UiA University of Agder

Master

The Potential and Limitations of the Tsetlin Machine in Model-Free Reinforcement Learning

DIDRIK KALLHOVD DRØSDAL and ANDREAS GRIMSMO

SUPERVISOR
Ole-Christoffer Granmo and Per-Arne Andersen

University of Agder, 2023
Faculty of Engineering and Science
Department of ICT

Obligatorisk gruppeerklæring

Den enkelte student er selv ansvarlig for å sette seg inn i hva som er lovlige hjelpemidler, retningslinjer for bruk av disse og regler om kildebruk. Erklæringen skal bevisstgjøre studentene på deres ansvar og hvilke konsekvenser fusk kan medføre. Manglende erklæring fritar ikke studentene fra sitt ansvar.

1.	Vi erklærer herved at vår besvarelse er vårt eget arbeid, og at vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen.	Ja
2.	**Vi erklærer videre at denne besvarelsen:** • Ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands. • Ikke refererer til andres arbeid uten at det er oppgitt. • Ikke refererer til eget tidligere arbeid uten at det er oppgitt. • Har alle referansene oppgitt i litteraturlisten. • Ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse.	Ja
3.	Vi er kjent med at brudd på ovennevnte er å betrakte som fusk og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. Universitets- og høgskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§ 31.	Ja
4.	Vi er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert.	Ja
5.	Vi er kjent med at Universitetet i Agder vil behandle alle saker hvor det forligger mistanke om fusk etter høgskolens retningslinjer for behandling av saker om fusk.	Ja
6.	Vi har satt oss inn i regler og retningslinjer i bruk av kilder og referanser på biblioteket sine nettsider.	Ja
7.	Vi har i flertall blitt enige om at innsatsen innad i gruppen er merkbart forskjellig og ønsker dermed å vurderes individuelt. Ordinært vurderes alle deltakere i prosjektet samlet.	Nei

Publiseringsavtale

Fullmakt til elektronisk publisering av oppgaven Forfatter(ne) har opphavsrett til oppgaven. Det betyr blant annet enerett til å gjøre verket tilgjengelig for allmennheten (Åndsverkloven. §2).
Oppgaver som er unntatt offentlighet eller taushetsbelagt/konfidensiell vil ikke bli publisert.

Vi gir herved Universitetet i Agder en vederlagsfri rett til å gjøre oppgaven tilgjengelig for elektronisk publisering:	Ja
Er oppgaven båndlagt (konfidensiell)?	Nei
Er oppgaven unntatt offentlighet?	Nei

Acknowledgements

We would like to express our deepest gratitude to our supervisor, associate professor Per-Arne Andersen, for his guidance, encouragement, and insightful critiques during the course of this research work. His knowledge and expertise in the field has been invaluable, and his guidance and suggestions have improved the quality of our work substantially.

Our thanks also go to the University of Agder, which provided us with the resources to work on this research. As well as provide us with guidance and help when we encountered challenges. We would especially like to thank professor Ole-Christoffer Granmo for his expertise and assistance with the Tsetlin Machine and how it functions.

Finally, we would also love to express our gratitude to our friends and families for their constant support and encouragement throughout our studies and this research work.

Abstract

This paper aims to investigate the potential of model-free reinforcement learning using the Tsetlin Machine by evaluating its performance in widely recognized benchmark environments for reinforcement learning: Cartpole and Pong. Our study is divided into two primary objectives. First, we analyze the effectiveness of the Tsetlin Machine in learning from the actions of expert agents in the Cartpole environment. Second, we assess the ability of the multiclass Tsetlin Machine to learn to play both Cartpole and Pong environments from scratch.

Our findings indicate that the Tsetlin Machine can successfully learn and solve the Cartpole environment. Although the Pong environment remains unsolved, the Tsetlin Machine demonstrates its learning capabilities by scoring several points in multiple test runs, even managing to win in some of them. Through our empirical investigation, we conclude that the Tsetlin Machine exhibits promise in the field of reinforcement learning. Nonetheless, further research is needed to address the limitations observed in its performance in some of the examined environments.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Rapid advancements in reinforcement learning (RL) have led to remarkable success in various video game environments, including StarCraft, Chess, Go, and Dota II, among others [17]. These environments serve as critical benchmarks for assessing and evaluating the performance of AI agents. Reinforcement learning research enables intriguing capabilities, such as fine-tuning the ChatGPT algorithm using reinforcement learning with human feedback[9]. One of the key advantages of RL is the ability to learn complex, non-linear decision-making processes that may be challenging or impossible to model using supervised learning, much due to the limited availability of data. Through continuous interaction with the user-defined environment, these learning machines progressively improve their performance over time.

The Tsetlin Machine (TM) is an emerging approach in machine learning that utilizes simple and interpretable rules to solve complex problems, demonstrating high computational efficiency and the ability to perform training and inference on-device. This paper aims to explore the potential of Tsetlin Machines in the context of model-free reinforcement learning (RL), an area where the TM has not yet been extensively studied. By combining the strengths of reinforcement learning with the unique capabilities of Tsetlin machines, we hope to advance the field of artificial intelligence and contribute to developing efficient, powerful, and portable learning algorithms. This study focuses on two well-known RL benchmarks, CartPole and Pong. Although these environments have simple state and action spaces, solving them in a fully model-free setting with TMs remains an open question, which we aim to address.

1.1 Contribution

To systematically evaluate the performance of Tsetlin machines, our study employs a two-step approach. First, we train Tsetlin machines to imitate established deep-learning-based agents, providing insight into TM's capacity to learn complex policies. Second, we explore the potential of Tsetlin machines to learn functional policies from scratch, enabling them to devise novel strategies and solutions in the selected game environments. Through our empirical study, identify the strengths and weaknesses of Tsetlin machines in dynamic scenarios and uncover any limitations in their clause formation and retention capabilities. Ultimately, this research contributes to a deeper understanding of Tsetlin machines' applicability in reinforcement learning tasks and helps to determine their viability as an alternative to state-of-the-art techniques. The objective of this paper is to introduce reinforcement learning to the Tsetlin Machine and empirically evaluate its effectiveness, offering valuable insights into TM's performance in such model-free RL settings[1]. All our work will be uploaded and added

[1]Part of this Master's thesis was submitted as a paper to the ISTM 2023 conference in Spring 2023. [6]

to the TMU GitHub[2]. Currently, it can be found on our own GitHub.[3]

1.2 Motivation

The key motivation behind this work is threefold. First, reinforcement learning (RL) has proven to be an exceptionally powerful technique, serving as the foundation for some of the most successful algorithms in the literature, such as ChatGPT. Second, there has been very limited work exploring TM-driven RL approaches, and no prior research has successfully developed a model-free RL mechanism using TMs. Third, the Tsetlin machine is rapidly advancing, showcasing significant improvements in computational efficiency, enabling algorithms to perform training and inference on-device[10]. This is a considerable advantage compared to the resource-intensive nature of Neural Networks and traditional machine-learning methods. By combining the strengths of reinforcement learning with the unique capabilities of Tsetlin machines, we aim to progress one step closer towards a groundbreaking RL approach that can, with the help of the Tsetlin machine, revolutionize the field of artificial intelligence and contribute to the development of efficient, powerful, and portable learning algorithms.

1.3 Overview of paper

The rest of the paper is organized as follows. Section 2 provides an overview of the relevant background information pertaining to the Tsetlin machine and the theory essential for this study. Section 2.4 discusses the foundations of our work and highlights other related research that informed our approach. Section 3 describes the proposed approach to imitation and model-free TM-based RL. Section 4 highlights our findings and results from the experiments included in this study. Section 5 presents a comprehensive analysis and theoretical discussion of our proposed strategies, explaining the motivations behind their performance and behaviour. Finally, we conclude the paper in Section 6, where we summarize our contributions and the significance of our proposed approach. Lastly, in Section 6.1, we outline a roadmap for future research directions that may build upon the findings of this study.

[2]TMU GitHub URL: https://github.com/cair/tmu, to be added soon
[3]GitHub URL: https://github.com/AndreasGRIMSMO/Tsetlin-Reinforcement

Chapter 2

Theory

2.1 Tsetlin Machine

2.1.1 Tsetlin Automata

The Tsetlin Automaton [7] is a type of finite-state automaton that was originally proposed as a mathematical model of learning and decision-making processes. A Tsetlin Automaton has a finite number of states, each representing a different action or policy. It learns by receiving feedback in the form of penalties or rewards and adjusts its state accordingly.
The original Tsetlin Automaton consists of the following components:

- A set of N states, $S = \{s_1, s_2, \ldots, s_N\}$, where each state s_i represents an action or policy.
- A learning parameter, p, which controls the rate of learning.
- A reward function, $R(s_i)$, which provides feedback based on the chosen action.
- A penalty function, $P(s_i)$, which also provides feedback based on the chosen action.

The state transition rules are defined as follows:

- If the automaton is in state s_i and receives a reward, it moves to state s_{i+1} with probability p, and remains in state s_i with probability $1 - p$.

$$s_i = s_i + I(\text{reward}) * (rand() < p)$$

- If the automaton is in state s_i and receives a penalty, it moves to state s_{i-1} with probability p, and remains in state s_i with probability $1 - p$.

$$s_i = s_i - I(\text{penalty}) * (rand() < p)$$

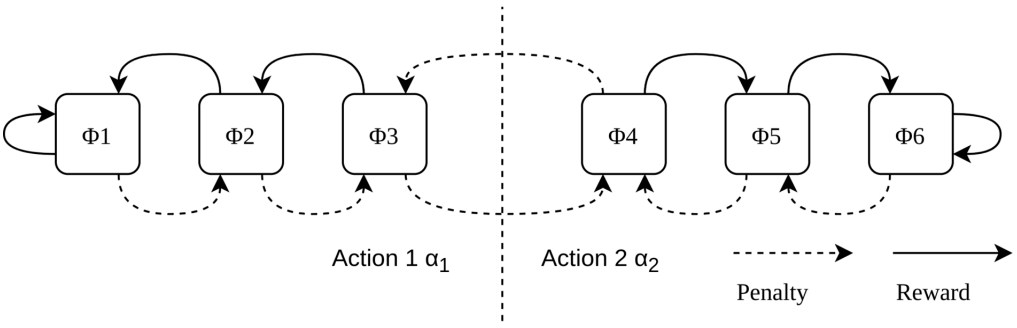

Figure 2.1: Illustration on the Tsetlin automata behavior [7]

The Tsetlin Automaton learns by adjusting its state based on the received feedback until it converges to an optimal state.

