11th Int Conference on Machine Learning, ICML'94. pp. 105–111. Elsevier, USA (1994). https://doi.org/10.1016/B978-1-55860-335-6.50021-0

[12] Levine, S., Finn, C., Darrell, T., Abbeel, P.: End-to-end training of deep visuomotor policies. Journal of Machine Learning Research **17**(1), 1334–1373 (2016)

[13] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing Atari with Deep Reinforcement Learning. Neural Information Processing Systems (dec 2013), http://arxiv.org/abs/1312.5602

[14] Moldovan, T.M., Abbeel, P.: Safe Exploration in Markov Decision Processes. Proc. of the 29th Int. Conference on Machine Learning (2012)

[15] Santana, E., Hotz, G.: Learning a Driving Simulator. arxiv preprint arXiv:1608.01230 (2016)

[16] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal Policy Optimization Algorithms. arxiv preprint arXiv:1707.06347 (2017),

[17] Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. A Bradford Book, MA, USA, 2 edn. (2018), https://dl.acm.org/doi/book/10.5555/3312046

[18] Wang, Y., He, H., Tan, X.: Robust Reinforcement Learning in POMDPs with Incomplete and Noisy Observations (2019)

[19] Xiao, T., Kesineni, G.: Generative Adversarial Networks for Model Based Reinforcement Learning with Tree Search. Tech. rep., University of California, Berkeley (2016), http://tedxiao.me/pdf/gans{~}drl.pdf

[20] Zhang, C., Patras, P., Haddadi, H.: Deep Learning in Mobile and Wireless Networking: A Survey. IEEE Communications Surveys & Tutorials **21**(3), 2224–2287 (2019). https://doi.org/10.1109/COMST.2019.2904897

Paper I: Safer Reinforcement Learning for Agents in Industrial Grid-Warehousing

Paper J

Title:	Interpretable Option Discovery using Deep Q-Learning and Variational Autoencoders
Authors:	Andersen, Per-Arne et al.
Affiliation:	Department of ICT, University of Agder, Grimstad Norway
Conference:	3rd International Conference on Intelligent Technologies and Applications
Year:	2020, September

J

Interpretable Option Discovery using Deep Q-Learning and Variational Autoencoders

Per-Arne Andersen
Department of ICT
University of Agder
Grimstad, Norway
per.andersen@uia.no

Morten Goodwin
Department of ICT
University of Agder
Grimstad, Norway
morten.goodwin@uia.no

Ole-Christoffer Granmo
Department of ICT
University of Agder
Grimstad, Norway
ole.granmon@uia.no

Abstract

Deep Reinforcement Learning (RL) is unquestionably a robust framework to train autonomous agents in a wide variety of disciplines. However, traditional deep and shallow model-free RL algorithms suffer from low sample efficiency and inadequate generalization for sparse state spaces. The options framework with temporal abstractions [18] is perhaps the most promising method to solve these problems, but it still has noticeable shortcomings. It only guarantees local convergence, and it is challenging to automate initiation and termination conditions, which in practice are commonly hand-crafted.

Our proposal, the Deep Variational Q-Network (DVQN), combines deep generative- and reinforcement learning. The algorithm finds good policies from a Gaussian distributed latent-space, which is especially useful for defining options. The DVQN algorithm uses MSE with KL-divergence as regularization, combined with traditional Q-Learning updates. The algorithm learns a latent-space that represents good policies with state clusters for options. We show that the DVQN algorithm is a promising approach for identifying initiation and termination conditions for option-based reinforcement learning. Experiments show that the DVQN algorithm, with automatic initiation and termination, has comparable performance to Rainbow and can maintain stability when trained for extended periods after convergence.

3rd International Conference on Intelligent Technologies and Applications (INTAP-2020)
Submitted 11. Mar, 2020, Accepted 22. Jun 2020, Published 15. Mar, 2021

Keywords: Deep Reinforcement Learning, Clustering, Options, Hierarchical Reinforcement Learning, Latent-space representation

1 Introduction

The interest in deep Reinforcement Learning (RL) is rapidly growing due to significant progress in several RL problems [2]. Deep RL has shown excellent abilities in a wide variety of domains, such as video games, robotics and, natural language progressing [16, 14, 13]. Current trends in applied RL has been to treat neural networks as black-boxes without regard for the latent-space structure. While unorganized latent-vectors are acceptable for model-free RL, it is disadvantageous for schemes such as options-based RL. In an option-based RL, the policy splits into sub-policies that perform individual behaviors based on the current state of the agent. A sub-policy, or option, is selected with initialization criteria and ended with a termination signal. The current state-of-the-art in option-based RL primarily uses hand-crafted options. Option-based RL algorithms work well for simple environments but have poor performance in more complicated tasks. There is, to the best of our knowledge, no literature that addresses good option selection for difficult control tasks. There are efforts for making automatic options selection [17], but no method achieves notable performance across various environments.

This paper proposes a novel deep learning architecture for Q-learning using variational autoencoders that learn to organize similar states in a vast latent-space. The algorithm derives good policies from a latent-space that feature interpretability and the ability to classify sub-spaces for automatic option generation. Furthermore, we can produce human-interpretable visual representations from latent-space that directly reflects the state-space structure. We call this architecture DVQN for deep Variational Q-Networks and study the learned latent-space on classic RL problems from the Open AI gym [4].

The paper is organized as follows. Section 3 introduces preliminary literature for the proposed algorithm. Section 4 presents the proposed algorithm architecture. Section 5 outlines the experiment setup and presents empirical evidence of the algorithm performance. Section 2 briefly surveys work that is similar to our contribution. Finally, Section 6 summarises the work of this paper and outlines a roadmap for future work.

2 Related Work

There are numerous attempts in the literature to improve interpretability with deep learning algorithms, but primarily in the supervised cases. [22] provides an in-depth survey of interpretability with Convolutional Neural Networks (CNNs). Our approach is similar to the work of [20], where the authors propose an architecture for visual perception of the

DQN algorithm. The difference, however, is primarily our focus on the interpretability of the latent-space distribution via methods commonly found in variational autoencoders. There are similar efforts to combine Q-Learning with Variational Autoencoders, such as [19, 11], and shows promising results theoretically but with limited focus on interpretability. [1] did notable work on interpretability among using a distance KL-distance for optimization but did not find convincing evidence for a deeper understanding of the model. The focus of our contribution deviates here and finds significant value in a shallow and organized latent-space.

Options The learned latent-space is valuable for the selection of options in hierarchical reinforcement learning (HRL). There is increasing engagement in HRL research because of several appealing benefits such as sample efficiency and model simplicity [3]. Despite its growing attention, there are few advancements within this field compared to model-free RL. The options framework [18] is perhaps the most promising approach for HRL in terms of intuitive and convergence guarantees. Specifically, the options framework defines semi-Markov decision processes (SMDP), which is an extension of the traditional MDP framework [21]. SMDP features temporal abstractions where multiple discrete time steps are generalized to a single step. These abstract steps are what defines an option, where the option is a subset of the state-space. In the proposed algorithm, the structure of the latent-space forms such temporal abstractions for options to form.

3 Background

The algorithm is formalized under conventional Markov decision processes tuples $< S, A, P, R, \gamma >$ where S is a (finite) set of all possible states, A is a (finite) set of all possible actions, P defines the probabilistic transition function $P(S_{t+1} = s'|s, a)$ where s is the previous state, and s' is the transition state. R is the reward function $R(r_{t+1}|s, a)$. Finally, the γ is a discount factor between $\gamma \in [0 \ldots 1]$ that determines the importance of future states. Lower γ values decrease future state importance while higher values increase.

4 Deep Variational Q-Networks

Our contribution is a deep Q-learning algorithm that finds good policies in an organized latent space from variational autoencoders. [1] Empirically, the algorithm shows comparable performance to traditional model-free deep Q-Networks variants. We name our method the **D**eep **V**ariational **Q-N**etwork (DVQN) that combines two emerging algorithms, the variational autoencoder (VAE) [12] and deep Q-Networks (DQN) [14].

[1]The code will be published upon publication.

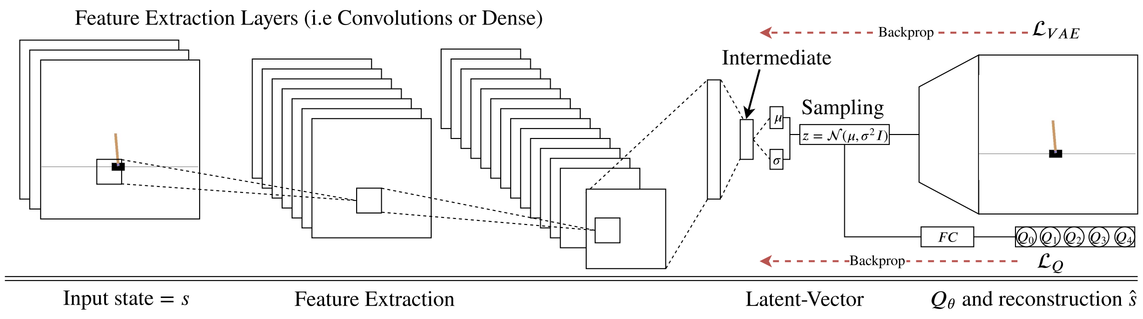

Feature Extraction Layers (i.e Convolutions or Dense)
Backprop
$\mathcal{L}_{VAE}$
Intermediate
Sampling
μ
σ
$z = \mathcal{N}(\mu, \sigma^2 I)$
FC
Q_0 Q_1 Q_2 Q_3 Q_4
Backprop
$\mathcal{L}_Q$
Input state = s
Feature Extraction
Latent-Vector
Q_θ and reconstruction $\hat{s}$

Figure 1: The deep variational Q-Networks architecture.

In traditional deep Q-Networks, the (latent-space) hidden layers are treated as a black-box. On the contrary, the objective of the variational autoencoder is to reconstruct the input and **organize** the latent-vector so that similar (data) states are adjacently modeled as a Gaussian distribution.

In DQN, the latent-space is sparse and is hard to interpret for humans and even option-based machines. By introducing a VAE mechanism into the algorithm, we expect far better interpretability for creating options in RL, which is the primary motivation for this contribution. Variational autoencoders are, in contrast to deep RL, involved with the organization of the latent-space representation, and commonly used to generate clusters of similar data with t-SNE or PCA [23]. The DVQN algorithm introduces three significant properties. First, the algorithm fits the data as a Gaussian distribution. This reduces the policy-space, which in practice reduces the probability of the policy drifting away from global minima. Second, the algorithm is generative and does not require exploration schemes such as ϵ-greedy because it is done in re-parametrization during training. Third, the algorithm can learn the transition function and, if desirable, generate training data directly from the latent-space parameters, similar to the work of [8].

Figure 1 illustrates the architecture of the algorithm. The architecture follows general trends in similar RL literature but has notable contributions. First, features are extracted from the state-input, typically by using convolutions for raw images and fully-connected for vectorized input. The extracted features are forwarded to a fully connected intermediate layer of a user-specified size commonly between 50 to 512 neurons. The intermediate layer splits into two streams that represent the variance μ and standard deviation σ and is used to sample the latent-vector using a Gaussian distribution through the re-parameterization. The latent-vector is forwarded to the decoder for state reconstruction and the Q-Learning stream for action-value (Q-value) optimization. The decoder and Q-Learning streams have the following loss functions:

$$\mathcal{L}_{VAE} = MSE(s, \hat{s}) + D_{KL}[q_\psi(z|s) || p_\theta(z|s)] \tag{1}$$

$$\mathcal{L}_{DQN} = (r + \gamma Q(s', \arg\max_{a'} Q(s', a'; \theta_i); \theta_i) - Q(s, a; \theta_i))^2 \tag{2}$$

$$\mathcal{L}_{DVQN} = c_1 \mathbb{E}_{\sim q_\psi(z|s)}[\mathcal{L}_{VAE}] + c_2 \mathbb{E}_{s,a,s',D\sim r}[\mathcal{L}_{DQN}]. \tag{3}$$

The global loss function $\mathcal{L}_{DVQN}$ is composed of two local objectives: $\mathcal{L}_{DQN}$ and $\mathcal{L}_{VAE}$. In the VAE loss, the first term is the mean squared error between the input s and its reconstruction $\hat{s}$. The second term is regularization using KL-distance to minimize the distance between the latent distribution and a Gaussian distribution. The DQN loss is a traditional deep Q-Learning update, as described in [14].

Algorithm 1 DVQN: Minimal Implementation

```
1: Initialise $\Omega$
2: Initialise DVQN model $\pi$
3: Initialise replay-buffer $D_\pi$
4: for N episodes do
5:     $D_\pi \leftarrow$ Collect samples from $\Omega$ under the untrained policy $\pi$ via the generative
       policy sampling.
6:     Train model $\pi$ on a mini-batch from $D_\pi$ with objective from Equation 3
```

Algorithm 1 shows a general overview of the algorithm. First, the environment is initialized. Second, the DVQN model from Figure 1 is initialized with the desired hyperparameters, and third, the replay-buffer is created. For a specified number of episodes, the algorithm samples actions from the generative policy for exploration and stores these as MDP tuples in the experience replay. After each episode, the algorithm samples mini-batches from the experience replay and performs parameter updates using stochastic gradient descent. The (loss function) optimization objective is described in equation 3. The process repeats until the algorithm converges.

5 Experiments and Results

In this section, we conduct experiments against four traditional environments to demonstrate the effectiveness of the DVQN algorithm. We show that the algorithm can organize the latent-space by state similarity while maintaining comparable performance to model-free deep Q-learning algorithms.

5.1 Experiment test-bed

We evaluate the DVQN in the following environments; CartPole-v0, Acrobot-v1, CrossingS9N3-v0, and FourRooms-v0, shown in Figure 2. These environments are trivial to solve using model-free reinforcement learning and hence, excellent for visualizing

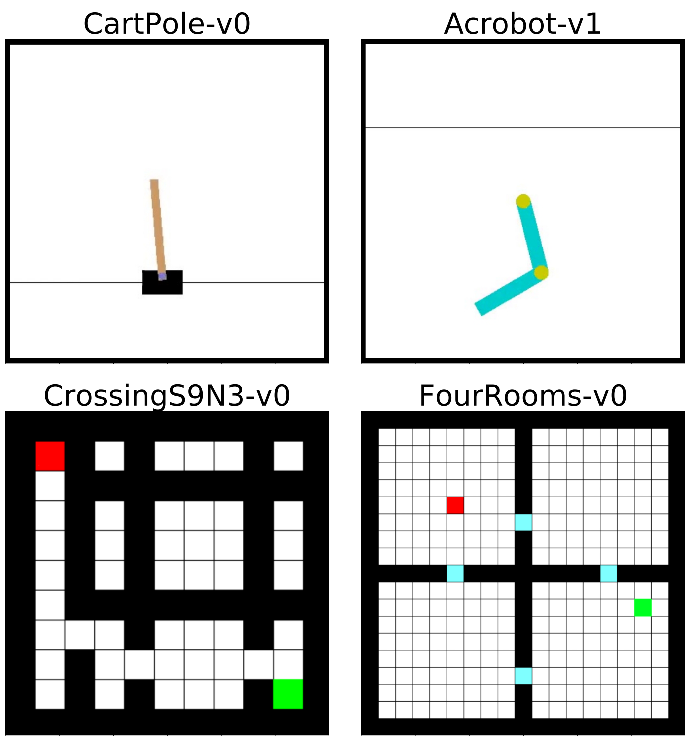


Figure 2: The experiment test-bed contains the following environments; CartPole-v0, Acrobot-v1, CrossingS9N3-v0, and FourRooms-v0

the learned latent-space. The FourRooms-v0 environment is especially suited for option-based RL and can solve the problem in a fraction of time steps compared to model-free RL. Although the DVQN algorithm **does not quantify options for analysis** (see Section 6), the primary goal is to organize the latent-space so that it is possible to extract meaningful and interpretable options automatically. The aim is to have **comparable performance** to vanilla deep Q-Network variants found in the literature [14, 5, 10]. DVQN benchmarks against vanilla DQN, Double DQN, and Rainbow.

FourRooms-v0 and CrossingS9N3-v0 are a grid-world environment where the objective is to reach the terminal-state cell (In the lower right of the image in both environments). In FourRooms-v0, the agent has to enter several doors and only complete a part of the goal for each door it enters. FourRooms-v0 is an ideal environment for option-based reinforcement-learning because each door is considered a sub-goal. While the environment is solvable by many deep reinforcement learning algorithms, option-based RL is more efficient. The agent receives small negative rewards for moving and positive rewards for entering the goal-state (global) or the doors (local). The Crossing is a simpler environment where the agent has to learn the shortest path to the goal state. In both grid-environments, the agent can move in any direction, one cell per time step.

To further show that the algorithm works in simple control tasks, we perform experiments in CartPole-v0 and Acrobot-v1. The objective in CartPole-v0 is to balance a pole on a cart.

Algorithm	**DQN**	**DDQN**	**Rainbow**	**DVQN (ours)**
Optimiser	Adam			RMSProp
Learning Rate	0.003			0.000025
Activation	ReLU			ELU
Batch Size	32			128
Replay Memory	1m			
Epsilon Start	1.0			N/A
Epsilon End	0.01			N/A
Epsilon Decay	0.001 (Linear)			N/A
Gamma	0.95			
Q-Loss	Huber			MSE

Table 1: Algorithm and hyperparameters used in the experiments. For the Rainbow algorithm, we used the same hyperparameters described in [10]. The DDQN had target weight updates every 32k frames.

Each step generates a positive reward signal while receiving negative rewards if the pole falls below an angle threshold. The agent can control the direction of the cart at every time step. The Acrobot-v1 has a similar aim to control the arm to hit the ceiling in a minimal number of time steps. The agent receives negative rewards until it reaches the ceiling. The CartPole-v0 and Acrobot-v1 environments origins from [4] while CrossingS9N3-v0 and the FourRooms-v0 origins from [6].[2]

5.2 Hyperparameters

During the experiments, we found DVQN to be challenging to tune. Initially, the algorithm used ReLU as activation but was discarded due to vanishing gradients resulting in divergence for both policy and reconstruction objectives. By using ELU, we found the algorithm to be significantly more stable during the experiments, and it additionally did not diverge if training continued after convergence. We will explore the underlying cause of our future work. Table 1 shows the hyperparameters used in our experiments where most of the parameters are adopted from prior work. Recognize that the DVQN algorithm does not use ϵ-greedy methods for exploration. The reason for this is that random sampling is done during training in the variational autoencoder part of the architecture. In general, the algorithm tuning works well across all of the tested domains, and better results can likely be achieved with extended hyperparameter searches. For DVQN, we consider such tuning for our continued work on DVQN using options, see Section 6.

[2]A community-based scoreboard can be found at `https://github.com/openai/gym/wiki/Leaderboard`.

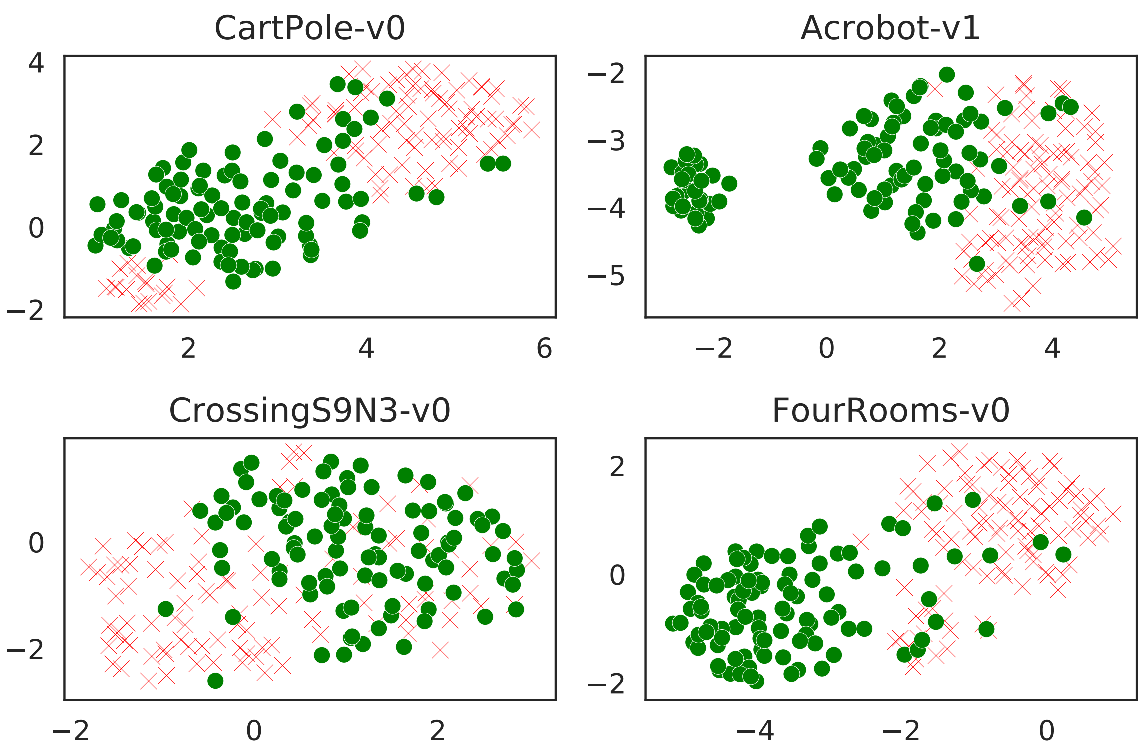


Figure 3: The learned latent space for all of the tested environments. DVQN successfully trivialise the selection of options as seen in the well-separated state clusters. The circular points illustrate states with positive reward while cross illustrates negative rewards.

5.3 Latent-Space evaluation

An attractive property of our model is that the latent-space is a Gaussian distribution. As seen in Figure 3, the DVQN algorithm can produce clustered latent-spaces for all tested environments. For example, in the CartPole-v0 environment, there are three clusters where two of them represent possible terminal states and one that represents states that give a reward. To fully utilize the capabilities of DVQN, the latent-space can be used to generate options for each cluster to promote different behavior for every region of the state-space.

Figure 4 illustrates the visualization of the latent-space representation in CartPole-v0. We find that each cluster represents a specific position and angle of the pole. The latent-space interpolates between these state variations, which explains its shape. Although the clusters are not perfect, it is trivial to construct separate classification for each cluster with high precision, and this way automatically construct initiation and termination signals for options.

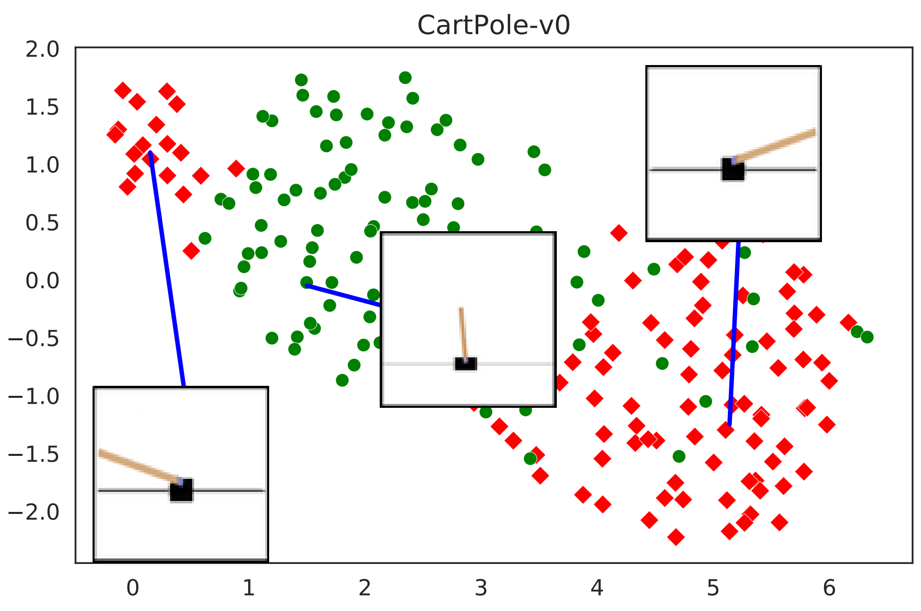


Figure 4: The relationship between states and the latent-space for the CartPole-v0 environment. DVQN can separate each angle, left, middle and right into separable clusters, which are especially useful in option-based reinforcement learning. Additionally, the visualization of the latent space that the Q-head uses to sample actions is trivial to interpret.

5.4 Performance evaluation

Figure 5 illustrates a comparison of performance between state-of-the-art Q-Learning algorithms and the proposed DVQN algorithm. The performance measurement is the mean of 100 trials over 1500 episodes for CartPole-v0,
CrossingS9N3-v0, FourRooms-v0, and 3000 episodes for Acrobot-v1. The performance is measured in accumulated rewards and is therefore negative for environments where each time step yields a negative reward.

The DVQN algorithm performs better than DQN and shows comparable performance to DDQN and Rainbow. DVQN is not able to find a good policy in the Acrobot-v1 environment but successfully learns a good visual representation of the latent space. In general, the DVQN algorithm is significantly harder to train because it requires the algorithm to find a good policy within a Gaussian distribution. We found this to work well in most cases, but it required fine-tuning of hyperparameters. The algorithm is also slower to converge, but we were able to improve training stability by increasing the batch-size and decreasing the learning-rate.

6 Conclusion and Future Work

This paper introduces the deep variational Q-network (DVQN), a novel algorithm for learning policies from a generative latent-space distribution. The learned latent-space is

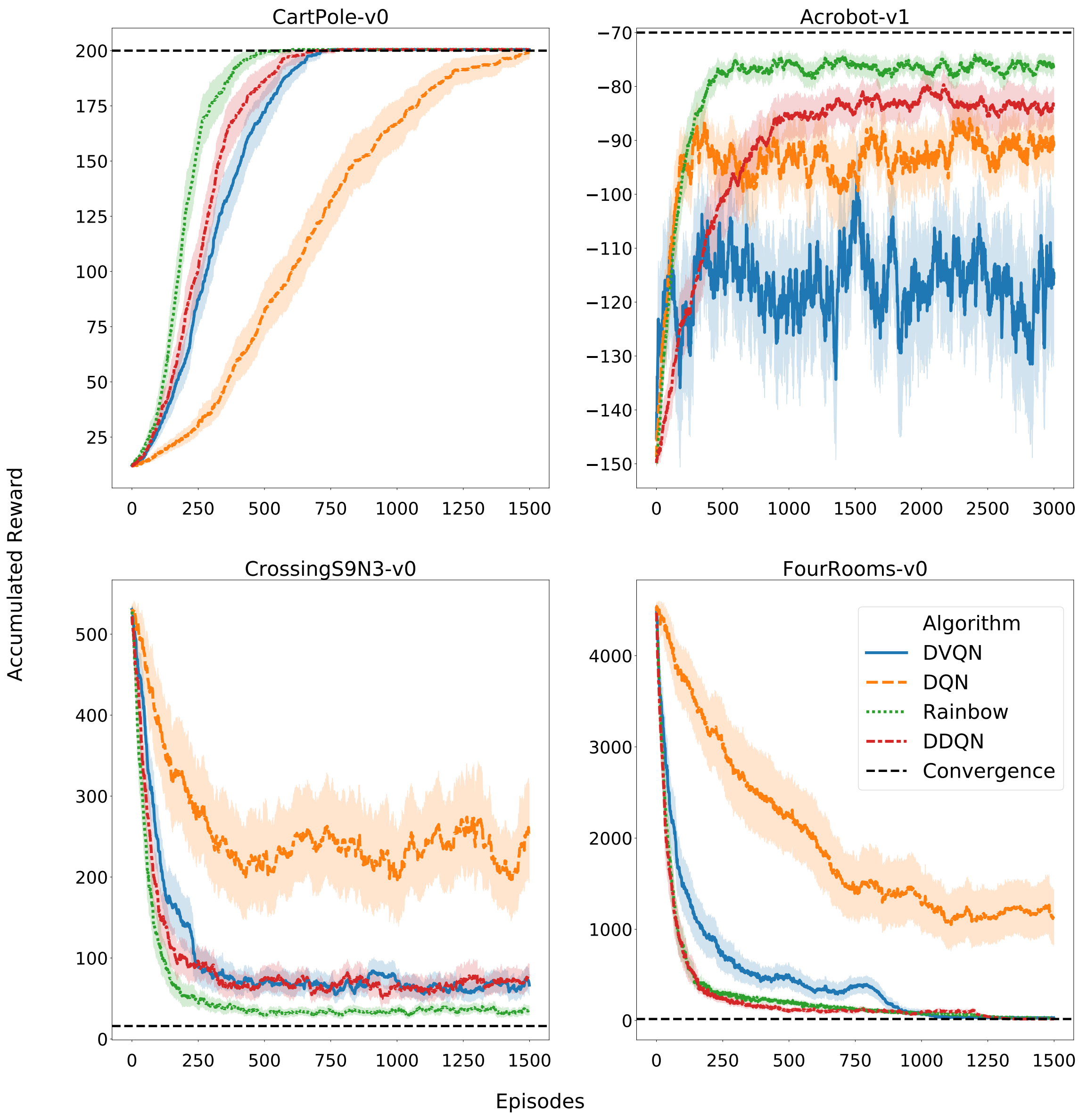


Figure 5: The accumulative sum of rewards of the DVQN compared to other Q-Learning based methods in the experimental environments. Our algorithm performs better than DQN from [14], and shows comparable results to DDQN from [9] and Rainbow from [10]. We define an episode threshold for each of the environments (x-axis) and accumulate the agent rewards as the performance metric for CartPole-v0 and Acrobot-v1. The scoring metric in CrossingS9N3-v0 and the FourRooms-v0 is based on how many steps the agent used to reach the goal state.

particularly useful for clustering states that are close to each other for **discovering options automatically**. In the tested environments, the DVQN algorithm can achieve **comparable performance** to traditional deep Q-networks. DVQN does not provide the same training stability and is significantly harder to fine-tune than traditional deep Q-learning algorithms. For instance, network capacity is increased. As a result of this, the algorithm takes longer to train, and during the experiments, only the RMSprop optimizer [15] with a small step size was able to provide convergence. Additionally, the exponential linear units from [7] had a positive effect on stability. On the positive side, the DVQN contributes a **novel approach for options discovery** in hierarchical reinforcement learning algorithms.

The combination of VAE and reinforcement learning algorithms has interesting properties. Under **optimal conditions**, the latent-space should, in most cases follow a true Gaussian distribution where policy evaluations always provide optimal state-action values, since this is the built-in properties of the latent space in any VAE. The difference between traditional deep Q-networks and DVQN primarily lies in the elimination of a sparse and unstructured latent-space. In deep Q-Networks, optimization does not provide a latent-space structure that reflects a short distance between states but rather a distance between Q-values [14]. By using KL-regularization from VAE, low state-to-state is encouraged. Another benefit of VAE is that we sample from a Gaussian distribution to learn μ and σ, which is especially satisfying for algorithms with off-policy sampling and therefore eliminates the need for (ϵ-greedy) random sampling.

In the continued work, we wish to do a thorough analysis of the algorithm to justify its behavior and properties better. A better understanding of the Gaussian distributed latent-space is particularly appealing because it would enable better labeling schemes for clustering, or perhaps fully automated labeling. Finally, we plan to extend the algorithm from model-free behavior to hierarchical RL with options. The work of this contribution shows that it is feasible to produce organized latent-spaces that could provide meaningful options, and the hope is that this will result in state-of-the-art performance in a variety of tasks in RL.

