UiA University of Agder

Master

# Interpretable Tsetlin Machine For Explaining Board Games With Complex Game States

VARPE JOAR

SUPERVISORS
Glimsdal Sondre
Granmo Ole Christoffer

**University of Agder, 2022**
Faculty of Engineering and Science
Department of Engineering and Sciences

## 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:**<br>• Ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands.<br>• Ikke refererer til andres arbeid uten at det er oppgitt.<br>• Ikke refererer til eget tidligere arbeid uten at det er oppgitt.<br>• Har alle referansene oppgitt i litteraturlisten.<br>• 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

I would firstly like to thank my supervisors for all help and guidance provided. I would especially thank Sondre Glimsdal for his invaluable guidance and patience. Furthermore would I like to thank all the people working at Forsta Grimstad for letting me use their office the entire time I was testing and writing this thesis. I would also like to thank the two servers Neuromancer and Flatline, for working tirelessly throughout this whole period. This would probably not have been possible if not for their help.

As a final note I would like to give Sondre Magnus Siljan a special thanks for entering the office, and as a side-note telling me that the due date was 4 days before I thought it was, if not for him, the report would not have been delivered at the correct time. As I thought the due date was 4 days later, leading to an exciting 22hour writing extravaganza.

# Abstract

Stefan Dorra's For Sale is both a turn-based and simultaneous action zero-sum game where the objective is to become as rich as possible. The first phase of the game is a sequence of turn-based English auctions that bids for properties selected at random. The game itself is complex having a mix of multiple players, hidden information, and stochastic elements. Although auctions themselves have been thoroughly studied in literature this particular setup remains an open problem.

In this thesis, we investigate the usage of the interpretable Coalesced Tsetlin Machine (CoTM) for solving these types of auction games providing both excellent play and an understanding of how to play.

To this end, we first develop a self-playing reinforcement learning algorithm that achieves near optimal play. Secondly, based on this algorithm we construct a dataset with examples of optimal play. Thirdly, using CoTM we investigate various ways of understanding why particular moves are made.

The CoTM is also shown to outperform popular methods such as decision trees, neural networks, and k-nearest neighbours. On average the CoTM accuracy is 84.55% significantly outperforming the other competitors.

We believe the resulting interpretability establishes that CoTM can be used for the interpretation of games that have a more complex game state than Hex and Go.

# Contents

**Acknowledgements** **ii**

**Abstract** **iii**

**List of Figures** **ix**

**List of Tables** **xi**

**1 Introduction** **1**

1.1 Motivation . . . . . 3

1.2 Thesis Description . . . . . 3

1.3 Goals and Research Questions . . . . . 4

1.4 Contribution . . . . . 4

1.5 Outline . . . . . 4

**2 Theory** **5**

2.1 Stefan Dorra's For Sale . . . . . 5

2.1.1 1. Auction / Bidding and property acquisition phase . . . . . 5

2.1.2 2. Auction / Selling phase . . . . . 6

2.1.3 Deciding the winner . . . . . 6

2.2 Reinforcement Learning . . . . . 6

2.2.1 On-policy . . . . . 7

2.2.2 Off-policy . . . . . 7

2.3 Multi Agent Reinforcement learning . . . . . 7

2.4 Sparse Rewards . . . . . 8

2.5 supervised learning . . . . . 8

2.6 OpenAi Gym . . . 8
2.7 PettingZoo . . . 8
2.8 Ray RLlib . . . 9
2.9 Intrinsic Curiosity Module . . . 9
2.10 Proximal Policy Optimization . . . 9
2.11 Parameter Sharing . . . 10
2.12 Apriori . . . 10
2.13 Nash Equilibrium . . . 10
2.14 Equilibrium in multiplayer games . . . 11
2.15 Learning Automata . . . 11
2.16 Tsetlin Automaton . . . 11
2.17 The Tsetlin Machine . . . 12
2.18 Coalesced Tsetlin Machine . . . 13

**3 Reinforcement learning** **15**

3.1 Creating the environment . . . 15
3.1.1 Framework . . . 15
3.1.2 Action masking . . . 15
3.1.3 Observational space . . . 16
3.1.4 Rewards . . . 18
3.2 Generating Winning games . . . 18
3.2.1 Black Box Solving the First Phase . . . 18
3.2.2 Experiments . . . 19