2.1.2 Tsetlin Machine

The Tsetlin Machine extends the Tsetlin Automaton concept to a multi-agent system, where each agent represents a feature or a pattern in the input data. The Tsetlin Machine is capable of performing classification and regression tasks by learning conjunctive clauses that represent relationships between input features and output labels.
The Tsetlin Machine consists of the following components:

- A set of M Tsetlin Automata, $A = \{A_1, A_2, \ldots, A_M\}$, where each automaton A_i learns to represent a feature or a pattern.
- An action space, C, which is a set of clauses that capture relationships between input features and output labels.
- A learning parameter, p, which controls the rate of learning for all automata.
- A reward function, $R(A_i, s_i)$, and a penalty function, $P(A_i, s_i)$, which provide feedback to each automaton based on the chosen action and the ground truth label.

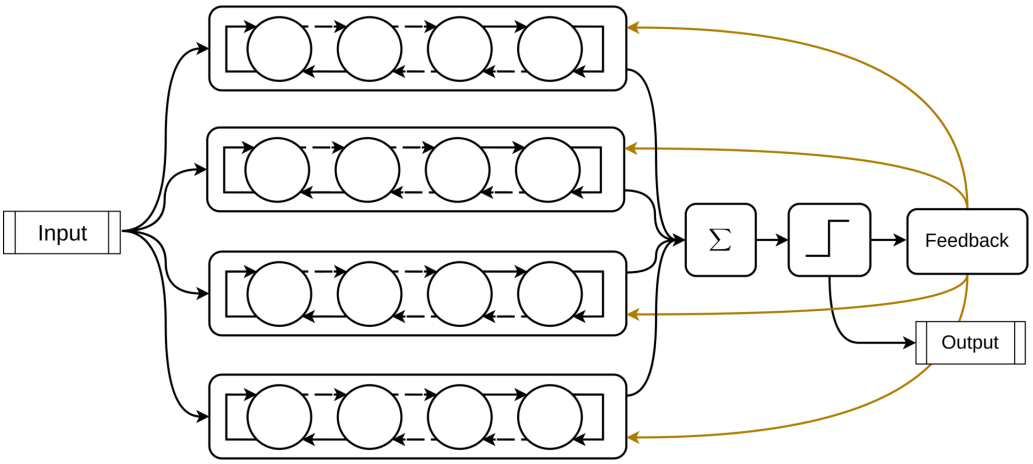

Figure 2.2: Tsetlin machine block [7]

Each Tsetlin Automaton in the Tsetlin Machine operates independently and follows the same state transition rules as the original Tsetlin Automaton. The Tsetlin Machine learns by adjusting the states of its automata based on the received feedback until it converges to an optimal set of clauses that can accurately predict the output labels.
The decision function of the Tsetlin Machine is a summation of the learned clauses:

$$y(x) = \sum_{i=1}^{M} c(x_i) * w_i$$

where $y(x)$ is the predicted output, x_i is an input feature, $c(x_i)$ is a learned clause, and w_i is the weight associated with the clause.

2.1.3 Tsetlin Machine's learning process

The Tsetlin Machine's learning process can be described as a three-step procedure:

Initialization
The Tsetlin Automata are initialized with random states. The weights for each clause, w_i, are initialized to zero.

Learning
The Tsetlin Machine iteratively processes the input data, receiving feedback from the reward and penalty functions. For each example in the input data, the Tsetlin Machine performs the following steps:

1. Each Tsetlin Automaton selects a clause based on its current state.
2. The decision function computes the predicted output, $y(x)$, based on the selected clauses and their weights.
3. The reward and penalty functions provide feedback to each Tsetlin Automaton based on the correctness of the prediction. The automata adjust their states according to the state transition rules and learning parameter p.
4. The weights of the clauses, w_i, are updated based on the prediction error and a user-defined learning rate.

$$w_i = w_i + \text{learning\_rate} * (y_{\text{true}} - y(x)) * c(x_i)$$

Convergence
The learning process continues until a stopping criterion is met, such as a maximum number of iterations or a minimum error threshold. The Tsetlin Machine's final output is a set of learned clauses and their weights, which can be used to make predictions on new input data. The Tsetlin Machine's transparent structure and simple learning process make it an attractive alternative to more complex models like neural networks, particularly in applications where interpretability is crucial. Its game-theoretic foundation enables it to learn optimal patterns and relationships in the input data, allowing it to perform well in various classification and regression tasks.

2.1.4 The feedback

The Tsetlin Machine uses two types of feedback to guide learning:

Type-I Feedback (Reward): This feedback is given when the automaton makes a correct decision. The purpose of the reward is to reinforce the current action. That means, if the current action has led to a correct decision, the probability of selecting the same action in the future should be increased. In the context of the Tsetlin Machine, a correct decision could be the correct inclusion or exclusion of a literal in a clause.

Type-II Feedback (Penalty): This feedback is given when the automaton makes an incorrect decision. The purpose of the penalty is to discourage the current action. That means, if the current action has led to an incorrect decision, the probability of selecting the same action in the future should be decreased. In the context of the Tsetlin Machine, an incorrect decision could be the incorrect inclusion or exclusion of a literal in a clause.

The balance between these two types of feedback allows the Tsetlin Machine to gradually learn the best combination of literals for each clause, and therefore the best decision-making model for the given task. By using this approach, the Tsetlin Machine can create interpretable models that perform comparably to other machine learning methods on a variety of tasks.

2.1.5 Binarizer process

The binarization process constitutes a vital pre-processing technique in TMs, transforming numerical data into binary values through a threshold value. Values exceeding the threshold

are assigned 1, while those equal to or below receive 0. This process proves advantageous for logistic-based machine learning models that necessitate binary outputs. Conventionally, the user determines the threshold; however, alternative approaches, such as utilizing a fitting function for input array processing, can establish this value. This method identifies unique feature values, downsampling them if they exceed the maximum bit limit. Consequently, the binarizer processes input data, generating binary representations for each unique feature value, resulting a representation that is compatible with the Tsetlin machine.

2.1.6 Multiclass classification

Tsetlin Machines can be employed for classification tasks by using an argmax function instead of a threshold. The output of a Multi-Class Tsetlin Machine, as illustrated in Figure 2.3, is given by:

$$\hat{y} = \arg\max_{i=1,\ldots,m} \left(\sum_{j=1}^{n/2} C_{i+j}(X) - \sum_{j=1}^{n/2} C_{i-j}(X) \right) \tag{2.2}$$

In this equation, m represents the total number of distinct classes in the problem, while the + and - symbols indicate the polarity of the clauses (including and excluding literals, respectively).

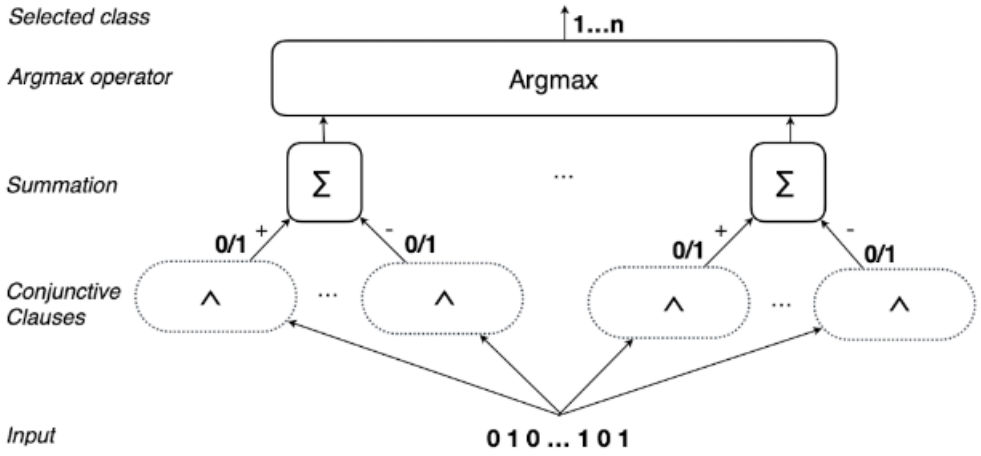

Figure 2.3: Multiclass classification [7]

2.2 Reinforcement learning

Reinforcement learning (RL)[17] is a machine learning paradigm that involves training an agent to make decisions by interacting with an environment. The core components of RL are the agent, environment, states, actions, and rewards. Here's a basic overview of reinforcement learning:

Agent: The agent is the decision-making entity that interacts with the environment. It can be anything from a robot to a software program.

Environment: The environment is the context in which the agent operates. It provides the agent with feedback in the form of rewards or penalties based on the agent's actions.

States: A state represents the current situation or context in the environment. The agent perceives the state and takes actions accordingly.

Actions: Actions are the set of possible moves or decisions the agent can make in a given state. The agent selects actions based on its policy, which is a mapping of states to actions.

Rewards: Rewards are the feedback given to the agent by the environment based on the

agent's actions. They can be positive (for desired actions) or negative (for undesired actions). The goal of the agent is to maximize the cumulative reward it receives over time.
Transition model: A probability distribution that represents the likelihood of transitioning from one state to another given a particular action. It is represented as $P(s'|s,a)$, where s and s' are states, and a is the action taken, it is defined as $\mathcal{R}(s,a) = \mathbb{E}[r_{t+1}|s_t = s, a_t = a]$.
γ: The discount factor, which controls the relative importance of immediate and future rewards.

The environment is typically modeled as a Markov Decision Process (MDP)[17] , which is characterized by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where:

- $\mathcal{S}$ is a set of states
- $\mathcal{A}$ is a set of actions
- $\mathcal{P}$ is the state transition probability function, defined as $\mathcal{P}(s'|s,a) = \Pr(s_{t+1} = s'|s_t = s, a_t = a)$
- $\mathcal{R}$ is the reward function
- γ is the discount factor

At each time step, the agent observes the current state s_t, takes an action a_t according to its policy $\pi(a_t|s_t)$, and receives a reward r_t from the environment. The goal of the agent is to learn a policy that maximizes the expected cumulative reward over time, also known as the return $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$.
In MDPs, the environment's dynamics are assumed to satisfy the Markov property, which states that the future state depends only on the current state and action, and not on the history of past states and actions. This property simplifies the learning process by reducing the problem to finding the optimal policy that considers only the current state.
Reinforcement learning problems can be formalized as MDPs, and various algorithms have been developed to learn the optimal policy in this framework. Examples include Q-learning, a value-based method that iteratively updates the action-value function to estimate the optimal policy, and policy gradient methods, which directly optimize the policy based on observed state transitions and rewards. By updating the policy or value function based on the interactions with the environment, RL algorithms aim to improve the agent's performance in achieving its goal of maximizing the expected cumulative reward.