References

[1] Annasamy, R.M., Sycara, K.: Towards Better Interpretability in Deep Q-Networks. Proceedings, The Thirty-Third AAAI Conference on Artificial Intelligence (sep 2018), `http://arxiv.org/abs/1809.05630`

[2] Arulkumaran, K., Deisenroth, M.P., Brundage, M., Bharath, A.A.: Deep reinforcement learning: A brief survey. IEEE Signal Processing Magazine **34**(6), 26–38 (2017). https://doi.org/10.1109/MSP.2017.2743240

[3] Barto, A., Mahadevan, S., Lazaric, A.: Recent Advances in Hierarchical Reinforcement Learning. Tech. rep., PIGML Seminar-AirLab (2003)

[4] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: OpenAI Gym. arxiv preprint arXiv:1606.01540 (jun 2016), `http://arxiv.org/abs/1606.01540`

[5] Chen, W., Zhang, M., Zhang, Y., Duan, X.: Exploiting meta features for dependency parsing and part-of-speech tagging. Artificial Intelligence **230**, 173–191 (sep 2016). https://doi.org/10.1016/j.artint.2015.09.002, `http://arxiv.org/abs/1509.06461`

[6] Chevalier-Boisvert, M., Willems, L., Pal, S.: Minimalistic Gridworld Environment for OpenAI Gym. \url{https://github.com/maximecb/gym-minigrid} (2018)

[7] Clevert, D.A., Unterthiner, T., Hochreiter, S.: Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). The International Conference on Learning Representations 16 (nov 2015), `http://arxiv.org/abs/1511.07289`

[8] Ha, D., Schmidhuber, J.: Recurrent World Models Facilitate Policy Evolution. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31, pp. 2450–2462. Curran Associates, Inc., Montréal, CA (sep 2018), `http://papers.nips.cc/paper/7512-recurrent-world-models-facilitate-policy-evolution.pdf`

[9] van Hasselt, H., Guez, A., Silver, D.: Deep Reinforcement Learning with Double Q-learning. Proceedings, The Thirtieth AAAI Conference on Artificial Intelligence p. 13 (sep 2015), `http://arxiv.org/abs/1509.06461`

[10] Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., Silver, D.: Rainbow: Combining Improvements in Deep Reinforcement Learning. In: Proc. 32nd Conference on Artificial Intelligence, AAAI'18. pp. 3215–3222. AAAI Press, New Orleans, Louisiana USA (oct 2018), `https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/download/17204/16680`

[11] Huang, S., Su, H., Zhu, J., Chen, T.: SVQN: Sequential Variational Soft Q-Learning Networks. In: International Conference on Learning Representations (2020), `https://openreview.net/forum?id=r1xPh2VtPB`

[12] Kingma, D.P., Welling, M.: Auto-Encoding Variational Bayes. Proceedings of the 2nd International Conference on Learning Representations (dec 2013). https://doi.org/10.1051/0004-6361/201527329, http://arxiv.org/abs/1312.6114

[13] Levine, S., Finn, C., Darrell, T., Abbeel, P.: End-to-end training of deep visuomotor policies. Journal of Machine Learning Research **17**(1), 1334–1373 (2016), http://www.jmlr.org/papers/volume17/15-522/15-522.pdf

[14] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning (dec 2015). https://doi.org/10.1038/nature14236, http://arxiv.org/abs/1312.5602

[15] Nair, V., Hinton, G.E.: Rectified Linear Units Improve Restricted Boltzmann Machines. Proceedings of the 27 th International Conference on Machine Learning p. 8 (2010)

[16] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., Van Den Driessche, G., Graepel, T., Hassabis, D.: Mastering the game of Go without human knowledge. Nature (2017). https://doi.org/10.1038/nature24270

[17] Stolle, M.: Automated Discovery of Options in Reinforcement Learning. Ph.D. thesis, McGill University (2004)

[18] Sutton, R.S., Precup, D., Singh, S.: Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. Artificial Intelligence **112**(1-2), 181–211 (1999)

[19] Tang, Y., Kucukelbir, A.: Variational Deep Q Network. In: Advances in Neural Information Processing Systems 30. Long Beach, CA, USA (nov 2017), http://arxiv.org/abs/1711.11225

[20] Wang, J., Gou, L., Shen, H.W., Yang, H.: DQNViz: A Visual Analytics Approach to Understand Deep Q-Networks. IEEE Transactions on Visualization and Computer Graphics **25**(1), 288–298 (jan 2019). https://doi.org/10.1109/TVCG.2018.2864504

[21] Younes, H.L.S., Simmons, R.G.: Solving Generalized Semi-Markov Decision Processes using Continuous Phase-Type Distributions. Proceedings, The Ninth AAAI Conference on Artificial Intelligence (2004), www.aaai.org

[22] Zhang, C., Patras, P., Haddadi, H.: Deep Learning in Mobile and Wireless Networking: A Survey. IEEE Communications Surveys & Tutorials (mar 2018), http://arxiv.org/abs/1803.04311

[23] Zheng, Y., Tan, H., Tang, B., Zhou, H., Jiang, Z., Zheng, Y., Tan, H., Tang, B., Zhou, H.: Variational Deep Embedding: A Generative Approach to Clustering. International Joint Conference on Artificial Intelligence 17 p. 8 (2017), http://arxiv.org/abs/1611.05148

Paper J: Interpretable Option Discovery using Deep Q-Learning and Variational Autoencoders

J

Paper K

Title:	ORACLE: End-to-End Model Based Reinforcement Learning
Authors:	Andersen, Per-Arne et al.
Affiliation:	Department of ICT, University of Agder, Grimstad Norway
Conference:	41th SGAI International Conference on Artificial Intelligence
Year:	2021

ORACLE: End-to-end Model Based Reinforcement Learning

Per-Arne Andersen
Department of ICT
University of Agder
Grimstad, Norway
per.andersen@uia.no

Morten Goodwin
Department of ICT
University of Agder
Grimstad, Norway
morten.goodwin@uia.no

Ole-Christoffer Granmo
Department of ICT
University of Agder
Grimstad, Norway
ole.granmon@uia.no

Abstract

Reinforcement Learning (RL) algorithms seek to maximize some notion of reward. There are two categories of RL agents, model-based or model-free agents. In the case of model-free learning, the algorithm learns through trial and error in the target environment in contrast to model-based where the agent train in a learned or known environment instead.

Model-free reinforcement learning shows promising results in simulated environments but falls short in the case of real-world environments. This is because trial and error do not fit with the reality where errors are related to an economic burden. On the other hand, Model-based reinforcement learning (MBRL) aims to exploit a known or learned dynamics model, which substantially increases sample efficiency. This paper focuses on learning a dynamics model and use the learned model to train several model-free algorithms by directly sampling the dynamics model. However, it is challenging to achieve good accuracy on dynamics models for highly complex domains due to stochasticity and compounding noise in the system. A majority of model-based RL focuses on dynamics models that derive policies from observation space. Deriving policies from observation space is problematic because it is often high dimensional with significant complexity.

Forty-first SGAI International Conference on Artificial Intelligence (AI-2021), Cambridge England
Submitted 16. Aug, 2021, Accepted 29. Sep 2021, Published. 2021

This paper proposes an end-to-end model-based reinforcement learning algorithm for learning model-free algorithms to act in environments without trial and error in the real environment. This method is beneficial for existing installations that employ existing decision-making systems, such as an expert system. The proposed algorithm has the same fundamental learning principles as the Dreaming Variational Autoencoder but is substantially different architecturally. We show that the algorithm is more sample efficient and performs comparably with existing model-free approaches. We also demonstrate how the algorithm is actor agnostic, enabling existing model-free algorithms to operate in a model-based context.

Keywords: Reinforcement Learning, Markov Decision Processes, Neural Networks, State space models, Model-based Reinforcement Learning

1 Introduction

Reinforcement Learning (RL) continues to be fruitful in many applications in recent literature. To mention a few, we have seen tremendous progress in learning computers to perform many tasks, such as complex strategies in games via self-play learning [25], autonomous driving [20], health care applications such as sequential decision making in tumor classification [30], and in industry decision making and efficiency optimization [8]. It is clear that RL has a significance in the present time, but also in going forward to understand and define artificial intelligence.

While reinforcement learning shows promise, it is still far from achieving super-intelligence and lacks sufficient generalization capabilities for mass adoption in disciplines such as industry. The various existing reinforcement learning algorithms suffers from catastrophic forgetting [29], Low sample efficiency [3], Few safety guarantees on decisions [11], and requires extensive hyper-parameter tuning [10] for optimal performance. On the bright side, RL algorithms often only have one of these negative traits making it possible to circumvent these issues altogether by choosing the correct algorithm for the correct task.

Reinforcement learning separates into two main categories, Model-based and Model-Free Reinforcement Learning. In Model-free RL, the goal is to derive a policy from samples, either in an on-policy or off-policy manner. Policy Gradients is an on-policy method and learns directly using samples collected from its policy. On-policy methods are regarded as less sample efficient and stable but can often reach higher convergence targets. On the other hand, off-policy algorithms use historic data sampled from previous policy snapshots or Monte-Carlo sampling. The benefit of reusing more distant experiences to prevent catastrophic forgetting and is widely accepted as more sample-efficient. On the other hand, Model-based reinforcement learning (MBRL) aims to exploit a known or learned dynamics model, which drastically increases sample efficiency and policy stability [21].

This paper focuses on learning a dynamics model and use the learned model to train several model-free algorithms by sampling directly from the dynamics model. However, it is challenging to achieve good accuracy on dynamics model trajectories for highly complex domains due to stochasticity and compounding noise in the system. A majority of model-based RL focuses on dynamics models to derive policies from observation space. Deriving policies from observation space is problematic because it is often high dimensional with significant complexity. One approach that we investigate in this work is to reduce the learning complexity by learning policies from latent space variables directly.

This paper attempts to address some fundamental issues with model-based RL by learning several agents concurrently using ensemble learning. The ensemble consists of on-policy and off-policy model-free algorithms that learn from samples of a learned world model. Each agent learns in parallel from interaction with its separate dream environment and, during inference, forms a majority voting system. The world model is trained separately by first observing expert systems or RL agents interact with the environment and finally learn its dynamics in a supervised manner. The world model learns markovian state transitions, rewards, state cost for measuring goal distance. ORACLE solve traditional reinforcement learning problems and shows promising progress on complex state-of-the-art test environments. The contributions of this work summarize as follows.

- A novel world-model approach based on stochastic recurrent neural networks (SRNN) from [9] for end-to-end model based RL.
- Ensemble Learning using multiple of model-free algorithms with majority voting for action selection

We organize the paper as follows. Section 2 presents a in-depth background into reinforcement learning topics that ORACLE builds on and moves on to detail the algorithm architecture, design decisions and theoretical justifications. Section 3 presents empirical results in classical reinforcement learning problems, game environments, and in complex real-time strategy scenarios. We show that ORACLE outperforms both on and off-policy model-free alternatives in sample efficiency while maintaining comparable performance in most experiments. Section 4 presents recent related work in the field of model-based reinforcement learning. Finally, Section 5 discusses the good, the bad, and the ugly of ORACLE and concludes on the significance of our findings, We lay forth potential future paths of research for improving ORACLE performance and sample efficiency.

2 ORACLE: Observations Rewards Actions Costs Ensemble Learning

The **O**bservations **R**ewards **A**ctions **C**osts **L**earning **E**nsemble (ORACLE) is a novel end-to-end architecture for training model-free algorithms on a dynamics model of the ground truth environment. The ORACLE is a combination of several state-of-the-art deep learning techniques such as SRNN [9], variational autoencoders [18], and vector-quantization for latent space [24], and state-space models (SSM) [9]. This section aims to detail the algorithm and give the reader thorough insight into how ORACLE operates to learn a dynamics model for planning and executing decision-making.

Model-based reinforcement learning learns a dynamics model of a environment to derive a policy for decision making. The underlying mechanism is a Markov Decision Process (MDP), which mathematically defines the synergy between states, actions, rewards, and transitions. The problem is formalized as a tuple $M = (S, A, T, R)$, where $S = \{s_n, \ldots, s_{t+n}\}$ is a set of possible states and $A = \{a_n, \ldots, a_{t+n}\}$ is a set of possible actions. The state transition function $T : S \times A \times S \rightarrow [0, 1]$, which the dynamics model tries to learn is a probability function such that $T_{a_t}(s_t, s_{t+1})$ is the probability that current state s_t transitions to s_{t+1} given that the agent choses action a_t. The reward function $R : S \times A \rightarrow \mathbb{R}$ where $R_{a_t}(s_t, s_{t+1})$ returns the immediate reward received on when taking action a in state s_t with transition to s_{t+1}. The policy takes the form $\pi = \{s_1, a_1, s_2, a_2, \ldots, s_n, a_n\}$ where $\pi(a|s)$ denotes chosen action given a state. Model-based reinforcement learning divides primarily into three categories: 1) Dyna-based, 2) Policy Search-based, and 3) Shooting-based algorithms in which this work concerns Dyna-based approaches. The Dyna algorithm from [28] trains in two steps. First, the algorithm collects experience from interaction with the environment using a policy from a model-free algorithm (i.e., Q-learning). This experience is part of learning an estimated model of the environment, also referred to as a dynamics model. Second, the agent policy samples imagined data generated by the dynamics model and update its parameters towards optimal behavior.

Dynamics model: ORACLE's most crucial component is the dynamics model. The aim is to learn some parameters such that we can predict $s_{t+1} = F(s_t, a_t, \theta)$, the future state of a system or environment. Our approach is a combination of Variational Autoencoders (VAE) from [18] and Stochastic Recurrent State Space Models (SRSSM) from [9]. VAE and SRSSM's are highly expressive model classes for learning patterns in time series data and for system identification [7]. System identification, i.e. learning dynamics models from data is central in model-based reinforcement learning [6] and hence is a attractive concept for learning a functional dynamics model. We train the algorithm similarly to VAE's using amortized variational inference where we have two models, the generative

Table 1: Set of tunable parameters in ORACLE. In addition to this incomplete list, the algorithm has options for controlling model complexity such as neuron counts and number of layers.

Hyperparameter	**Values**	**Selected**	**Comment**
Batch Size	$\mathbb{Z}^+$	48	Number of sequence batches
Sequence Size	$\mathbb{Z}^+$	48	Number of frames in a sequence
Buffer Size	$\mathbb{Z}^+$	9 000	Replay buffer
Reward Scaling	$\mathbb{R}$	1.0	Scaling of the reward objective
Cost Scaling	$\mathbb{R}$	1.0	Scaling of the cost objective
VQ Scaling	$\mathbb{R}$	0.1	Scaling of the VQ objective
KL Scaling	$\mathbb{R}$	1.0	Scaling of the KL objective (KL-β)
KL Minimum Nats	$\mathbb{R}$	3.0	Minimal information loss
Optimizer		AdamW	AdamW improves generalization, see [19]
Gradient Clipping	$\mathbb{R}$	100.0	Clip gradients to increase learning stability
Adaptive GC	$\mathbb{B}$	1	Based on the history of gradient norms [27]
Learning Rate	$\mathbb{R}$	0.0001	Low Learning rate to improve stability.
Latent Leaps	$\mathbb{Z}^+$	30	Number of leaps into future states.
Dynamics Model RNN		LSTM	
Activation Functions		ELU	
Enc/Dec Neurons	$\mathbb{Z}^+$	1024	
Stochastic Reward	$\mathbb{B}$	1	Sample rewards under Gaussian assumptions
Stochastic Costs	$\mathbb{B}$	1	Sample costs under Gaussian assumptions

model (prior) pr_θ and the inference model (posterior) po_θ.

$$\textbf{Prior Model}: pr(z_t, h_t | h_{t-1}, a_{t-1}) \tag{1}$$

$$\textbf{Posterior Model}: po(z_t | x_t, h_{t-1}, a_{t-1}) \tag{2}$$

Equation 1-2 define our models where a_t is the event that triggers the transition from observation o_t to o_{t+1} in the real dynamics system. The prior model has a recursive dependency on previous hidden state h_{t-1}, action a_{t-1}, and outputs the hidden state h_t including the latent state z_t as seen in equation 1. Internally, the prior performs the following operations:

1. Compute $u_t = concat(h_{t-1}, a_{t-1})$
2. forward concatenation to RNN such that $h_t = RNN(u_t)$
3. parameterize mean $\mu_t = NN_1(h_t)$ diagonal covariance matrix $\sigma = NN_2(h_t)$
4. sample from Gaussian distribution $p_\theta(z_t|h_t) \sim \mathcal{N}(z_t; \mu, \sigma)$

where all steps are performed for every sample and forms our prior beliefs of the latent variables. Note that we do input any information about the visual landscape (observation) into the prior, but the objective function, which we will describe later aims to learn these dynamics implicit via the posterior function.

We now move the attention to calculating the latent variable through a posterior model. As seen in equation 2, the posterior depends on the previous hidden state h_{t-1}, action a_{t-1} but has exclusive access to the encoded ground truth observation x_t. Similarly to the prior model, the input are concatenated and directly parameterize a Gaussian distribution. The posterior model can be summarized to the following procedure:

1. Compute $u_t = concat(h_{t-1}, a_{t-1}, x_{t-1})$
2. parameterize mean $\mu_t = NN_3(u_t)$ diagonal covariance matrix $\sigma = NN_4(u_t)$
3. sample from Gaussian distribution $p_\psi(z_t|h_t) \sim \mathcal{N}(z_t; \mu, \sigma)$

To optimize the dynamics model, we use variational bayes where the goal is to fit the posterior approximation such that $pr_\theta(z) = po_\theta(z|x)$, however, since $po(z|x) = \int_z \frac{po(x|z)po(z)}{po(x)} dz$, it becomes intractable as we are concerned with computing the integral over the entire latent space z. For this reason, we approximate the posterior using $po_\theta(z)$ and choose the *Kullback-Leibler* distance to approximate. Following the work in [18], we end up with **E**vidence **L**ower **BO**und [17] given by

$$(\underbrace{\mathbb{E}[logpr(X|z)]}_{transition-loss} + \underbrace{\mathbb{E}[logpr(R|z)]}_{reward-loss} + \underbrace{\mathbb{E}[logpr(C|z)]}_{cost-loss}) - \gamma \cdot \underbrace{D_{KL}[po_\theta(z|X) \| pr_\theta(z)]}_{KL-divergence} \tag{3}$$

where log-likelihoods forms the reconstruction loss for observations, rewards, and costs, and KL-divergence act as a regularizer.

Rewards and Costs: ORACLE can predict rewards and costs using its dynamics model. During a transition in MDP's a feedback signal is emitted, which reinforcement learning agents utilize to fuel learning. We propose a novel, goal-directed approach where a cost-metric is added to the reward signal. This way, the reward signal is motivated by reaching the goal terminal state quickly.

Learning Ensemble : After ORACLE has trained in a supervised manner, an ensemble of model-free algorithms are trained on the dynamics model. Each actor learns separately in concurrent dynamics models. When the algorithm is sufficiently trained[1], the ensemble can perform decision making in the real environment. ORACLE can be configured to use the ensemble for decision making via majority voting or to use a single agent's decision.

Vector Quantization (VQ): We follow the VQ-VAE architecture from [24], and the motivation is to transform continuous latent-space variables into discrete latent variables, which has shown to be significantly better for reasoning planning, and predictive learning. Furthermore, VQ-VAE can model very long-term dependencies as it has a high compression rate compared to continuous space. [24]

Stochastic Weight Averaging (SWA) is a novel approach to ensemble learning where the objective is to widen the optima space such that it is easier to find and to give a better generalization of the model [16]. Compared to other ensemble learning techniques, SWA only requires a single model where snapshots are stored every n epochs. And are after m epochs averaged over. SWA has different learning rate strategies (i.e., cyclical learning rate), but we choose a fixed learning rate throughout training.

Hyperparameters: The ORACLE algorithm has a magnitude of different hyperparameters for tuning stability and performance. During the experiments, we found the algorithm to be rather robust to small changes in hyperparameters, and hence, we limit the scope to analyzing VQ, SWA, and Adaptive Gradient clipping (AGC) in section 3. ORACLE supports numerous hyperparameters, and to limit the scope, we have focused on only a few in this work. The motivation for having such a substantial set of hyperparameters is that different environments have different requirements to learn a good generalized model. Specifically, we choose a long-short term memory (LSTM) layer in the Dynamics model for our deterministic prediction of a future state. Another notable choice is to enable adap-

[1]The definition of 'sufficient' is to train up until a satisfactory performance in terms of average return.

tive gradient clipping, a novel approach to clip the gradient from historical norms [27]. Additionally, we clip the gradient if it exceeds 100.0 to limit training steps.

Algorithm 1: ORACLE Training Routine

```
1  **Result 1**: Learned ensemble of policies $\pi = \{ \pi_1 \cdots \pi_n \}$
2  **Result 2**: Learned dynamics model $z_{t+1} = f(z_t, h_t, a_t)$
3  **Hyperparameters**: See Table 1.
4  **Assumptions:** Expert-system $\Omega$
5  while dynamics model is not trained do
6      Train dynamics model $po_\theta(\hat{o}_{t+1}, z_{t+1} | o_t, s_{t+1}, a_t)$ using equation 3
7  end
8  while training model-free ensemble is not trained do
9      Choose action $a$ from policy strategy
10     Execute $a$ at state $s$ and get $z_{t+1}, r, c$ via dynamics model $po_\theta(z_{t+1}, r_{t+1}, c_{t+1} | z_t)$
11     Perform policy update (depedening on algorihm)
12 end
```

Algorithm 1 shows pseudo-code for training ORACLE[2]. In essence, the algorithm has two training steps. First, the algorithm observes some policy-making decisions in the ground truth environment. The algorithm trains either directly as the samples are observed or store them in a buffer for delayed training. When the dynamics model is trained, which is indicated by learning objective graphs, the second training procedure for a model-free algorithm begins. The programmer is allowed to use any model-free algorithm but should note that the training procedure is different for off and on-policy algorithms. In off-policy algorithms, the algorithm should utilize a replay buffer and sample actions from an external policy for exploration, while on-policy algorithms should train directly without such storage. Each model-free algorithm is assigned a fixed number of batches, each representing a dream-world instance. When the model-free agent has reached a sufficient level of performance, the ensemble is ready for making decisions in the ground truth environment. In this implementation, we use majority voting, and when there is no consensus, random actions are selected.

3 Experiments

This section reveals that ORACLE can perform well across many different environments and outperforms existing RL approaches in the classical RL environment CartPole where the aim is to balance a pole on a moving cart. Furthermore, we show promising results in Deep RTS Deathmatch, a one versus one real-time strategy game [1]. Finally, we evaluate

[2]We refer the reader to `https://github.com/perara/oracle` for a detailed implementation in python.

performance in the HalfCheetahPyBulletEnv-v0 from PyBullet, an open-source physics engine [5].[3] Our experimental evaluation aims to study the following questions:

- How well does ORACLE perform on reinforcement learning problems, compared to state-of-the-art model-based and model-free algorithms?
- What conclusions can we draw about ORACLE performance and are there any lessons for future study?

3.1 Hyperparameter and Sample Efficiency Evaluations

In the sample efficiency evaluations, we focus on how we can tune the ORACLE algorithm to improve sample efficiency while also maintaining acceptable performance. In this study, we only look at ORACLEbut significantly increase the number of hyperparameter variations in the experiments. We will investigate if SWA, VQ, latent leap (LL), and AGC have an advantageous effect on sample efficiency and if certain combinations show better performance. We intentionally leave out many hyperparameters that did not impact the performance in any meaningful capacity. However, we will detail our significant findings and discuss a guideline for choosing the correct hyperparameters for different environment types. We run the experiments five times for all environments and average the results. The round the convergence step to the nearest thousand and the algorithms fails the experiment if exceeding 1 million steps without reaching convergence. Finally, we use a ensemble of model-free algorithms for decision making, which we detail further in section 3.2

Table 2 illustrates the sample efficiency in terms of convergence rate for different hyperparameter settings with separate dynamics models per environment. The results clearly show that latent leap set to 30, AGC enabled, VQ disabled, and SWA enabled is the best choice for CartPole. For DeepRTS Deathmatch and HalfCheetahPyBulletEnv-v0, we observe the best results when VQ is enabled. We conclude that VQ performs worse in CartPole because it is a far simpler environment, and the algorithm cannot generalize well environments with few steps before termination state. This makes sense, as the VQ architectures double the number of trainable parameters in the model. The primary function of the VQ layer is to allow for encoding multiple environments in the same dynamics models, and hence it is natural to continue the experiments by using the same dynamics model for all environments.

Table 3 illustrates the sample efficiency of ORACLEwhen using the same model for all environments. This experiment aims to see if it is beneficial to feed the latent vector into a

[3]We take this opportunity to welcome the RL community to consider open-source benchmarks for easier comparison of scientific results.

Table 2: The experiment setup for the limited hyperparameter search. The table clearly shows that a latent-leap of 30 is superior in reaching a convergence score for all tested environments.

CartPole-v1				
SWA	**VQ**	**LL**	**AGC**	**Convergence Step**
on	on	10	on	N/A
on	on	30	on	755 000
on	on	60	on	N/A
on	**off**	**30**	**on**	**390 000**
off	on	30	off	825 000
DeepRTS Deathmatch				
SWA	**VQ**	**LL**	**AGC**	**Convergence Step**
on	on	10	on	N/A
on	**on**	**30**	**on**	**600 000**
on	on	60	on	N/A
on	off	30	on	N/A
off	on	30	off	N/A
HalfCheetahPyBulletEnv-v0				
SWA	**VQ**	**LL**	**AGC**	**Convergence Step**
on	on	10	on	N/A
on	**on**	**30**	**on**	**725 000**
on	on	60	on	N/A
on	off	30	on	N/A
off	on	30	off	N/A

Table 3: The data depicts the average performance using a single dynamics model to learn all environments. The table clearly shows that enabling VQ has a positive effect on sample efficiency, and without, the environment can not converge before the step limit has passed.

SWA	**VQ**	**LL**	**AGC**	**Average Convergence Step**
on	on	10	on	895 000
off	**on**	**30**	**on**	**565 000**
on	on	60	on	N/A
on	off	10	on	N/A
on	off	30	on	N/A
on	off	60	on	N/A
on	on	10	off	N/A
on	on	30	off	695 000
on	on	60	off	N/A
off	on	10	on	N/A
off	on	30	on	596 000
off	on	60	on	N/A
off	on	10	off	N/A
off	on	30	off	650 000
off	on	60	off	N/A

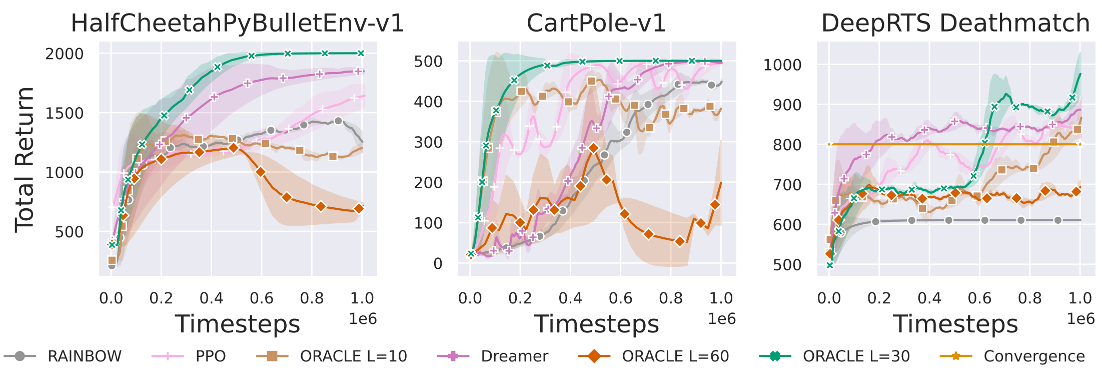


Figure 1: The figure illustrates the accumulated return performance and sample efficiency of PPO, RAINBOW, Dreamer, and ORACLE. We observe that in all environments, ORACLE outperform the state-of-the-art algorithms with LL set to 30. The x-axis describes the environment step, while the y-axis describes the average return.

VQ to structure the latent space categorically. The results clearly show that ORACLE can generalize across several environments using the same parameters, using a VQ layer after the generative network.

We make the following conclusions on how to tune ORACLE. For simple environments with less than 1 000 timesteps before forced termination, we recommend disabling VQ and using LL=30. If the environment exceeds 1 000 timesteps, enable VQ. When training the algorithm on all environments, we recommend having all hyperparameters enabled using LL=30.

3.2 Comparative performance Evaluation

In our comparative performance evaluation, we aim to understand how the ORACLE algorithm performs in contrast to state-of-the-art model-based and model-free methods and how to tune the algorithm for different environments properly. We compare the ORACLE to RAINBOW [15] and PPO [26] for model-free and Dreamer [12] for model-based methods. We select PPO, DQN, RAINBOW, A3C, and VPG as the ensemble and perform majority voting for each evaluated action. In the case of a draw, we randomly select one of the actions. We run each experiment 5 times and measure steps for model-free algorithms parallel with training and model-based methods towards the real environment every 5 000 steps.[4] For the comparison, we use reference hyperparameters found in [15] for RAINBOW and [26] respectively.

As seen in figure 1 ORACLE outperforms both model-free and model-based approaches for the selected environments. A dependent factor on how good ORACLE performs is

[4]We make the reader aware that the experiments are compute-heavy, hence few experiment iterations. In total, the experiments take ∼5 days of wall-clock time to train on consumer-level hardware.

the latent-leap parameter which represents how many steps the algorithm *leaps into the future* before resampling the real environment. Specifically, we see that a latent leap of 30 is a good compromise between sample efficiency and return performance.

3.3 Design Evaluation

We next discuss the findings in the hyperparameter tuning experiments and the comparative performance evaluation to understand better why ORACLE outperforms prior approaches.

Specifically, we think that ORACLE has strength in capturing large state-spaces and has a good ability to generalize well over sparse datasets. The dynamics model is primarily continuous with a combination of deterministic and stochastic variables, and it is well understood in the literature that variational inference is an excellent approach to model powerful generative models. Putting a VQ layer between the continuous probabilistic latent vector and transform it into a discrete latent vector before reconstructing the output. In parallel to our work [22] has the same conclusion and shows outstanding results on solving chess, outperforming previous methods. We conclude

- Using VQ in combination with VAE and SRSSM provides a powerful enhancement to model robustness, but it falls short when used for more simple problems,
- generally, we see that ORACLEis best suited for larger problems with more than 1 000 timesteps, and
- it remains an open question to justify the combination of VAE, SRSSM, and VQ analytically.

4 Recent Related Work

Recent literature shows that Model-based RL is becoming the frontier with several new algorithms that outperform model-free variants with a large margin. The most recent achievement takes form as discrete world models with DreamerV2 from Hafner et al. [14]. DreamerV2 is the first reinforcement learning agent that achieves human-level performance on the Atari benchmark by learning behaviors fully offline in a world-model. Prior work in [12] use similar architecture to [2] by deriving latent dynamics that form estimations of future observations given an action. Very Recently, Ozair et al. proposed Vector Quantized Models for planning in Reinforcement Learning [22]. The authors use a stochastic variant of the Monte Carlo tree search algorithm to plan the agent's actions and the discrete latent variables representing the system's dynamics model. This approach shows state-of-the-art results in chess and illustrates that the approach scales to DeepMind

Lab, a first-person 3D environment with complex visual state observations with only partial observability.

Deep Planning Network (PlaNet) is a model-based agent that interpret the pixels of a state to learn the dynamics of an environment. The environment dynamics are stored into latent-space, where the agent sample actions based on the learned representation. The proposed algorithm showed significantly better sample efficiency compared to algorithms such as A3C ([13]).

[4] recently proposed Probabilistic Ensembles with Trajectory Sampling (PETS). The algorithm uses an ensemble of bootstrap neural networks to learn a dynamics model of the environment over future states. The algorithm then uses this model to predict the best action for future states. The authors show that the algorithm significantly lowers sampling requirements for environments such as half-cheetah compared to SAC and PPO.