**4 Training TM** **21**

4.1 Pipeline . . . 21
4.2 Comparison Between other Algorithms for Interpretability and Accuracy . . . 22
4.3 Encoding of the Parameters . . . 22
4.4 Tsetlin Machine Rule Extraction . . . 23
4.4.1 Features and Feature Extraction . . . 24
4.4.2 Graphical interval mining . . . 24

4.4.3 Game scenario . . . . . 24

**5 Results** **25**

5.1 Training BlackBox AI player . . . . . 25

5.2 Creating the training data . . . . . 26

5.3 Data Preprocessing . . . . . 27

5.4 Black Box Model Y Distributions . . . . . 28

5.4.1 3-rounds ICM trained . . . . . 28

5.4.2 3-rounds PPO trained . . . . . 28

5.4.3 4-rounds ICM trained . . . . . 29

5.4.4 4-rounds PPO trained . . . . . 29

5.5 Comparing CoTM accuracy to other methods applied to the data . . . . . 30

5.6 Interpretability of other methods . . . . . 31

5.7 Explaining CoTM output . . . . . 31

5.7.1 Naive example . . . . . 31

5.7.2 Top Weighted . . . . . 32

5.7.3 Frequent item mining . . . . . 33

5.8 Ensamble CoTM . . . . . 34

5.8.1 Frequent item mining . . . . . 34

5.9 Game scenario . . . . . 34

5.9.1 Graphical Interval Mining . . . . . 35

**6 Discussions and conclusion** **38**

6.1 Environment Related Improvements . . . . . 38

6.1.1 Subsequent pass feature vs Actual pass feature . . . . . 38

6.1.2 Scaling starting coins . . . . . 38

6.1.3 Hidden vs not Hidden Coins and/or Cards . . . . . 38

6.2 PPO outperforming ICM to a large extent . . . . . 39

6.3 Extensive Hyperparameter Search . . . . . 39

6.4 Parameter encoding/ features . . . . . 39

6.4.1 Changing Standard Deviation bit encoding to large, medium and low 39

6.5 Readability and state accuracy of the intervals of the plots . . . . . 40
6.6 Revisiting research questions and goals . . . . . 40
6.6.1 Can we through the use of the CoTM explain how to play For Sale? . 40
6.6.2 Can we explain the behavior of the self learned algorithm? . . . . . 40
6.6.3 How does the CoTM compare to other interpretable algorithms? . . . 40
6.7 Future work . . . . . 40
6.7.1 Further experiments with different input features . . . . . 40
6.7.2 Bigger TM Ensemble . . . . . 41
6.7.3 Other Potentional Black Box Algorithms . . . . . 41
6.7.4 Using other Multi-Agent Methods . . . . . 41
6.7.5 Training on the Full 8 Rounds . . . . . 41
6.7.6 Train a black box model on the second round . . . . . 41
6.7.7 Finish the whole For Sale environment with trained black box agent . 41
6.8 Conclusion . . . . . 42

**Bibliography** **43**

# List of Figures


1.1 For sale set up for 3 players for the first round of the first phase. . . . . 2

1.2 The general overview of the setup. . . . . 3

2.1 Tsetlin Machine structure[31] . . . . . 13

3.1 A visualized example state. . . . . 16

4.1 The general pipeline utilized in the project . . . . . 22

5.1 Visualization of the data prepossessing process, here with the results from 100000 games with ICM 3-rounds. Top left is the raw data, top right all games where the agent had no choice other than betting 0, bottom left is the data after duplicates have been removed and finally bottom right is the data after oversampling has been applied. . . . . 27

5.2 The y distribution after cleaning 100000 ICM 3-round games. . . . . 28

5.3 The y distribution after cleaning 100000 PPO 3-round games. . . . . 29

5.4 The y distribution after cleaning 100000 ICM 4-round games. . . . . 29

5.5 The y distribution after cleaning 100000 PPO 4-round games. . . . . 30

5.6 Plotted DecicionTree after training. . . . . 31

5.7 The game scenario, AI players turn to bid. . . . . 35

5.8 Visualization of selected ranges after item mining the literals and negated literals from the clauses, for the example game scenario for CoTM ensemble in favor of betting 4. . . . . 36

5.9 Visualization of selected ranges after item mining the literals and negated literals from the clauses, for the example game scenario for CoTM ensemble against betting 6. . . . . 37