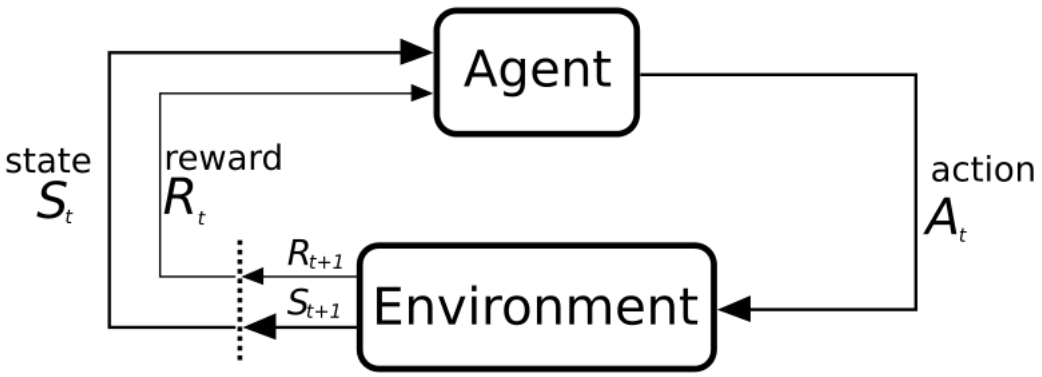

Figure 2.4: Reinforcement learning diagram of a Markov decision process[**mar**].

2.2.1 ϵ-Greedy

A central part of RL is the use of sampling techniques where one such technique is the ϵ-greedy algorithm. With the ϵ-greedy approach, the agent selects a random action with probability ϵ and the highest Q-value with probability $1-\epsilon$. The value of ϵ typically decreases over time, allowing the agent to prioritize exploration during the early stages of learning and gradually shift towards exploitation as it gains experience.

Algorithm 1: Epsilon-Greedy Algorithm

```
Generate a random number p between 0 and 1
if p < epsilon then
    pull random action
else
    pull best action
end if
observe reward of action
```

2.2.2 Deep Q-Network

Deep Q-Learning (DQL) is an advanced variant of the classical Q-Learning algorithm [11], which is a model-free, value-based reinforcement learning method. Q-Learning aims to learn the optimal action-value function $Q^*(s, a)$, which represents the expected return when taking action a in state s and following the optimal policy thereafter. The core update rule for Q-Learning is given by the Temporal Difference (TD) update: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left(r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)\right)$, where α is the learning rate and γ is the discount factor.

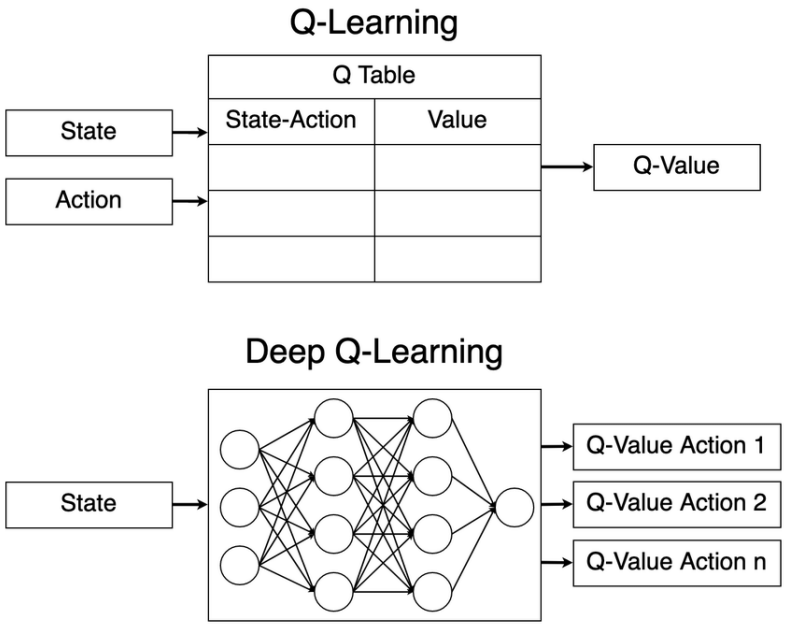

Figure 2.5: Figure comparing Q-Learning with Deep Q-Learning[1]

In DQL, deep neural networks are used to approximate the action-value function $Q(s, a; \theta)$, enabling the algorithm to handle high-dimensional state spaces and complex tasks. However, learning with neural networks can be unstable and challenging due to issues such as

correlations between consecutive samples and overestimation of action-values. To address these problems, DQL introduces two key techniques: experience replay and target networks. Experience replay is a technique that maintains a memory buffer D to store past transitions (s_t, a_t, r_t, s_{t+1}). During training, instead of using online samples, the algorithm samples random minibatches of transitions from the buffer D. This method breaks the correlations between consecutive samples and promotes more stable learning.
Target networks, on the other hand, involve maintaining a separate network $Q(s, a; \theta^-)$ that is a slower, lagged copy of the main network. This auxiliary network is used to compute target values for the loss function. Periodically, the parameters of the target network are updated to match the main network. By decoupling the target values from the current network parameters, target networks help mitigate the overestimation bias that can arise in Q-Learning.

The DQL algorithm optimizes the following loss function:
$L(\theta) = \mathbb{E}(s_t, a_t, r_t, st + 1) \sim D \left[(r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-) - Q(s_t, a_t; \theta))^2 \right]$. By minimizing this loss, the algorithm learns an approximation of the action-value function, enabling the agent to make better decisions and achieve improved performance in various tasks.

2.2.3 Proximal Policy Optimization

Proximal Policy Optimization (PPO)[16] is a widely-used policy optimization algorithm in reinforcement learning that adeptly balances exploration and exploitation. PPO achieves this balance by incorporating the advantages of trust region methods and first-order optimization. Trust region methods ensure that updates to the policy do not deviate too far from the current policy, while first-order optimization techniques are computationally efficient.
In PPO, the objective is to maximize the surrogate objective function
$L(\theta) = \mathbb{E}(s_t, a_t, r_t) \sim \pi_{\text{old}} \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} A^{\pi_{\text{old}}}(s_t, a_t) \right]$, where $A^{\pi_{\text{old}}}(s_t, a_t)$ represents the advantage function under the old policy. The advantage function quantifies the relative value of taking a specific action compared to the expected value of following the current policy.
To prevent excessively large policy updates that could destabilize the learning process, PPO introduces a clipping mechanism for the policy ratio $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$. This ratio is clipped within a predefined range $[1 - \epsilon, 1 + \epsilon]$, where ϵ is a hyperparameter controlling the allowed deviation from the old policy. This clipping mechanism ensures that the new policy remains close to the old policy, effectively constraining the policy update within a trust region.
With its combination of trust region constraints and first-order optimization, the PPO algorithm is computationally efficient and exhibits stable performance across a broad spectrum of tasks. The capacity to strike a balance between exploration and exploitation has popularized its use in reinforcement learning contexts, from robotics to gaming scenarios.

2.2.4 Advantage Actor Critic

Advantage Actor-Critic (A2C)[12] is a model-free reinforcement learning algorithm that unifies the strengths of policy gradient methods and value-based approaches. The central idea of A2C is to learn both the policy and the state-value function simultaneously using two separate neural networks, the actor and the critic.
The actor network, represented by $\pi(a|s; \theta_\pi)$, is responsible for learning the policy, which maps states to actions. It aims to maximize the expected return by selecting actions that yield higher rewards in a given state. On the other hand, the critic network, represented by $V(s; \theta_v)$, estimates the state-value function. The state-value function predicts the expected return from a given state, assuming the actor follows its current policy. The purpose of the critic is to evaluate the quality of the current policy and provide a baseline for policy improvement.
To improve the policy, A2C computes the advantage function, $A(s_t, a_t)$, which measures the

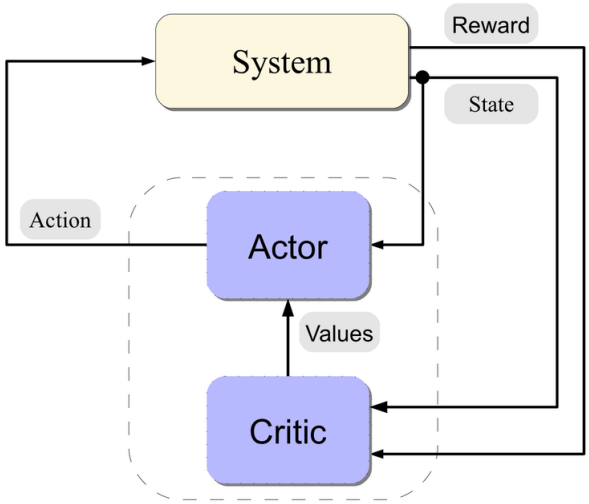

Figure 2.6: Illustration of the advantage actor critic algorithm [15]

relative value of taking action a_t in state s_t over following the policy's average behavior. The advantage function is defined as the difference between the observed return and the estimated state-value: $A(s_t, a_t) = r_t + \gamma V(s_{t+1}; \theta_v) - V(s_t; \theta_v)$. Intuitively, a positive advantage indicates that the chosen action is better than the average action in the current state, while a negative advantage suggests that the chosen action is worse.

The actor network updates its policy by performing gradient ascent on the expected return, weighted by the advantage function. This encourages the actor to increase the probability of actions with positive advantages while decreasing the probability of actions with negative advantages. By using the advantage function as a baseline, the A2C algorithm reduces the variance of policy gradient updates, leading to more stable learning and improved performance.

2.3 Environments

In this part we discuss the environments used in our experiments.