5 Discussion and Future Work

We have investigated if the model-based reinforcement learning algorithm ORACLE performs well in classical and novel environments through a theoretical and empirical approach. We have shown that using state-space models combined with recurrent neural networks and variational inferences yields promising results towards advanced artificial intelligence that can perform well in tasks without directly interacting with the target environment. While it is difficult to explain the model analytically fully, we can show empirically that the model can generalize well in all tested environments. Furthermore, we show empirically that ORACLE substantially outperforms all tested model-free algorithms in performance and sample efficiency in tested environments, which is required for industry-near mission-critical environments. However, the algorithm still has several shortcomings, which we intend to approach in our continued work. First, we wish to increase the robustness to posterior collapse, a well-known problem in variational inference. Second, we wish to expand the scope of our experiments to include a substantially more comprehensive quantitive study with qualitative support to better understand the algorithm's strengths and weaknesses. Otherwise, we wish to

- Adopt the work in [23] to alleviate the posterior collapse phenomena and hopefully be able to provide better stability guarantees for the training procedure,

- Experiment with liquid time-constant networks (LTC) [27] for better learning the environment dynamics in the state-space model

References

[1] Andersen, P., Goodwin, M., Granmo, O.: Deep RTS: A Game Environment for Deep Reinforcement Learning in Real-Time Strategy Games. In: 2018 IEEE Conference on Computational Intelligence and Games (CIG). pp. 1–8 (aug 2018). https://doi.org/10.1109/CIG.2018.8490409

[2] Andersen, P.A., Goodwin, M., Granmo, O.C.: The Dreaming Variational Autoencoder for Reinforcement Learning Environments. In: Max Bramer, Petridis, M. (eds.) Artificial Intelligence XXXV, vol. 11311, pp. 143–155. Springer, Cham, xxxv edn. (dec 2018). https://doi.org/10.1007/978-3-030-04191-5_11

[3] Arulkumaran, K., Deisenroth, M.P., Brundage, M., Bharath, A.A.: Deep reinforcement learning: A brief survey. IEEE Signal Processing Magazine **34**(6), 26–38 (2017). https://doi.org/10.1109/MSP.2017.2743240

[4] Chua, K., Calandra, R., McAllister, R., Levine, S.: Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31, pp. 4754–4765. Curran Associates, Inc. (may 2018)

[5] Coumans, E., Bai, Y.: PyBullet, a Python module for physics simulation for games, robotics and machine learning, `http://pybullet.org`

[6] Deisenroth, M., Rasmussen, C.E.: PILCO: A model-based and data-efficient approach to policy search. In: Proceedings of the 28th International Conference on machine learning ICML'11. pp. 465–472. Citeseer (2011)

[7] Doerr, A., Daniel, C., Schiegg, M., Duy, N.T., Schaal, S., Toussaint, M., Sebastian, T.: Probabilistic Recurrent State-Space Models. In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 80, pp. 1280–1289. PMLR (2018), `http://proceedings.mlr.press/v80/doerr18a.html`

[8] Draganjac, I., Miklic, D., Kovacic, Z., Vasiljevic, G., Bogdan, S.: Decentralized Control of Multi-AGV Systems in Autonomous Warehousing Applications. IEEE Transactions on Automation Science and Engineering **13**(4), 1433–1447 (oct 2016). https://doi.org/10.1109/TASE.2016.2603781

[9] Fraccaro, M.: Deep latent variable models for sequential data (2018), `https://orbit.dtu.dk/en/publications/deep-latent-variable-models-for-sequential-data`

[10] Fuchs, A., Heider, Y., Wang, K., Sun, W.C., Kaliske, M.: DNN2: A hyper-parameter reinforcement learning game for self-design of neural network based elasto-plastic constitutive descriptions. Computers and Structures **249**, 106505 (jun 2021). https://doi.org/10.1016/j.compstruc.2021.106505

[11] García, J., Fernández, F.: A Comprehensive Survey on Safe Reinforcement Learning. Journal of Machine Learning Research **16**, 1437–1480 (2015)

[12] Hafner, D., Lillicrap, T., Ba, J., Norouzi, M.: Dream to Control: Learning Behaviors by Latent Imagination. In: Proc. 8th International Conference on Learning Representations, ICLR'20 (2020), https://openreview.net/forum?id=S1lOTC4tDS

[13] Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., Davidson, J.: Learning Latent Dynamics for Planning from Pixels. In: Chaudhuri, K., Salakhutdinov, R. (eds.) Proc. 36th International Conference on Machine Learning, ICML'18. vol. 97, pp. 2555–2565. PMLR, Long Beach, CA, USA (jun 2019), http://proceedings.mlr.press/v97/hafner19a/hafner19a.pdf

[14] Hafner, D., Lillicrap, T.P., Norouzi, M., Ba, J.: Mastering Atari with Discrete World Models. In: Proc. 9th International Conference on Learning Representations, ICLR'21 (2021), https://openreview.net/forum?id=0oabwyZbOu

[15] Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., Silver, D.: Rainbow: Combining Improvements in Deep Reinforcement Learning. In: Proc. 32nd Conference on Artificial Intelligence, AAAI'18. pp. 3215–3222. AAAI Press, New Orleans, Louisiana USA (oct 2018), https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/download/17204/16680

[16] Izmailov, P., Podoprikhin, D., Garipov, T., Vetrov, D., Wilson, A.G.: Averaging Weights Leads to Wider Optima and Better Generalization. In: R. Silva, A.G., Globerson, A. (eds.) 34th Conference on Uncertainty in Artificial Intelligence 2018. pp. 876–885. Association For Uncertainty in Artificial Intelligence (mar 2018), http://arxiv.org/abs/1803.05407

[17] Jordan, M.I., Ghahramani, Z., Jaakkola, T.S., Saul, L.K.: Introduction to variational methods for graphical models. Machine Learning **37**(2), 183–233 (nov 1999). https://doi.org/10.1023/A:1007665907178, https://link.springer.com/article/10.1023/A:1007665907178

[18] Kingma, D.P., Welling, M.: Auto-Encoding Variational Bayes. Proceedings of the 2nd International Conference on Learning Representa-

tions (dec 2013). https://doi.org/10.1051/0004-6361/201527329, http://arxiv.org/abs/1312.6114

[19] Loshchilov, I., Hutter, F.: Decoupled Weight Decay Regularization. In: Proc. 7th International Conference on Learning Representations, ICLR'19 (2019), https://openreview.net/forum?id=Bkg6RiCqY7

[20] Mallozzi, P., Pelliccione, P., Knauss, A., Berger, C., Mohammadiha, N.: Autonomous Vehicles: State of the Art, Future Trends, and Challenges. In: Automotive Systems and Software Engineering, pp. 347–367. Springer International Publishing (2019). https://doi.org/10.1007/978-3-030-12157-0_16

[21] Moerland, T.M., Broekens, J., Jonker, C.M.: Model-based Reinforcement Learning: A Survey. arxiv preprint arXiv:2006.16712 (jun 2020), https://arxiv.org/abs/2006.16712

[22] Ozair, S., Li, Y., Razavi, A., Antonoglou, I., van den Oord, A., Vinyals, O.: Vector Quantized Models for Planning. In: Proc. 39th International Conference on Machine Learning, ICML'21 (jun 2021), http://arxiv.org/abs/2106.04615

[23] Razavi, A., van den Oord, A., Poole, B., Vinyals, O.: Preventing Posterior Collapse with delta-VAEs. In: Proc. 7th International Conference on Learning Representations, ICLR'19 (2019), https://openreview.net/forum?id=BJe0Gn0cY7

[24] Razavi, A., van den Oord, A., Vinyals, O.: Generating Diverse High-Fidelity Images with VQ-VAE-2. In: Wallach, H., Larochelle, H., Beygelzimer, A., Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32. pp. 14837–14847. Curran Associates, Inc., Vancouver, BC, Canada (2019), http://papers.nips.cc/paper/9625-generating-diverse-high-fidelity-images-with-vq-vae-2

[25] Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T., Silver, D.: Mastering Atari, Go, chess and shogi by planning with a learned model. Nature **588**(7839), 604–609 (dec 2020). https://doi.org/10.1038/s41586-020-03051-4, https://doi.org/10.1038/s41586-020-03051-4

[26] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal Policy Optimization Algorithms. arxiv preprint arXiv:1707.06347 (jul 2017), http://arxiv.org/abs/1707.06347

[27] Seetharaman, P., Wichern, G., Pardo, B., Roux, J.L.: Autoclip: Adaptive gradient clipping for source separation networks. In: IEEE International Workshop on Machine Learning for Signal Processing, MLSP. vol. 2020-Septe. IEEE Computer Society (sep 2020). https://doi.org/10.1109/MLSP49062.2020.9231926

[28] Sutton, R.S.: Dyna, an integrated architecture for learning, planning, and reacting. ACM SIGART Bulletin **2**(4), 160–163 (jul 1991). https://doi.org/10.1145/122344.122377, `https://dl.acm.org/doi/10.1145/122344.122377`

[29] Vithayathil Varghese, N., Mahmoud, Q.H.: A Survey of Multi-Task Deep Reinforcement Learning. Electronics **9**(9) (2020). https://doi.org/10.3390/electronics9091363, `https://www.mdpi.com/2079-9292/9/9/1363`

[30] Yu, C., Liu, J., Nemati, S.: Reinforcement Learning in Healthcare: A Survey. arxiv preprint arXiv:1908.08796 (aug 2019), `https://arxiv.org/abs/1908.08796`

Paper K: ORACLE: End-to-End Model Based Reinforcement Learning

Paper L

Title:	CaiRL: A High-Performance Reinforcement Learning Environment Toolkit
Authors:	Andersen, Per-Arne et al.
Affiliation:	Department of ICT, University of Agder, Grimstad Norway
Journal:	In Review: IEEE Conference on Games
Year:	2021

CaiRL: A High-Performance Reinforcement Learning Environment Toolkit

Per-Arne Andersen
Department of ICT
University of Agder
Grimstad, Norway
per.andersen@uia.no

Morten Goodwin
Department of ICT
University of Agder
Grimstad, Norway
morten.goodwin@uia.no

Ole-Christoffer Granmo
Department of ICT
University of Agder
Grimstad, Norway
ole.granmon@uia.no

Abstract

This paper addresses the dire need for a platform that efficiently provides a framework for running reinforcement learning (RL) experiments. We propose CaiRL Environment Toolkit as an efficient, compatible, and more sustainable alternative for training learning agents and details recommendations on developing efficient simulations.

There is an increasing focus on developing sustainable artificial intelligence, but little effort has been made to improve the environmental efficiency for running simulations. The most popular development toolkit for reinforcement learning, OpenAI Gym, is built using Python, a powerful but slow programming language. To overcome the slowness of Python, we propose a platform on C++ that gives the same flexibility but at magnitudes faster speeds.

CaiRL also presents the first reinforcement learning Toolkit with a built-in Adobe Flash emulator for running legacy flash games for reinforcement learning research. We empirically demonstrate that CaiRL performs significantly better through a thorough comparison of the classic control domains and further illustrate that CaiRL is fully compatible with OpenAI Gym for running reinforcement learning experiments.

Keywords: Reinforcement Learning, Environments, Sustainable AI

IEEE Conference on Games
Submitted 24. Dec 2021, In Review.

1 Introduction

Reinforcement Learning (RL) is a machine learning area concerned with sequential decision-making in real or simulated environments. RL has a strong theoretical background and has shown outstanding capabilities in learning to maneuver in unknown non-stationary state-spaces [1–3]. In most recent literature, Deep RL has mastered many complex games such as Go [4], StarCraft II [5] and progressively moves towards mastering autonomous control [1]. Furthermore, RL has potential in health care for tumor classification [6], Finances [7], and Industry-4.0 [8] applications. RL solves problems iteratively and needs to interact with a model, also referred to as an environment.

There are, however, fundamental challenges with how RL operates when learning to behave correctly. First, trial and error are seldom compatible with real-world systems, as they often require safety throughout the learning process. RL defines a reward function that the algorithm actively attempts to maximize, and it is suggested that an RL system can learn only following reward signals [9]. Given that it is feasible to craft an optimal reward function, agents can quickly learn to reach the desired behavior but require making uneducated guesses during their learning trajectory. RL also requires many samples to learn optimal trajectories, rendering it challenging to learn optimal behaviors promptly. While there are efforts into addressing the safety and sample efficiency concerns in RL, it remains an open question [10]. However, the problem is that the training procedure is time-consuming, which leads to a large climate footprint and is compute-budget inefficient.

While it is not trivial to change the concepts of RL in favor of better computation times, there are ways to reduce the time used on evaluating the RL model and the environment model. Environment models in RL take the form of a mathematical model or a computer program. The model is expressed using a programming language where Python dominates the ML and RL field in both cases. Python is a dynamically typed interpreted language used much due to its simplicity reducing the prototyping time drastically. Interpreted languages are considerably slower than compiled languages because

- computer code is read line for line and translated to machine code on the fly compared to statically compiled programming languages such as C++ [11]

- programs written in Python are not trivial to parallelize because Python uses a global interpreted lock (GIL)

- dynamically typed languages are slower due to the extra computation cost of inferred data types.

Due to the shortcomings mentioned above of RL, a significant portion of research in RL takes place in simulated environments. By learning in a simulated environment, the agent can freely make decisions that lead to catastrophic results without any harm to the real system, and it is multitudes faster sample efficiency than real-world systems. Currently, a majority of environments are implemented in Python and runs using the OpenAI Gym toolkit [12]. OpenAI Gym is an interface to generalize the execution of environments so that it is trivial to test algorithms across several environments. Nevertheless, it suffers a significant performance cost because of its implementation in Python. Consequently, reinforcement learning experiments yield unnecessary computing costs, computation time, and a high carbon emission footprint.

CaiRL approaches these negative side-effects by improving computation efficiency to reduce computing costs and lower the carbon emission footprint for a more sustainable AI.

We propose the CaiRL Environment toolkit to fill the gap of a flexible and high-performance toolkit for running reinforcement learning experiments. CaiRL uses C++ as its programming language and focuses on being close to the computer hardware while maintaining ease of use to reduce experiment setup times. The toolkit uses a combination of templating and const expression functions to move a considerable amount of computing to compile-time, leaving less work for run-time. CaiRL aims to have a near-identical interface to OpenAI Gym to ensure that migrating existing code requires minimal effort. To the best of our knowledge, we present the first Adobe Flash compatible reinforcement learning interface with support for Actionscript 2 and 3 without installing third-party applications. CaiRL embeds the Java Virtual Machine (JVM) and Python interpreter, enabling the toolkit to run Java and Python-based environments seamlessly if porting to C++ is impractical. This contribution aims to

1. implement a more efficient solution to running reinforcement learning experiments,
2. introduce novel problems for reinforcement learning research
3. briefly tutorial the reader on adding new environments
4. show the effectiveness of the proposed solution
5. empirically show that reinforcement learning agents learn faster
6. show the effectiveness in reducing the carbon emission footprint following [13]

The paper is organized as follows. In Section 2, we dive into the existing literature on reinforcement learning game design and compare the existing solution to find the gap for our research question. Section 3 details reinforcement learning and sets it in perspective to the problem CaiRL set out to solve. Section 4 details the design choices of CaiRL and provides a thorough justification for design choices. Section 5 presents our empirical findings of performance, adoption challenges, and how they are solved, and finally compares the interface of the CaiRL framework to OpenAI Gym. Section 6 presents a brief design recommendation for developers of new environments aimed at reinforcement learning research. Finally, we conclude our work and outlines a path forwards for adopting CaiRL.

2 Background

2.1 Reinforcement Learning

Reinforcement Learning is modeled according to an **M**arkov **D**ecision **P**rocess (MDP) described formally by a tuple $(S, A, T, R, \gamma, s_0)$, where S is the state-space, A is the action-space, $T\colon S \times A \rightarrow S$ is the transition function, $R\colon S \times A \rightarrow \mathbb{R}$ is the reward function [14], γ is the discount factor, and s_0 is starting state. RL operates iteratively until reaching a terminal state, of which the program terminates. Q-Learning is an off-policy RL algorithm and seeks to find the best action to take given the current state. The algorithm operates off a Q-table, an n-dimensional matrix that follows the shape of state dimensions where the final dimension is the q-values. Q-Values quantify how good it is to act a at time t. In this paper, we replace the Q-table with a function approximator, specifically a neural network. This forms the algorithm Deep Q-Network (DQN), which is one of the first deep learning-based approaches to RL and is commonly known to solve ATARI 2600 with superhuman performance [15]. In section 6.3, we use DQN to measure the run-time and carbon emission performance to validate the usefulness of CaiRL.

2.2 Graphics Acceleration

A graphics accelerator takes charge of evaluating program code into a graphical representation through a renderer. The graphics accelerator runs either as a software implementation such as program code or a hardware implementation such as graphics cards with specialized electronics for graphics rendering. It is natural to assume that hardware-accelerated graphics yield the most performance because of task specialization, but this is not always the truth in simple 2D graphics where the use-case is to copy the drawn frame to CPU memory.

According to [16], software rendering in modern CPU chips performs 2-10x faster due to specialized bytecode instructions. The study concludes that although GPU can render frames faster, provided that the frame permanently resides in GPU memory. Having frames in the GPU memory is impractical for machine learning applications because of the copy between CPU and GPU. The authors in [17] propose to use Single Instruction Multiple Data (SIMD) optimizations to improve the performance in games. SIMD is an extension to the CPU instruction set for doing vectorized arithmetic to increase instruction throughput. The authors find that using SIMD instruction increases performance by over 80% compared to traditional CPU rendering techniques.

The findings in these studies suggest that software acceleration is beneficial in some graphic applications, and similarly, we find it useful in a reinforcement learning context. Empirically, we see that software rendering performs better for simple 2D and 3D graphic applications due to the high-latency copy operation needed between GPU and CPU. Much of the success in CaiRL lies in the fact that software rendering, while being slower for advanced games such as StarCraft, significantly outperforms hardware rendering for simple graphics. One alternative to improve performance in hardware rendering is to pixel buffer objects or equivalent implementation. A pixel buffer object (PBO) is a buffer storage which allows the user to retrieve frame buffer pixels asynchronously while a new frame buffer draws to the screen frame buffer. In particular, copying pixels without PBO is slow because rendering must halt while the buffer is read.

CaiRL aims to increase the game repository over time and encourage submissions of new environments. For this reason, it is essential to recommend implementation methods that yield the highest performance. We find that software rendering is the best choice for simple 2D and 3D-based games, and in complex 3D games such as StarCraft II, the programmer should implement PBO-based rendering if it is possible to gain access to the rendering context.

2.3 Programming Languages

Historically, many different programming languages are used to perform machine learning research and application development. In more recent history, the Python language is used more frequently among the scientific community and more specifically in machine learning, and deep learning [11]. Unfortunately, Python's most used implementation is CPython, a single-threaded implementation with little regard for efficiency compared to compiled languages. However, Python's most popular toolkits for machine learning are implemented in a compiled languages and use glue code to interact to increase performance. A study from Zehra et al. suggests that C++ has approximately a 50 times performance advantage over Python, and Python has advantages in code readability for beginners in programming [18]. It is clear from these studies that Python is great for prototyping and learning programming but falls behind for time-consuming tasks. It falls natural to seek an approach that can preserve the simplicity of Python while also maintaining good performance to reduce the task execution time.

The choice of programming language is essential in CaiRL because it aims to be efficient and reduce the carbon emission footprint as much as possible. C++ seems like a natural choice as it is mature, has a stable standard library, and is trivial to maintain, compared to C. It is also possible to compile C++ code to a python compatible binary.

Pybind11 is one such framework that provides a method to create an efficient bridge between C++ and python code. pybind11 is a lightweight library that exposes C++ types in Python and vice versa but focuses mainly on exposing C++ code paths to python applications. There is a minor performance penalty during conversion between Python and C++ objects. Hence implementations in C++ will run near-native performance in Python. For this reason, we follow the path of implementing an efficient experiment toolkit for reinforcement learning in C++ with binding code to allow Python to interface with CaiRL.

3 Design Specifications

The design goal of CaiRL is to have interoperability with OpenAI Gym, but with magnitudes better performance and flexibility to support environments in a multitude of programming languages. Keeping full compatibility with OpenAI Gym is central to trivialize the two frameworks without significant amendments to existing code.

CaiRL is a novel reinforcement learning environment toolkit for running experiments with high performance. By designing such a toolkit, reinforcement learning becomes more affordable due to reduced execution costs and strives to reach more sustainable AI. A bi-effect of these goals is that experiments run significantly faster and most CPU cycles on training AI instead of evaluating game-states. The CaiRL environment toolkit supports classical RL problems such as (1) Cart-Pole, Acro-Bot, Mountain-Car, and Pendulum, (2) Novel high-complexity games such as Deep RTS [19], Deep Line Wars, X1337 Space Shooter, and (3) over 1 000 flash games available for experimentation. [1]

The engine of CaiRL relies upon C++ with highly performant fast-paths such as Single instruction, multiple data (SIMD) for vectorized calculation that fits into the processor registry in a single instruction. The design mimics OpenAI Gym but relies on templating and const expressions that enable calculations to evaluate at compile-time instead of runtime. CaiRL is split into modules, and we dedicate this section to describe the design decision and the resulting interaction layer and benefits compared to similar solutions.

3.1 Module Layer

CaiRL has a modular design aiming towards having little cross-dependencies between module categories to decrease compile times. CaiRL splits into three categories:

[1]We invite the reader to `http://github.com/cair/rl` for in-depth documentation.

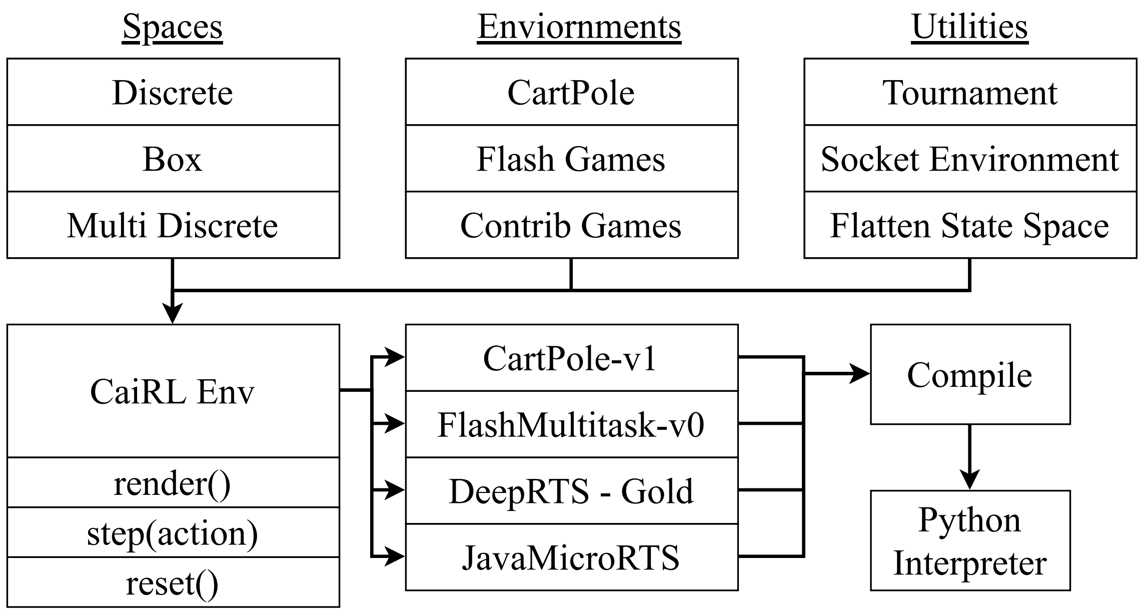


Figure 1: Brief overview over components in CaiRL. The framework is split into modules similar to Gym: spaces, environments, and utilities. The modules interface with the CaiRL Env class, from which all environments derive. The environments compile to a shared library and are callable from python code.

- Spaces for action and observation definitions,
- Environments for wrappers of supported environments,
- Utilities for enriching the framework, such as the tournament module.

As seen in Figure 3.1, the CaiRL framework has a streamlined dependency structure, where the primary dependency lies in the CaiRL Env class. The CaiRL Env class is the glue that binds together the environment definition and logic. Finally, the environment is compiled to machine code and wrapped to a python compatible format as described in section 2.3.

3.2 Interaction Layer

There are two ways of building reinforcement learning experiments with CaiRL using C++ directly or through the Python to C++ bindings. CaiRL runs efficiently in both programming languages, but there is a computational cost to run the Python interpreted and translate calls between C++ and Python. The primary goal of the API design is to match the Gym API as close as possible to reduce the needed effort of migrating existing codebases to CaiRL.

```
e =  Flatten<TimeLimit<200,CartPoleEnv>>()
```

```
2  for(int ep = 0; ep < 100; ep++){
3   e.reset();
4   int terminal, steps = 0;
5   while(!terminal){
6    steps++;
7    const auto [s1, r, t, info] =
8    e.step(e.action_space.sample());
9    auto obs = e.render();
10   terminal = t;
11  }
12 }
```

Listing 1: Minimal Example of CaiRL-CartPole-v1 in C++

As seen in listing 1, the C++ interface is similar to OpenAI Gym. The difference is that CaiRL modules use template classes, as seen in line 2. A template defines a class that can evaluate much of the program logic during compile-time. There are considerable run-time benefits because code initialization resolves during compile-time at the cost of longer compile times. Another downside of templates is reduced flexibility because it is impractical to create environment definitions during run-time, which is required when building Python-based environment wrappers. However, it is possible to define run-time-defined environments in Python at the cost of performance.

A very central component of CaiRL is the ability to run experiments natively in Python. This becomes possible through exposing C++ code through wrapper code in pybind11. Pybind11 is a tool that provides methods to translate between the python interpreter automatically and the CaiRL C++ shared library. Using the Python wrapper code, there is no need for C++ experience, and it is possible to both use and extend CaiRL solely using Python. The Python interface is very similar to the C++ interface but focuses more on compatibility with the OpenAI Gym interface.

```
1  #e =  gym.make("CartPole-v1")
2  e =  cairl.make("CartPole-v1") # Use CaiRL
3  for ep in range(100):
4  e.reset()
5  terminal, steps = 0
6  while not terminal:
7  steps++
8  a = e.action_space.sample()
9  s1, r, t, info = e.step(a)
10 obs = e.render()
11 terminal = t
```

Listing 2: Minimal Example of OpenAI and CaiRL CartPole-v1 in Python

Listing 2 illustrates the use of CaiRL in Python as compared to Gym. In particular, the only change that is required to switch between the two frameworks is to comment on Line 1 and comment out Line 2.

3.3 Affordable and Sustainable AI

AI is a constantly growing field of research, and with the shifted focus on Deep Learning, it is well understood that the need for computing power has increased sharply. Deep Learning models have a range of a few thousand parameters, up to several billion parameters that require carefully tuning with algorithms such as stochastic gradient descent. Hence, compute power plays an essential role in the performance of the trained model. In Deep Reinforcement Learning, the same applies but also requires extensive data sampling from an environment. It is safe to conclude that running experiments becomes exponentially more expensive and goes against more sustainable AI. CaiRL aims to minimize the cost of reinforcement learning by reducing environment execution time. In essence, this has the bi-effect of reducing the carbon emission footprint in RL significantly compared to existing solutions, as observed in section 6.1.

4 Supported Games

4.1 Java Applications

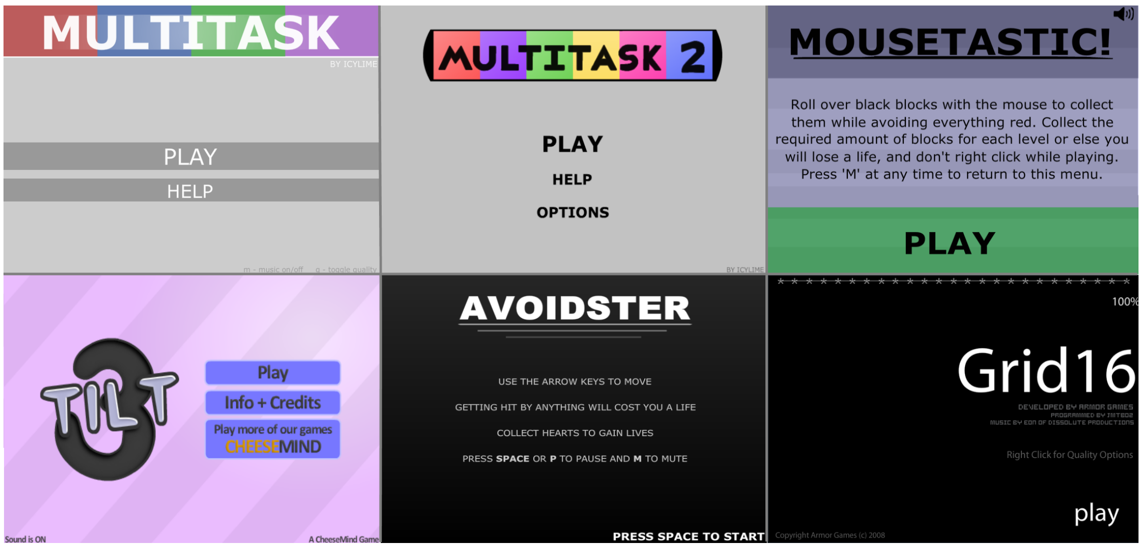
Java is a popular language that runs in the Java Virtual Machine (JVM). While Java is not the dominant language for developing games in the industry, there are a few notable examples such as MicroRTS [20] and the Showdown AI competition [21]. These environments have shown significant value to several research communities in reinforcement learning, evolutionary algorithms, and planning-based AI. Configuration for integrating java games in CaiRL is minimal but requires defining a CMakeList configuration that builds Java source code to a Java archive (JAR) file. Then the programmer must define a C++ class that extends the abstract Env interface. The bindings procedure uses the Java Native Interface (JNI) to initialize a Java Virtual Machine (JVM). Through the JNI, it is possible to interact with Java code from C++ and hence becomes trivial to create binding code that runs efficiently through the C++ library but at the expense of performance drawbacks of the JVM.

4.2 Python Applications

Python is perhaps the most used programming language for machine learning research in recent literature. Consequently, many of the popular reinforcement learning environments run natively in Python. We approach this task with two possible solutions where. The first approach is to automatically convert python code into C++ using the Nuitka package found at `https://github.com/Nuitka/Nuitka`. For complex environments with many third-party dependencies, CaiRL can wrap python code by embedding the python interpreter. The benefit of the latter approach is that the original code remains untouched but at the cost of significantly less performance. All third-party environments reside in the `cairl.contrib` package and are freely available through the C++ and Python interface. For an environment to be fully compatible with the CaiRL interface, the environment must inherit the abstract Env class and implement the `step(action)`, `reset()` , and `render()` function.

4.3 Flash Games

The most notable feature of CaiRL is the ability to run flash games without external applications at an accelerated speed. CaiRL extends the LightSpark flash emulator for Actionscript 3 and falls back to GNU Gnash for ActionScript 2. CaiRL features a repository of over 1300 flash games for conducting AI research and reinforcement learning research.



Roll over black blocks with the mouse to collect them while avoiding everything red. Collect the required amount of blocks for each level or else you will lose a life, and don't right click while playing. Press 'M' at any time to return to this menu.

USE THE ARROW KEYS TO MOVE
GETTING HIT BY ANYTHING WILL COST YOU A LIFE
COLLECT HEARTS TO GAIN LIVES
PRESS SPACE OR P TO PAUSE AND M TO MUTE

Figure 2: The figure shows a few select flash games in CaiRL. Flash environments are an excellent playground for RL and Machine Learning algorithms.

Figure 4.3 illustrate a subset of the implemented games from [22]. The first game, Multitask and Multitask 2 is similar to CartPole but requires the agent to perform different tasks concurrently. If the agent fails one of the tasks, the game terminates. The reward function is defined as positive rewards while the game is running and negative rewards when the game is shut down, indicating that the game is lost. The game observations are either raw pixels or memory, and the actions-space is discrete. The Mousetastic game is a continuous action space with mouse cursor positions. The goal of the game is to collect black blocks while avoiding every red object. The reward function is a positive reward for all states except for terminal states. Similarly, the remaining games in figure 4.3 are games with simple objectives but are novel experiments for reinforcement learning research.

4.4 Puzzle Games

The initial version of CaiRL supports a comprehensive collection of puzzles from the Simon Tatham collection [23]. This collection aims to provide logical puzzles that are solvable either by humans or by algorithms. While reinforcement learning is not mainly known for solving logical puzzles, we find it excellent to add them for future research. Some literature suggests that reinforcement learning works for solving puzzles [24] including the options framework from [25]. All puzzles include a heuristic-based solver, which enables transfer and curriculum learning research.

5 Competition Platform

The CaiRL features a competition platform for running tournaments in multi-player games. Although the Gym interface is incompatible with two-player games, the CaiRL interface allows for such setups. It is possible to run tournaments in two different configurations, but creating custom configurations through the abstract tournament class is possible. The tournament and competition are useful at conferences, workshops, or academic game jams. Currently, CaiRL supports the single-elimination model and the swiss tournament model [26].

Single elimination model. As seen in figure 5, the single-elimination model is perhaps the most simple setup for tournaments. The tournament starts with a randomized selection of matches for the first round. Winning participants from round 1 match together and losing participants are eliminated from the tournament. The same procedure continues until only one participant is standing as the tournament's winner.