# List of Tables


3.1 An example state from the 3 round version, only showing player one for simplicity. . . . 17

5.1 All possible orders of play with the three models, with results. . . . 25

5.2 The total wins of the models, with results from random agent vs PPOv0 and PPOv1 . . . 26

5.3 The total wins per player position after PPOv0 played 1 million games against itself. . . . 26

5.4 The PPOv0 model playing 100.000 games against 2 random agemts. . . . 26

5.5 Results from training on the generated dataset. . . . 30

5.6 Example reduced naive output clause. . . . 32

5.7 Top 5 weighted clauses for class 6 for single CoTM. . . . 33

5.8 Example apriori result from single CoTM with support over 0.01 and length more than 1. . . . 33

5.9 Biggest support entries from ensamble CoTM. . . . 34

5.10 Example apriori result from ensamble CoTM with support over 0.01 and length more than 1. . . . 34

# Chapter 1

# Introduction

In recent years a new top contender in an increasing amount of fields has emerged, outperforming humans in many disciplines. These superhuman AIs play, among other games, Go[1], Poker[2], Chess[3], and Dota 2[4] to an extraordinary degree of skill. Even though humans are outperformed by AIs, humans are still playing games. A study from 2014 showed that video games were part of the everyday life of 97% of all children and adolescents in the United States. This suggests that games will be played for the currently foreseeable future[5]. Games have several benefits, and an enriched sense of mastery may lead to an increased sense of enjoyment during activity[5, 6]. To increase human abilities beyond the current, and/or evolve the competitive scene in variety of games, AI assistance could be necessary. Understanding how the AI chooses its move is essential to transfer knowledge from the AI to human players, and this is where the main issue presents itself. The current state-of-the-art AI models have a difficulty in explaining why the actions that are chosen are optimal. As they are consisting of black box methods. This hinders human-computer collaboration, especially when a human is trying to learn from an AI. For instance take an E-sport player in the game of Dota 2, wanting to perfect his game by learning from OpenAi 5, the best current Dota 2 algorithm. Currently, it would prove to be a very difficult and time-consuming task, as there is little to no explanation by the AI during or after an action has been chosen. The player would have to observe the AI and take notes on what the AI did and try to figure out why, which may or may not be successful. Valuable time that the player could have used for practice, would have to be invested into trying to understand why the AI behaves as it does instead, with no guarantee that the effort would be rewarded. With explainable AI, a more profound understanding of why instead of what could be achieved. Currently, there are some ways of interpreting black box AI[7, 8]. It is proved somewhat possible to extract information from AlphaZero[1] by having access to cutting-edge expertise in the game of chess, but following this approach for every discipline would be inefficient and expensive[9].

One of the more recent works regarding the Tsetlin Machine (TM) shows how to use propositional logic expressions to describe winning and losing positions in the board game Hex, leading to a description of strong moves[10]. Applying this knowledge to other games with the suggestion from Rudin[11], that interpretable models can perform competitively if the representation of the problem is adequate, it may be possible to learn rules from games that could teach potential human players why strong moves are strong, and conversely why weak moves are weak in other games than Hex. Applying a similar approach to the auction game Stefan Dorra's For Sale could prove both challenging for an AI to play, as well as having a more complex game space in terms of modeling. The TM, which is based on propositional logic, a set of AND and OR, is easily interpreted given input in a suitable form.

This thesis will be looking at the aforementioned game, Stefan Dorra's For Sale: The Game of Property and Prosperity[12], which was nominated for the 2017 Hungarian Board Game Award and a finalist in the 2016 Lys Grand Public. The game is a zero-sum auction game, which even children can play. For Sale is an interesting game for reinforcement learning, as it has a mix of multiple players, hidden information, stochastic elements and ease of play. As opposed to hex that has a natural binary representation, the multi faceted board representation of For Sale makes both the representation and the following analysis significantly more difficult. It consists of 3-6 players, who each start with the same amount of money. For simplicity we will in this thesis limit us to the 3 player version.


Figure 1.1: For sale set up for 3 players for the first round of the first phase.

The game has simple rules and gameplay is divided into two phases. First the property acquisition phase, and then the currency acquisition phase. The first phase is played out as an english first bid auction where each player bids on the highest of the three properties, and an example setup can be seen in Figure 1.1[13]. The player with the highest bid wins the highest card, the next highest bid the next highest card and the lowest bid gets the lowest card. This scheme continues for all rounds until all of the property cards have been awarded to the partaking players. See section 2.1 for an in depth explanation.