2.3.1 Cartpole

The CartPole-v1 environment is one of the classic control problems in reinforcement learning and is part of the OpenAI Gym environment suite. The goal of the environment is to balance a pole attached by an un-actuated joint to a cart, which can only move horizontally along a frictionless track. The pole starts upright position, and the cart can move left or right to keep the pole from falling over[3]. The episode is terminated when the cart's position gets

Table 2.1: Observation space for cartpole [4]

Num	**Observation**	**Min**	**Max**
0	Cart Position	-4.8	4.8
1	Cart Velocity	-Inf	Inf
2	Pole Angle	∼-0.418 rad (-24°)	∼0.418 rad (24°)
3	Pole Angular Velocity	-Inf	Inf

below -2.4 or above 2.4, as well as when the angle goes above 12 degrees or below -12 degrees. An episode is truncated when the score gets to 500, in other words, when it goes 500 steps

without being terminated[4][5]. The cartpole problem is generally considered solved when the agent gets an average reward of 195.0 over 100 episodes.

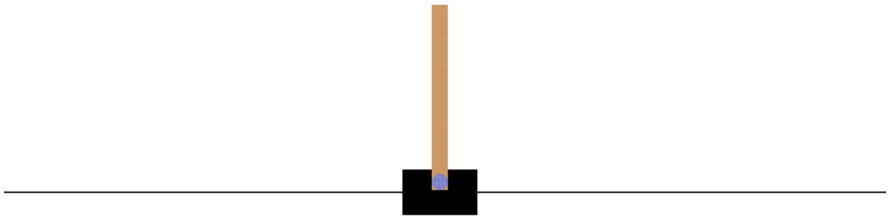

Figure 2.7: Graphical representation of the cartpole environment[5]

2.3.2 Pong

Pong is a simple two-player game where each player controls a paddle that can be moved up or down to hit a ball back and forth. The game's goal is to prevent the ball from passing the player's paddle while trying to get the ball past the opponent's paddle. In the Pong-ram-v4 environment from the Gymnasium API[13], the game ends when one player reaches a score of 21 points, or after a certain number of game steps have been reached, meaning the smallest possible reward throughout an episode is -21, and the highest is 21.

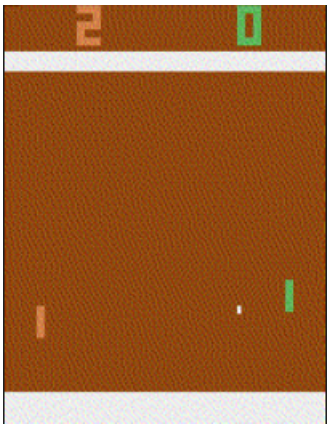

Figure 2.8: RGB representation of the Pong environment[13]

Table 2.2: Action and observation space for cartpole [4]

Action Space	Discrete(2)
Observation Shape	(4,)
Observation High	[4.8 inf 0.42 inf]
Observation Low	[-4.8 -inf -0.42 -inf]

2.4 Related work

2.4.1 Interpretable Reinforcement Learning with the Regression Tsetlin Machine

Interpretable Reinforcement Learning with the Regression Tsetlin Machine[18] is a Master's thesis authored by Varun Ravi Varma, which explores the application of the regression Tsetlin machine (RTM)[2] in reinforcement learning. Varma employs RTM to simulate Q-learning in Cartpole and mountain car environments. Although the CartPole agents' rewards do not surpass 200, the graphical representation of results exhibits a progressive reward increase over time.

2.4.2 Off-policy and on-policy reinforcement learning with the Tsetlin machine

Off-policy and on-policy reinforcement learning with the Tsetlin machine[14] is a research paper by Saeed Rahimi Gorji and Ole-Christoffer Granmo that investigates off-policy and on-policy reinforcement learning methodologies using Tsetlin machines (TM). The authors assess the performance of their approach in gridworld instances and utilize a multi-layer perceptron (MLP) as a benchmark for comparison. The study's results demonstrate the RTM's ability to effectively navigate a small grid world.

2.4.3 Tsetlin machine limited research

In TM-driven reinforcement learning, it is important to note that the existing body of research remains limited. While the studies above have made significant strides in exploring TM-based approaches for RL tasks, there is a notable gap in the literature concerning direct policy learning in a model-free manner. This highlights the need for further investigation into how Tsetlin machines can be utilized effectively for model-free reinforcement learning and emphasizes the potential for novel contributions in this area. As such, we focus on developing and evaluating model-free methodologies that leverage Tsetlin machines for learning policies directly, thereby expanding our understanding of the capabilities and limitations of TM-driven reinforcement learning approaches.

Chapter 3

Model-Free Reinforcement Learning using Tsetlin Machines

In this section, we present a comprehensive exploration of Tsetlin Machines' (TMs) potential in reinforcement learning. Our approach to this study is that of Design Science [8], meaning that we seek to create an artifact and evaluate its performance to gain insight and understanding. Our investigation encompasses two primary studies: (1) Imitation Learning, where TMs acquire policies by emulating expert agents or optimal mathematical models, and (2) Model-free Learning, where TMs develop policies from scratch through various strategies. We address the challenges posed by continuous observation spaces in the CartPole and Pong environments and outline the essential data preprocessing steps required to make these environments compatible with TMs. Moreover, we introduce innovative learning strategies tailored to uncover TMs' capabilities in complex reinforcement learning tasks.

3.1 Data Preprocessing and Reward Functions

The **CartPole** environment features a continuous observation space, which the Tsetlin machine cannot process directly since it only supports binary data. Therefore, we first need to convert the continuous observations into binary representations. To achieve this, we employ a binarizer that preprocesses the game-state observations into binary state representations. This binarizer is trained on a diverse set of actions, randomly generated to ensure a comprehensive range of different observations, which allows the Tsetlin machine to learn effectively from the CartPole environment.

In cartpole the agent gets 1 point as reward for each step it is up and hasn't been truncated. The max episode length is 500, meaning that the max reward for an episode is 500[3][5]. The **Pong** environment provides two distinct types of data representation: visual and RAM. The visual representation is an RGB image that emulates the view of a player interacting with the Atari 2600 version of Pong. The RAM representation, on the other hand, consists of memory values ranging from 0 to 255. For our experiments, we chose the RAM representation due to its simplicity and more straightforward conversion to binary format. To preprocess the RAM data for the Tsetlin machine, we transform each memory value into an 8-bit binary representation, which effectively discretizes the continuous state space and makes it compatible with the Tsetlin machine's requirements.
In Pong the agent gets 1 point for each time it scores, and gets -1 point for each time the opponent scores. One episode continues until either the agent has scored 21 points or the opponent has. Meaning that the max reward for one episode is 21 and the minimum reward is -21[13].

3.2 Hyperparameter Selection

Selecting appropriate hyperparameters is an essential aspect of constructing effective TM models tailored to specific problems. In the current state-of-the-art, particularly within the reinforcement learning context, it remains unclear which hyperparameters are most sensitive and the extent to which they impact the final model's performance. To investigate these relationships, we conduct experiments in which up to four hyperparameters are varied simultaneously, exploring their effects within a reinforcement learning framework. The hyperparameter selection and with corresponding key search space and findings is as follows:

- **Clauses:** Our experiments span a range of clause numbers, from 100 to 5000.
- **Specificity (s):** Following initial tests, we determine that a specificity value of 2.7 consistently produces satisfactory results across various settings.
- **Threshold (T):** We consider several alternatives, such as $T = 1$ and $T =$ (number of clauses)$/2$, before ultimately selecting $T =$ (number of clauses) $* 0.3$, which were found to yield the best results.
- **Number of bits in the binarizer:** We examine binarizer configurations with values ranging from 2 to 20 bits.

For advanced solutions employing the Tsetlin machine from scratch, the increased complexity of the algorithm necessitates the inclusion of additional hyperparameters. These parameters are considered in conjunction with the aforementioned parameters:

- **Batch size:** We evaluate batch sizes varying from 10 to 20.
- **Batch change:** We explore batch changes within a range of 0 to 20.

We further detail these findings in Section 4.

3.3 Study 1: Imitation Learning

In this study, we explore the potential of Tsetlin Machines for learning policies by implementing imitation learning techniques. Our research focuses on two distinct strategies that aim to enable the Tsetlin Machine to effectively emulate the behavior of expert agents: (1) utilizing a pre-trained model of a Proximal Policy Optimization (PPO) agent, a widely recognized reinforcement learning algorithm, and (2) employing an optimal mathematical model specifically designed to address the challenges posed by the CartPole Environment.

Strategy 1: PPO Imitation. The first strategy involves using experience samples from a pre-trained PPO agent to investigate whether Tsetlin Machines can successfully learn policies through imitation. This approach requires a PPO agent to either (1) train and store all experiences in a buffer or (2) sample from an already trained policy. Tsetlin Machines then sample state-action pairs from this buffer, perform inference, and compare their output with the label, which represents the action taken by the "expert" (in this case, the PPO agent). By adopting this strategy, we can evaluate how often the Tsetlin Machine makes decisions that align with those of the PPO agent, thereby providing an estimate of its ability to emulate similar behavior. This serves as a vital preliminary step towards developing reinforcement learning driven by Tsetlin Machines.

Strategy 2: Perfect Model Imitation The second strategy involves employing an optimal mathematical model, as presented in Algorithm 2, to address the CartPole Environment

Algorithm 2: Optimal policy for Cart Pole

Require: *obs*

$\theta, w \leftarrow obs[2:4]$
if $|\theta| < 0.03$ **then**
  return 0 **if** $w < 0$ **else** 1
else
  return 0 **if** $\theta < 0$ **else** 1
end if

with the highest level of expertise. The policy works by analysing the angle and the angular velocity of the pole. The angle of the pole is assigned to theta θ and the angular velocity to omega ω, obtained from the environments observation space. The policy functions by first evaluating whether the absolute value of the angle of the pole, θ, is less than 0.03, indicating that the angle of the pole significantly deviates from the upright position. If this criterion is met, the policy will return action 0 if the pole's angular velocity, ω, is less than 0 or action 1 if the angular velocity is greater. If this condition is not met, the policy will instead attempt to stabilize the pole towards the most upright position. This works by checking for the pole's angle, θ, and outputting action 0 if the angle is less than 0, and action 1 if the angle is greater than 0. This carefully crafted policy efficiently manages the cartpole's position and velocity, ensuring optimal behaviour within the environment.