Swiss tournament model. A major drawback with the single-elimination model is that players that lose are not able to compete for the rest of the tournament. As a consequence of this, it is difficult to measure the overall performance of all participating players. The Swiss tournament system is a bracket-based system that allows players to play for the whole tournament. The tournament initialization starts with matching players using ELO rating or by random selection. Figure 5 illustrates the tournament setup formed as brackets. Winners of the first match are transferred to the winning bracket, denoted as $1-0$ bracket, while the losers play in the $0-1$) bracket. This procedure continues until players are eliminated such as in bracket $0-2$, $1-1$, $2-0$, $2-1$. There are several ways to

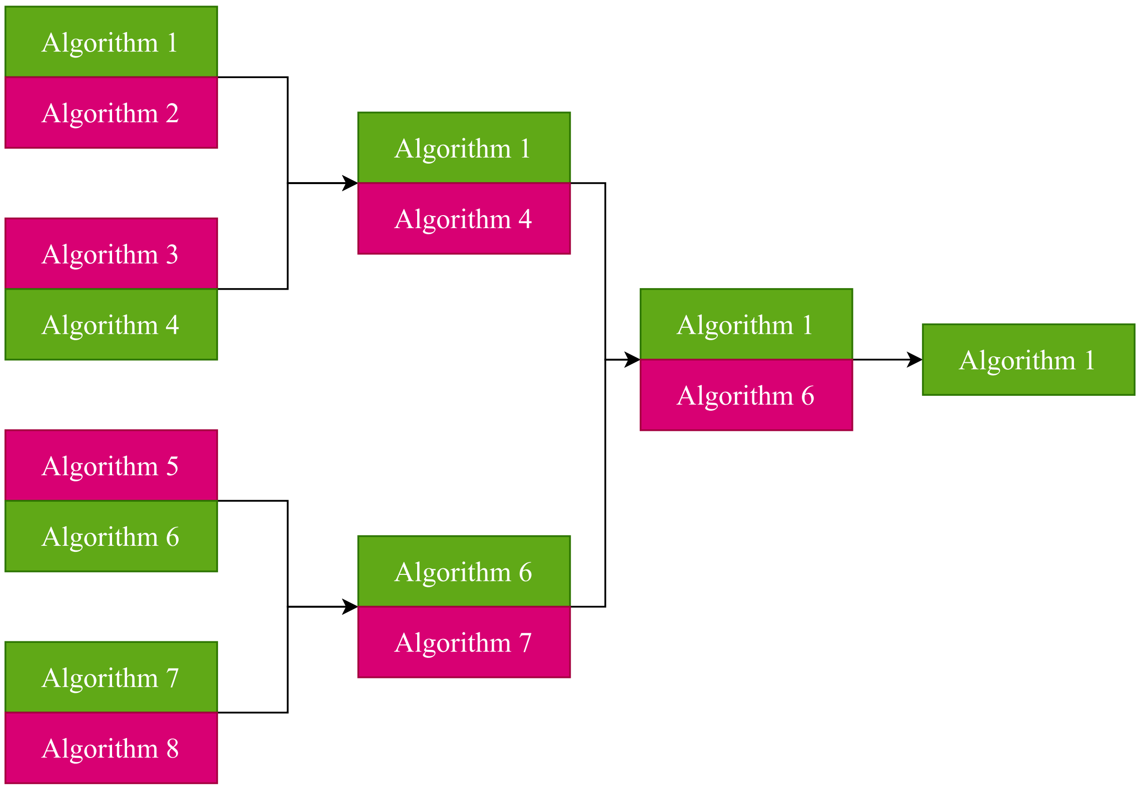


Figure 3: Minimal example of the CaiRL single-elimination tournament model

determine winners of the tournament, and the system in CaiRL enables to either end the tournament when there are winners from three brackets or match all players until there is a single winner. The Swiss tournament system is widely used in e-sports, and competitive board games [27]

6 Evaluations

6.1 Performance Evaluation

To evaluate the performance of CaiRL, we compare the classic control environments from OpenAI Gym with a similar implementation in CaiRL. We run the experiments 100 000 times averaged over 100 trials. We test the environments with graphical rendering and without graphical rendering.

Figure 6.1 clearly shows that CaiRL performs 5x faster in simulations and over 80x faster on rendering compared to the OpenAI Gym equivalent.

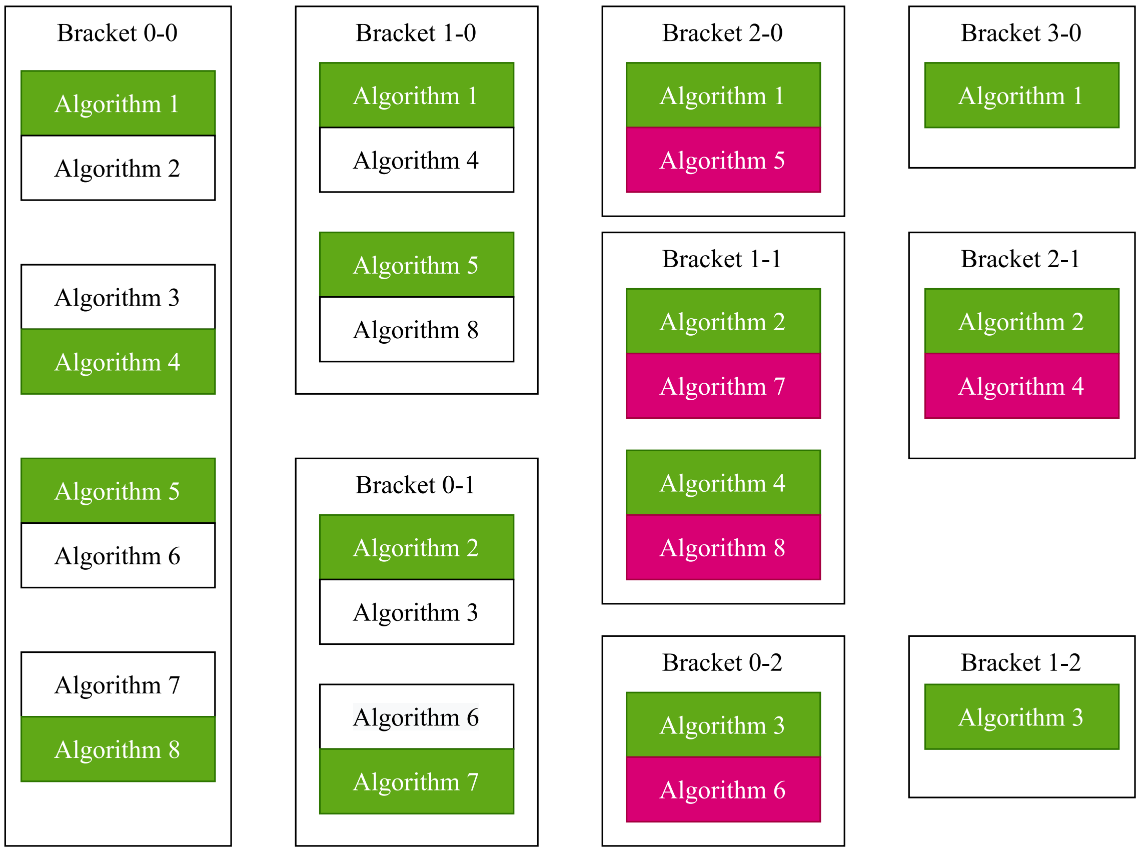


Figure 4: The picture illustrates a minimal example of the CaiRL Swiss tournament model. The green color illustrates the winner of a tournament match, while the red illustrates an eliminated player.

6.2 Algorithm Evaluation

We evaluate the DQN algorithm in the classical control environments and compare the agent performance between CaiRL and Gym. Furthermore, we test DQN in the Multitask game and report our findings. We use the standard hyperparameters from [15] and use raw images as input to the algorithm.

Figure 6.2 clearly shows that the CaiRL environment runs magnitudes faster in a training context. The algorithms are trained until mastering the task and are averaged over 100 trials. As expected, the algorithm performs similarly in both Gym and CaiRL, but in terms of sustainability, using CaiRL reduces the environmental footprint by approximately 30% for all tested environments.

As seen in Figure 6.2, the DQN algorithm successfully solves the multitask environment after approximately 1 500 000 frames averaged over ten trials. The simulation ran in 140 frames per second, which amounts to 4.6x faster execution than real-time. Each training trial took approximately 6 hours to finish, and in total, the experiment lasted for 60 hours.

6.3 Carbon Mission Evaluation

In this section, we aim to answer the following question: *Is CaiRL a better alternative for lowering carbon emissions in RL.* To begin answering this question, we rerun experiments with the novel experiment-impact-tracker from [13]. The experiment-impact-tracker is a drop-in method to track energy usage, carbon emissions, and compute utilization of the system and is recently proposed to encourage the researcher to create more sustainable AI. Our experiments run a DQN agent on the classical control environment CartPole-v1 in CaiRL and OpenAI Gym. We compare the toolkits using the console-only version and the graphical variant. We use the following environment configurations and DQN parameters:

Table 1: The DQN hyperparameters for the carbon emission experiment

Hyperparameter	**Value**
Discount	0.99
Units	32, 32
Activation	elu
Optimizer	Adam
Loss Function	Huber
Batch Size	32
Learning Rate	3e-4
Target Update Freq	150
Memory Size	50 000
Exploration Start	1.0
Exploration Final	0.01

The experiment runs for 1 000 000 times in the console version and 10 000 times for the graphical version[2]

Table 2: The table descripts the total carbon emission values and power consumption used during the experiments. The carbon emission is measured in CO2/kg and power draw is measured in milliwatt-hour (mWh).

Measurement	Environment	CaiRL	Gym	Ratio
CO2/kg	Console	**0.000014**	0.000067	20.8955
CO2/kg	Graphical	**0.000051**	0.075265	147578.431373
Power (mWh)	Console	**0.000319**	0.001483	21.5104
Power (mWh)	Graphical	**0.001131**	1.673959	148006.9849

Table 2 shows that CaiRL has a considerably lower carbon emission compared to OpenAI Gym. In the console variant, CaiRL has 20.89x less carbon emission compared to Gym.

[2]The experiments code be accessed at `https://git.io/JEPzR`.

The graphical experiment shows a more significant difference with over 147578x less carbon emissions. The reason OpenAI Gym has high emission rates is that it is locked to capturing images from the game window. We measure the emissions by subtracting the DQN time usage with the total time to only account for the environment run-time costs.

7 Conclusion

CaiRL is a novel platform for RL and AI research and aims to reduce program execution time for experiments to reduce budget costs and the carbon emission footprint of AI. CaiRL outperforms OpenAI Gym implementations significantly while also being compatible with existing OpenAI Gym experiments. We have demonstrated how CaiRL is used and outlined recommendations for defining a graphical interface for games to reduce the rendering time. Furthermore, we have illustrated that CaiRL supports many programming languages, including C++, Java, Python, and ActionScript 2 and 3. CaiRL supports over 1000 games in ActionScript, Several C++ games, MicroRTS, and Showdown in Java and supports building python games out of the box. In the evaluations of CaiRL, we demonstrate superiority in performance and positively impact the carbon footprint of AI.

8 Future Work

This paper has presented CaiRL, a reinforcement learning toolkit for running a wide range of environments in a unified framework.

CaiRL is an ambitious project to improve the tools required to conduct efficient reinforcement learning research. In fulfilling its role, the complexity of the toolkit demands extensive testing and verification to ensure that all experiments are run deterministically and provide reliable results. While CaiRL is now released, we intend to work towards improving the framework in several ways. For the continuation of this project, we intend to

- Improve documentation of the framework to increase ease-of-use
- Build easier to use Java, Python, and Flash support modules so that it requires less work for researchers to add new games
- Add support for other programming languages, for instance, Rust and LUA.
- Increase awareness of CaiRL and create an interactive scoreboard and discussion channel for environment and algorithms, much like the efforts of OpenAI Gym

References

[1] F. Berkenkamp, M. Turchetta, A. P. Schoellig, and A. Krause, “Safe Model-based Reinforcement Learning with Stability Guarantees,” in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Long Beach, CA, USA: Curran Associates, Inc., may 2017, pp. 908–918. [Online]. Available: https://papers.nips.cc/paper/6692-safe-model-based-reinforcement-learning-with-stability-guarantees

[2] W. C. Cheung, D. Simchi-Levi, and R. Zhu, “Reinforcement Learning for Non-Stationary {M}arkov Decision Processes: The Blessing of ({M}ore) Optimism,” in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, H. D. III and A. Singh, Eds., vol. 119. PMLR, 2020, pp. 1843–1854. [Online]. Available: https://proceedings.mlr.press/v119/cheung20a.html

[3] S. Padakandla, P. K. J., and S. Bhatnagar, “Reinforcement learning algorithm for non-stationary environments,” *Applied Intelligence 2020 50:11*, vol. 50, no. 11, pp. 3590–3606, jun 2020. [Online]. Available: https://link.springer.com/article/10.1007/s10489-020-01758-5

[4] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver, “Mastering Atari, Go, chess and shogi by planning with a learned model,” *Nature 2020 588:7839*, vol. 588, no. 7839, pp. 604–609, dec 2020. [Online]. Available: https://www.nature.com/articles/s41586-020-03051-4

[5] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver, “Grandmaster level in StarCraft II using multi-agent reinforcement learning,” *Nature 2019 575:7782*, vol. 575, no. 7782, pp. 350–354, oct 2019. [Online]. Available: https://www.nature.com/articles/s41586-019-1724-z

[6] C. Yu, J. Liu, and S. Nemati, “Reinforcement Learning in Healthcare: A Survey,” *arxiv preprint arXiv:1908.08796*, aug 2019. [Online]. Available: https://arxiv.org/abs/1908.08796

[7] Y. Li, “Deep Reinforcement Learning: An Overview,” *arxiv preprint arXiv:1701.07274*, jan 2017. [Online]. Available: http://arxiv.org/abs/1701.07274

[8] Y. P. Pane, S. P. Nageshrao, J. Kober, and R. Babuška, “Reinforcement learning based compensation methods for robot manipulators,” *Engineering Applications of Artificial Intelligence*, vol. 78, pp. 236–247, feb 2019.

[9] D. Silver, S. Singh, D. Precup, and R. S. Sutton, “Reward is enough,” *Artificial Intelligence*, vol. 299, p. 103535, oct 2021.

[10] T. M. Moerland, J. Broekens, and C. M. Jonker, “Model-based Reinforcement Learning: A Survey,” *arxiv preprint arXiv:2006.16712*, jun 2020. [Online]. Available: https://arxiv.org/abs/2006.16712

[11] S. Raschka, J. Patterson, and C. Nolet, “Machine Learning in Python: Main Developments and Technology Trends in Data Science, Machine Learning, and Artificial Intelligence,” *Information 2020, Vol. 11, Page 193*, vol. 11, no. 4, p. 193, apr 2020. [Online]. Available: https://www.mdpi.com/2078-2489/11/4/193/htmhttps://www.mdpi.com/2078-2489/11/4/193

[12] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “OpenAI Gym,” *arxiv preprint arXiv:1606.01540*, jun 2016. [Online]. Available: http://arxiv.org/abs/1606.01540

[13] P. Henderson, J. Hu, J. Romoff, E. Brunskill, D. Jurafsky, and J. Pineau, “Towards the systematic reporting of the energy and carbon footprints of machine learning,” *Journal of Machine Learning Research*, vol. 21, no. 248, pp. 1–43, 2020.

[14] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, 2nd ed. Cambridge, MA, USA: A Bradford Book, 2018. [Online]. Available: https://dl.acm.org/doi/book/10.5555/3312046

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” pp. 529–533, dec 2015. [Online]. Available: http://arxiv.org/abs/1312.5602

[16] P. Mileff and J. Dudra, “Efficient 2D software rendering,” *Production Systems and Information Engineering*, vol. 6, no. 2, pp. 55–66, 2012.

[17] O. Mendel and J. Bergström, “SIMD Optimizations of Software Rendering in 2D Video Games,” p. 27, 2019.

[18] F. Zehra, M. Javed, D. Khan, and M. Pasha, “Comparative Analysis of C++ and Python in Terms of Memory and Time,” dec 2020. [Online]. Available: https://www.preprints.org/manuscript/202012.0516/v1

[19] P. Andersen, M. Goodwin, and O. Granmo, “Deep RTS: A Game Environment for Deep Reinforcement Learning in Real-Time Strategy Games,” in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, aug 2018, pp. 1–8.

[20] S. Ontanon, “The combinatorial multi-armed bandit problem and its application to real-time strategy games,” in *Proceedings, The Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2013, pp. 58–64. [Online]. Available: http://www.aaai.org/ocs/index.php/AIIDE/AIIDE13/paper/viewPaper/7377

[21] S. Lee and J. Togelius, “Showdown AI competition,” *2017 IEEE Conference on Computational Intelligence and Games, CIG 2017*, pp. 191–198, oct 2017.

[22] P.-A. Andersen, M. Goodwin, and O.-C. Granmo, “FlashRL: A Reinforcement Learning Platform for Flash Games,” *Norsk Informatikkonferanse*, 2018. [Online]. Available: http://arxiv.org/abs/1801.08841

[23] S. Bauer, “Simon Tatham’s Portable Puzzle Collection,” Linux User Group, Frankfurt, Tech. Rep., feb 2021. [Online]. Available: https://www.lugfrankfurt.de/talks/SGTPuzzles{_}FraLug.pdf

[24] F. Dandurand, D. Cousineau, and T. R. Shultz, “Solving nonogram puzzles by reinforcement learning,” *Proceedings of the Annual Meeting of the Cognitive Science Society*, vol. 34, no. 34, p. 6, 2012.

[25] R. S. Sutton, D. Precup, and S. Singh, “Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning,” *Artificial Intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[26] C. Hua, “The Swiss Tournament Model,” Ph.D. dissertation, University of Pennsylvania, jan 2017. [Online]. Available: https://repository.upenn.edu/wharton{_}research{_}scholars/149

[27] L. Csató, “On the ranking of a Swiss system chess team tournament,” *Annals of Operations Research 2017 254:1*, vol. 254, no. 1, pp. 17–36, feb 2017. [Online]. Available: https://link.springer.com/article/10.1007/s10479-017-2440-4

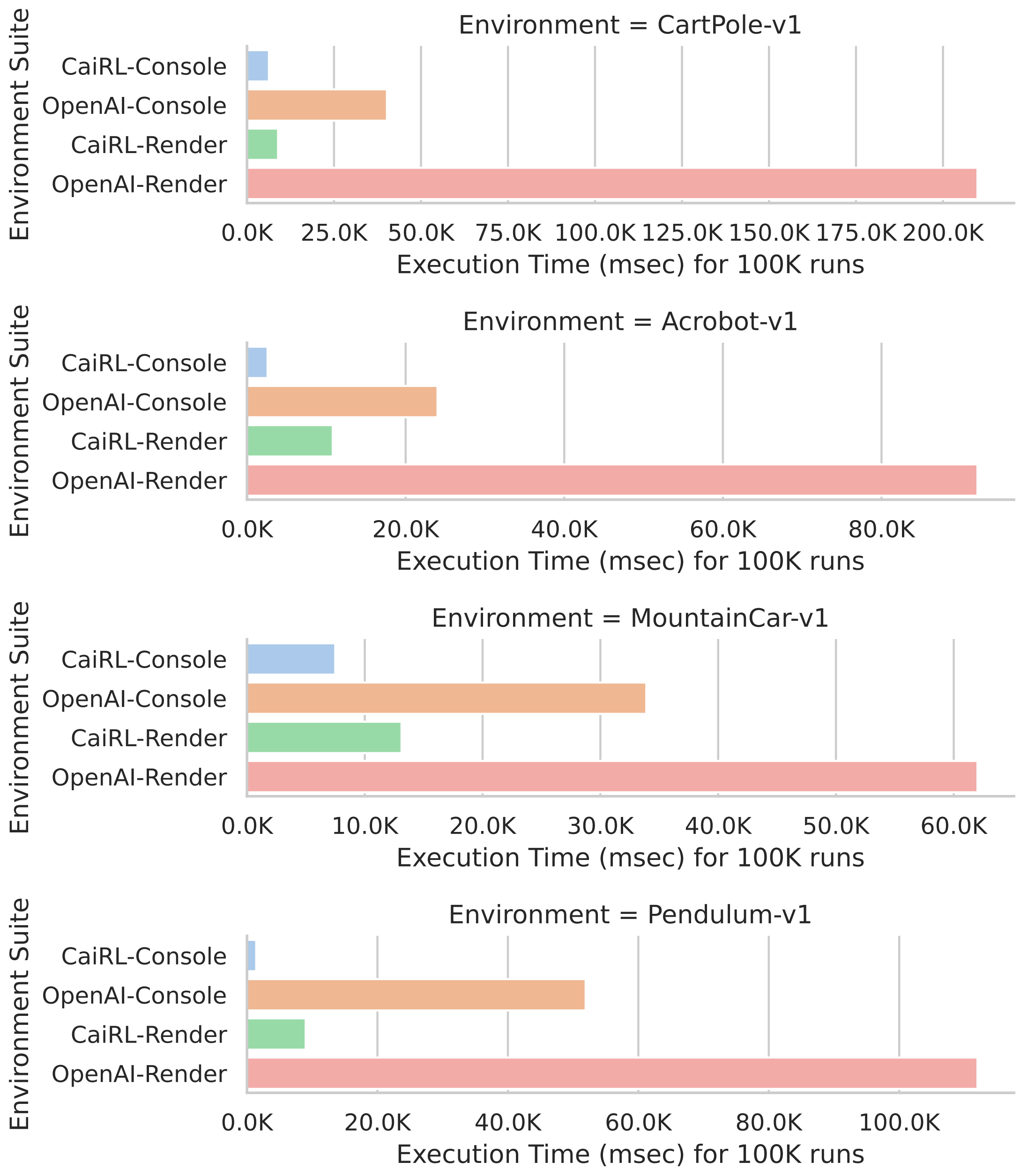


Figure 5: Performance evaluation between CaiRL and OpenAI Gym in the classical control tasks. The figure clearly shows that CaiRL uses significantly less time running the experiments. The figure illustrates how much time the environment toolkit uses to execute 100 000 episodes of the game.

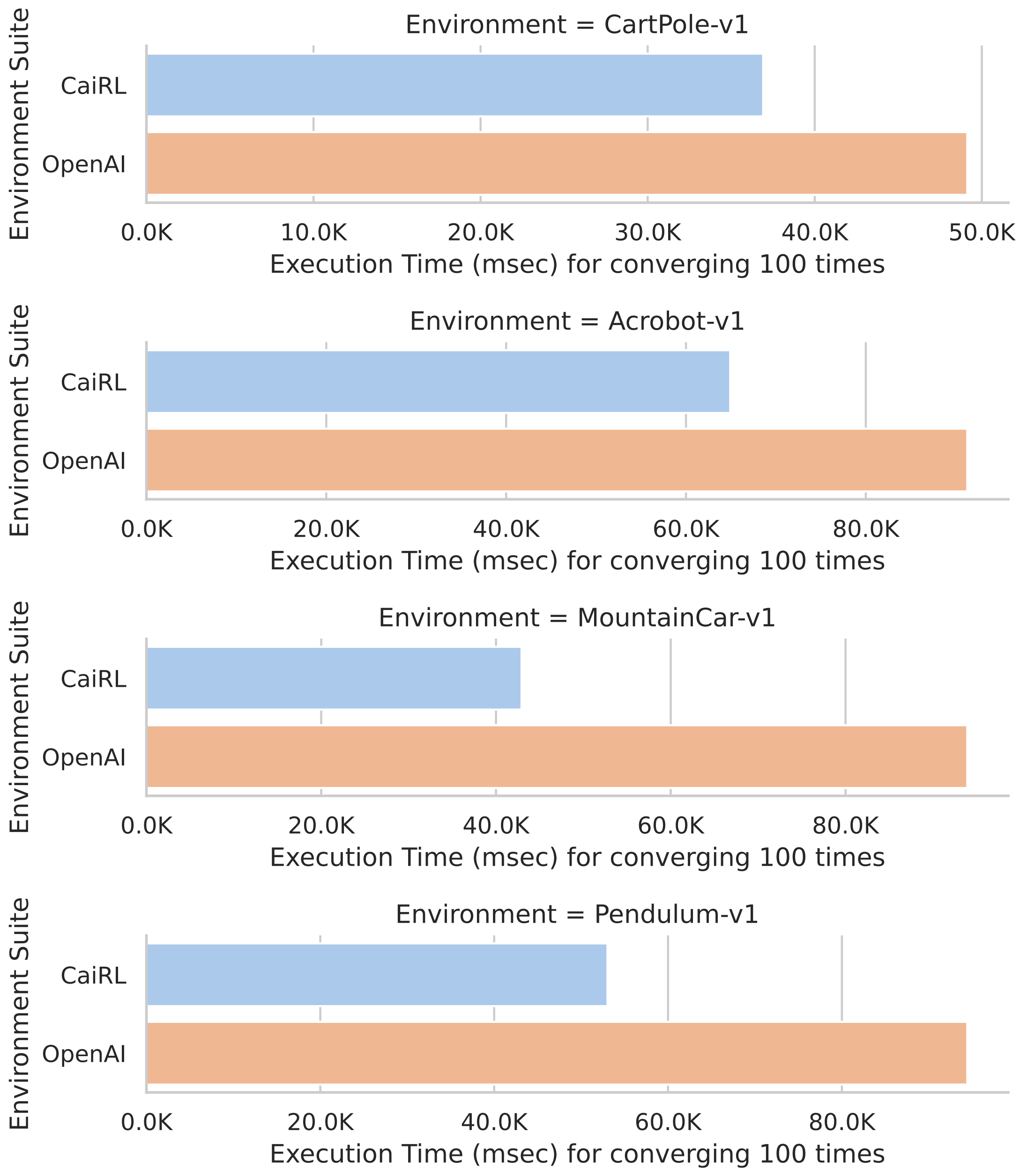


Figure 6: DQN tested in all classical control environments. The CaiRL environment toolkit clearly reduce the wall-clock execution time which in turn, reduce the environmental footprint.

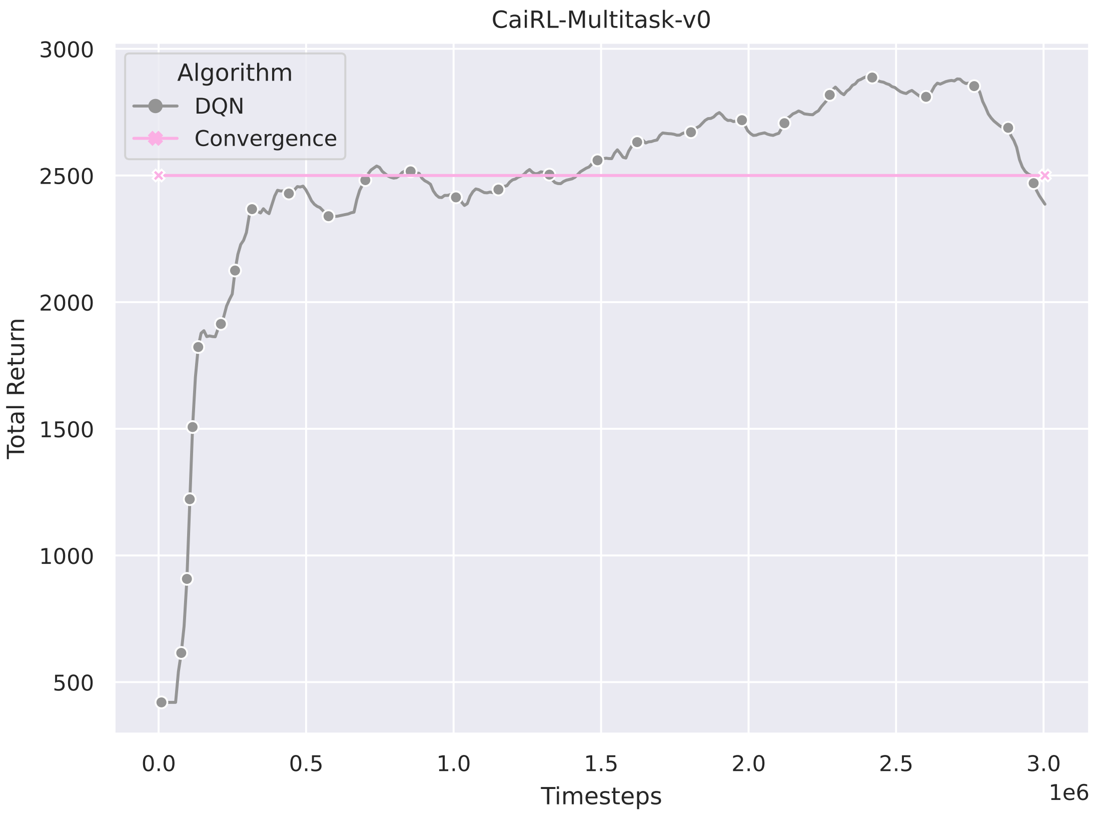


Figure 7: DQN performance in the Multitask environment. The algorithm solves the environment after approximately 3 000 000 frames where the training procedure is averaged over 10 trials.

Paper L: CaiRL: A High-Performance Reinforcement Learning Environment Toolkit

Paper M

Title:	Towards Safe and Sustainable Reinforcement Learning for Real-Time Strategy Games
Authors:	Andersen, Per-Arne et al.
Affiliation:	Department of ICT, University of Agder, Grimstad Norway
Journal:	In Review: Artificial Intelligence . Impact factor in 2020: 9.088
Year:	2021

M

Towards Safe and Sustainable Reinforcement Learning for Real-Time Strategy Games

Per-Arne Andersen
Department of ICT
University of Agder
Grimstad, Norway
per.andersen@uia.no

Morten Goodwin
Department of ICT
University of Agder
Grimstad, Norway
morten.goodwin@uia.no

Ole-Christoffer Granmo
Department of ICT
University of Agder
Grimstad, Norway
ole.granmon@uia.no

Abstract

The combination of Deep Neural Networks and Reinforcement Learning, namely Deep Reinforcement Learning (DRL), shows continued success in solving complex problems across many areas such as medicine, industry, and game playing. Perhaps most notably, DRL has shown outstanding performance in advanced Real-Time Strategy (RTS) games such as StarCraft II and Dota 2, arguably the most challenging games used in RL literature to date. RTS games have highly uncertain environments with little observable information, making them perfect for evaluating the robustness and safety of RL algorithms.

There are, however, still significant limitations with DRL algorithms in the literature, such as an ever-increasing computational cost and little focus on safety-aware approaches. Most published algorithms are computationally expensive to train, making it near-impossible to retrain without significant resources. A consequence is an ever-increasing gap between state-of-the-art algorithms trained on supercomputers and algorithms trained on commodity hardware. Training these computationally intensive DRL algorithms impacts CO2 emission significantly, which is arguably not sustainable.

Elsevier: Journal of Artificial Intelligence
Impact Factor: 9.088 (2020)
Submitted 15. Oct 2021, In Review.

A majority of RL algorithms in the literature are risk-neutral, are demanding to deploy in safety-critical systems, and can result in catastrophic failures. While there are some efforts to improve RL safety, few methods transfer well from theoretically verified scenarios to complex real-world autonomous systems validated experimentally.

This article presents a novel model-based DRL approach for tackling complex environments with the aim to reduce the need for failures during training. Specifically, our approach demonstrates successful learning while still considering robust safety awareness, minimizing risk, and reducing computational costs compared to model-free RL methods.

Our approach, the **Safe O**bservations **R**ewards **A**ctions **C**osts **L**earning **E**nsemble (S-ORACLE), is empirically verified in multiple complex and uncertain game enviroments: Deep RTS, ELF: MiniRTS, MicroRTS, Deep Warehouse, and StarCraft II, outperforming state-of-the-art model-free and model-based approaches.

Keywords: Reinforcement Learning, Markov Decision Processes, Neural Networks, State-space models, Model-based Reinforcement Learning, Risk-aware RL, Mission-critical safety

1 Introduction

The desired property of artificial intelligence is its ability to solve complex real-world challenges. Reinforcement Learning (RL) is a field of study which concerns agents that ought to do sequential decision making in a dynamic system or environment. RL takes inspiration from nature and the art of learning through trial and **error** and seeks to learn optimal behavior policy adjusting through feedback after each consecutive action. The theoretical background of RL stems from Markov chains, with augmentations to form Markov decision processes (MDP) [1]. This mathematical framework describes the probability of transitioning from one state to another given a decision. However, because RL aims to maximize rewards long-term, the likelihood of entering catastrophic states increases dramatically. RL classifies into two categories of algorithms. [2]

• **Model-free RL (MFRL)** algorithms aim to find the behavior policy without a known transition probability distribution and the corresponding reward function but instead learn through balancing the exploration/exploitation dilemma under the assumption of an unknown MDP [2].

• **Model-based RL (MBRL)** aims to learn its policy through a known model of the environment. It consists of a state transition probability function and a reward function and is referred to as a dynamics model in the literature. A dynamics model is either known *a priori* or learned through estimations based on observations and interactions with the environment. In real-world and complex computational tasks, the complete state information is rarely available, and hence, the estimated dynamics models often inherit the uncertainty about missing information. Model-based RL is shown to have better sample efficiency and more flexibility for safe RL, which are appealing traits for mission-critical and sustainable applications. [3]

• **Safety** is a central problem to RL because algorithms deployed in the real world risk damaging equipment or humans during the learning process. This gives us clear motivation to address these shortcomings in the realm of **safe reinforcement learning** (safe RL) [4]. Safe RL aims to learn a policy that maximizes rewards while also maintaining safety. There are several directions in the literature towards safer RL, and we discuss this more closely in Section 3.

• **Sustainability** meets the needs of the present without compromising the ability of future generations to meet their own needs. Machine Learning has a notorious reputation for an ever-increasing computational cost, demonstrated well through reports of state-of-the-art results using millions of dollars in computing to achieve high accuracy [5]. The computation growth is not sustainable and produces a high amount of climate emission gasses during experiments, negatively impacting the environment. However, recent literature from Henderson et al. proposes a systematic method of reporting climate emission for experiments that aim to reduce the overall climate footprint of machine learning research. [6] To the best of our knowledge, this is the first article presenting safe reinforcement learning that systemically reports climate emission, detailed in Section 6.3.