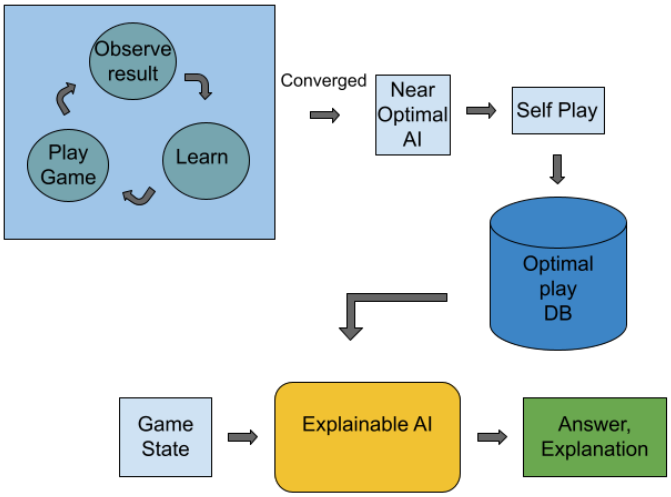



Figure 1.2: The general overview of the setup.

The main idea of the project is to use a traditionally hard to understand black box AI player to play a board game, then use the TM to interpret reasoning behind the moves. Figure 1.2 shows an illustration o the general overview. There currently is no competitive benchmark or current state of the art in regards to strictly playing For Sale, although some high performance algorithms tackling multi-agent auction games exists[14].The black box AI player will be trained to such a degree that when the AI plays the game and wins, it would be considered optimal play. The agent then plays against itself to generate the database. The complete history of states and actions, also known as trajectories, are saved from all the winning games during the agents' games, and saved in a "Optimal play database". This database will then be used to train an explainable AI to understand the game to such an extent that it is considered optimal as well. As a result of the database of moves, the problem is transformed from a reinforcement learning problem, into a supervised learning problem. The Explainable AI will then analyze different gamestates, where it will output what it considers the best move, as well as a reasoning behind why this move is the best. The explainable AI this thesis will utilize is the Coalesced Tsetlin Machine (CoTM).

## 1.1 Motivation

Machines playing imperfect information games acquire superhuman results, but are often regarded as a black box of actions. Through the recent breakthrough of the TM, more interpretable AI solutions are being developed. Improvement and additional knowledge regarding gameplay in games can be achieved through the use of interpretable AI.

## 1.2 Thesis Description

This thesis seeks to increase the knowledge regarding the possible use cases for TM, by applying it to a more complex game environment. We wish to explore how a complete pipeline where the end goal is to understand a game through an interpretable method.

## 1.3 Goals and Research Questions

- RQ 1: Can we through the use of the CoTM explain how to play For Sale?
- RQ 2: Can we explain the behavior of the self learned algorithm?
- RQ 3: How does the CoTM compare to other interpretable algorithms?

## 1.4 Contribution

This thesis explores how a self-playing reinforcement learning method may be used in conjunction with the CoTM to explain the best moves in the board game For sale. The following list summarizes the contributions:

- A trained model that play the first phase of For Sale.
- A fully functional PettingZoo[15] environment for the two phases of the game For Sale
- A database consisting of optimal trajectories a reduced first phase of the game For Sale.
- An explanation of the behavior of the self learned algorithm
- A method of graphically analysing complex gamestates through the use of the CoTM
- A comparison between CoTM and other algorithms in terms of interpretability.
- An end-to-end example of creating an environment, training a reinforcement learning algorithm and then explaining the behavior of the learned model through the use of the CoTM

## 1.5 Outline

* Chapter 2 introduces the game For Sale. Followed by the relevant technical theory of machine learning and its components used in this thesis.

* Chapter 3 describes the methodology of preparing input data for the CoTM, training the CoTM on the processed data and then setting up for further interpretable analysis of the CoTM. using the and is divided into two parts, the first part describes the training of a reinforcement learning model

* Chapter 4 presents the creation of an reinforcement learning environment, and the training of a reinforcement learning agent using an existing framework with a custom backbone network

* Chapter 5 presents the findings and results of the project

* Chapter 6 discusses the findings, looks at further work and concludes the thesis

# Chapter 2

# Theory

This chapter introduce relevant background material for understanding the subsequent chapters.