3.4 Study 2: Model-free Learning

The second study is to learn Tsetlin Machines to play games from scratch model-free. We propose three methods that facilitate efficient learning in the Cart Pole environment:

Strategy 3: ϵ-greedy. The first strategy uses ϵ-greedy to balance exploration and exploitation. The ϵ-greedy scheme is widely used in RL algorithm to balance exploration and exploitation. The idea is that by finding a suitable balance to the exploration-exploitation dilemma, one would efficiently learn policies, as observed in other algorithms, including Q-Learning. As an alternative approach, we added tests to fully disable exploitation (e.g., set epsilon to 1.0 without annealing enabled), ultimately forcing the Tsetlin Machine to learn from only random sampling. The algorithm is as follows:

1. Initialize binarizer, MultiClassTsetlinMachine, gym environment, and variables such as steps, exploration rate, and its rate of change.
2. **For** each step in the range of total steps:
   (a) Process the current state of the cartpole and sample the absolute value of the pole's angle.
   (b) Perform epsilon-greedy/random action and execute it on the cartpole.
   (c) Sample the absolute value of the pole's angle after performing the action.
   (d) **If** the difference between the first and second angles is positive, fit the Tsetlin Machine on the action and state. **Otherwise**, fit the Tsetlin Machine on the opposite action and state.

The above depicts TMs set in a traditional RL loop, much like what is used for algorithms like Q-Learning. We use the Multiclass version of TM where we predict actions based on the observation at the current timestep. Contrary to conventional reinforcement learning techniques, we craft a special reward signal. More specifically, we tune the reward signal to give feedback if the pole angle deviates from the upright position, as opposed to giving sparse

rewards, i.e., positive rewards for each time step and negative ones for terminal states. In essence, we train the TM on actions that minimize the deviation of the pole's angle from the optimal 90-degree angle. In certain situations where the pole's angle deviates in an increasing pattern, the TM is trained on the opposite action to encourage an attempt at recovering from a catastrophic state. As mentioned, we test this algorithm using both annealing ϵ-greedy, and a static ϵ for fully random sampling of actions.
However, during empirical evaluations, we found this approach to have substantial instabilities in learning, resulting in (1) unreliable convergence, and (2) inadequate performance.

Strategy 4: Double TM-Learning. In order to address the instability issues encountered in Strategy 3, we propose an advanced approach, dubbed Double TM-Learning, which is inspired by the well-established double Q-Learning techniques. This novel modification, akin to double Q-Learning, utilizes a pair of Tsetlin Machines: one dedicated to predicting the agent's actions and another for training based on actions considered favourable according to the performance of the primary Tsetlin Machine. Following a user-defined Nth step, the second Tsetlin Machine's learned parameters are transferred to and overwritten the parameters of the first Tsetlin Machine, effectively facilitating periodic updates.

Strategy 5: Batching. In our pursuit to enhance the previous designs, we developed Strategy 5, which uses a sequential batch technique to prioritize which samples the TM should learn from:

1. Initialize the binarizer, MultiClassTsetlinMachine, and gym environment.
2. Initialize variables, such as the total number of steps, exploration rate, and the rate of change for exploration, as well as arrays for storing actions and observations.
3. Initialize variables for batch size, batch counter, and batch change.
4. **For** each step within the specified range:
   (a) Determine whether the batch size should be increased. If so, increase the batch size by the batch change.
   (b) Process the current state of the cartpole.
   (c) Apply the ϵ-greedy algorithm or choose a random action.
   (d) Perform the chosen action on the cartpole.
   (e) Increment the batch counter.
   (f) **If** the episode is terminated; either by the pole falling, moving to far to the side or reaching the maximum amount of steps, reset the observation and action arrays as well as the batch counter.
   (g) **If** the batch counter reaches the batch size, fit the Tsetlin Machine on the observation and action arrays, and reset them along with the batch counter.

This method determines the selection of batches based on the model's ability to maintain the pole in an upright position for a predefined X number of steps. The Tsetlin Machine is subsequently trained on the recorded observations and actions for all the steps. A key feature of this strategy is the batch-change mechanism, predetermined points in the training the batch size is increased by the value of the batch change variable. The intuition is that as the algorithm becomes better, we must increase the difficulty, much like in curriculum learning. In this context, we set a constraint that the algorithm must perform a sequence of good actions, where the sequence length increases as the algorithm learns. This ensures

that we prioritize samples that lead to successful policies.

During the learning process, we either apply the ϵ-greedy algorithm to sample an action or select random actions. The observations and actions that meet the acceptable criteria for keeping the pole upright are stored in an array. If the pole falls, the arrays are cleared; if the specified number of steps is reached, the Tsetlin Machine trains on the dataset before emptying the array. This approach is designed to prevent the machine from learning actions that yield unfavourable outcomes.

The algorithm's architecture exhibits considerable flexibility, allowing for easy adjustments to the number of steps required before the TM initiates training. This adaptability enables fine-tuning the learning process, promoting optimal performance in intricate reinforcement learning tasks. We demonstrate this capability by detailing adjustments to the strategy employed in the pong game.

Fine-tuning Strategy 5 for Pong. In adapting Strategy 5 for pong, we disable the ϵ-greedy annealing mechanism by setting $\epsilon = 1.0$. This modification ensures that the Tsetlin machine is trained exclusively on random actions and states. The TM is responsible for controlling the paddle's vertical movement, either up or down. In this refined version of Strategy 5, we introduce multi-batching: a technique that involves collecting batches of experiences for TM training, subject to differing conditions. For the first batch, the TM trains on N samples in the absence of negative rewards. For the second batch, the TM trains on all samples occurring between the previous negative reward and the attainment of a positive terminal state.

To investigate the efficacy of this fine-tuned strategy, we examined two variations. In the first version, we trained the TM twice on samples that led to positive rewards. However, subsequent experimentation revealed that training once, as is common in model-free RL, also yielded satisfactory results when appropriate hyperparameters were used.

Chapter 4

Results

4.1 Study 1: Imitation Learning

In the first study, we investigated the performance of Tsetlin Machines (TMs) in learning to imitate other models, specifically, the Proximal Policy Optimization (PPO) agent and the Perfect Model (Algorithm 2). We maintained the same hyperparameters across both approaches to ensure a fair comparison between the imitation learning strategies. Through an extensive hyperparameter tuning process, we determined that the optimal configuration consisted of 500 clauses, a specificity value of 2.6, a threshold set to N_CLAUSES $\times$ 0.3, and a 4-bit binarizer. This configuration facilitated the best performance in the imitation learning tasks for TMs.

We initially conducted a baseline evaluation for the expert algorithms, PPO and the Perfect Model, which we will refer to as "Math" henceforth. We assessed the performance of both PPO and Math agents in the CartPole environment over 100 episodes, and the results are summarized in Table 4.1.

The PPO agent demonstrates considerable variability in its performance, as evidenced by the standard deviation of 121.8. With a mean reward of 380.4, this indicates that the reward for the PPO agent ranges between 160 and 500 during the episodes. In contrast, the Math agent exhibits perfect performance, achieving a consistent mean reward of 500 with a standard deviation of 0.0 across all episodes.

Figure 4.1 illustrates the outcomes of employing imitation strategies with Proximal Policy Optimization (PPO) and the Math model. The primary objective is to investigate the extent to which Tsetlin Machines (TMs) can successfully imitate expert algorithms, such as PPO and the Math-based model. The experiment encompasses 100 episodes, with the y-axis in the plot representing the total reward obtained for a specific episode. Each model was tested for 100 runs, and the results gathered are the average over all the runs. The experiment on the PPO model is two fold, one which is the normal PPO model and one which is a demo version of the PPO model. Included in the illustration are three other RL based models, added for comparison. These models are Double Q-Network, Advantage Actor Critic and Proximal Policy Optimization, and all of these models were pre-trained on the environment before running for the experiment.

Table 4.1: PPO (Both Normal and Demo version) & Math, DQN, A2C etc... on cartpole

Agent	**Mean reward**	**Standard deviation**
PPO Demo	380.4	121.8
PPO, Math, DQN, etc...	500	0.0

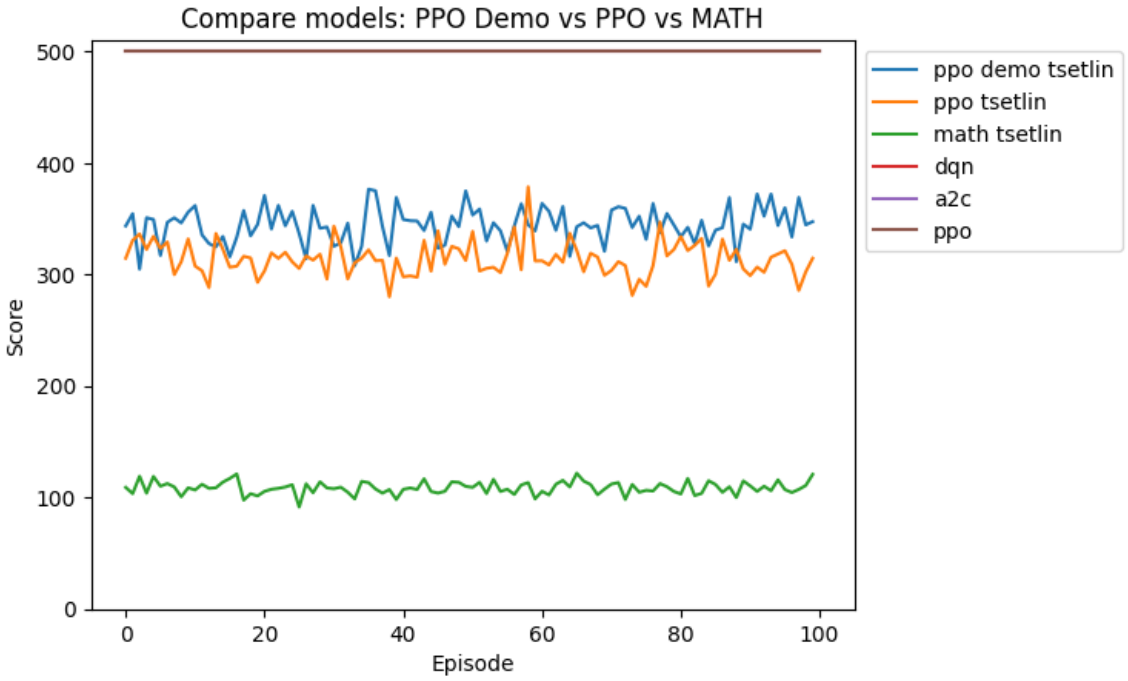

Figure 4.1: The reward-performance of Tsetlin Machine while imitating PPO and Math. We compare the results to Deep Q Networks, Advantage Actor Critic and normal Proximal Policy Optimization. DQN, A2C and PPO all have the same score of 500, hence only one line is shown. Each model was tested for 100 runs and the findings shown are the average from all the runs combined

Table 4.2: Comparison between the TM emulating Demo & Normal PPO vs emulating Math

Agent Type	**Performance Metrics**		
	***Accuracy* & Time**	***Average reward***	***Std***[a]
Demo PPO	89.93% & 32.91s	344.53	161.15
Normal PPO	93.56 & 26.64s	266.18	178.89
Math	95.09% & 30.69s	114.58	47.12

[a]Standard Deviation

Analyzing the results, we see a considerable contrast in the performance of the Tsetlin Machine when imitating the PPO and Math models. The TM imitates the PPO agent and exhibits a distribution of outcomes, with seemingly arbitrary rewards ranging between 280 to 400. In contrast, the TM emulating the Math agent demonstrates a more concentrated distribution of results, albeit with overall lower rewards, spanning from 80 to 120.