• **Recent literature** shows that RL can perform at a superhuman level several complex problems [7] such as robotic control [8]; optimizing portfolio and performing algorithmic trading in stock markets [9]; database optimization [10]; planning and path optimization [11]; medical applications such as breast cancer classification and biological sciences [12, 13]; autonomous control in vehicles [14, 15]. Much of the recent success of applied RL stems from research into solving complex game environments. Recent literature shows that RL algorithms can learn purely from pixels, such as DQN in the Atari 2600 environment suite [16], mastering board games such as Chess and Go using a combination of tree-search and Deep RL [17, 18]. RL can likewise learn highly complex games such as Real-Time Strategy (RTS) games, a highly complex stochastic environment with near-infinite state spaces and action spaces [5]. While the mentioned studies perform well according to a reward signal, no approach exists to maximize **safety**, minimize **risk**, or address the unsustainable computational costs during training and inference, which real-world applications often demand.

1.1 Challenges

Although there is steady advancement in solving complex tasks using RL, there are many challenges left unchecked in RL algorithms for the broad adoption of industry, real-world, and safety-critical applications. These challenges include:

- safe learning without *a priori* knowledge [4],
- sample efficiency [2],
- high carbon emission footprint [6],
- exploration (trial and error)-exploitation dilemma [2],
- challenging to design reward structures [19],
- hyperparameter-sensitivity [20],
- and a clear gap between affordable RL and state-of-the-art [21]

It is well understood that RL requires millions of training samples to converge in complex tasks, and hence it naturally follows a high cost of computational power. The high computational power increases the demand for hardware which directly impacts the climate footprint negatively [6]. Furthermore, it is seldom that an RL algorithm functions well without extensive hyperparameter tuning and careful reward design; hence requires several training sessions to find a composition that yields an excellent behavioral policy. Combining these problems makes it difficult to reproduce algorithms and makes RL less suitable for applied intelligence in the industry. The consequence is that most RL research concerns risk-neutral algorithms with the primary goal of performing well in simulated environments, with little regard for applicability in safety-critical systems. Lastly, there is a significant gap in the literature where prior state-of-the-art solved simple RTS game setups such as MicroRTS [22], and DeepRTS [23] wherein contrast, the latest state-of-the-art beats the world champion in Dota 2 [24] and achieves Grandmaster rank in Starcraft II [5]. We believe that the research in RTS games is far from solved and by no means mastered, considering that recent state-of-the-art does not transfer well to real-world applications and is not safe, affordable, or sustainable in long-term operations.

1.2 Motivation

Solving a complex game environment is an exciting feat because it illustrates that RL gradually matures towards the ultimate challenge of solving advanced and complex real-world problems in a safe manner[21]. RL is still in its infancy, and while many fundamental problems are solved, there is still room for improvement in sample efficiency, safety, and generalization [7]. In the interim, RTS games are an appealing platform to benchmark

Table 1: RTS games share many characteristics with Real-World applications. The table draws parallels between central challenges in Real-world and RTS games.

Challenge	**Real World**	**RTS Game**
Reward Sparsity	Taking actions towards less climate emissions are rewarded several years later. [6]	Immediate actions are rewarded when a game ends and not after each consecutive action [19]
Partial Observability	Autonomous driving having an unknown number of participants in the traffic with the uncertainty of which actions other cars will make. [14]	Parts of the game are hidden in ”Fog of War”. The opponent makes changes to the environment without the agent knowing. [22]
High Dimensionality	Data from patients with thousands of dimensions (features) or high-resolution images from cameras [13]	RTS games have large maps (e.g., 256x256) with the ability to have 500 units with 100 different actions at each timestep. [5]
Planning over long time horizons	Autonomous driving requires planning from the start point to the destination point [12]	To succeed in RTS games, the player must plan ahead hundreds of time steps to beat the opponent.[5]
Learning Safely	Learning to drive a car happens under the supervision of an experienced (arguably expert) driver, and no accident must occur during the learning phase or after learning to drive the car.	Many RTS games have environmental damage that regresses the agent’s progress. The agent must learn to avoid this damage, preferably without damage ever occurring. [20]

algorithms because they share similar characteristics to real-world problems [25]. Table 1 demonstrate common characteristics between RTS games and real-world dynamics. It falls natural to use RTS games as a platform for researching task solvability to find techniques applicable in safety-critical real-world applications [26]. The hope is that RL can play a substantial role in solving safety-critical real-world applications through learning problems virtually. These problems range from minor optimizations such as vehicle routing to major problems such as driving autonomously in traffic and optimizing the climate emissions of the engine exhaust system [27].

1.3 Contributions

The contribution of this article is composed of four highlights:

1. This work presents the **Safe O**bservations **R**ewards **A**ctions **C**osts **L**earning **E**nsemble (S-ORACLE), a novel model-based RL approach that aims towards **safer**, more sustainable, and more efficient performance. The algorithm uses an ensemble of originally risk-neutral model-free approaches to make decisions motivated using risk-averse reward signals and is trained purely on a model learned on a fraction of samples compared to only using model-free training.[1]

2. We perform a thorough performance evaluation of the proposed algorithm in RTS problems against well-known model-free approaches such as DQN, PPO, A2C, and RAINBOW in six environments; Deep RTS, Deep Line Wars, ELF: MiniRTS, StarCraft II, Deep Warehouse, and MicroRTS. The work presents baseline results of obtained performance in the tested environments and extensively evaluates the carbon emissions budget for used algorithms in the tested environment.

3. Safety of the proposed method is evaluated in the Deep RTS Lava environment and Deep Warehouse environment, an industry-like grid-warehouse simulator. The work demonstrates that the proposed algorithm outperforms risk-neutral RL algorithms while also being significantly more sample efficient.

4. Lastly, safety is discussed in context to RL, what the limits are and possible approaches towards truly generalizable safe algorithms, and concludes the findings of this work.

1.4 Outline

This article is organized in the following manner. Section 2 thoroughly details the theoretical background of S-ORACLE. It details Markov decision processes, model-free RL,

[1]The source code is located at `https://github.com/s-oracle/s-oracle`

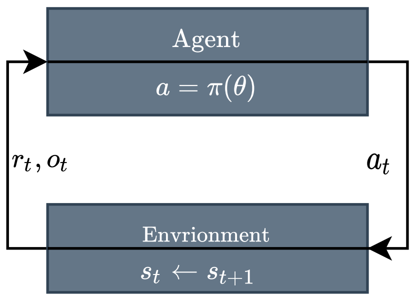


Figure 1: The agent-environment synergy in a Markov decision process [2]. The agent makes actions in the environment, which transitions the environment to the next state. The agent observes the new state with the corresponding reward signal.

model-based RL, and risk-aware RL techniques. Section 3 outlines a selection of fundamental research in the field of safe RL and explains the concepts thoroughly. Section 4 presents the RTS and safety-critical environments used in the evaluation of S-ORACLE and summarizes the configurations utilized during experiments. Section 5 presents the S-ORACLE algorithm and details the algorithm thoroughly seen from the perspective of safety and performance. Section 6 details the empirical evaluation of S-ORACLE in comparison to other state-of-the-art algorithms. Specifically, the Section presents safety evaluations and performance evaluations. Lastly, we report and document the sustainability in terms of computational costs and climate emissions. Section 7 reflects on performance and safety and possibilities of safety-aware RL. Finally, Section 8 concludes the presented work and lays a path for future research.

2 Background

2.1 Markov Decision Processes

The essence of Reinforcement Learning is learning machine algorithms to make a sequence of decisions in a dynamic system. Markov decision processes (MDP's) are a mathematical framework that defines a class of stochastic sequential decision processes and are the fundamental building block of RL algorithms. In this article, we are concerned with games with a finite number of states and actions, and hence, we consider finite MDP's. The MDP framework is visualized in Figure 1 as a discrete-time sequential process of making decisions and then observe o_t with its corresponding reward r_t. The observation is either fully-observable such that $o_t = s_t$ or partially-observable such that $o_t \neq s_t$. [28]

Definition 1 *An MDP model is expressed as a tuple* $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ *where* $\mathcal{S}$ *is the state space,* $\mathcal{A}$ *is the action space available to the agent at every time-step,* $\mathcal{P}$:

$\mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \rightarrow [0,1]$ is the transition function, $\mathcal{R} : S \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function, and $\gamma \rightarrow [0,1]$ is the discount-factor [29].

The transition function $\mathcal{P}$ with four-arguments and describes the probability of transitioning from a particular state s to s' with the corresponding reward r after taking action a. The transition function contains all information about the MDP,

$$\begin{aligned} \mathcal{P}(s', r|s, a) &= Pr\left[S_{t+1} = s', R_{t+1}|S_t = s, A_t = a\right] \\ &= \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) = 1, \end{aligned} \tag{1}$$

where $\mathcal{P}$ is defined for the next state s', $\forall s \in \mathcal{S}$, and $\forall a \in \mathcal{A}(s)$. From the four-argument transition function, we can derive a state-transition function $\mathcal{T} : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \rightarrow [0,1]$ and a reward function $\mathcal{R} : S \times \mathcal{A} \rightarrow \mathbb{R}$ for all state-action pairs. The state-transition function is defined,

$$\begin{aligned} \mathcal{T}(s'|s, a) &= Pr\left[S_t = s'|S_{t-1} = s, A_{t-1} = a\right] \\ &= \sum_{r \in \mathcal{R}} \mathcal{P}(s', r|s, a), \end{aligned} \tag{2}$$

where the probability of entering the next state s' is dependent on the current state s and the taken action a. The reward function is defined,

$$\begin{aligned} \mathcal{R}(s, a) &= \mathbb{E}\left[R_{t+1}|S_t = s, A_t = a\right] \\ &= \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} \mathcal{P}(s', r|s, a), \end{aligned} \tag{3}$$

where $\mathcal{R}$ is the expected reward that the agent receives after making action a to transition to state s'. An environment has the Markov property if it is possible to predict the next state and expected next reward given only the current state and action. The Markov property exists for a MDP only if,

$$\begin{aligned} &Pr\left[S_{t+1} = s', R_{t+1} = r|s_t, a_t, \ldots s_0, a_0\right] \\ &= Pr\left[S_{t+1} = s', R_{t+1} = r|s_t, a_t\right]. \end{aligned} \tag{4}$$

For complex game environments or real-world applications, the Markov property is rarely present because full observations of the system dynamics are hidden ($o_t \neq s_t$), making it difficult to characterize the next state given the current observed state fully. [30].

2.2 Partially Observable MDP

In complex games such as Starcraft II and Dota 2, only partial information of the state is accessible and, therefore, the agent must use that partial observation or construct beliefs of what the true state may be. Partially Observable Markov Decision Processes (POMDP)'s are a generalization of MDPs that accounts for partial observability of the state and is

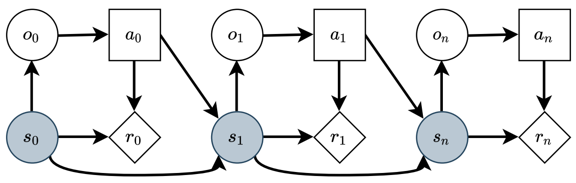


Figure 2: A POMDP system. o, a, s, r denotes observations, actions, states, and rewards, respectively. The agent can only observe $o_0 \dots o_n$ after making an action where the underlying state $s_t \dots s_n$ is hidden from the agent.

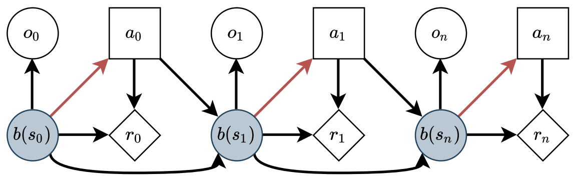


Figure 3: A Belief MDP. b, a, s, r denotes belief-state distributions, actions, states, and rewards, respectively. In contrast to POMDP's, the underlying state s is replaced with a belief state distribution. States sampled from the belief state distribution are used to make decisions in the MDP, which has similar traits to fully observable MDP's.

therefore widely used in literature to formalize optimization problems in game environments. POMDP is defined as a tuple $\mathcal{M}_{\mathcal{POMDP}} = \langle \mathcal{S}, \mathcal{A}, \Omega, \mathcal{P}, \mathcal{R}, \mathcal{O}, \gamma \rangle$ of three sets and three functions. The definition is similar to regular MDP's but extends with a set of observations $\Omega = \{o_1, o_2, \dots, o_n\}$ and the agent perception model $O : \mathcal{S} \times \mathcal{A} \rightarrow \Pi(\Omega)$ where $\Pi(\Omega)$ represent the probability distribution on Ω. [31]. Figure 2 illustrates a POMDP where the agent only can make actions a_t that are dependent on observations o_t. The problem with observations is that they may not capture the necessary information to succeed in the environment. For example, if we consider a game of pong where the observation is a pixel image, the agent cannot determine the direction or velocity of the ball. To alleviate these problems, the agent can make decisions based on a history of observations. However, keeping a history of prior observations is infeasible for games with near-infinite states and hence, is not a sufficient method for complex games.

2.3 Belief MDP

Another approach to capture information from history is to encode observations into a belief state using belief MDPs. A belief state b is a summarization of previous observations into a probability distribution over all states $s \in \mathcal{S}$, where $b(s) = Pr(s_t | o_{1 \dots t})$, represents the probability that the environment is in state s [32]. Given that we have an initial belief state b_0, we can compute belief states,

$$\begin{aligned} b'(s') &= P(s'|o,a,b) \\ &\propto P(o|s',a,b)P(s'|a,b) \\ &\propto O(o|s',a)P(s'|a,b) \\ &\propto O(o|s',a)\sum_{s\in\mathcal{S}} P(s'|a,b,s)P(s|a,b) \\ &\propto \underbrace{O(o|s',a)}_{\text{Observation Model}} \sum_{s\in\mathcal{S}} \underbrace{\mathcal{T}(s'|s,a)}_{\text{State-Transition}} \underbrace{b(s)}_{\text{belief}}, \end{aligned} \tag{5}$$

which is a sufficient statistic for $b_t \equiv o_{1\ldots t}$. The reward function $R(s,a)$ requires hidden-state information, but we can introduce belief states such that $R(b,a) = \sum_{s\in\mathcal{S}} b(s)R(s,a)$. The belief state distribution b can memorize historical observations and we assume that this is enough information to represent the hidden-state s_t so that we can treat the POMDP similarly to fully-observable MDP's, seen in Figure 3. [31, 33]

2.4 Reinforcement Learning

Analogous to human intelligence, a behavior policy is considered the brain of the algorithm and expresses parameters for a function that aims to behave optimally in a given problem. This Section is dedicated to explain classical RL commonly uses tables or simple function approximators to parameterize the behavior policy. These methods are shown to work well for simple problems, but for games or real-world applications, both fall inadequate because of large state and action spaces. [2]

The ultimate goal of a reinforcement learning agent is to find the optimal policy π^*. A policy represents a mapping from a state observation to action probabilities $\pi(a|s)$. RL algorithms categorize into three learning types; value-based, policy-based, and a combination called actor-critic-based algorithms, where our work is based on value-based approaches. There are also variations of on-policy and off-policy where off-policy algorithms can learn using historical data. Lastly, there are model-based and model-free algorithms where model-based algorithms learn using a predicted model or a known environment model, in contrast to model-free algorithms that learn solely by trial and error in an unknown environment. An RL problem is commonly modeled as an MDP and usually have an update procedure as follows:

1. Read current observation s_t
2. Make a decision based on observation $\pi(a|s_t)$
3. Receive reward r_t
4. Update policy estimates with a learning algorithm. Go back to step 1.

The optimal policy π^* is the policy in policy-space that maximize the return,

$$\pi^* = \arg\max_{\pi \in \Pi} \mathbb{E}\left[G|\pi\right] \tag{6}$$

where the return G_t,

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \\ &= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \end{aligned} \tag{7}$$

is the cumulative discounted return. The discount factor $0 \leq \gamma \leq 1$ quantifies the importance between immediate rewards and distant rewards where $\gamma = 0$ considers only immediate rewards and $\gamma = 1$ weights immediate and distant rewards equally. [2]. Perhaps the most central Equation of RL is the Bellman Expectation Equations,

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi\left[G_t|S_t = s\right] \\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s)\right] \end{aligned} \tag{8}$$

where $v_\pi(s)$ quantifies how good it is for the agent to be in state s_t following policy π henceforth [15]. The Bellman Equation, famously from dynamic programming, defines a recursive function that expresses the relationship between the value of a state and the successor state [34, 2]. However, the state-value function is not practical when quantifying how good actions are in state s_t because it summarizes all actions. The state-action value function $Q : S \times A \to \mathbb{R}$ quantifies how good action a_t being in state s_t. The state-action value function,

$$\begin{aligned} &Q_\pi(s, a) \\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma v(S_{t+1})|S_t = s, A_t = a\right] \\ &= \mathbb{E}[R_{t+1} + \gamma \mathbb{E}_{a \sim \pi} Q(S_{t+1}, a) \mid S_t = s, A_t = a], \end{aligned} \tag{9}$$

is similar to the state-value function $Q_\pi(s, a)$ but decomposes the value-function into values for individual actions. [35]. Consequently, we know that the optimal policy π^* is found indirectly if $Q^*(s, a)$ or $V^*(s)$ is known through solving the Bellman Equation,[2] [7],

$$Q^*(s, a) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} Q^*(S_{t+1}, a)]. \tag{10}$$

2.5 Model-Based RL

Model-based RL follows the standard MDP derivation and involves learning the transition-function $\mathcal{T}$ from observed data [37]. The goal is to find some parameters θ_m

[2]RL theory has significantly more ground to cover, but we have left out non-essential parts for this article. We recommend [36] for further reading.

so that the estimated transition function $\hat{\mathcal{P}}(s', r|s, a; \theta_m) \cong \mathcal{P}(s', r|s, a)$. In this work, we learn to derive the reward-function $\hat{\mathcal{R}} : \mathcal{S} \times \mathcal{A}$ and state-transition function $\hat{\mathcal{T}} : \mathcal{S} \times \mathcal{A}$ because we aim to quantify the uncertainty of the reward function estimates for risk-sensitive RL, which is further detailed in Section 2.9. While it is common to incorporate decision-making into the model with algorithms such as cross-entropy methods (CEM), we have a model agnostic approach that allows model-free algorithms (e.g., Q-learning). Model-free approaches are more studied, hence has a significantly better performance than model-based approaches [5]. Contrary to model-free algorithms, model-based algorithms are much more sample efficient [38, 39] and in combination with a learned dynamics model, the aim is to get the best of two worlds: sample efficiency, performance, and risk-awareness incorporated together.

• **Dynamics Model.** The goal of an estimated dynamics model $\hat{\mathcal{T}} : \mathcal{S} \times \mathcal{A}$ is to learn parameters θ_m that best can reflect the behavior of the unknown dynamics $\mathcal{T}$ from Equation 2. The estimated predictive model is with this referred to as a dynamics model and is defined,

$$\hat{\mathcal{T}}(\hat{s}'|\hat{s}, a; \theta_m) \cong \mathcal{T}(s'|s, a) \tag{11}$$

where we assume that the estimated model is in some way captures information of the unknown MDP similar to HMM's [40]. There are many approaches to learning such models, and this article focuses on a combination of variational autoencoders (VAE) [41], state-space models [42], and recurrent neural networks [43].

• **Variational Autoencoder** is a generative autoencoder that uses neural networks to parametrize probability distributions used to define a probabilistic latent variable model efficiently. VAE's define a generative model and an inference network used to learn the parameters that best fit observed data. The generative model is the joint probability distribution $pr_\theta(x, z) = pr_\theta(x|z)pr_\theta(z)$ where $pr_\theta(z) = \mathcal{N}(z; 0, I)$ and the decoder is usually $pr_\theta(x|z) = \mathcal{N}(x; \mu, \sigma)$ with μ and σ being neural network estimators [41]. The inference network or the encoder allows computing a posterior approximation given a particular data point (e.g., an observation of a game). In particular, we use *amortized variational inference* that shares parameters overall observed data points [41]. The posterior approximation is defined as $p(z|x) = \mathcal{N}(z; \mu, \sigma)$ following the same principles as for the decoder. Parameter learning is performed using **E**vidence **LO**wer **B**ound (ELBO), which consists of a reconstruction term and a regularization term (Equation 24 and 26) where the first term encourages good reconstruction and the second term aims to model the data similarly to the prior (e.g., a Gaussian distribution). Intuitively, the aim is to create a generative model, meaning it is possible to reconstruct belief data over unseen data given the best estimates of data seen thus far. This fits well with a model in RL but requires adjustments to account for temporal dimensions. Motivated by the challenges of posterior collapse in VAE, Oord et al. proposed a categorical generative network, detailed in Section 5 [44].

• **State-Space Models** are particularly interesting because their background is founded in estimating trajectories such as the Apollo project in the '60s [42]. There are three types of SSMs, *filtering*, **prediction**, and *smoothing*, but we use prediction in this work and assumes the SSM model as a non-linear Gaussian function similar to VAE. The goal is to predict a future latent vector $p_\theta(z_t|b_{t-1}, a_{t-1})$ where b_{t-1} is the belief state and a_{t-1} is the action performed. Here we combine amortized variational inference from VAE's with learning a prediction model conditioned on belief states and actions from RL. [3]
• **Recurrent Neural Networks** are interesting because they aim to learn data dependencies stretched through time. While we have tested Liquid-Time Constant (LTC) networks and Gated Recurrent Units (GRU), we found that Long Short-Term Memory (LSTM) networks perform better in several tested environments. Following the theory of Belief MDP, we can learn a belief distribution with sufficient statistics of observations $b_t \equiv o_{1...t}$. We use an LSTM layer in our generative network to produce belief states b_t dependant on previous belief state b_{t-1} and action a_{t-1} but only use the notation z_t after stochasticity is added to the LSTM prediction [4].
• **To summarize**, the proposed predictive dynamics model is a probabilistic state-space model that using LSTMs to learn the belief distribution that parameterizes a Gaussian prior, learned using amortized variational inference from VAE's.

2.6 Q-Learning

Arguably the most central algorithm for fundamental RL is the Q-Learning algorithm [46]. Q-Learning is a value-based algorithm and uses the Q-function from Equation 9 as the basis for updating the policy parameter. Q-Learning is a model-free algorithm it meaning that it works independently of the underlying MDP dynamics $\mathcal{T}$ and is an off-policy algorithm, meaning that it can learn from samples collected by other policies. The Q-Learning algorithm follows the Bellman equations (9), where

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\delta_t, \tag{12}$$

where $0 \leq \alpha \leq 1.0$ is the learning rate, and δ_t is the Bellman residual,

$$\delta_t = R_{t+1} + \underbrace{\gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a)}_{\text{off-policy}} - Q(s_t, a_t). \tag{13}$$

The Bellman residual δ_t denotes the temporal-difference (TD) error between current and subsequent state estimates. This procedure is named bootstrapping, e.g., we update our estimates of $Q(s_t, a_t)$ with another estimation $\max_{a \in \mathcal{A}} Q(s', a)$ where Q-Learning assumes

[3]We recommend [41] for further reading of VAE and [45] for SSM's.

[4]The belief-state is deterministic from the LSTM, and we found the latent-space variable much better if modeling as a Gaussian distribution

best action is taken given current knowledge [47]. The max operator assumes that all future actions are optimal, which allows other policies to make decisions in an off-policy manner. It is also possible to derive another popular algorithm, SARSA (state-action-reward-state-action), by omitting the max operator such that the bellman residual,

$$\delta_t^{SARSA} = R_{t+1} + \gamma \underbrace{Q(s_{t+1}, a_{t+1})}_{\text{on-policy}} - Q(s_t, a_t) \tag{14}$$

is an on-policy TD update instead. The Q-Values are usually stored in a table in computer memory. However, it becomes infeasible to use traditional RL in larger problems because the algorithms store information in tables in the computer memory. For this reason, we use function approximators to learn a latent representation of the state-action value table.

2.7 Deep Q-Networks

Since traditional RL relies on tables to store parameters, it becomes impossible to assume an exact function to solve our policy optimization problem [2]. Instead, function approximation and specifically using neural networks is an appealing approach as they have demonstrated the capability to learn high-dimensional functions [16]. Deep Q-Networks tries to estimate the Q-table so that $Q(s, a; \theta) \approx Q * (s, a)$. Learning of the parameters θ is done through minimizing the following loss-objective,

$$\mathcal{L}(\theta_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{P}(.)} \left[(\delta_i | \theta_i)^2 \right], \tag{15}$$

where δ_i is the bellman residual from Equation 13. The first term is the reward, the second term is the greedy estimation from the target Q-network, and the third term is the inference Q-network.

2.8 Policy Gradient Algorithms

In contrast to value-based methods, **Policy Gradient (PG)** algorithms aim to find an optimal behavior policy through direct policy search. The policy is defined as a parametrized function with respect to θ and computes gradients based on an objective function,

$$\begin{aligned} J(\theta) &= \sum_{s \in \mathcal{S}} d_{\pi_\theta}(s) v_{\pi_\theta}(s) \\ &= \sum_{s \in \mathcal{S}} d_{\pi_\theta}(s) \sum_{a \in \mathcal{A}} \pi_\theta(a|s) Q_{\pi_\theta}(s, a) \\ &\propto \mathbb{E}_{s,a,r,s' \sim \mathcal{P}(.)} \left[\ln \pi_\theta(a|s) Q_{\pi_\theta}(s, a) \right] \end{aligned} \tag{16}$$

where $d_{\pi_\theta}(s)$ denotes the stationary distribution for π_θ. [48]. Updates are performed using *gradient ascent*, that is, we wish to find parameters θ for π_θ that yields the highest return, written as $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$. The gradient theorem [2] find that the gradient respect to θ of the objective function $J(\theta)$ is,

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[Q_{\pi_\theta}(s,a) \nabla_\theta \ln \pi_\theta(a|s)] \tag{17}$$

where $Q_{\pi_\theta}(s,a)$ is interchangeable with any return based function (e.g., advantage function) and $\mathbb{E}_{\pi_\theta}$ indicates the empirical average over a finite set of samples, sampled using the algorithm [49]. Many algorithms build on the policy gradient framework and perhaps most notably the Proximal Policy Optimization (PPO) for its substantial empirical performance and simplicity. PG algorithms are notoriously difficult to train because estimates have high variance and no bias. Vanilla PG is perhaps most known for this behavior, making the algorithm more susceptible to local optima and perform poorly across larger state spaces. Many algorithms build on the policy gradient framework, and perhaps the most promising direction is the Proximal Policy Optimization (PPO) family for its substantial empirical performance and simplicity [50]. We define the ratio $ro_t(\theta)$ between current policy $\pi_\theta(a_t|s_t)$ and previous policy $\pi_{\theta_{old}}(a_t|s_t)$ such that $ro_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}}$. Substituting $\ln \pi_\theta(a|s)$ in Equation 16, the objective becomes,

$$J^{CPI}(\theta) = \mathbb{E}_{s,a,r,s' \sim \mathcal{P}(.)}\left[ro(\theta) Q_{\pi_\theta}(s,a)\right], \tag{18}$$

which is the Trust Region Policy Optimization (TRPO) objective in [51]. The problem with J^{CPI} is that maximization leads to excessively large policy updates, hence learning becomes unstable. The work on PPO propose a clipping Scheme to reduce the size of policy updates,

$$J^{CLIP}(\theta) = \mathbb{E}_{s,a,r,s' \sim \mathcal{P}(.)}\left[\min(ro(\theta) Q_{\pi_\theta}(s,a), \text{clip}(ro(\theta), 1-\epsilon, 1+\epsilon) Q_{\pi_\theta}(s,a))\right], \tag{19}$$

where ϵ is a hyperparameter $0 \leq \epsilon \leq 1$, typically set in the range of $\epsilon \approx 0.2$.

2.9 Risk-Aware RL

From a traditional RL view, most algorithms are considered risk-neutral, as clearly seen in the update Equation for Q-Learning (Equation 9) and its deep learning counterpart (Equation 15.) The problem with risk-neutrality is that for safety-critical systems such as the Deep Warehouse simulator, traditional algorithms fail to learn without relying on experience from catastrophic states [20]. This work draws parallels to a safety-critical system using the Deep Warehouse and Deep RTS Lava environment[5] mini-game (Section 6), where the goal is to retrieve gold without entering catastrophic lava states.

• **Uncertainty.** One approach to safe RL is to quantify the *epistemic* or *aleatoric* uncertainty and define it as a notion of *risk*. Aleatoric uncertainty stems from the observations

[5]The Deep RTS Lava mini-game can be found at `https://git.io/JzpnJ`

done of the environment. An environment might have some inherent noise and stochasticity that is not controllable by the agent, which further amplifies belief MDP (however, the belief distribution is epistemic uncertainty) and POMDP's because of added uncertainty beyond the observable state-space. Generally, we think of aleatoric uncertainty as something that is not changeable but quantifiable. On the other hand, epistemic uncertainty is the model uncertainty, or in other words, the uncertainty of whether our prediction is correct with current knowledge. However, epistemic uncertainty is learnable and is reduced as the model becomes more certain of predictions. [52] In a stochastic process such as an MDP, several measurable uncertainties exist, such as model uncertainty, reward uncertainty, and value uncertainty [53]. **Model uncertainty** considers the uncertainty in transition function $\mathcal{P}$ of an MDP and has successfully guided agents during learning [54]. This article only consider MDP's where the transition function is unknown, and hence, we must estimate the model, which adds even more uncertainty to the model. **Reward uncertainty** defines the uncertainty of receiving a specific reward r_t at state s_t given the action made. Most notably is the intrinsic motivation from [55] that defines an auxiliary reward signal from the extrinsic reward function $\mathcal{R}$. Work from [56] similarly proposes a dynamics-based prediction error that is used as a secondary reward signal. **Value uncertainty** considers the value function $v_\pi(s)$ as a source of risk. Due to the recursive nature of the Bellman equations (Equation 8), the value-function estimates increasingly accumulate errors as time $t \to \infty$.

• **Risk-directed exploration.** Following the work of [57], risk-directed exploration is an auxiliary signal that guides action selection in RL algorithms. We quantify exploration risk $\Psi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$,

$$\Psi(s,a) = w\mathrm{H} - (1-w)\frac{\mathbb{E}[\mathcal{R}(s,a)]}{\max\limits_{a \in A} |\mathbb{E}[\mathcal{R}(s,a)]|}, \tag{20}$$

where H is the entropy,

$$\mathrm{H}(s,a) = -\hat{\mathcal{T}}(s'|s,a)\log\hat{\mathcal{T}}(s'|s,a). \tag{21}$$

The entropy H of a stochastic process is a measurement uncertainty that suits well for quantifying risk. The risk is denoted Ψ and, as seen in Equation 20, is used as a trade-off between normalized expected return and system entropy [58]. The risk is weighted $0 \leq w \leq 1$ where $w \geq .5$ values the stochastic nature of the system more as a notion of risk. Furthermore, we define a utility function $\mathcal{U} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$,

$$\mathcal{U}(s,a) = \rho(1 - \Psi(s,a)) + (1-\rho)\pi(a|s), \tag{22}$$

M

where $\leq \rho \leq 1$ controls the risk-awareness of the agent. For $\rho = 0$, risk awareness is disabled, and as $\rho \to 1$, the agent becomes increasingly aware of risk during decision making. Note that the utility function $\mathcal{U}$ is policy agnostic and works with policy-based and value-based methods, and is usable with sampling techniques such as ϵ-greedy and as a Boltzmann distribution (softmax). [57, 59]

• **Risk-Sensitive RL** refers to the branch of safe RL, which expresses a balance between a weighted risk and the return,

$$\max_{\pi \in \Pi}(\mathbb{E}_{\pi}(\mathcal{R}(s,a)) - \beta\omega). \tag{23}$$

The first term of Equation 23 is the expectation of the return (Equation 3), and the second term is the weight $0 \leq \beta \leq 1$ of the risk-function $\omega : \mathcal{R} \times \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ (note that this is omega ω not to be confused with w in Equation 20) [60]. Literature has studied different definitions of risk, such as using uncertainty from the TD-Error [61, 62], reward uncertainty [63], and using a set of error-states [60]. We use two sources of uncertainty in the system, particularly the dynamics model entropy and the variance of a set n predicted rewards, similar to [63]. This article follows the work of [64] but uses the measured variance as a risk signal for a more risk-averse agent. We combine risk-sensitive RL with Equation 20, where safety is adjusted long term with exploration [57] and short term through risk-averse weighting of the return function. We detail further our particular approach in Section 5.