## 2.1 Stefan Dorra's For Sale

The game For Sale is a short game by Stefan Dorra about buying and selling real estate. It can be summarized to consisting of two phases, first the auction phase, and then the selling phase. The winning player is the richest player at the end of the game. The game is playable for 3-6 people and the setup of the game varies depending on the amount of players. The game consists of 30 property cards numbered 1 - 30, 30 currency cards valued 0 - 15,000, where 1,000 are skipped, and two of each, 60 1,000 coins, and 12 2,000 coins. The initial setup of the 3-player version results in each player getting two 2,000 dollar coins, and fourteen 1,000 dollar coins resulting in 18,000 dollars total, as well as removing 6 random property cards and 6 random currency cards from the game.

### 2.1.1 1. Auction / Bidding and property acquisition phase

The first phase is a Turn-based continuous open bidding ascending price auction / english auction, which means that each player bids and the next player has to raise the bid, and the auction ends when only one player is left. The auction starts with three property cards laying face up on the board ( as there are 3 players in the game), then the players start bidding on the highest card, beginning with the player who lives in the largest house, and continues clockwise around the table. The next player must then decide whether to bid or pass. When a player eventually says they are no longer interested in betting higher than the current bid, the player passes. After passing the player gets the current lowest card laying on the board, and has to concede half of the players' last bid this round, rounded up to the closest integer in the case of a number not being whole. In the case of the player not having given a bid at all the resulting payment will be 0, meaning that it is not necessary to bid anything to gain the least valuable property. This repeats until only the winning player is left in the round, who then pays the full extent of their bid and gets the last card. A new round starts and this repeats until the stack of property cards are empty, then the game moves on into the next phase.

### 2.1.2 2. Auction / Selling phase

The second phase is a Simultaneous one-time blind bidding / First-price sealed bid auction. A first-price sealed bid auction operates in the following way: All players enters their bid, but does not show anyone else the bid. When all players have entered their bid, all the bids are revealed and the winner is the one with the highest bid. The auction in the game starts with three currency cards being revealed on the board, then the players each enter one of their property cards as their bids. The currency cards are then awarded the players in descending order coupled with the descending order of the bids, the player that bid the highest property card will be awarded the currency card of the highest value, then the second highest bid gets the second highest property card etc, until the last player gets the last card. This is repeated until all currency cards have been dealt to all the players and no player has any property cards left.

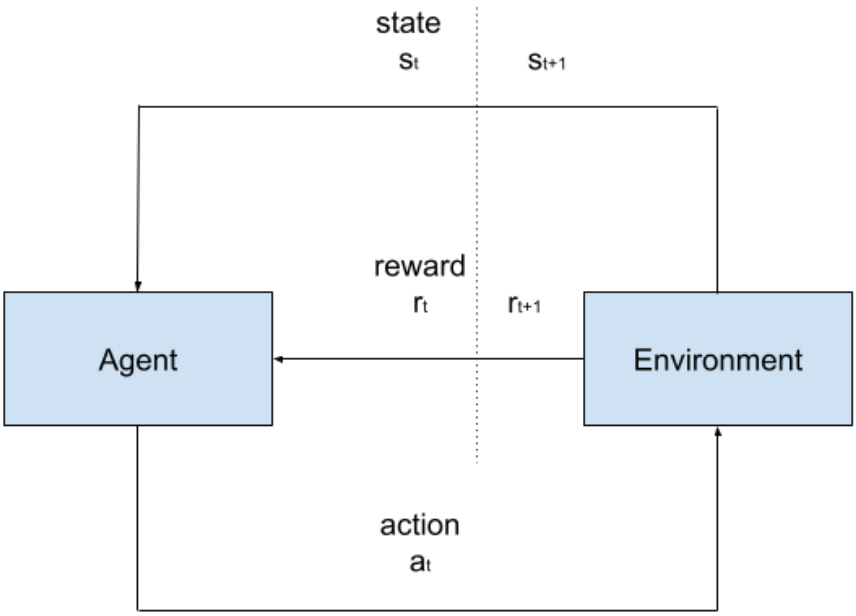
### 2.1.3 Deciding the winner

After the two phases have concluded, it is time to count the value of each player and decide the winner. Each currency card is worth the amount displayed on the card, and each coin the player may have is worth their displayed value. In the case of a draw, the player with the most coins is crowned the winner.

## 2.2 Reinforcement Learning

In the book by Barton and Sutton[16], reinforcement learning is introduced as the discipline of machine learning where the machine (an agent) learns what to do, how to map situations to actions resulting in a maximized numerical reward signal. The agent is not told what actions to do, but instead has to discover which actions yield the most reward, in what state, by doing the action and receiving a reward. In the most interesting and challenging cases, the actions may affect not only the immediate reward, but also the next situations, which in turn may affect all subsequent rewards. The most important distinguishing features of reinforcement learning are trial-and-error search and delayed reward.