Table 4.2 presents a comprehensive summary of these findings. It reveals that the PPO-based Tsetlin Machine yields a higher average reward, but is accompanied by a greater standard deviation. Conversely, the Math-based Tsetlin Machine proves to be more efficient in terms of processing speed and attains superior accuracy.

4.2 Study 2: Model-free Learning

Strategy 3: ϵ-greedy. In the context of Strategy 3, we conduct a hyperparameter optimization experiment, focusing on the following parameters: Clauses (with values of 50, 100, 500, 1000, and 3000), Specificity (set to 3.9), Threshold (calculated as Clauses $\times$ 0.3),

Table 4.3: Test runs of step-by-step strategy

Run	**ϵ-greedy**	**Clauses**	**Mean reward**	**Std**
1	True	50	41.8	26.9
2	True	100	31.8	20.8
3	True	500	36.8	19.0
4	True	1000	30.8	16.4
5	True	3000	23.1	14.7
6	False	50	27.5	12.6
7	False	100	30.0	15.7
8	False	500	24.0	12.5
9	False	1000	27.9	17.5
10	False	3000	34.3	36.6

Table 4.4: Double Tsetlin test run

Clauses	**s**	**Threshold**	**Average reward**	**Std**
1000	3.9	clauses * 0.4	31.8	26.0

and the number of bits in the binarizer (fixed at 6). The experiment executes individual configurations over 10 000 steps, repeating each configuration 10 times to obtain an average performance. The results of this experiment are presented in Table 4.3. For the current investigation, only the number of clauses is assessed, as the other parameters have been previously optimized. This decision is made to conserve computational resources and expedite the final experimental run.

As indicated in Table 4.3, Strategy 3 exhibits suboptimal performance in the cartpole environment. The average rewards consistently fall below 100, with no observable improvement over time. Additionally, we perform the experiment with both ϵ-greedy set to True and False to evaluate its potential impact on performance. However, this parameter proves to be inconsequential for the outcomes observed.

Strategy 4: Double TM-Learning. Pertaining to the fourth strategy, which entails using a double Tsetlin machine architecture, we did not find any benefits for this approach, offering no visible advantages in our experiments. Strategy 4 did not improve compared to Strategy 3; in some cases, it demonstrates inferior results. Table 4.4 summarises the subpar performance and shows the hyperparameters evaluated for this strategy. We also perform additional evaluations in seek of better hyperparameters, but no substantial improvements are observed using the Double TM-Learning strategy.

Strategy 5: Batching. In the pursuit to find a Tsetlin Machine capable of solving complex RL problems, we discover that Strategy 5 emerges as the most promising approach. This technique consistently delivers average rewards between 300 and 400, a remarkable improvement compared to previous strategies. We carry out a hyperparameter search to further assess the effectiveness of this approach. The experiment focuses on the following parameters: Clauses, Specificity (s), Threshold (T), Number of bits in binarizer, Batch size, and Batch change. We showcase the top five and bottom five configurations in Table 4.5 and Table 4.6, respectively. The full hyperparameter search, which includes 60 distinct configurations, can be found in the appendix A at the end. All 60 runs are conducted with $s = 3.9$, $T = clauses * 0.4$, and the number of bits in $binarizer = 6$.

The insights we present in both tables highlight that optimal performance is consistently achieved with clause counts ranging between 1,000 and 3,000 when not employing the ϵ-

Table 4.5: The five best batch runs with corresponding parameters

Run	**ϵ-greedy**	**Clauses**	**B-size**	**B-change**	**M-reward**	**Std**
20	False	1000	10	10	474.9	38.5
27	False	3000	10	20	474.6	26.3
26	False	3000	10	10	474.3	30.0
28	False	3000	20	0	457.3	47.9
21	False	1000	10	20	454.5	56.0

Table 4.6: The five worst batch runs with corresponding parameters

Run	**ϵ-greedy**	**Clauses**	**B-size**	**B-change**	**M-reward**	**Std**
1	False	50	10	0	77.8	23.5
7	False	100	10	0	153.5	133.4
13	False	500	10	0	176.0	106.7
25	False	3000	10	0	180.9	117.4
31	True	50	10	0	195.0	50.7

greedy approach. Conversely, less successful runs tend to involve a batch size of 10 and an absence of batch size modifications. Taking into account that some of the best-performing runs achieve rewards close to 470, we can confidently conclude that the Tsetlin Machine is effective in solving the Cart Pole problem.

It is important to note that there is still room for improvement, as the algorithm does not consistently achieve a mean reward of 500. Despite exhibiting variability in performance, the algorithm represents a significant advancement in the domain of model-free RL using Tsetlin Machines. This breakthrough paves the way for further exploration and enhancement of Tsetlin Machine capabilities within the realm of RL.

Strategy 5 learning graphs In order to better understand the learning process of strategy 5 and identify areas for improvement we ran some tests where the TM was trained over X amount of episodes and after each training episode we ran 10 episodes for evaluation, and added the mean reward to the graphs in figure 4.2. For this experiment we used the following hyperparameters: Clauses: 3000, Threshold: Clauses * 0.3, Batch Size: 20, Batch Change: 0, and specificity from 3.7 to 20. The tests were ran without the epsilon-greedy algorithm. The tests shows instability in the learning, and that a specificity of 3.7 is the best of the lot. Due to the instability of the learning we ran each test 10 times and put the average reward in the graphs seen in figure 4.3 to get a better understanding of the expected learning from each set of hyperparameters. The experiment shows that a specificity of 3.9 gave the best results.

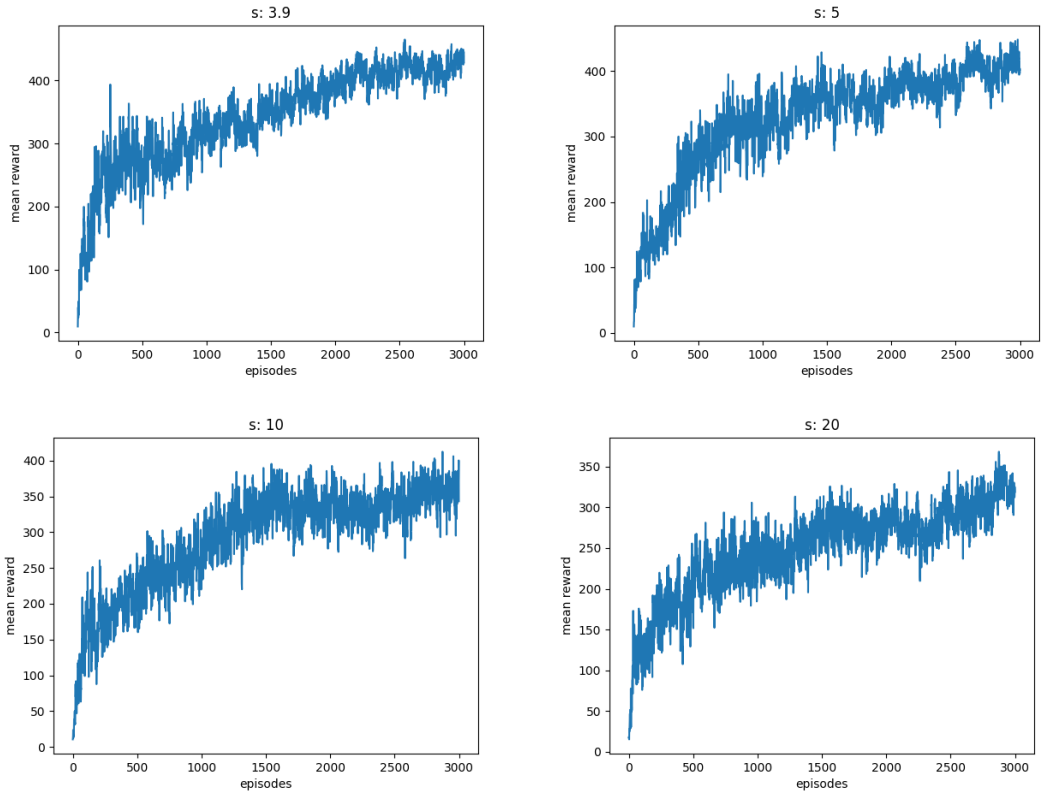

Figure 4.3: Comparison of the average learning of different s values. Reward calculated by running the experiments used for figure 4.2 10 times and adding the average for each episode to the graph.