• **Goal-directed RL** is not directly a technique for reducing risk but has appealing properties that reduce the probability of entering catastrophic states and hence, is used in literature towards risk reduction [65]. Goal-directed reinforcement learning (GDRL) separates the learning into two phases, where phase one aims to solve the goal-directed exploration problem (GDE). To solve the GDE problem, the agent must determine at least one viable path from the initial state to the goal state. In phase two, the agent uses the learned path to find a near-optimal path. The two phases iterate until the agent policy is converged. [66] The modeling task is to compute costs using neural network approximators that follow an *episodic* training Scheme. During exploration, the algorithm records a buffer of visited states, and at the time the agent enters a terminal state, the buffer is labeled with the corresponding Euclidean distance from the goal. The training is a supervised learning problem, and if enough data is collected, the estimator can accurately predict the distance between the current state and the terminal state. We denote the cost $c_t \in C$ in Section 5 as the *normalized* distance from the current observed state to the goal state.

3 Related Work

In the majority of established systems in the industry, an expert system made from human reasoning acts as the controller for the environment [67]. It is critical for safe and stable learning in real-world environments so that ongoing operations are not interrupted, and this Section details related work that aims towards improving and solving safe model-based RL (SMBRL).

• **Lyapunov-functions**. Perhaps the most notable recent work in solving SMBRL problems is the work of Berkenkamp et al. proposing a region of attraction-based method that is guaranteed safe constrained to a set of safe-states. The author presents a learning algorithm with two assumptions (1) the model is Lipschitz continuous, and (2) a reliable and well-calibrated statistical model exists to allow accurate exploration, close to the ground truth model function (the environment). The author adopts the region of attraction from control theory and uses Lyapunov functions that constrain the problem to an invariant set of the policy space. Furthermore, the author finds that a Lyapunov function is derivable from the value function $v_\pi(s)$, given that all rewards are positive. The experiments and theoretical justifications demonstrate that the algorithm functions well for the inverted pendulum environment and that their proposed algorithm improves performance while also holding safety constraints. [68]. Chow et al. similarly use Lyapunov functions to solve constrained MDP (CMDP) problems with the assumption of a feasible baseline policy. The author proposes updating the Lyapunov function using bootstrapping and showing that the method integrates well with Q-Learning. The author does not provide convergence proof to the optimal policy but empirically demonstrates that their approach performs better than Lagrangian methods. [69].

• **Barrier Functions.** Similar to Lyapunov-based approaches, the use of barrier function aims to constrain the policy-space to a *safe-set* of possible policies that are guaranteed to work safely. Cheng et al. propose a combination of model-based dynamics learning, shielded RL, and model-free algorithms as actors. Barrier functions are forward invariant, similar to Lyapunov functions, and uses a temperature hyperparameter to determine how constrained the policy space is. The method guarantees safety. However, it builds on the assumption that a determined set of safety policies are given before training. The authors demonstrate that their method outperforms novel model-free algorithms in sample efficiency and safety but at the cost of return performance. [70]. In a similar direction, Yang et al. propose a novel actor-critic barrier function structure for multi-agent safety-critical systems. Specifically, the authors propose a two-player game architecture for guaranteed safety during learning and solve a known model with safety guarantees [71].

• **Human Intervention.** One of the fundamental questions to raise in a safety-critical environment is *when to trust your model.* In earlier work on Human Intervention techniques, Saunders et al. propose a simple framework for training an algorithm safely using human intervention for catastrophic states. For every timestep in the environment, a human participant evaluated the state and proposed action of the algorithm, and for catastrophic actions, the human corrects the agent and gives negative feedback. While this is set in a model-free context, the human can be considered a *corrected* model of the policy space, and the authors found their method to perform well but failed to scale due to the time used by the human. [72]. Similarly, Turchetta et al. propose a curriculum-based approach, Curriculum Induction for Safe Reinforcement Learning (CISR), which assumes a teacher that intervenes in the event of catastrophic states. The teacher policy guides the student, intervenes, and puts the student in safe states if the CMDP criteria are not met. The author describes the CISR framework as a meta-learning framework where the teacher policy is an optimizable hyperparameter. However, the algorithm makes assumptions that there exists an intervention set defined before learning. The authors demonstrate that the student policy performs significantly better in the Frozen Lake environment when erroneous states are intervened compared to no intervention. [73]

• **Summary.** Throughout the last decade, many different approaches have been proposed toward improving safety in RL algorithms. While this article is out of scope to describe all methods, this Section describes promising directions in safe RL. This article draws inspiration from several other works in safe RL, which the respective authors best described in [57, 60, 61, 62, 64, 65, 53, 74], but otherwise referred to throughout this work. Furthermore, a brief overview of traditional safe-RL work is described in [4].

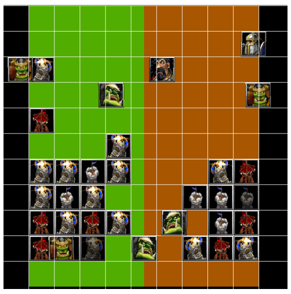
4 Environments

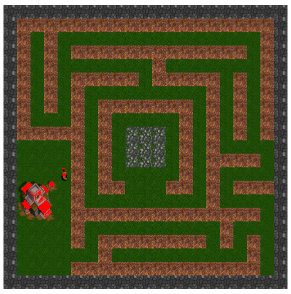
In the extreme pace that RL is moving, it is essential to use benchmarks that provide trivially replicable environment conditions. It is also important to have flexible benchmarks and can grow alongside the progress of research. Game-Complexity is a metric to determine the difficulty of a task, and while it does not account for uncertainty or stochasticity in the environment, it is often in literature. For example, Chess has state-complexity of $\sim 10^{50}$, Go (19x19) = $\sim 10^{360}$, and for StarCraft II, some estimations range from 10^{1685} to 10^{36000} [78]. As iterated, there is a clear gap in the literature of experimental environments, and the goal is that efforts such as MicroRTS [22], ELF [76], along with Deep Line Wars [75], Deep Warehouse [20], and Deep RTS [23] help fill the gap. In particular, all of these environments are flexible in that they allow to design of mini-games of nearly any state-complexity and allow configuration that tests the safety of algorithms. This Section thoroughly presents the experimental environments used in Section 6 and outlines the evaluation setup for each environment.

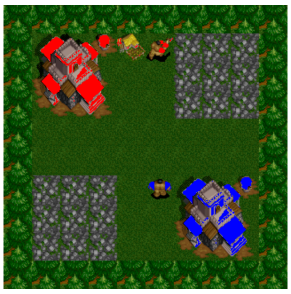
4.1 Deep Line Wars

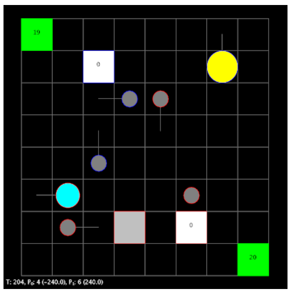
The Deep Line Wars (DLW) RTS environment provides a lightweight version of the Deep RTS game that offers a way to train algorithms on offensive and defensive strategies. Figure 4a illustrates the graphical observation, and to succeed in the DLW environment, the player should master a balance between (1) base construction, (2) economy management, (3) defensive planning, and (4) offensive evaluation.

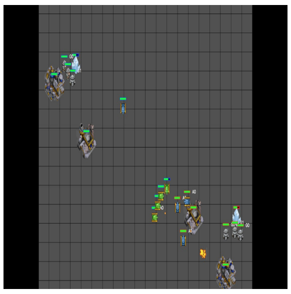
These objectives seem trivial to master individually but pose a significant challenge when combined. The DLW game is a two-player RTS game where the goal is to build a base

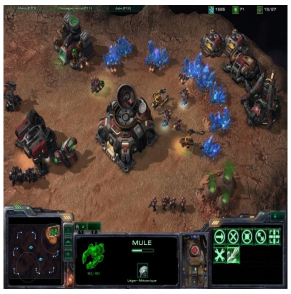

(a) Deep Line Wars [75]


(b) Deep RTS Maze Objective [23]


(c) Deep RTS One Versus One [23]


(d) MicroRTS [22]


(e) ELF: Mini-RTS [76]


(f) StarCraft II Mini-Games [77]

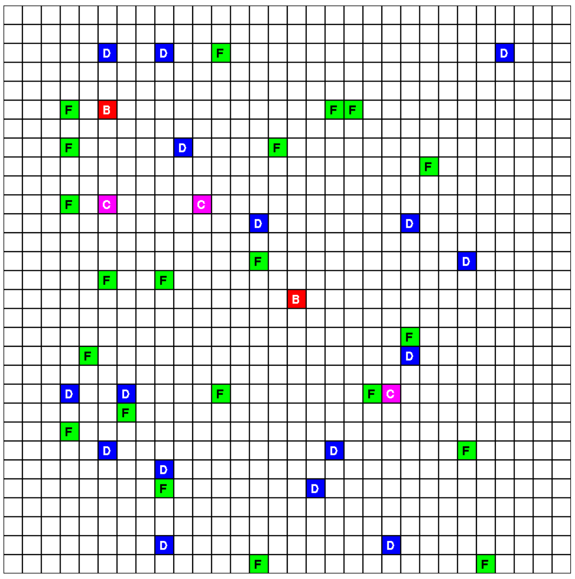

(g) Deep Warehouse. In a simple cube-based ASRS system, the environment consists of (**B**) passive and (**C**) active delivery-points, (**D**) pickup-points, and (**F**) taxis.[20]

Table 2: MicroRTS maps with the corresponding map-size.

Map	Size
basesWorkers8x8A	8×8
basesWorkers16x16A	16×16
BWDistantResources32x32	32×32
(4)BloodBath.scmB	64×64
FourBasesWorkers8x8	8×8
TwoBasesBarracks16x16	16×16
NoWhereToRun9x8	9×8
DoubleGame24x24	24×24

and send units towards your opponent's base. DWL supports several modes from fully deterministic and observable state-space to partial observable stochastic state-spaces. The observations are available as a compact vector representation and represented as pixel images [75].

4.2 Deep RTS

The Deep RTS game environment enables research at different difficulty levels in planning, reasoning, and control. Deep RTS aims to narrow the research gap between Micro RTS [79] and StarCraft II [77]. Deep RTS compose games as scenarios that define a particular goal to win the game. For the most primitive scenario, the goal is to gather a set amount of gold before the agent receives a reward signal and the game terminates. The most complex environments support observability, environmental danger, and other opponents. It is possible to define delayed actions and delayed rewards to increase the difficulty of the task.

4.3 Micro RTS

Micro RTS is a simple RTS game designed to conduct AI research. The idea behind Micro RTS is to strip away the computational heavy game logic and graphics, seen in Figure 4d, to increase the performance and enable researchers to test theoretical concepts quickly [22]. The microRTS game logic is deterministic and includes options for fully and partially observable state spaces. The primary field of research in microRTS is game-tree search techniques such as variations of Monte-Carlo tree search and minimax [80, 22, 81] but is also used for deep learning techniques, especially in the larger maps, seen in Table 2.

Table 3: List of mini-game maps in SC2LE

Map
MoveToBeacon
DefeatRoaches
BuildMarines
CollectMineralShards
CollectMineralAndGas
FindAndDefeatZerglings
DefeatBanelingsAndZerglings

4.4 ELF: Mini-RTS

The goal in Mini-RTS is for the agent to destroy the opponent's base with its troops. Players have units, resources, and a base and must balance economics for defensive and offensive planning. It is possible to expand the base with the worker unit type to build barracks and expand the offensive powers. The ELF game engine is tick-driven, meaning that the agent must make a decision per tick. Mini-RTS is partially observable due to the fog-of-war, and thus the agent is only presented with imperfect information. However, compared to StarCraft II, Mini-RTS is significantly less complex both logic-wise and graphically wise, seen in Figure 4e. The Mini-RTS environment features two built-in strategies; AI-Simple, AI-Hit-and-run where both are used in the experiments.

4.5 StarCraft II

SC2LE (StarCraft II Learning Environment) bridges the programming language Python and the StarCraft II state-space and action space. SC2LE is an initiative from Blizzard and DeepMind to demonstrate the capabilities of AI in RTS. StarCraft II is depicted in Figure 4f and is a complex environment that requires short and long-term planning. It is difficult to observe a correlation between actions and rewards due to the imperfect state information and delayed rewards, making StarCraft II one of the hardest challenges for RL algorithms to solve [77]. In addition to the full-game case, SC2LE features mini-games suitable for RTS research, as seen in Table 3.

4.6 Deep Warehouse

Training algorithms in real-world environments have severe safety challenges during training and suffer from low sampling speeds [82]. The Deep Warehouse environment features discrete and continuous action and state spaces. The environment has a wide range of configurations and is the only open-source implementation that aims to simulate proprietary automated storage and retrieval systems to the best of our knowledge[6].

In the context of warehousing, an Automated Storage and Retrieval System (ASRS) is a composition of computer programs working together to maximize the incoming and outcoming throughput of goods. Using an ASRS system in logistics has many benefits, including high scalability, increased efficiency, reduced operating expenses, and operation safety. We consider a cube-based ASRS environment which can be thought of as a 3-dimensional asymmetrical rectangle with goods stored depth-wise. Taxi agents collect and deliver goods to delivery points on the rectangle's uppermost layer and usually work alongside other agents. A computer program controls the taxi agent that reads its sensory data to determine the following action. Although these systems are far better than manual labor warehousing, there is still significant improvement potential in the current state-of-the-art. Most ASRS systems are manually crafted expert systems, which due to the high complexity of the multi-agent ASRS systems, only performs sub-optimally. [83]. The Deep Warehouse experiments use a grid-size of 11×11 with 10 agents, 22×22 with 50 agents, and 41×41 with 100 agents, respectively.

4.7 Summary

Zooming out on the landscape of RL for RTS games, it is clear that the challenge is far from mastered when considering sustainability, safety, or flexibility. There is, without doubt, a progressive pace towards methods that perform better within specific environments, but as of yet, StarCraft remains the ultimate goal for sustainable RL algorithms [48, 79]. For the experiments in Section 6, we use the following environments:

- **Deep Line Wars**
  For the experiments, we use three map sizes in Section 6, 12×11 , 22×22, and 40×30, respectively, where the opponent uses the built-in *expert* strategy.

- **Deep RTS**
  In Section 6, we present results for two scenarios in Deep RTS where the first is a single-player objective to navigate through a maze and find gold, seen in Figure 4b. This environment is used to evaluate safety during training in RTS games. The second scenario, seen in Figure 4c is a one versus one, 10×10 map with four available units, four available buildings, and over 100 possible action combinations every game state update.

[6]The deep warehouse environment is open-source and freely available at `https://github.com/cair/deep-warehouse`

- **Deep Warehouse**
  To demonstrate safety improvement of S-ORACLE compared to fully model-free methods, we train the algorithms on three grids with various concurrent agents in each grid size. The experiments run in grid-sizes of 11×11, 22×22, and 41×41 with 10, 50, and 100 manhattan-distance based co-agents, respectively.

- **MicroRTS**
  We use the same environments as in the IEEE Conference of Games for recent and previous years, listed in Table 2

- **ELF: Mini-RTS**
  The Mini-RTS environment features two built-in strategies; AI-Simple, AI-Hit-and-run where both are used in the experiments.

- **Starcraft II**
  We use all mini-games available to SC2LE, which are listed in Table 3.

5 Learning dynamics for Planning and control

The S-ORACLE algorithm is a novel end-to-end architecture for training model-free algorithms on a dynamics model learned through observations of the true environment. S-ORACLE is a combination of state-of-the-art deep learning techniques; stochastic recurrent state-space models (SRSSM) [84], variational autoencoders (VAE) [41], and vector-quantization [85]. The model contains a deterministic encoder and a stochastic decoder, a stochastic dynamics model, and a policy. The full architecture is visualized in Figure 4 and the scope of this Section is to describe each component of the S-ORACLE model explain the desired outcome running model-free algorithms for safer planning and decision-making.

5.1 Deterministic Encoder and Stochastic Decoder

When the agent observes a high-dimensional input such as images, we must reduce dimensional to reduce computational complexity. The convolutional layers are interchangeable with fully connected layers for vector-based inputs, but we consider pixels for this article.
• **The decoder** is a deconvolution network that upsamples the compressed latent variables to the spatial structure of the input variable o_t. The last layer of convolutions predicts μ and σ that parameterize gaussian distributions for all output *pixels* in $\hat{o}_{t+1}$. Experiments revealed significant performance benefits of a stochastic decoder output because it added variability during training, making the predictions far more accurate than fully deterministic layers. There are no particular differences during inference, as we sample the mean,

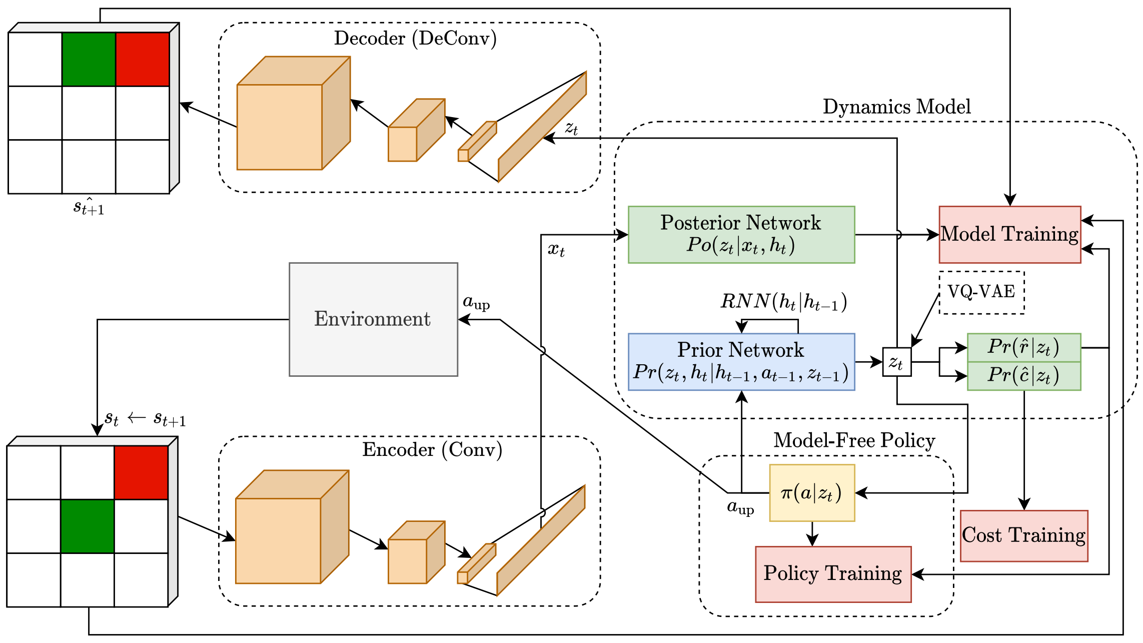


Figure 4: The S-ORACLE model. The environment produces an initial state s_t that feeds into a ConvNet. The flattened representation x_t is fed into the posterior distribution and produces a latent vector z_t. Concurrently, the prior distribution uses the previous hidden-state to predict a belief of the latent vector. The latent vector is the backbone for predicting a reward and the state-cost. The policy use trained parameters to predict an action and is sent to the environment. Every component of the model trains jointly in where each *training-block* (e.g., the colored squares) plays a part in the final optimization objective.

which roughly equates to the mean value of deterministic neural networks [84].

• **The encoder** is a three-layer deterministic convolutional neural network (CNN) that aims to capture information from pixel observations o_t and consequently reduce dimensionality before reshaping the compressed representation to a vector x_t[86]. The idea is that we compress observation to a compact representation, compute an internal state in the dynamics model, and decompress the internal representation. During training, the overall goal is for the dynamics model to resemble the same behavior as $\mathcal{T}(s'|s,a)$, where the model is denoted $\hat{\mathcal{T}}(s'|s,a)$. The encoder and decoder are learned using the MSE (cross-entropy) between the predicted observation $\hat{s}_{t+1}$, and the *after-the-fact* observation s_{t+1},

$$\mathcal{L}_{OBS} = -\mathbb{E}[logp(o|z)]. \tag{24}$$

5.2 Dynamics Model

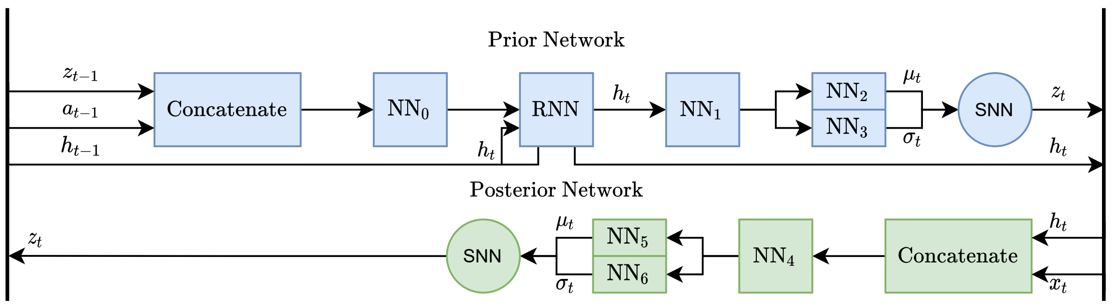


Figure 5: Detailed overview of the dynamics model forward-pass from Figure 4. The prior network uses last-step information, along with an action (the action that leads up to state s_t is denoted a_{t-1}), and outputs the computed hidden-state and the sampled latent-state variable z_t. For LSTM, we store the cell memory along with the hidden-state. The posterior network takes in the observed information x_t and our prior belief state h_t and predicts the *informed* latent-state variable z_t. Although both latent-state variables are denoted z_t, the prior latent-state variable is denoted $\hat{z}_t$ during training.

S-ORACLE is a model-based approach and attempts to estimate the true MDP transition function, described in Section 2.5. We use neural networks to estimate and combine VAE and SRSSM to create a highly expressive probabilistic model. S-ORACLE models two distributions, a prior model and a posterior model. Figure 5 depicts the forward pass of S-ORACLE after the observation o_t encodes to x_t.

• **The prior network** (generative network predicts the next state latent-space variables z_t using information from the last time-step; the latent-state z_{t-1}, current action a_{t-1}, and hidden-state h_{t-1}. Note that for LSTM, which is used in the experiments, the cell-state is preserved between inferences when using LSTM. The RNN node calculates the next hidden-state splits into μ and σ to parameterize Gaussians that predict z_t.

• **The posterior network** (inference model) concatenates hidden-state h_t and the encoded observation x_t and parametrizes Gaussian distributions similar to the prior network, which predicts *informed after-the-fact* latent-space variables.
• **The forward-pass** of the dynamics model is as follows where the prior model performs the following operations:

1. Compute $u_t = \text{concat}(z_{t-1}, a_{t-1})$
2. Compute RNN state $h_t = \text{RNN}(u_t)$
3. Parameterize mean $\mu_t = \text{NN}_2(h_t)$ diagonal covariance matrix $\sigma = \text{NN}_3(h_t)$
4. Sample from Gaussian distributions $\text{SNN}_\theta(z_t|h_t) \sim \mathcal{N}(h_t; \mu, \sigma)$

where all steps are performed for every sample and form our prior beliefs of the latent variables. The posterior model, seen in Equation 25b, depends on previous hidden-state h_{t-1}, action a_{t-1} including the encoded state-observation x_t. The posterior model can be summarized to the following procedure:

1. Compute $u_t = \text{concat}(h_t, x_t)$
2. Parameterize mean $\mu_t = \text{NN}_5(u_t)$ diagonal covariance matrix $\sigma = \text{NN}_6(u_t)$
3. Sample from Gaussian distributions $\text{SNN}_\psi(z_t|h_t) \sim \mathcal{N}(x_t; \mu, \sigma)$

• **Training** takes inspiration from previous work in [87] using variational inference [41] combined with Stochastic Recurrent State Space Models (SRSSM) from [84]. VAE and SRSSM are highly expressive model classes for learning patterns in time series data and system identification (e.g., learning dynamics model from observed data) [88]. We train the algorithm similarly to VAE's using amortized variational inference since $po_\theta(z|x) = \int_z \frac{po_\theta(x|z)po_\theta(z)}{po_\theta(x)} dz$ is intractable [89]. The generative model (prior) pr_θ and the inference model (posterior) po_θ is denoted,

$$\textbf{Prior Model} : pr_\theta(z_t, h_t|h_{t-1}, a_{t-1}) \tag{25a}$$
$$\textbf{Posterior Model} : po_\theta(z_t|x_t, h_t), \tag{25b}$$

where Equation 25a is the prior distribution that attempts to learn parameters θ that best fit the posterior distribution (Equation 25b) by minimizing the Kullback-Leibler (KL) distance. The intuition is that the posterior model learns dynamics of the MDP (environment) through observations x_t while the prior distribution must learn indirectly through optimization (parameters θ). To make the optimization tractable, we use the ELBO [41, 90] such that the optimization term for the SSM (generative and inference network) becomes,

$$\mathcal{L}_{SSM} = \mathcal{L}_{OBS} - D_{KL}[po_\theta(z|x)|pr_\theta(z)]. \tag{26}$$

5.3 Categorical Latents

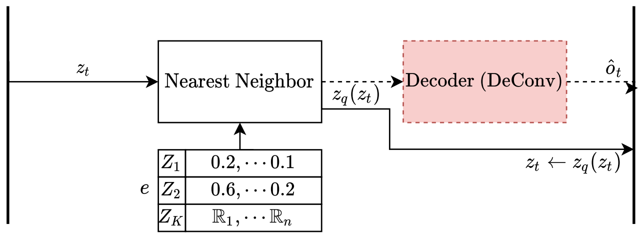


Figure 6: A detailed overview of the vector quantization network for mapping latent-space variables to categories after the generative network has predicted a continuous latent-space variable. The VQ-VAE module is depicted in Figure 4 and attempts to map variables from the generative network (prior) to categories. The dashed lines illustrate the process during training, while solid lines illustrate inference time computation. The algorithm finds an appropriate category using the nearest neighbor.

Following the work in [85], we use a variation of the VQ-VAE architecture, seen in Figure 6. However, we use it to categorize latent variables similar to [66] that allows for selecting policies to specific areas of the state-space and is a promising approach towards automating the options selected in the options framework [91]. Furthermore, VQ-VAE shows efficiency for planning and predictive learning [92] and does not suffer from posterior collapse. We observe that the dynamic model recovers from posterior collapse combined with stochastic weight averaging (SWA) combined with categorical latent motivate using VQ-VAE.

Inference and training is straightforward where the sampled latent-variables from the SSM mapped to a categorical codebook $Z_1 \cdots Z_K$ with K (hyperparameter, see Table 18) possible categories of latent-space variables. The output is the category closest to the input, and we updated the latent variable $z_t \leftarrow z_t^k$. Training occurs jointly with rest of the model,

$$\mathcal{L}_{VQ} = ||sg\left[z_e\right] - e||_2^2 + \beta_{\text{vq}}||z_e - sg\left[e\right]||_2^2, \tag{27}$$

where e is the codebook. The first term is the codebook alignment loss which updates the selected category closer to the SSM latent vector. The sg denotes the stop-gradient operator, which prevents gradients computation for the SSM as this term only concerns updating the codebook. The second term moves the SSM latent-vector towards the codebook but we wish to limit the influence on SSM, hence, $\beta_{\text{vq}} \approx 0.05$ in contrast to [44] using $0.25 \leq \beta_{\text{vq}} \leq 2.0$.

5.4 Rewards, Risks, and Costs

Using the learned dynamics model, we estimate rewards and costs as part of the feedback that fuels the learning of the RL agent. The feedback signal comprises regular rewards, uncertainty-based risk rewards, and costs that measure the distance between the current state and a positive terminal state. We follow the background in Section 2.9 and use Equation 23 to define the return function. The reward function $\hat{\mathcal{R}}$ is a parameterized Gaussian distribution learned using the squared loss between predicted and actual reward during training. The cost is learned separately after a terminal state is experienced by labeling previous states with the temporal difference (e.g., how many steps it took from state s_{t-n} until the terminal state s_t). The risk feedback is learned as a byproduct of the learned predictive reward model as we are interested in using uncertainty as a risk measurement $Risk = \text{Var}(\hat{\mathcal{R}})$, which gives us the reward function,

$$\mathcal{R}_{oracle} = \mathcal{R}(z) - Var(\hat{\mathcal{R}}) + (1 - C(z)). \tag{28}$$

The S-ORACLE risk-aware approach summarizes in two steps for reducing risk during training and action inference. The first term in Equation 22 is the inference-time risk reduction, while during learning, the agent utilizes 22 for action selection and Equation 23 with $\omega = \text{Var}(\hat{\mathcal{R}}(z))$. The second term is the Cost function $C(z))$ that predicts the normalized distance to a positive goal state. To optimize and learn the reward function and cost function, we minimize the following,

$$\mathcal{L}_{Rew} = \underbrace{\mathbb{E}[logpr(R|z)]}_{reward-loss} + \underbrace{sg(\mathbb{E}[logpr(C|z)])}_{cost-loss}. \tag{29}$$

5.5 Actor Policy Ensemble

The actor ensemble's primary objective is to make informed decisions that lead to safe trajectories to a goal state. In this work, S-ORACLE uses an ensemble of model-free approaches, specifically A2C, DQN, RAINBOW, IMPALA, and PPO. While all of the algorithms are different, they fundamentally share the objective of maximizing rewards which fuel the motivation for modeling the reward function in Equation 28. Specifically, all actor algorithms optimize towards the risk-aware reward function and empirically improve safety significantly. The actor ensemble is defined,

$$\pi_{ensemble} = \text{majority\_vote}(\{\pi_{A2C}(a|z), \pi_{DQN}(a|z), \pi_{RAINBOW}(a|z), \pi_{IMPALA}(a|z), \pi_{PPO}(a|z))\}, \tag{30}$$

where $\pi_{ensemble}$ is the majority voting policy that selects action a_t. Note that all policies select actions according to the risk-adjusted utility function in Equation 22 first proposed by [57], but using $\hat{\mathcal{R}}$ instead of the environment reward signal $\hat{R}$. Although the ensemble actor does not guarantee safety during training or learning, it shows a promising leap towards safer algorithms empirically, and Section 8 discusses the implications and possible methods to improve safety further.

5.6 Tuning and Training S-ORACLE

The S-ORACLE algorithm has many hyperparameters for tuning stability and performance, and is located in Table 18. Experiments illustrate S-ORACLE robust between most environments but required additional hyperparameter tuning for StarCraft II and Deep RTS. LTC and GRU were tested but found LSTM to perform better for tested environments. Another notable hyperparameter choice enables adaptive gradient clipping (AGC), a novel approach to clip the gradient from historical norms [93]. Additionally, gradients are clipped between -100.0 and 100.0 to increase training stability. The S-ORACLE model optimizes all objectives jointly,

$$\mathcal{L}_{ORACLE} = \mathcal{L}_{OBS} + \mathcal{L}_{Rew} + \gamma\mathcal{L}_{SSM} + \mathcal{L}_{VQ}, \tag{31}$$

using **S**tochastic **W**eight **A**veraging (SWA) with the AdamW optimizer [94]. SWA is a novel approach to ensemble learning where the objective is to widen the optima space such that it is easier to find and to give a better generalization of the model [95][7].Compared to other ensemble learning techniques, SWA only requires a single model where snapshots are stored every n epochs that are averaged every m epochs. SWA has different learning rate strategies (e.g., cyclical learning rate), and a linear cyclical learning rate is chosen for S-ORACLE.

5.7 Summary

This Section summarizes S-ORACLE as a novel approach towards safer reinforcement learning in RTS and industry-near environments. In contrast to prior work, **no assumptions on prior knowledge** at the cost of *not guaranteeing safety* if left unsupervised. However, using an **expert system** significantly decreases the likelihood of entering catastrophic states if used to pre-train the dynamics model. Section 6 shows that the **actor ensemble performs well within a presumable safe state-space set, without external guidance** if pretraining of the dynamics model is allowed. The S-ORACLE is summarized to the following procedure:

1. Read state observation o_t

[7]The author further demonstrated effectiveness in *Improving Stability in Deep Reinforcement Learning with Weight Averaging* but the work is not peer-reviewed. However similar results are demonstrated in [96]

2. Extract features of the observation x_t
3. Compute latent-state using x_t for the prior and posterior $\hat{z}_t$
4. Compute categorical latent-state for latent-space using VQ-VAE architecture z_t
5. Predict using latent-state z_t
    - Decode latent-state in decoder and sample possible future observation $\hat{o}_t$
    - Predict action for policy ensemble given latent-state using majority voting $\pi_{ensemble}(a_t|z_t)$
    - Predict reward of the latent-state $\hat{r}_t$
    - Predict cost of the separate model using latent-state $\hat{c}_t$
6. Compute gradients jointly following Equations 24, 26, 27,29, and separately for the policy ensemble and cost network. Use the Adam optimizer with SWA and gradient clipping.