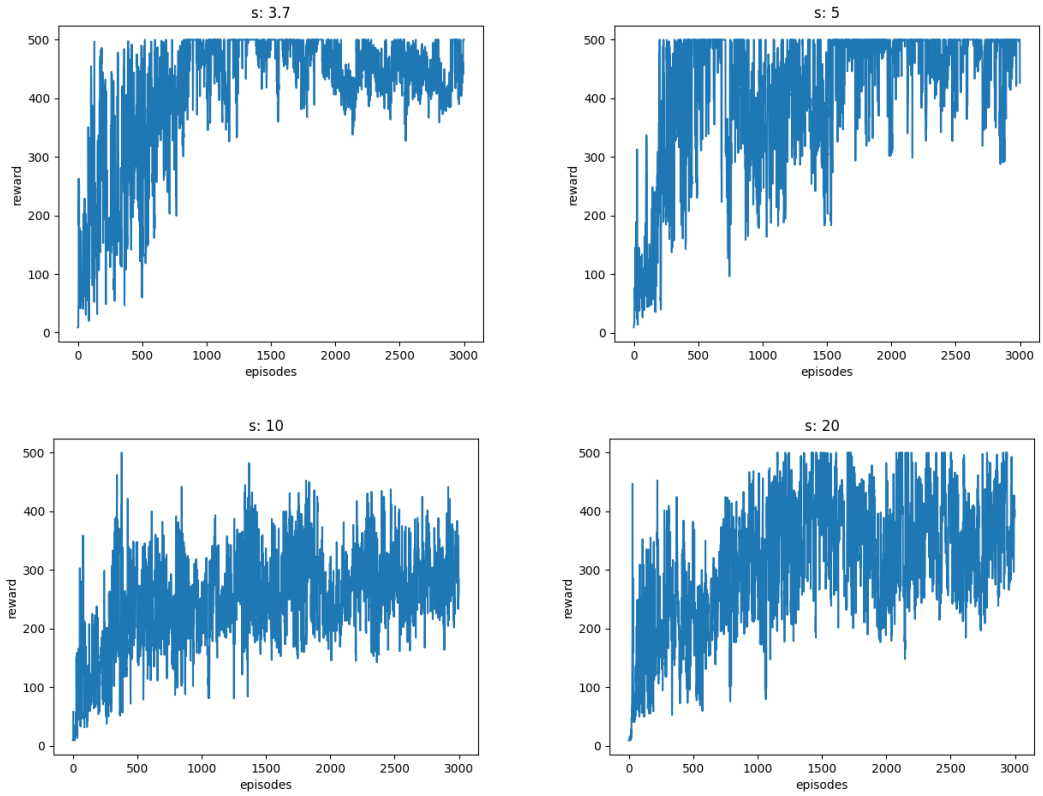

Figure 4.2: Comparison of the learning of different s values. Reward calculated by training for one episode and then validating for 10 and adding the mean to the graph.

Table 4.7: Parameters for the first and second versions of the Pong strategy.

Parameter	**First Version**	**Second Version**
Clauses	20,000	3,000
S	3.9	3.9
Threshold	Clauses * 0.3	Clauses * 0.3
Episodes	2,000	500
N	30	30

Table 4.8: Rewards per episode for the first and second versions of the Pong strategy and random agent.

Version 1	**Version 2**	**Random Rewards**
-19.0	-20.0	-21.0
-19.0	**-19.0**	-21.0
-20.0	**-17.0**	-21.0
-20.0	-21.0	-20.0
-19.0	-20.0	-20.0
-21.0	**-18.0**	-20.0
-19.0	-20.0	-21.0
-19.0	-20.0	-20.0
-19.0	-20.0	-20.0
-20.0	-21.0	-20.0

4.3 Pong

We present the results of TM agents trained on two different versions of the Pong strategy. The agents were evaluated over 10 episodes, and their parameters and performance are reported in two separate tables. Table 4.7 presents the hyperparameters used in the experiment, while Table 4.8 displays the results of our preliminary study on learning Pong from scratch. As can be seen from the table, TMs outperform random policies, indicating that they indeed learn a policy containing information about how to play the game of Pong. However, we are still far from mastering this game, and we further discuss these results in Section 5.

In order to get a better overview over the learning process of the TM in the pong environment we ran version 2 of the pong algorithm over 3000 episodes, and validated for 10 episodes after each training episode. The highest score and the mean score of the 10 validation episodes were added to figure 4.4. The frequency of high scores increases around 1500 episodes, showing that the model improved over time. The model produced varying results, and the best ones are presented in the aforementioned tables.

Additionally we ran the same experiment with the same configuration as the previous run, except this time we increased the clauses to 10 000. This run showcased better performance, being able to achieve victories after much fewer episodes as shown in figure 4.5. As seen in the figure, this new experiment showcased more stability with a higher frequency of better results.

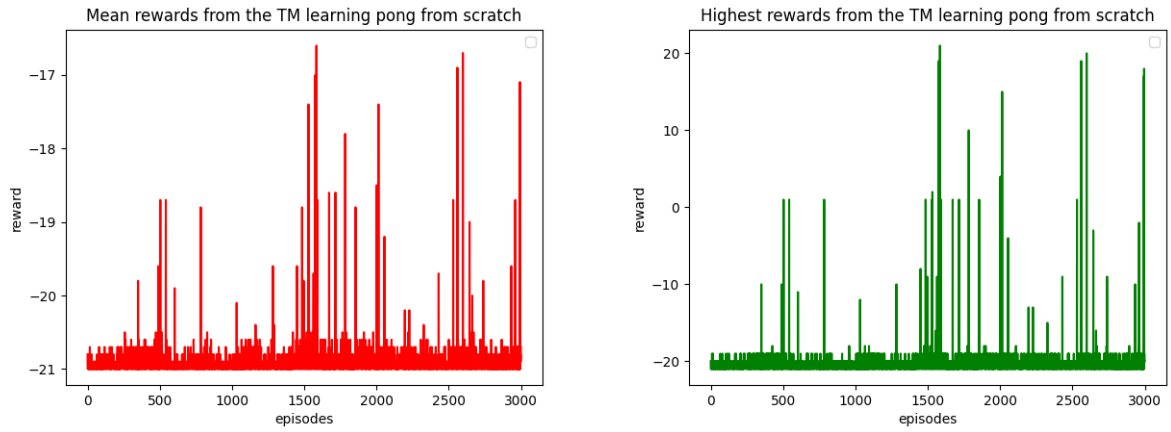

Figure 4.4: Graphs showing the rewards from Pong being learnt from scratch, with 3000 clauses. After one episode of training, 10 episodes of validation is run, saving the highest reward and mean reward in separate graphs.

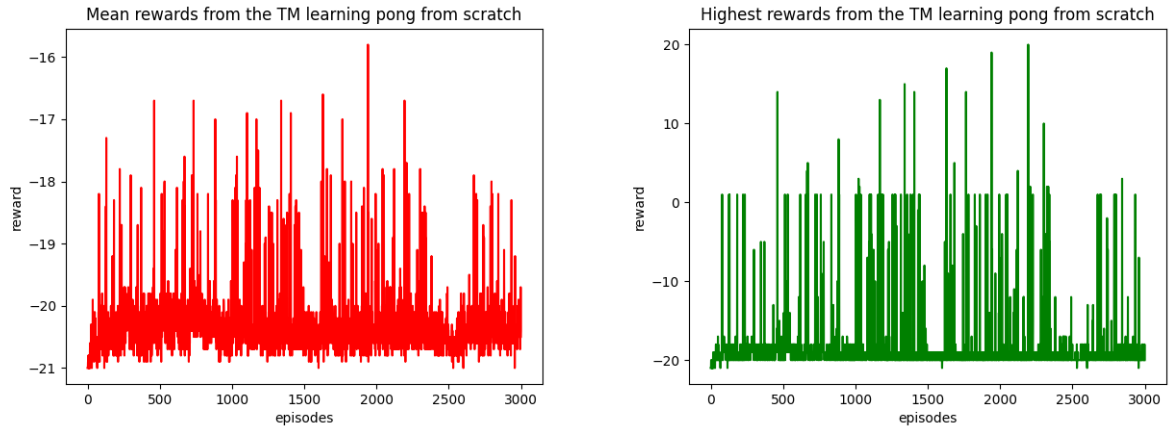

Figure 4.5: Graphs showing the rewards from Pong being learnt from scratch, with 10 000 clauses. After one episode of training, 10 episodes of validation is run, saving the highest reward and mean reward in separate graphs.

Chapter 5

Discussion

5.1 Cartpole

The imitation study, consisting of Strategy 1 with PPO and Strategy 2 with Math model, shows the Tsetlin Machine's capacity to learn from a pre-trained agent in the Cartpole environment. These results demonstrate that the Tsetlin Machine is capable of learning complex behaviours from pre-trained models, hence it favours the idea that TMs can also learn policies in more complex environments, such as Chess.

We hypothesize that the Tsetlin Machine trained on the demo-PPO outperforms the TM trained on the Math model and the perfect PPO model because their overly perfect nature leads to a lack of data on edge cases and experience from catastrophic states. The key problem is that with perfect expert knowledge, the TM makes trials, but no errors, which is crucial to learn stable policies. As a result, the Tsetlin Machine is uncertain about the appropriate actions in situations where it has deviated from perfect behaviour. When trained on the PPO agent, the Tsetlin Machine achieves an average score in the range that considers the Cartpole problem solved.

Moreover, both Strategy 3 (angle method) and Strategy 5 (sequential batch method) employ the Epsilon-greedy algorithm in certain variations, as well as pure exploration in others. The experiments utilizing the Epsilon-greedy algorithm exhibit inferior outcomes when compared to their pure exploration counterparts. One potential explanation for this phenomenon is that the TM lacks sufficient samples that enable feedback signals to penalize incorrect actions, thereby only allowing correct actions to be reinforced. This observation aligns with the findings from Strategy 2 (Perfect Math Model). Consequently, the pure exploration method generates a more extensive dataset for the TM's training, resulting in a superior agent.

Based on the results from strategy 3 and strategy 5, we theorize that strategy 5 achieved higher results due to the fact that strategy 3 was purely trained on actions which lead to smaller angles. Meaning that it would completely disregard how the current angular velocity would affect the angle. The effect of this would lead the TM to sacrifice the current stability for a smaller angle. The TM also did not hold any regards to the current position of the cart, considering that moving too much to either side would terminated the current episode. We theorize that strategy 5 handled this problem better, as it was trained on data in which the cartpole "survived" over time, meaning that it would learn to handle cases which was not directly connected to the pole's angle.