6 Empirical Evaluation

This Section presents empirical results from the five RTS environments Deep RTS, ELF, MicroRTS, StarCraft II, and Deep Warehouse, for safety evaluations. The background for the environments is found in Section 4. Each environment is tested with five different model-free algorithms, A2C, DQN, RAINBOW, IMPALA, PPO using standard hyperparameters from the respective literature. Additionally, we test with the DVAE algorithm, a model-based approach that uses variational autoencoders to learn a dynamics model that is later used to train a PPO policy [96].

- **Experiment Setup.** The experiments are allocated one GPU per algorithm from a pool of 2x 2080 TI and 2x 1080 TI cards. Experiments allocated with a 1080 TI are given 37% longer train time to compensate for slower training speeds. For climate footprint estimations, we limit the GPU power draw to 250w and divide the total energy consumption of the 1080 TI-based experiments to compensate for the additional time. Model-free algorithms train for 24 hours and model-based trains for 16 hours to demonstrate efficiency. Performance is tested for several episodes, where the number of episodes is specified per environment due to computational limitations.
- **Experiment Scope.** The goal of the **experiment is to demonstrate the raw performance and safety performance of S-ORACLE**. Concurrently, a baseline is constructed for future work. Experiments demonstrate that S-ORACLE generalizes well for multiple problems, although the algorithm still has improvement

potential in behavior performance, sample efficiency, and safety. The results also systematically report S-ORACLE's energy and carbon footprint compared to other tested algorithms as suggested by Henderson et al. [6].

6.1 Safety Evaluation

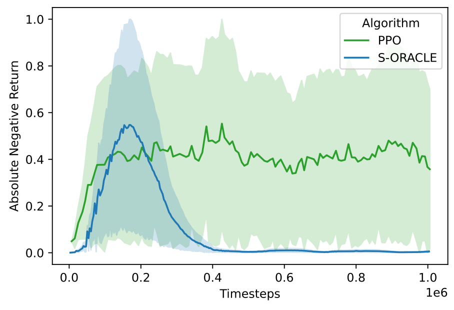
S-ORACLE aims to learn an agent agnostic feedback signal that directs the risk-neutral agents towards safer trajectories during training. S-ORACLE learns in a two-objective process where the first objective is to learn the dynamics model and the second is to learn an actor ensemble (see Section 5). S-ORACLE offer two training schemes to training the predictive model:

1. **Scheme #1**. Train dynamics model and actor ensemble concurrently, balancing the exploration-exploitation dilemma for risk management.
2. **Scheme #2**. Train dynamics model using external knowledge, using existing expert systems known to operate safely in the environment and subsequently, train actor ensemble off-line using dynamics model before carefully evaluate safety in live systems. This approach is seen as successful in prior work [96, 20].

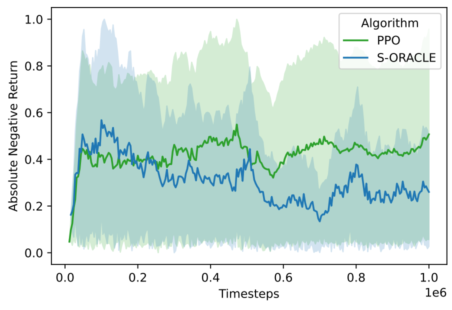
The first method is the most versatile because it does not require any knowledge *a priori* to solving the problem. However, the first method is at increased risk of entering catastrophic states. The second method improves empirical safety but relies on external systems to succeed. This Section aims to empirically demonstrate the safety performance of S-ORACLE compared to PPO in Scheme #1 and DVAE in Scheme #2. The experiments measure the ratio of incoming absolute negative returns to determine safety, normalized between $0.0 \leq x \leq 1.0$. For every 100 000 timesteps, the algorithm is evaluated for 100 episodes, and the measured min-max variance is illustrated using shades in the plots. S-ORACLE is evaluated in two environments using the described training schemes:

1. **Deep RTS** lava environment, seen in Figure 4b. The agent performs safe behavior when avoiding lava states. The goal is to reach the gold in the middle of the map.
2. **Deep Warehouse** logistics challenge. Transport goods from source to destination as fast as possible while avoiding collision with other agents. The algorithm controls a single agent, while the remaining agents use a manhattan distance-based heuristic. The agent is part of a larger logistics system with other taxi agents, depending on map size (see Section 4).

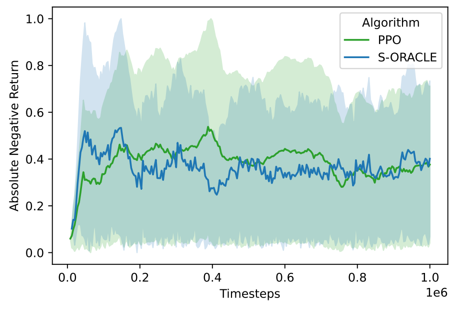
Using Scheme #1, the algorithms balance the exploration-exploitation dilemma during dynamics model training. Hence they are more susceptible to error states because of missing knowledge about the dynamics. Figure 7a-7d shows the safety violations ratio in Deep



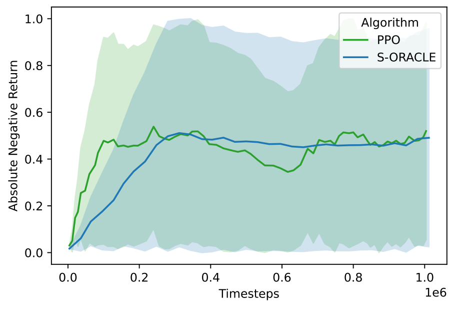
(a) Deep RTS Lava.



(b) Deep Warehouse 11×11.

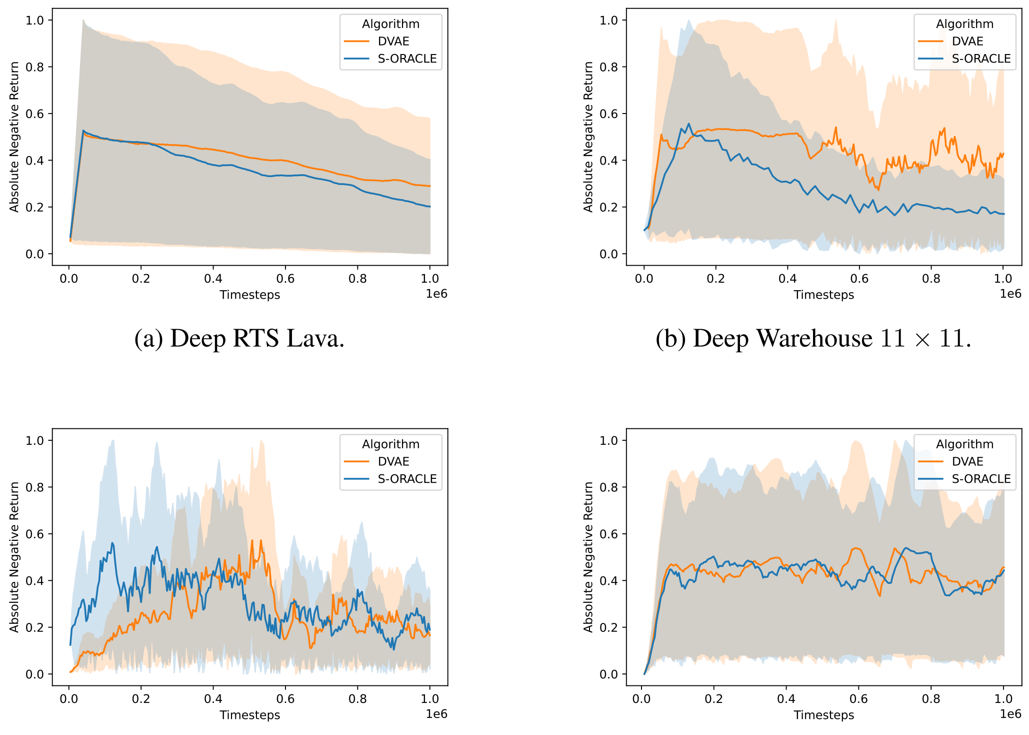


(c) Deep Warehouse 22×22.



(d) Deep Warehouse 41×41.

Figure 7: Safety violations rate following training Scheme #1. The y-axis is the mean absolute negative return where the absolute negative return is averaged per 10 000 timesteps.



(a) Deep RTS Lava. (b) Deep Warehouse 11×11.

(c) Deep Warehouse 22×22. (d) Deep Warehouse 41×41.

Figure 8: Safety violations rate following training Scheme #2. The y-axis is the mean absolute negative return where the absolute negative return is averaged per 10 000 timesteps.

RTS and Deep Warehouse, respectively. In the Deep RTS Lava environment, it is clear that S-ORACLE learns to avoid lava, and for Deep Warehouse 11×11 (Figure 7b), the S-ORACLE gradually reduce the rate of negative returns compared to PPO. Similarly, for Deep Warehouse 22×22 and 41×41 (Figure 7c and Figure 7d), S-ORACLE accumulate fewer negative rewards, but the effect seems to diminish for more complex state-spaces. Collectively, the figures demonstrate that a risk-neutral algorithm (PPO) explores negative states at a relatively uniform rate, while the risk-averse agent (S-ORACLE) impacts the number of visited error-states significantly less.

In Scheme #2, the dynamics model is trained from observations of an expert system before training using the dynamics model occurs for the agent algorithm. The second approach demonstrated better safety awareness, naturally, because the algorithm learned much of the environment through the dynamics model before making actions live. Figure 8 shows the safety-awareness of S-ORACLE compared to DVAE, clearly showing that there is a downwards trend in the mean absolute negative return. However, as the environment becomes more complex (Figure 8d), the effect seems to diminish in contrast to Figure 8b.

The empirical evidence clearly shows that S-ORACLE improves safety in Scheme #1 and Scheme #2 compared to model-free RL algorithms and the prior work of DVAE.

Specifically, as the algorithm learns to navigate the environment, the agent receives fewer negative rewards than other tested environments.

6.2 Performance Evaluations

- **Deep Line Wars.** Table 4 shows the performance of the tested algorithms against the hard-coded expert agent. Each algorithm runs 1000 times and is averaged over. Specifically, we observe that all tested algorithms perform well, even for large maps. The S-ORACLE agent performs comparably to the tested model-free algorithms and outperforms all tested algorithms on average. The action space comprises 4 actions for cursor position, 4 build actions, and 4 spawn unit actions.
- **Deep RTS.** We evaluate the algorithms for 100 episodes in a maze-line problem as seen in Figure 4b and in a one-versus-one as in Figure 4c between the tested algorithms. For the maze-like environment, results and averaged. Table 5 shows the performance of tested algorithms in the maze-environment. All algorithms perform well, where the best algorithm, IMPALA, consistently found the best path after training. The other algorithms performed in the $\sim 95\% \pm 5$ range. Table 6 shows that S-ORACLE outperforms 5 out of 7 algorithms and scores comparably with PPO in 1v1 matches. We observed that S-ORACLE plays conservatively and wins by strategically blocking the opponent's path with houses during experiments. In the long run, the conservative behavior wins because the cost of units is more expensive than houses, and additionally, houses allow the player to build a larger army that trivially defeats the opponent with limited housing. The agent had access to 13 discrete actions at every timestep and received positive rewards for victories and zero during the game.
- **ELF: Mini-RTS.** The partial observability of Mini-RTS makes the learning task significantly harder to master compared to Deep RTS. Table 7 shows that S-ORACLE average 83% win ratio but had difficulties defeating the AI-Simple algorithm consistently. The Mini-RTS action space comprises 9 strategic actions in which the agent can construct complex strategies. We observe that the average-length game is approximately 3500 ticks where a positive reward is given for victories, a negative reward for defeats, and zero rewards in other states. Algorithms are tested for 100 episodes.
- **MicroRTS.** Perhaps the most widely used environment for competitions is the MicroRTS environment [22]. We test each algorithm for 100 episodes against strategies presented in the IEEE Conference on Games (COG-2019) competition. Table 8 shows the results, where each row illustrates the win rate against the column algorithm. We see that S-ORACLE outperforms the other algorithms, closely

followed by DVAE, PPO, and IMPALA. The action space and state-space vary between maps but are thoroughly described in [97].

- **StarCraft II.** Perhaps the most compute-heavy experiment is the StarCraft II environments, albeit we only focus on mini-games of the original game. In particular, we use the mini-games from [77] already standardized in literature, and we adopt these findings for our comparison. As seen in Table 9 no algorithm remains dominant in all environments, but the professional player from DeepMind (DM) [77]. However, the S-ORACLE algorithm scores on average better than the model-free approaches, likely because of the ensemble technique when voting for actions. The action space is challenging to implement because it comprises several hundred actions for every timestep. For this reason, we choose to adopt the same method as in [98].

Table 4: Deep Line Wars result. Each agent (row) plays against the built-in *expert* agent at different map-sizes (column). The last column is the average performance for all map-sizes. The cell values represent win ratio ranging from 0 to 1.

Algorithm | Map	12x11	22x22	40x30	64x64	Avg
A2C	1.00 ±0.0	0.96 ±0.01	0.86 ±0.07	0.69 ±0.3	0.88
DQN	1.00 ±0.0	0.92 ±0.03	0.89 ±0.1	0.85 ±0.08	0.92
RAINBOW	1.00 ±0.0	0.95 ±0.03	0.90 ±0.08	0.64 ±0.01	0.87
IMPALA	1.00 ±0.0	0.96 ±0.04	0.95 ±0.04	0.85 ±0.11	0.94
PPO	1.00 ±0.0	0.99 ±0.01	0.95 ±0.01	0.66 ±0.18	0.90
DVAE	1.00 ±0.0	0.99 ±0.01	0.89 ±0.1	0.77 ±0.05	0.91
S-ORACLE	1.00 ±0.0	0.98 ±0.01	0.97 ±0.01	0.87 ±0.07	**0.96**

6.3 Sustainability Report

Table 10 reports the contribution of $\frac{CO2}{kg}$ of the experiments following the work of Henderson et al. [6]. Algorithm performance is tested in 41 experiments and additional six safety experiments for PPO, DVAE, and S-ORACLE. Note that the model-based approaches have a training budget of 16 hours, and model-free algorithms have 24 hours. The Figure shows that model-based approaches utilize time significantly more, considering that total CPU and GPU time is close to model-free algorithms. The reason for better utilization is that model-based RL algorithms better saturate the GPUs when training using the dynamics model.

Table 5: The results of the Deep RTS Maze environment. The environment as seen in 4b defines the reward function as $r_t = optimalRoute \times 2 - t$. The results are averaged over 100 episodes. The cell values represent accumulated score ranging from 0 to 113.

Algorithm	Score
A2C	91 $\pm$5
DQN	98 $\pm$9
RAINBOW	99 $\pm$4
IMPALA	100 $\pm$2
PPO	96 $\pm$6
DVAE	97 $\pm$11
S-ORACLE	94 $\pm$5

Table 6: DeepRTS 1v1 results. The row is the win rate of the respective algorithm agains the column wise algorithm for 100 episodes. The cell values represent win ratio ranging from 0 to 1.

Algorithm	A2C	DQN	RAINBOW	IMPALA	PPO	DVAE	S-ORACLE	Avg
A2C	-	0.35 $\pm$0.11	0.23 $\pm$0.05	0.44 $\pm$0.11	0.14 $\pm$0.05	0.39 $\pm$0.11	0.36 $\pm$0.05	0.32
DQN	0.65 $\pm$0.03	-	0.45 $\pm$0.02	0.36 $\pm$0.05	0.25 $\pm$0.03	0.36 $\pm$0.32	0.25 $\pm$0.14	0.39
RAINBOW	0.77 $\pm$0.02	0.55 $\pm$0.15	-	0.59 $\pm$0.05	0.47 $\pm$0.07	0.55 $\pm$0.05	0.45 $\pm$0.02	0.56
IMPALA	0.56 $\pm$0.07	0.64 $\pm$0.11	0.41 $\pm$0.08	-	0.45 $\pm$0.04	0.59 $\pm$0.22	0.36 $\pm$0.18	0.50
PPO	0.86 $\pm$0.07	0.75 $\pm$0.01	0.53 $\pm$0.11	0.55 $\pm$0.02	-	0.52 $\pm$0.05	0.56 $\pm$0.03	**0.63**
DVAE	0.61 $\pm$0.25	0.64 $\pm$0.05	0.45 $\pm$0.05	0.41 $\pm$0.06	0.48 $\pm$0.05	-	0.25 $\pm$0.02	0.47
S-ORACLE	0.64 $\pm$0.11	0.75 $\pm$0.02	0.55 $\pm$0.09	0.64 $\pm$0.03	0.44 $\pm$0.02	0.75 $\pm$0.02	-	**0.63**

Table 7: ELF: Mini-RTS results. Each experiment runs for 100 episodes and is averaged over. The cell values represent win ratio ranging from 0 to 1.

Algorithm	AI-Simple	AI-Hit-and-run	Average
A2C	0.85 $\pm$0.07	0.86 $\pm$0.08	0.86
DQN	0.56 $\pm$0.05	0.85 $\pm$0.02	0.71
RAINBOW	0.75 $\pm$0.11	0.88 $\pm$0.05	0.82
IMPALA	0.66 $\pm$0.02	0.96 $\pm$0.09	0.81
PPO	0.78 $\pm$0.04	0.97 $\pm$0.11	**0.88**
DVAE	0.73 $\pm$0.05	0.99 $\pm$0.05	0.86
S-ORACLE	0.67 $\pm$0.09	0.98 $\pm$0.04	0.83

7 Discussion

Performance is not the dominant factor in safety-critical systems. However, the *niceness* of performing superhuman or beyond existing expert systems is appealing if the algorithm

Table 8: MicroRTS averaged results for all tested mini-games. Results for individual environments are found in appendix A. The cell values represent win ratio ranging from 0 to 1.

Agorithm	RandomBiasedAI	NaiveMCTS	Tiamat	Droplet	UTS_Imass	Win Ratio
A2C	0.89 ±0.01	0.86 ±0.06	0.84 ±0.11	0.79 ±0.14	0.84 ±0.13	0.84
DQN	0.91 ±0.04	0.86 ±0.12	0.79 ±0.03	0.82 ±0.17	0.7 ±0.26	0.82
RAINBOW	0.89 ±0.1	0.88 ±0.06	0.83 ±0.08	0.79 ±0.21	0.79 ±0.11	0.83
IMPALA	0.88 ±0.05	0.86 ±0.09	0.86 ±0.05	0.87 ±0.06	0.85 ±0.12	0.86
PPO	0.91 ±0.02	0.92 ±0.02	0.83 ±0.07	0.84 ±0.05	0.83 ±0.07	0.86
DVAE	0.90 ±0.07	0.87 ±0.09	0.80 ±0.18	0.90 ±0.07	0.84 ±0.15	0.86
S-ORACLE	0.90 ±0.06	0.88 ±0.02	0.86 ±0.02	0.84 ±0.04	0.86 ±0.05	**0.87**

Table 9: StarCraft II results. The A2C, A3C, and DM are results from relevant literature, and the remainder is novel results in this work. We ran the experiments ten times and averaged the results. The cell values represent total accumulated return.

Environment	A2C [98]	A3C [99]	DM [77]	DQN	RAINBOW	IMPALA	PPO	ORACLE	S-ORACLE
MoveToBeacon	21.3	24	26	26	30	32	35	24	29
DefeatRoaches	72.5	47	41	100	81	91	75	60	77
BuildMarines	0.55	0.6	138	0	0	2	8	12	2
CollectMineralShards	81	45	133	3	12	41	53	55	58
CollectMineralAndGas	3320	371	6880	3978	3911	4251	4102	5212	5102
FindAndDefeatZerglings	22.1	25	46	45	21	23	19	29	35
DefeatBanelingsAndZerglings	56.8	43	729	62	20	423	251	305	530
Average Score	510.6	79.3	**1141.8**	602	582.1	694.7	649	813.8	**833.2**

can provide safety guarantees. S-ORACLE cannot provide safety guarantees, but it provides empirical evidence of acceptable performance while lowering erroneous agent decisions. The algorithm demonstrates that, although following a generic and noise-inducing risk-averse reward signal, the algorithm demonstrates that good policies are possible to obtain and to perform adequately in most tested environments.

Safety is perhaps the most challenging trait to learn in an RL setting because it is not naturally present in the foundation of which the framework builds. Perhaps the first hint is found in the word *reinforcement* and the most understanding of the concept is to *build upon something* or *the process of encouraging or establishing a belief or pattern of behavior*. The RL literature often depicts the analogy of a child in a behavioristic way that through sensory stimuli such as vision, hearing, smell, feeling, and taste, the child learns through trial and error. However, to compare RL algorithms to a child, we must strip

Table 10: This work contributed 55.08 kg of CO2eq to the atmosphere, and used 1377.0 kWh of electricity. The columns describes the following from left to right: (1) Number of experiments performed, (2) Average hours of CPU time per experiment, (3) Average hours of GPU time per experiment, (4) Total CPU time for all experiments, (5) Total CPU time for all experiments, (6) Total kWh for all experiments, and finally, the total $\frac{CO2}{kg}$ contribution of the experiments.

Algorithm	Num Exp	Avg CPU H.	Avg GPU H.	Tot CPU H.	Tot GPU H.	Tot. kWh	$\frac{CO2}{kg}$
A2C	41	13.53	21.41	554.73	877.81	233.04	9.32
DQN	41	11.71	20.22	480.11	829.02	219.02	8.76
RAINBOW	41	19.24	15.74	788.84	645.34	180.66	7.23
IMPALA	41	15.55	14.55	637.55	596.55	164.76	6.59
PPO	45	16.22	19.22	729.90	864.90	234.11	9.36
DVAE	45	14.33	11.24	644.85	505.80	142.25	5.69
S-ORACLE	49	13.11	12.53	642.39	613.97	169.23	6.77

away all its sensory information and memory and feed it noisy and sparse information about the world. If so, the child (algorithm) has very few sensory sensations, and at the beginning of time t do not know the environment, it seems impossible to learn anything without trial and error. Or hence, the child analogy is inadequate. Drawing parallels to S-ORACLE, it is only natural that the algorithm performs erroneous decisions during learning. While there are methods to provide true safety guarantees via prior information and policy constraints [68], no method exists for learning a task perfectly without balancing the exploration-exploitation dilemma. Drawing further parallels to the mission-critical industry setting, One approach to **alleviate** this problem is to use existing algorithms or expert systems operational in the environment to build a world model accurate enough to guide model-free algorithms towards safe policies. As seen in the experiments using expert-system knowledge to train the dynamics model works empirically well in the Deep Warehouse environment. S-ORACLE cannot theoretically guarantee safety but empirically shows that it improves safety significantly compared to risk-neutral approaches during learning and inference. Combined with generalizable self-tuning constraints, the hope is that future work brings such work to allow fully safe decision-making in safety-critical systems.

8 Conclusion and Future Directions

This work investigates whether the safer model-based reinforcement learning algorithm S-ORACLE has (1) better performance, (2) sample efficient, (3) more sustainable, and (4) safer than traditional model-free deep RL algorithms. Based on experiments in Section 6, empirical evidence suggests that 2-4 hold except for 1, where the algorithm is on-par or inferior to model-free approaches.[8] This Section details further 1 to 4 and concludes the findings of this work.

Performance is evaluated through the empirical score the algorithm obtained during experiments. Generally, we find S-ORACLE less performant and to have a larger spread in obtained scores. While S-ORACLE is not performing better, it closely follows in most experiments, although there is still work that needs addressing lesser performance. Our analysis is that the agent performs more conservative action sequences, which leads to defeat as the opponent gain territory dominance in the RTS games. However, in Deep Warehouse, we observe that the algorithm is on par with PPO and DQN, which is a good measurement of how performance is in industry-like environments (e.g., stationary environments). One possible solution would also allow for longer training, but at the cost of a less climate-sustainable model.

Sample Efficiency is significantly better than model-free algorithms, as naturally expected of a model-based approach. However, we do not S-ORACLE as a high sample efficient algorithm. For instance, we expect work such as [101] to have better sample efficiency. Our findings in Section 6 indicate that the model prediction error closely follows the sample efficiency. For instance, the StarCraft II environment is notoriously difficult to learn, and hence, it takes far longer to obtain a good model. In future work, we would like to improve the prediction model by using discrete latent space techniques more closely to work in [101].

Sustainability is an increasingly important topic in machine learning because most state-of-the-art algorithms in the literature have a high computational cost which has an immediate impact on global warming. For this reason, we have thoroughly evaluated our model through the lens of sustainability, that is, how much power draw our model takes to train. We follow the work of [6], and while this approximation is far from perfect, it is a good indication of the overall sustainability of the model. Unfortunately, we could not find any model that reports CO2 emissions, but we hope that our work serves as a good baseline for future work in safe model-based RL.

Safety is crucial for an algorithm that aims to reach industry-like production environments outside the research lab, and by no means are S-ORACLE the answer to a perfectly safe algorithm, but rather towards a more safe alternative. In our experiments, we quantify the occurrences of catastrophic states during learning, and we observe that S-ORACLE has significantly fewer error-states visited than fully model-free approaches. Our vision is for an algorithm without *assumptions built on assumptions* that work in practice and can scale to more complex environments. To the best of our knowledge, there is no similar

[8]Similar conclusions are found in [100] emphasizing having smaller rollout horizons for better model prediction.

work in safe RL for complex games such as RTS games, and we hope that future work may include impressive progress towards more safety-aware agents than S-ORACLE.

Summary We present a novel, safer model-based RL agent for RTS games and industry-like applications. We empirically show that it performs well while maintaining better safety decision-making and is more sustainable than model-free approaches.

Future work will attempt to address several shortcomings in our model and simplify the model where possible, similar to the works on TRPO and PPO. One particular problem is that the algorithm is susceptible to posterior collapse during training, well known in variational inference. Specifically, [102] is an appealing approach to increase stability, which we aim to address in future work. Lastly, we aim to reduce the number of model hyperparameters resulting from immense testing of theoretically justified methods. However, some hyperparameters are found to work only for particular values. Hence it is sensible to better define hyperparameters in closed sets compared to definitions for any real number, as seen in 18

9 Acknowledgements

We would like to take this opportunity to thank the reviewers for their immense contribution to evaluate this contribution. Also, we thank the editors for their interest in producing a special issue on risk-aware autonomous systems for artificial intelligence.

References

[1] R. Bellman, A Markovian Decision Process, Journal of Mathematics and Mechanics 6 (5) (1957) 6. doi:10.1512/iumj.1957.6.56038.
URL https://www.jstor.org/stable/24900506

[2] R. S. Sutton, A. G. Barto, Reinforcement learning: An introduction, 2nd Edition, A Bradford Book, Cambridge, MA, USA, 2018.
URL https://dl.acm.org/doi/book/10.5555/3312046

[3] R. S. Sutton, Dyna, an integrated architecture for learning, planning, and reacting, ACM SIGART Bulletin 2 (4) (1991) 160–163. doi:10.1145/122344.122377.
URL https://dl.acm.org/doi/10.1145/122344.122377

[4] J. García, F. Fernández, A Comprehensive Survey on Safe Reinforcement Learning, Journal of Machine Learning Research 16 (2015) 1437–1480.

[5] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, D. Silver, Grandmaster level in StarCraft II using multi-agent reinforcement learning, Nature 2019 575:7782 575 (7782) (2019) 350–354. doi:10.1038/s41586-019-1724-z.
URL https://www.nature.com/articles/s41586-019-1724-z

[6] P. Henderson, J. Hu, J. Romoff, E. Brunskill, D. Jurafsky, J. Pineau, Towards the Systematic Reporting of the Energy and Carbon Footprints of Machine Learning, Journal of Machine Learning Research 21 (2020) 1–43.
URL http://jmlr.org/papers/v21/20-312.html.

[7] K. Arulkumaran, M. P. Deisenroth, M. Brundage, A. A. Bharath, Deep reinforcement learning: A brief survey, IEEE Signal Processing Magazine 34 (6) (2017) 26–38. doi:10.1109/MSP.2017.2743240.

[8] T. Aotani, T. Kobayashi, K. Sugimoto, Bottom-up multi-agent reinforcement learning by reward shaping for cooperative-competitive tasks, Applied Intelligence 2021 51:7 51 (7) (2021) 4434–4452. doi:10.1007/S10489-020-02034-2.
URL https://link.springer.com/article/10.1007/s10489-020-02034-2

[9] S. Carta, A. Corriga, A. Ferreira, A. S. Podda, D. R. Recupero, A multi-layer and multi-ensemble stock trader using deep learning and deep reinforcement learning, Applied Intelligence 2020 51:2 51 (2) (2020) 889–905. doi:10.1007/S10489-020-01839-5.
URL https://link.springer.com/article/10.1007/s10489-020-01839-5

[10] G. Paludo Licks, J. Colleoni Couto, P. de Fátima Miehe, R. de Paris, D. Dubugras Ruiz, F. Meneguzzi, SmartIX: A database indexing agent based on reinforcement learning, Applied Intelligence 2020 50:8 50 (8) (2020) 2575–2588. doi:10.1007/S10489-020-01674-8.
URL https://link.springer.com/article/10.1007/s10489-020-01674-8

[11] G. Zou, J. Tang, L. Yilmaz, X. Kong, Online food ordering delivery strategies based on deep reinforcement learning, Applied Intelligence 2021 (2021) 1–

13doi:10.1007/S10489-021-02750-3.
URL https://link.springer.com/article/10.1007/s10489-021-02750-3

[12] M. Kusy, R. Zajdel, Probabilistic neural network training procedure based on Q(0)-learning algorithm in medical data classification, Applied Intelligence 2014 41:3 41 (3) (2014) 837–854. doi:10.1007/S10489-014-0562-9.
URL https://link.springer.com/article/10.1007/s10489-014-0562-9

[13] M. Mahmud, M. S. Kaiser, A. Hussain, S. Vassanelli, Applications of Deep Learning and Reinforcement Learning to Biological Data, IEEE Transactions on Neural Networks and Learning Systems 29 (6) (2018) 2063–2079. doi:10.1109/TNNLS.2018.2790388.

[14] T. Kobayashi, Student-t policy in reinforcement learning to acquire global optimum of robot control, Applied Intelligence 2019 49:12 49 (12) (2019) 4335–4347. doi:10.1007/S10489-019-01510-8.
URL https://link.springer.com/article/10.1007/s10489-019-01510-8

[15] B. Kiumarsi, K. G. Vamvoudakis, H. Modares, F. L. Lewis, Optimal and Autonomous Control Using Reinforcement Learning: A Survey, IEEE Transactions on Neural Networks and Learning Systems 29 (6) (2018) 2042–2062. doi:10.1109/TNNLS.2017.2773458.

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529–533. doi:10.1038/nature14236.

[17] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, D. Hassabis, Mastering the game of Go with deep neural networks and tree search, Nature 529 (7587) (2016) 484–489. arXiv:1610.00633, doi:10.1038/nature16961.

[18] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, D. Hassabis, A general reinforcement learning algorithm that masters chess, shogi, and

Go through self-play., Science (New York, N.Y.) 362 (6419) (2018) 1140–1144. doi:10.1126/science.aar6404.
URL http://www.ncbi.nlm.nih.gov/pubmed/30523106

[19] D. Silver, S. Singh, D. Precup, R. S. Sutton, Reward is enough, Artificial Intelligence 299 (2021) 103535. doi:10.1016/J.ARTINT.2021.103535.

[20] P.-A. Andersen, M. Goodwin, O.-C. Granmo, Towards safe reinforcement-learning in industrial grid-warehousing, Information Sciences 537 (2020) 467–484. doi:10.1016/j.ins.2020.06.010.

[21] Z. Ding, H. Dong, Challenges of Reinforcement Learning, Deep Reinforcement Learning: Fundamentals, Research and Applications (2020) 249–272doi:10.1007/978-981-15-4095-0_7.
URL https://link.springer.com/chapter/10.1007/978-981-15-4095-0{_}7

[22] S. Ontanon, The combinatorial multi-armed bandit problem and its application to real-time strategy games, in: Proceedings, The Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, 2013, pp. 58–64.
URL http://www.aaai.org/ocs/index.php/AIIDE/AIIDE13/paper/viewPaper/7377

[23] P.-A. Andersen, M. Goodwin, O.-C. Granmo, Deep RTS: A Game Environment for Deep Reinforcement Learning in Real-Time Strategy Games, in: 2018 IEEE Conference on Computational Intelligence and Games (CIG), 2018, pp. 1–8. doi:10.1109/CIG.2018.8490409.

[24] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. de Oliveira Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, S. Zhang, Dota 2 with Large Scale Deep Reinforcement Learning, CoRR abs/1912.0 (2019).
URL http://arxiv.org/abs/1912.06680

[25] H. Sethy, A. Patel, V. Padmanabhan, Real Time Strategy Games: A Reinforcement Learning Approach, Procedia Computer Science 54 (2015) 257–264. doi:10.1016/J.PROCS.2015.06.030.