While observing the graphs in figure 4.2 which shows training over time using strategy 5, it is clear that the training is unstable with sudden growths and declines in accuracy. It also shows that the tests with S values of 3.7 and 5 reaches 500 multiple times. The model with s: 20 also reaches 500, but not as frequent as the previously mentioned ones. Considering

the fact that our method is unstable, we chose to run the experiment 10 times. This was done to get a better idea of what the expected learning rate would be. The results shows that a specificity of 3.9 has a better average learning rate, and reaches higher scores towards the end of the run. We theorize that introducing experience replay to the training could improve the learning stability, as it improves the stability in deep Q learning. Another way the stability could be improved would be to take backups of the state of the TM during training, and revert back to the better performing version.

5.2 Pong

The comparison between the two methods used for pong suggests that the second version performs better. Likely because training twice on certain observations lead to overfitting. Further experimenting on the second method showed great potential, as seen in figure 4.4. The figure shows that it is able to have a perfect game in certain episodes, scoring 21 points and not letting the opponent score a single point. And overall the average reward over multiple episodes shows that in several cases it is able to score points before losing. As seen in figure 4.5 the TM with 10 000 clauses is able to achieve victory quicker and more frequent than the TM with only 3000 clauses, suggesting that 3000 is likely not enough to solve Pong. While the TM is unable to consistently beat it's opponent these results shows that it is able to learn Atari environments by reading from the ram, as well as great potential for the TM to learn more complex environments. The results from our experiments shows that the TM trained on pong suffers from the same learning instability as the TM trained on the cartpole environment.

One weakness of our pong methods is the fact that it was difficult to figure out how many steps it should go without having points scored against it before it got trained, one potential solution to this problem is listed in 6.1.

Chapter 6

Conclusions

In this paper, we investigate the Tsetlin Machine's performance in various video game environments, specifically assessing its capacity to imitate existing agents or learn from scratch. Our experiments show that the Tsetlin machine is able to imitate a perfect mathematical solution to the cartpole environment with a high accuracy. It is also able to learn from a perfect PPO agent trained on PONG and an imperfect PPO agent. When the TM trained on the existing agents played the cartpole environment, the TM trained on the imperfect PPO achieved the best results. Further experiments exploring the TM's capability to learn game environments from scratch reveal that it can not only solve but also master the Cartpole environment, and while it isn't able to solve the Pong environment it shows that the TM is able to learn from it. The findings of our paper leads us to conclude that the Tsetlin machine is able to learn from existing agents, and learn to play environments from scratch, meaning that the TM shows great potential within reinforcement learning.

6.1 Future Work

Future research in this domain can be bifurcated into two primary directions. First, investigate the performance of the multiclass Tsetlin Machine in other commonly used benchmarks for reinforcement learning by treating them as classification problems. Second, delving deeper into the capabilities of the Regression Tsetlin Machine (RTM)[2] and exploring solutions and methods more closely related to actual reinforcement learning, such as having the RTM predict rewards for specific actions.

Going further, considering the current state-of-the-art in this field of research, there are many areas to explore. Some interesting avenues to explore could include the following:

- **Double TM policy** could potentially stabilize training, as seen in Double Q-Learning. In success of discovering such mechanism, TM could tackle far greater problems than in current state-of-the art.
- Using **Type-II feedback** to penalize negative actions, to reinforce correct behavior. Utilizing this might assist in addressing the challenges we faced in some of our strategies that were unable to make use of incorrect actions.
- Use **Regression Tsetlin Machine** to make a model based solution. This involves using the TM to predict the next state on the environments. Many RL based models have this functionality, and adding this to the TM could benefit in its development.
- Attempt to **fix the instability** observed during the training of the CartPole agent, as fixing the instability would also improve the results of other environments within RL with the TM. This can be done by looking at experience replay, commonly used in Q-Learning, or the backup method mentioned in chapter 5.

- Future work on the **Pong** algorithm could be to replace the requirement of max steps being reached, into an algorithm which instead checks for contact between the ball and the TM controlled paddle. This can be achieved by allowing the algorithm to read from RAM and detect for contact between the ball and the TM controlled paddle, and then train the TM on actions that lead to this contact.

Appendix A

Strategy 5: Sequential Batch Experiment

A.1 Epsilon Greedy off

Run	Epsilon greedy	Clauses	Batch Size	Batch Change	Mean Reward	Std
1	False	50	10	0	77.8	23.5
2	False	50	10	10	338.2	117.0
3	False	50	10	20	278.7	109.4
4	False	50	20	0	335.4	107.6
5	False	50	20	10	387.5	90.0
6	False	50	20	20	332.3	90.2
7	False	100	10	0	153.5	131.4
8	False	100	10	10	359.5	111.7
9	False	100	10	20	395.0	91.2
10	False	100	20	0	340.0	85.0
11	False	100	20	10	364.5	61.4
12	False	100	20	20	353.9	110.9
13	False	500	10	0	176.0	106.7
14	False	500	10	10	439.2	69.2
15	False	500	10	20	396.6	117.1
16	False	500	20	0	412.7	74.9
17	False	500	20	10	435.5	53.3
18	False	500	20	20	381.1	68.2
19	False	1000	10	0	204.5	153.4
20	False	1000	10	10	474.9	38.5
21	False	1000	10	20	454.5	56.0
22	False	1000	20	0	393.9	77.1
23	False	1000	20	10	392.0	76.3
24	False	1000	20	20	392.7	79.1
25	False	3000	10	0	180.9	117.4
26	False	3000	10	10	474.3	30.0
27	False	3000	10	20	474.6	26.3
28	False	3000	20	0	457.3	47.9
29	False	3000	20	10	394.5	33.3
30	False	3000	20	20	372.8	81.1

A.2 Epsilon Greedy on

Run	Epsilon greedy	Clauses	Batch Size	Batch Change	Mean Reward	Std
31	True	50	10	0	195.0	50.7
32	True	50	10	10	234.8	51.3
33	True	50	10	20	310.8	125.3
34	True	50	20	0	198.3	12.0
35	True	50	20	10	189.0	25.4
36	True	50	20	20	265.4	110.3
37	True	100	10	0	223.2	97.8
38	True	100	10	10	287.3	152.6
39	True	100	10	20	229.7	90.6
40	True	100	20	0	217.1	43.6
41	True	100	20	10	197.2	7.9
42	True	100	20	20	202.9	14.0
43	True	500	10	0	208.5	146.4
44	True	500	10	10	267.2	64.7
45	True	500	10	20	239.7	84.4
46	True	500	20	0	198.3	5.9
47	True	500	20	10	231.7	94.9
48	True	500	20	20	227.1	95.9
49	True	1000	10	0	218.0	58.6
50	True	1000	10	10	239.7	115.6
51	True	1000	10	20	262.0	88.4
52	True	1000	20	0	227.2	96.0
53	True	1000	20	10	199.0	5.9
54	True	1000	20	20	222.8	78.7
55	True	3000	10	0	202.8	81.0
56	True	3000	10	10	210.5	122.3
57	True	3000	10	20	244.7	100.3
58	True	3000	20	0	199.8	4.3
59	True	3000	20	10	196.4	5.1
60	True	3000	20	20	197.6	7.8

Bibliography

[1] *A Deep Q-Learning based approach applied to the Snake game - Scientific Figure on ResearchGate*. Available from: https://www.researchgate.net/figure/Q-Learning-vs-Deep-Q-Learning_fig1_351884746 [accessed 11 May, 2023]. Accessed on 11 May, 2023.

[2] K.D. Abeyrathna et al. "The regression Tsetlin Machine: A Tsetlin machine for continuous output problems." In: *Progress in Artificial Intelligence*. 2019, pp. 268–280. DOI: 10.1007/978-3-030-30244-3_23.

[3] A. G. Barto, R. S. Sutton, and C. W. Anderson. "Neuronlike adaptive elements that can solve difficult learning control problems." In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13.5 (1983), pp. 834–846. DOI: 10.1109/TSMC.1983.6313077.

[4] Greg Brockman et al. *Openai Gym*. 2016. URL: https://arxiv.org/abs/1606.01540 (visited on 04/22/2023).

[5] *Cart Pole - Gym Documentation*. URL: https://www.gymlibrary.dev/environments/classic_control/cart_pole/.

[6] Didrik K. Drøsdal, Andreas Grimsmo, Per A. Andersen, Ole-Christoffer Granmo, and Morten. Goodwin. "*Exploring the Potential of Model-Free Reinforcement Learning in Tsetlin Machines*." Paper in submission. 2023.

[7] Ole-Christoffer Granmo. *The Tsetlin Machine – A Game Theoretic Bandit Driven Approach to Optimal Pattern Recognition with Propositional Logic*. 2018. URL: https://arxiv.org/abs/1804.01508.

[8] A. Hevner et al. "Design Science in Information Systems Research." In: *MIS Quarterly* 28.1 (2004), pp. 75–105.

[9] Yiheng Liu et al. *Summary of ChatGPT/GPT-4 Research and Perspective Towards the Future of Large Language Models*. 2023. arXiv: 2304.01852 [cs.CL].

[10] S. M. et al. "REDRESS: An Embedded Machine Learning Methodology Using Tsetlin Machines." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2023).

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, et al. "Human-level control through deep reinforcement learning." In: *Nature* 518 (2015), pp. 529–533. DOI: 10.1038/nature14236.

[12] Volodymyr Mnih et al. "Asynchronous Methods for Deep Reinforcement Learning." In: *CoRR* abs/1602.01783 (2016). arXiv: 1602.01783. URL: http://arxiv.org/abs/1602.01783.

[13] *Pong - Gymnasium Documentation*. URL: https://gymnasium.farama.org/environments/atari/pong/.

[14] S. Rahimi Gorji and O.C. Granmo. "Off-policy and on-policy reinforcement learning with the Tsetlin machine." In: *Applied Intelligence* (2023). DOI: 10.1007/s10489-022-04297-3.

[15] ResearchGate. *The Actor-Critic Architecture*. Available from: ResearchGate [accessed 11 May 2023]. ResearchGate. 2023. URL: https://www.researchgate.net/figure/The-Actor-Critic-Architecture_fig2_220696313.

[16] John Schulman et al. "Proximal Policy Optimization Algorithms." In: *CoRR* abs/1707.06347 (2017). arXiv: 1707.06347. URL: http://arxiv.org/abs/1707.06347.

[17] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[18] V. R. Varma. *Interpretable Reinforcement Learning with the Regression Tsetlin Machine*. 2022. URL: https://fse.studenttheses.ub.rug.nl/264831/Master_Thesis_VRVarma%5C%20(1).pdf (visited on 04/27/2023).