[26] S. Levine, C. Finn, T. Darrell, P. Abbeel, End-to-end training of deep visuomotor policies, Journal of Machine Learning Research 17 (1) (2016) 1334–1373.
URL http://www.jmlr.org/papers/volume17/15-522/15-522.pdf

[27] X. Hu, T. Liu, X. Qi, M. Barth, Reinforcement Learning for Hybrid and Plug-In Hybrid Electric Vehicle Energy Management: Recent Advances and Prospects, IEEE Industrial Electronics Magazine 13 (3) (2019) 16–25. doi:10.1109/MIE.2019.2913015.

[28] G. E. Monahan, State of the Art—A Survey of Partially Observable Markov Decision Processes: Theory, Models, and Algorithms, Management Science 28 (1) (1982) 1–16. doi:10.1287/MNSC.28.1.1.
URL https://pubsonline.informs.org/doi/abs/10.1287/mnsc.28.1.1

[29] M. L. Puterman, Chapter 8 Markov decision processes, Handbooks in Operations Research and Management Science 2 (C) (1990) 331–434. doi:10.1016/S0927-0507(05)80172-0.

[30] M. Hausknecht, P. Stone, Deep Recurrent Q-Learning for Partially Observable MDPs, in: AAAI Fall Symposium Series, 2015, pp. 1–9. arXiv:1507.06527.

[31] M. L. Littman, A. R. Cassandra, L. P. Kaelbling, Learning policies for partially observable environments: Scaling up, Machine Learning Proceedings 1995 (1995) 362–370doi:10.1016/B978-1-55860-377-6.50052-9.

[32] A. Rodriguez, R. Parr, D. Koller, Reinforcement Learning Using Approximate Belief States, in: S. Solla, T. Leen, K. Müller (Eds.), Advances in Neural Information Processing Systems, Vol. 12, MIT Press, 2000, pp. 1036–1042.
URL https://proceedings.neurips.cc/paper/1999/file/158fc2ddd52ec2cf54d3c161f2dd6517-Paper.pdf

[33] X. Xiang, S. Foo, Recent Advances in Deep Reinforcement Learning Applications for Solving Partially Observable Markov Decision Processes (POMDP) Problems: Part 1—Fundamentals and Applications in Games, Robotics and Natural Language Processing, Machine Learning and Knowledge Extraction 2021, Vol. 3, Pages 554-581 3 (3) (2021) 554–581. doi:10.3390/MAKE3030029.
URL https://www.mdpi.com/2504-4990/3/3/29/htmhttps://www.mdpi.com/2504-4990/3/3/29

[34] R. Bellman, On the Theory of Dynamic Programming, Proceedings of the National Academy of Sciences 38 (8) (1952) 716–719. doi:10.1073/PNAS.38.8.716.
URL https://www.pnas.org/content/38/8/716https://www.pnas.org/content/38/8/716.abstract

[35] L. P. Kaelbling, M. L. Littman, A. W. Moore, Reinforcement Learning: A Survey, Journal of Artificial Intelligence Research (apr 1996). arXiv:9605103, doi:

10.1.1.68.466.
URL http://arxiv.org/abs/cs/9605103

[36] M. Van Otterlo, M. Wiering, Reinforcement learning and markov decision processes, Adaptation, Learning, and Optimization 12 (2012) 3–42. doi:10.1007/978-3-642-27645-3_1.

[37] T. M. Moerland, J. Broekens, C. M. Jonker, Model-based Reinforcement Learning: A Survey, arxiv preprint arXiv:2006.16712 (jun 2020). arXiv:2006.16712.
URL https://arxiv.org/abs/2006.16712

[38] P.-A. Andersen, M. Goodwin, O.-C. Granmo, Towards Model-Based Reinforcement Learning for Industry-Near Environments, in: M. Bramer, M. Petridis (Eds.), Artificial Intelligence XXXVI, Springer International Publishing, Cham, 2019, pp. 36–49. doi:10.1007/978-3-030-34885-4_3.

[39] R. S. Sutton, Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming, Machine Learning Proceedings 1990 (1990) 216–224doi:10.1016/B978-1-55860-141-3.50030-4.

[40] S. R. Eddy, What is a hidden Markov model?, Nature Biotechnology 2004 22:10 22 (10) (2004) 1315–1316. doi:10.1038/nbt1004-1315.
URL https://www.nature.com/articles/nbt1004-1315

[41] D. P. Kingma, M. Welling, Auto-Encoding Variational Bayes, Proceedings of the 2nd International Conference on Learning Representations (dec 2013). arXiv:1312.6114, doi:10.1051/0004-6361/201527329.
URL http://arxiv.org/abs/1312.6114

[42] R. E. Kalman, A new approach to linear filtering and prediction problems, Journal of Fluids Engineering, Transactions of the ASME 82 (1) (1960). doi:10.1115/1.3662552.

[43] S. Hochreiter, J. Schmidhuber, Long Short-Term Memory, Neural Computation 9 (8) (1997). doi:10.1162/neco.1997.9.8.1735.

[44] A. Van Den Oord, O. Vinyals, K. Kavukcuoglu, Neural discrete representation learning, in: Advances in Neural Information Processing Systems, Vol. 2017-Decem, 2017, pp. 1–11.

[45] J. Durbin, S. J. Koopman, Time Series Analysis by State Space Methods: Second Edition, Oxford Statistical Science Series (2012).

[46] C. J. C. H. Watkins, P. Dayan, Q-learning, Machine Learning 1992 8:3 8 (3) (1992) 279–292. doi:10.1007/BF00992698.
URL https://link.springer.com/article/10.1007/BF00992698

[47] T. Jaakkola, M. Jordan, S. Singh, On the Convergence of Stochastic Iterative Dynamic Programming Algorithms, Neural Computation 6 (6) (1994) 1185–1201. doi:10.1162/neco.1994.6.6.1185.

[48] R. S. Sutton, The Bitter Lesson (2019).
URL http://www.incompleteideas.net/IncIdeas/BitterLesson.html

[49] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, P. Abbeel, High-dimensional continuous control using generalized advantage estimation, in: 4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings, 2016, pp. 1–14.

[50] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal Policy Optimization Algorithms, arxiv preprint arXiv:1707.06347 (jul 2017). arXiv:1707.06347.
URL http://arxiv.org/abs/1707.06347

[51] J. Schulman, S. Levine, P. Abbeel, M. Jordan, P. Moritz, Trust Region Policy Optimization, in: F. Bach, D. Blei (Eds.), Proceedings of the 32nd International Conference on Machine Learning, Vol. 37 of Proceedings of Machine Learning Research, PMLR, Lille, France, 2015, pp. 1889–1897.
URL http://proceedings.mlr.press/v37/schulman15.html

[52] A. Kendall, Y. Gal, What uncertainties do we need in Bayesian deep learning for computer vision?, in: Advances in Neural Information Processing Systems, Vol. 2017-Decem, 2017, pp. 5580–5590.
URL http://papers.nips.cc/paper/7141-what-uncertainties-do-we-need-in-bayesian-deep-learning-for-computer-vision

[53] J. A. Bagnell, A. Y. Ng, J. G. Schneider, Solving Uncertain Markov Decision Processes, Carnegie Mellon Research Showcase (2001).

[54] M. Kearns, S. Singh, Near-Optimal Reinforcement Learning in Polynomial Time, Machine Learning 2002 49:2 49 (2) (2002) 209–232. doi:10.1023/A:1017984413808.
URL https://link.springer.com/article/10.1023/A:1017984413808

[55] J. Schmidhuber, Formal theory of creativity, fun, and intrinsic motivation (1990-2010), IEEE Transactions on Autonomous Mental Development 2 (3) (2010) 230–247. doi:10.1109/TAMD.2010.2056368.

[56] D. Pathak, P. Agrawal, A. A. Efros, T. Darrell, Curiosity-driven exploration by self-supervised prediction, in: Proc. 34th International Conference on Machine Learning, ICML'17, Vol. 70, JMLR.org, Sydney, NSW, Australia, 2017, pp. 2778–2787.
URL https://dl.acm.org/doi/10.5555/3305890.3305968

[57] L. L. Edith, C. Melanie, P. Doina, R. Bohdana, Risk-directed Exploration in Reinforcement Learning, IJCAI 2005 Workshop on Planning and Learning in A Priori Unknown or Dynamic Domains (2005).

[58] J. Yang, W. Qiu, A measure of risk and a decision-making model based on expected utility and entropy, in: European Journal of Operational Research, 2005, pp. 792–799. doi:10.1016/j.ejor.2004.01.031.

[59] A. Tijsma, M. Drugan, M. Wiering, Comparing exploration strategies for Q-learning in random stochastic mazes, 2016 IEEE Symposium Series on Computational Intelligence, SSCI 2016 (feb 2017). doi:10.1109/SSCI.2016.7849366.

[60] P. Geibel, F. Wysotzki, Risk-Sensitive Reinforcement Learning Applied to Control under Constraints, Journal of Artificial Intelligence Research 24 (2005) 81–108. doi:10.1613/jair.1666.

[61] O. Mihatsch, R. Neuneier, Risk-sensitive reinforcement learning, Machine learning 49 (2) (2002) 267–290.

[62] P. Campos, T. Langlois, Abalearn: Efficient self-play learning of the game abalone, INESC-ID, neural networks and signal processing group (2003).

[63] A. Gosavi, Reinforcement learning: A tutorial survey and recent advances, INFORMS Journal on Computing 21 (2) (2009) 178–192.

[64] J. Romoff, P. Henderson, A. Piche, V. Francois-Lavet, J. Pineau, Reward Estimation for Variance Reduction in Deep Reinforcement Learning, in: A. Billard, A. Dragan, J. Peters, J. Morimoto (Eds.), Proceedings of The 2nd Conference on Robot Learning, Vol. 87 of Proceedings of Machine Learning Research, PMLR, 2018, pp. 674–699.
URL https://proceedings.mlr.press/v87/romoff18a.html

[65] J. del R. Millán, Reinforcement learning of goal-directed obstacle-avoiding reaction strategies in an autonomous mobile robot, Robotics and Autonomous Systems 15 (4) (1995) 275–299. doi:10.1016/0921-8890(95)00021-7.

[66] P.-A. Andersen, M. Goodwin, O.-C. Granmo, CostNet: An End-to-End Framework for Goal-Directed Reinforcement Learning, in: Artificial Intelligence XXXVII, Vol. 12498 LNAI, Springer, Cham, 2020, pp. 94–107. doi:10.1007/978-3-030-63799-6_7.

[67] Y. Lu, Artificial intelligence: a survey on evolution, models, applications and future trends, https://doi.org/10.1080/23270012.2019.1570365 6 (1) (2019) 1–29. doi:10.1080/23270012.2019.1570365.
URL https://www.tandfonline.com/doi/abs/10.1080/23270012.2019.1570365

[68] F. Berkenkamp, M. Turchetta, A. P. Schoellig, A. Krause, Safe Model-based Reinforcement Learning with Stability Guarantees, in: I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems 30, Curran Associates, Inc., Long Beach, CA, USA, 2017, pp. 908–918.
URL https://papers.nips.cc/paper/6692-safe-model-based-reinforcement-learning-with-stability-guarantees

[69] Y. Chow, M. Ghavamzadeh, L. Janson, M. Pavone, Risk-Constrained Reinforcement Learning with Percentile Risk Criteria, Journal of Machine Learning Research 18 (1) (2017) 6070–6120.
URL http://www.jmlr.org/papers/volume18/15-636/15-636.pdf

[70] R. Cheng, G. Orosz, R. M. Murray, J. W. Burdick, End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks, in: 33rd AAAI Conference on Artificial Intelligence, AAAI 2019, 31st Innovative Applications of Artificial Intelligence Conference, IAAI 2019 and the 9th AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, 2019. doi:10.1609/aaai.v33i01.33013387.

[71] J. Yang, W. Qiu, A measure of risk and a decision-making model based on expected utility and entropy, European Journal of Operational Research 164 (3) (2005) 792–799. doi:10.1016/J.EJOR.2004.01.031.

[72] W. Saunders, G. Sastry, A. Stuhlmüller, O. Evans, Trial without Error: Towards Safe Reinforcement Learning via Human Intervention, in: Proceedings of the 17th

International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '18, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2018, pp. 2067–2069.

[73] M. Turchetta, A. Kolobov, S. Shah, A. Krause, A. Agarwal, Safe Reinforcement Learning via Curriculum Induction, in: H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems, Vol. 33, Curran Associates, Inc., 2020, pp. 12151–12162.
URL https://proceedings.neurips.cc/paper/2020/file/8df6a65941e4c9da40a4fb899de65c55-Paper.pdf

[74] F. Berkenkamp, A. Krause, A. P. Schoellig, Bayesian optimization with safety constraints: safe and automatic parameter tuning in robotics, Machine Learning 2021 (2021) 1–35doi:10.1007/S10994-021-06019-1.
URL https://link.springer.com/article/10.1007/s10994-021-06019-1

[75] P.-A. Andersen, M. Goodwin, O.-C. Granmo, Towards a Deep Reinforcement Learning Approach for Tower Line Wars, in: M. Bramer, M. Petridis (Eds.), Artificial Intelligence XXXIV, Springer International Publishing, Cham, 2017, pp. 101–114. doi:10.1007/978-3-319-71078-5_8.

[76] Y. Tian, Q. Gong, W. Shang, Y. Wu, C. L. Zitnick, ELF: An Extensive, Lightweight and Flexible Research Platform for Real-time Strategy Games, Advances in Neural Information Processing Systems (2017) 2656–2666arXiv:1707.01067.
URL http://arxiv.org/abs/1707.01067

[77] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, J. Quan, S. Gaffney, S. Petersen, K. Simonyan, T. Schaul, H. van Hasselt, D. Silver, T. Lillicrap, K. Calderone, P. Keet, A. Brunasso, D. Lawrence, A. Ekermo, J. Repp, R. Tsing, StarCraft II: A New Challenge for Reinforcement Learning, Proceedings, The Thirteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (2017) 86–92arXiv:1708.04782, doi:https://deepmind.com/documents/110/sc2le.pdf.
URL http://arxiv.org/abs/1708.04782

[78] Z.-J. Pang, R.-Z. Liu, Z.-Y. Meng, Y. Zhang, Y. Yu, T. Lu, On Reinforcement Learning for Full-Length Game of StarCraft, Proceedings of the AAAI Conference on Artificial Intelligence 33 (01) (2019) 4691–4698. doi:10.1609/AAAI.V33I01.33014691.

URL https://ojs.aaai.org/index.php/AAAI/article/view/4394

[79] S. Ontanon, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, M. Preuss, A survey of real-time strategy game AI research and competition in starcraft, IEEE Transactions on Computational Intelligence and AI in Games 5 (4) (2013) 293–311. doi:10.1109/TCIAIG.2013.2286295.
URL http://ieeexplore.ieee.org/document/6637024/

[80] N. A. Barriga, M. Stanescu, M. Buro, Game Tree Search Based on Non-Deterministic Action Scripts in Real-Time Strategy Games, IEEE Transactions on Computational Intelligence and AI in Games (2017) 1–1doi:10.1109/TCIAIG.2017.2717902.
URL http://ieeexplore.ieee.org/document/7954767/

[81] A. Shleyfman, A. Komenda, C. Domshlak, On combinatorial actions and CMABs with linear side information, in: Frontiers in Artificial Intelligence and Applications, Vol. 263, 2014, pp. 825–830. doi:10.3233/978-1-61499-419-0-825.

[82] M. Botvinick, S. Ritter, J. X. Wang, Z. Kurth-Nelson, C. Blundell, D. Hassabis, Reinforcement Learning, Fast and Slow., Trends in cognitive sciences 23 (5) (2019) 408–422. doi:10.1016/j.tics.2019.02.006.
URL http://www.ncbi.nlm.nih.gov/pubmed/31003893

[83] K. J. Roodbergen, I. F. A. Vis, A survey of literature on automated storage and retrieval systems, European Journal of Operational Research (2009). doi:10.1016/j.ejor.2008.01.038.

[84] M. Fraccaro, Deep latent variable models for sequential data, Ph.D. thesis, Technical University of Denmark (2018).
URL https://orbit.dtu.dk/en/publications/deep-latent-variable-models-for-sequential-data

[85] A. Razavi, A. van den Oord, O. Vinyals, Generating Diverse High-Fidelity Images with VQ-VAE-2, in: H. Wallach, H. Larochelle, A. Beygelzimer, F. Alché-Buc, E. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems 32, Curran Associates, Inc., Vancouver, BC, Canada, 2019, pp. 14837–14847.
URL http://papers.nips.cc/paper/9625-generating-diverse-high-fidelity-images-with-vq-vae-2

[86] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings, IEEE 86 (11) (1998) 2278–2323. arXiv:

1102.0183, doi:10.1109/5.726791.
URL http://ieeexplore.ieee.org/document/726791/

[87] P.-A. Andersen, M. Goodwin, O.-C. Granmo, The Dreaming Variational Autoencoder for Reinforcement Learning Environments, in: Max Bramer, M. Petridis (Eds.), Artificial Intelligence XXXV, xxxv Edition, Vol. 11311, Springer, Cham, 2018, pp. 143–155. doi:10.1007/978-3-030-04191-5_11.

[88] A. Doerr, C. Daniel, M. Schiegg, N.-T. Duy, S. Schaal, M. Toussaint, T. Sebastian, Probabilistic Recurrent State-Space Models, in: J. Dy, A. Krause (Eds.), Proceedings of the 35th International Conference on Machine Learning, Vol. 80 of Proceedings of Machine Learning Research, PMLR, 2018, pp. 1280–1289.
URL http://proceedings.mlr.press/v80/doerr18a.html

[89] C. Zhang, J. Butepage, H. Kjellstrom, S. Mandt, Advances in Variational Inference, IEEE Transactions on Pattern Analysis and Machine Intelligence 41 (8) (2019) 2008–2026. doi:10.1109/TPAMI.2018.2889774.

[90] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, L. K. Saul, Introduction to variational methods for graphical models, Machine Learning 37 (2) (1999) 183–233. doi:10.1023/A:1007665907178.
URL https://link.springer.com/article/10.1023/A:1007665907178

[91] R. S. Sutton, D. Precup, S. Singh, Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning, Artificial Intelligence 112 (1-2) (1999) 181–211.

[92] S. Ozair, Y. Li, A. Razavi, I. Antonoglou, A. van den Oord, O. Vinyals, Vector Quantized Models for Planning, in: Proc.. 38th International Conference on Machine Learning, PMLR 139, 2021, 2021, pp. 1–15. arXiv:2106.04615.
URL http://arxiv.org/abs/2106.04615

[93] P. Seetharaman, G. Wichern, B. Pardo, J. L. Roux, Autoclip: Adaptive gradient clipping for source separation networks, in: IEEE International Workshop on Machine Learning for Signal Processing, MLSP, Vol. 2020-Septe, IEEE Computer Society, 2020, pp. 1–6. arXiv:2007.14469, doi:10.1109/MLSP49062.2020.9231926.

[94] I. Loshchilov, F. Hutter, Decoupled Weight Decay Regularization, in: Proc. 7th International Conference on Learning Representations, ICLR'19, 2019, pp. 1–19.
URL https://openreview.net/forum?id=Bkg6RiCqY7

[95] P. Izmailov, D. Podoprikhin, T. Garipov, D. Vetrov, A. G. Wilson, Averaging Weights Leads to Wider Optima and Better Generalization, in: A. G. R. Silva, A. Globerson (Eds.), 34th Conference on Uncertainty in Artificial Intelligence 2018, Association For Uncertainty in Artificial Intelligence, 2018, pp. 876–885. arXiv:1803.05407.
URL http://arxiv.org/abs/1803.05407

[96] P.-A. Andersen, M. Goodwin, O.-C. Granmo, Increasing sample efficiency in deep reinforcement learning using generative environment modelling, Expert Systems (mar 2020). doi:10.1111/exsy.12537.

[97] S. Huang, S. Ontanon, C. Bamford, L. Grela, Gym-MicroRTS: Toward Affordable Full Game Real-time Strategy Games Research with Deep Reinforcement Learning, in: Proc. 3rd IEEE Conference on Games, 2021, p. 19. arXiv:2105.13807.
URL https://arxiv.org/abs/2105.13807v3

[98] H. Hu, Q. Wang, Implementation on benchmark of SC2LE environment with advantage actor - Critic method, 2020 International Conference on Unmanned Aircraft Systems, ICUAS 2020 (2020) 362–366doi:10.1109/ICUAS48674.2020.9214032.

[99] Z. Zha, B. Wang, X. Tang, Evaluate, explain, and explore the state more exactly: an improved Actor-Critic algorithm for complex environment, Neural Computing and Applications 2021 (2021) 1–12doi:10.1007/S00521-020-05663-3.
URL https://link.springer.com/article/10.1007/s00521-020-05663-3

[100] M. Janner, J. Fu, M. Zhang, S. Levine, When to Trust Your Model: Model-Based Policy Optimization, in: H. Wallach, H. Larochelle, A. Beygelzimer, F. Alché-Buc, E. Fox, R. Garnett (Eds.), Proc. 33rd Conference on Neural Information Processing Systems (NeurIPS), Curran Associates, Inc., Vancouver, BC, Canada, 2019, pp. 12519–12530.
URL http://papers.nips.cc/paper/9416-when-to-trust-your-model-model-based-policy-optimization.pdf

[101] D. Hafner, T. Lillicrap, J. Ba, M. Norouzi, Dream to Control: Learning Behaviors by Latent Imagination, in: Proc. 8th International Conference on Learning Representations, ICLR'20, 2020, pp. 1–26.
URL https://openreview.net/forum?id=S1lOTC4tDS

[102] A. Razavi, A. van den Oord, B. Poole, O. Vinyals, Preventing Posterior Collapse with delta-VAEs, in: Proc. 7th International Conference on Learning Representa-

tions, ICLR'19, 2019, pp. 1–24.
URL https://openreview.net/forum?id=BJe0Gn0cY7

A Micro RTS Results

Table 11: MicroRTS-basesWorkers8x8A results.

basesWorkers8x8A						
Agorithm	RandomBiasedAI	NaiveMCTS	Tiamat	Droplet	UTS_Imass	Win Ratio
A2C	0.99 ±0.01	0.88 ±0.06	0.46 ±0.06	0.65 ±0.27	0.46 ±0.19	0.69
DQN	0.99 ±0.01	0.86 ±0.06	0.67 ±0.07	0.56 ±0.11	0.64 ±0.24	0.74
RAINBOW	1	0.89 ±0.07	0.76 ±0.16	0.62 ±0.27	0.67 ±0.09	0.79
IMPALA	0.99 ±0.01	0.79 ±0.08	0.51 ±0.27	0.73 ±0.2	0.42 ±0.4	0.69
PPO	1	0.92 ±0.06	0.66 ±0.3	0.83 ±0.12	0.75 ±0.25	0.83
DVAE	0.96 ±0.03	0.87 ±0.03	0.42 ±0.07	0.73 ±0.03	0.56 ±0.19	0.71
S-ORACLE	0.98 ±0.02	0.87 ±0.01	0.63 ±0.32	0.75 ±0.08	0.57 ±0.41	0.76

Table 12: MicroRTS-basesWorkers16x16A results.

basesWorkers16x16A						
Agorithm	RandomBiasedAI	NaiveMCTS	Tiamat	Droplet	UTS_Imass	Win Ratio
A2C	0.93 ±0.05	0.75 ±0.12	0.54 ±0.36	0.47 ±0.02	0.61 ±0.1	0.66
DQN	0.91 ±0.01	0.72 ±0.21	0.57 ±0.3	0.75 ±0.21	0.52 ±0.04	0.69
RAINBOW	0.9 ±0.05	0.63 ±0.15	0.73 ±0.25	0.55 ±0.31	0.75 ±0.15	0.71
IMPALA	0.85 ±0.06	0.64 ±0.16	0.66 ±0.07	0.76 ±0.19	0.57 ±0.26	0.7
PPO	0.87 ±0.03	0.7 ±0.24	0.46 ±0.36	0.86 ±0.09	0.57 ±0.19	0.69
DVAE	0.9 ±0.1	0.7 ±0.04	0.66 ±0.22	0.67 ±0.15	0.77 ±0.18	0.74
S-ORACLE	0.94 ±0.06	0.75 ±0.23	0.61 ±0.03	0.66 ±0.18	0.75 ±0.21	0.74

Table 13: MicroRTS-BWDistantResources32x32 results.

BWDistantResources32x32						
Agorithm	RandomBiasedAI	NaiveMCTS	Tiamat	Droplet	UTS_Imass	Win Ratio
A2C	0.87 ±0.03	0.67 ±0.12	0.86 ±0.13	0.66 ±0.04	0.83 ±0.12	0.78
DQN	0.84 ±0.08	0.87 ±0.12	0.77 ±0.04	0.86 ±0.06	0.82 ±0.12	0.83
RAINBOW	0.81 ±0.16	0.99 ±0.01	0.76 ±0.05	0.89 ±0.08	0.86 ±0.13	0.86
IMPALA	0.87 ±0.11	0.94 ±0.05	0.97 ±0.02	0.95 ±0.04	0.91 ±0.07	0.93
PPO	0.9 ±0.08	0.97 ±0.02	0.93 ±0.01	0.91 ±0.01	0.81 ±0.02	0.9
DVAE	0.9 ±0.03	0.95 ±0.01	0.78 ±0.19	0.97 ±0.02	0.78 ±0.03	0.88
S-ORACLE	0.91 ±0.05	0.94 ±0.06	0.97 ±0.02	0.91 ±0.02	0.83 ±0.11	0.91

Table 14: MicroRTS-DoubleGame24x24 results.

DoubleGame24x24						
Agorithm	RandomBiasedAI	NaiveMCTS	Tiamat	Droplet	UTS_Imass	Win Ratio
A2C	0.78 ±0.01	0.97 ±0.03	0.89 ±0.09	0.89 ±0.05	0.95 ±0.05	0.9
DQN	0.81 ±0.13	0.91 ±0.04	0.94 ±0.02	0.89 ±0.03	0.32 ±0.67	0.77
RAINBOW	0.8 ±0.13	0.95 ±0.02	0.84 ±0.14	0.69 ±0.25	0.64 ±0.16	0.78
IMPALA	0.84 ±0.16	0.92 ±0.02	0.88 ±0.1	0.96 ±0.02	0.85 ±0.11	0.89
PPO	0.85 ±0.01	0.99 ±0.01	0.89 ±0.03	0.68 ±0.04	0.85 ±0.1	0.85
DVAE	0.75 ±0.04	0.96 ±0.03	0.93 ±0.07	0.96 ±0.03	0.95 ±0.01	0.91
S-ORACLE	0.73 ±0.27	0.92 ±0.08	0.95 ±0.05	0.91 ±0.03	0.91 ±0.01	0.88

Table 15: MicroRTS-FourBasesWorkers8x8 results.

FourBasesWorkers8x8						
Agorithm	RandomBiasedAI	NaiveMCTS	Tiamat	Droplet	UTS_Imass	Win Ratio
A2C	1.0 ±0.0	0.91 ±0.08	0.95 ±0.05	0.84 ±0.14	0.81 ±0.17	0.9
DQN	1.0 ±0.0	0.89 ±0.03	0.78 ±0.03	0.64 ±0.12	0.71 ±0.08	0.8
RAINBOW	1.0 ±0.0	0.96 ±0.02	0.76 ±0.12	0.75 ±0.23	0.64 ±0.09	0.82
IMPALA	1.0 ±0.0	0.67 ±0.09	0.75 ±0.25	0.77 ±0.11	0.8 ±0.06	0.8
PPO	1.0 ±0.0	0.87 ±0.02	0.81 ±0.1	0.75 ±0.08	0.81 ±0.01	0.85
DVAE	1.0 ±0.0	0.75 ±0.12	0.75 ±0.03	0.96 ±0.03	0.64 ±0.36	0.82
S-ORACLE	1.0 ±0.0	0.87 ±0.02	0.82 ±0.13	0.67 ±0.24	0.78 ±0.02	0.83

Table 16: MicroRTS-NoWhereToRun9x8 results.

NoWhereToRun9x8						
Agorithm	RandomBiasedAI	NaiveMCTS	Tiamat	Droplet	UTS_Imass	Win Ratio
A2C	1.0 ±0.0	0.86 ±0.13	0.75 ±0.14	0.85 ±0.03	0.89 ±0.02	0.87
DQN	1.0 ±0.0	0.88 ±0.04	0.66 ±0.1	0.81 ±0.19	0.86 ±0.02	0.84
RAINBOW	0.98 ±0.01	0.82 ±0.17	0.89 ±0.05	0.86 ±0.02	0.84 ±0.16	0.88
IMPALA	1.0 ±0.0	0.93 ±0.02	0.96 ±0.02	0.87 ±0.02	0.88 ±0.01	0.93
PPO	0.99 ±0.01	0.96 ±0.02	0.88 ±0.07	0.84 ±0.11	0.82 ±0.14	0.9
DVAE	0.99 ±0.01	0.91 ±0.04	0.75 ±0.23	0.83 ±0.15	0.88 ±0.07	0.87
S-ORACLE	1.0 ±0.0	0.91 ±0.02	0.89 ±0.03	0.88 ±0.12	0.89 ±0.05	0.91

Table 17: MicroRTS-TwoBasesBarracks16x16 results.

TwoBasesBarracks16x16						
Agorithm	RandomBiasedAI	NaiveMCTS	Tiamat	Droplet	UTS_Imass	Win Ratio
A2C	0.97 ±0.01	0.91 ±0.04	0.97 ±0.01	0.89 ±0.07	0.89 ±0.04	0.93
DQN	0.96 ±0.02	0.9 ±0.02	0.89 ±0.1	0.89 ±0.1	0.78 ±0.03	0.88
RAINBOW	0.96 ±0.02	0.97 ±0.03	0.88 ±0.05	0.83 ±0.1	0.8 ±0.18	0.89
IMPALA	0.96 ±0.03	0.98 ±0.02	0.92 ±0.04	0.88 ±0.12	0.92 ±0.06	0.93
PPO	1.0 ±0.0	0.99 ±0.01	0.98 ±0.01	0.85 ±0.1	0.97 ±0.02	0.96
DVAE	0.96 ±0.01	0.89 ±0.03	0.85 ±0.02	0.99 ±0.01	0.86 ±0.12	0.91
S-ORACLE	0.97 ±0.01	0.9 ±0.04	0.82 ±0.06	0.9 ±0.09	0.88 ±0.04	0.89

B Hyperparameters

Table 18: Set of tunable parameters in S-ORACLE. In addition to this incomplete list, the algorithm has options for controlling model complexity such as neuron counts and number of layers.

Hyperparameter	**Values**	**Selected**	**Comment**
Batch Size	$\mathbb{Z}^+$	48	Number of sequence batches
Sequence Size	$\mathbb{Z}^+$	48	Number of frames in a sequence
Buffer Size	$\mathbb{Z}^+$	9 000	Replay buffer
Reward Scaling	$\mathbb{R}$	1.0	Scaling of the reward objective
Cost Scaling	$\mathbb{R}$	1.0	Scaling of the cost objective
VQ Scaling	$\mathbb{R}$	0.1	Scaling of the VQ objective
KL Scaling	$\mathbb{R}$	1.0	Scaling of the KL objective (KL-β)
KL Minimum Nats	$\mathbb{R}$	3.0	Minimal information loss
Optimizer		AdamW	AdamW improves generalization, see [94]
Gradient Clipping	$\mathbb{R}$	100.0 $\pm$	Clip gradients to increase learning stability
Adaptive GC	$\mathbb{B}$	1	Based on the history of gradient norms[93]
Learning Rate	$\mathbb{R}$	0.0001	Low Learning rate to improve stability.
Latent Leaps	$\mathbb{Z}^+$	30	Number of leaps into future states.
Dynamics Model RNN		LSTM	
Activation Functions		ELU	
Enc/Dec Neurons	$\mathbb{Z}^+$	1024	
Stochastic Reward	$\mathbb{B}$	1	Sample rewards under Gaussian assumptions
Stochastic Costs	$\mathbb{B}$	1	Sample costs under Gaussian assumptions
Discount Factor γ	$\mathbb{R}$	0.96	Discount factor used in value-based algorithms
Risk-Entropy Weight w (Eq 20)	$\mathbb{R}$	0.6	Risk-Directed Exploration entropy weight
Risk-Utility Function ρ(Eq 22)	$\mathbb{R}$	0.75	Weight of risk in utility function.
Risk function weight β (Eq 23)	$\mathbb{R}$	0.40	Weight of the risk functipn ω.
VQ-VAE weight β_{vq}	$\mathbb{R}$	0.05	Weight of VQ-VAE encoding updates.