Essentially reinforcement learning can be boiled down to the illustration shown in figure ???. The agent receives a state $s_t$ and a reward $r_t$ and proceeds with the action $a_t$. The agent then receives the new state $s_{t+1}$ and reward $r_{t+1}$ and the cycle continues with the actions usually being selected in order to maximize the reward over a set timespan or actions.




### 2.2.1 On-policy

On-policy methods attempt to evaluate or improve the policy that is used to make decisions. The policy in On-policy control methods are generally soft, which means that the probability of choosing an action given a state is larger than 0, for all states and all actions are possible in every state, however, the policy is gradually shifted closer and closer to a deterministic optimal policy[16].

### 2.2.2 Off-policy

Off-policy methods evaluate or improve a policy different from the one used to generate the data. The policy used to generate behavior, is called the behavior policy, may be unrelated to the policy that is evaluated and improved, which is called the target policy. One of the advantages of the separation between the policies is that the target policy may be deterministic, while the behavior policy can continue to sample all possible actions[16].

## 2.3 Multi Agent Reinforcement learning

Multi-agent reinforcement learning (MARL) approaches the sequential decision-making issue of several autonomous agents operating in a shared environment, each of which seeks to maximize its own long-term return by interacting with the environment and other agents[17]. The information structure, as in who knows what in MARL is more involved, as each agent may have limited access to the observations of others, leading to possibly sub-optimal decision rules locally.

## 2.4 Sparse Rewards

In reinforcement learning a sparse reward signal is a series of rewards produced by the environment when an agent interacts with it, where most of the rewards received by the agent are non-positive. It is a difficult task for reinforcement learning algorithms to connect a long series of actions when the rewards are mostly of no guidance. In a very sparse environment, the agent may never find a reward signal, and the correct action sequence may never be learned[18].

## 2.5 supervised learning

Supervised learning occurs when one or several algorithms build a function that translates inputs to desired outputs. The classification issue is one common formulation of the supervised learning task: the learner is supposed to learn a function that translates a vector into one of multiple classes by examining several input-output instances of the function[19].

## 2.6 OpenAi Gym

OpenAi Gym is a set of tools for studying reinforcement learning. It consists of a set of benchmark challenges that expose a common interface, as well as a website where individuals can post their results and compare algorithm performance. Gym focuses on reinforcement learning in an episodic framework, in which the agent's experience is split down into a sequence of episodes. The agent's beginning state is randomly chosen from a distribution in each episode, and the interaction continues until the environment achieves a terminal state.[20]. The design of Gym is based around five main points:

- Environments, not agents.
- Emphasize sample complexity, not just final performance.
- Encourage peer review, not competition.
- Strict versioning for environments.
- Monitoring by default.

## 2.7 PettingZoo

PettingZoo is a library of diverse sets of multi-agent environments with a universal Python API. It was developed with the goal of accelerating research in Multi-Agent Reinforcement Learning, by making work more interchangeable, accessible and reproducible similarly to what OpenAi's Gym have done for single-agent reinforcement learning. PettingZoo's API, inherits many features from Gym, is unique amongst Multi-Agent Reinforcement learning API, due to it being based around the Agent Environment Cycle games model[15].

## 2.8 Ray RLlib

RLlib is a library that provides scalable software primitives for Reinforcement Learning. The primitives enable abroad range of algorithms to be implemented with high performance, scalability, and substantial code reuse. RLlib is a part of the open source Ray project[21][22].

## 2.9 Intrinsic Curiosity Module

The agent is composed of two subsystems: A reward generator that outputs a curiosity-driven intrinsic reward signal, and a policy that outputs a sequence of actions to maximize that reward signal. In addition to intrinsic rewards, the agent moptionally may also receive some extrinsic reward from the environment. Let the intrinsic curiosity reward generated by the agent at time $t$ be $r_t^i$ and the extrinsic reward be $r_t^e$. The policy sub-system is trained to maximize the sum of these two rewards $r_t = r_t^i + r_t^e$, with $r_t^e$ mostly if not always zero. The policy $\pi(s_t; \theta_P)$ is represented by a deep neural network with parameters $\theta_P$. Given the agent in state $s_t$, it executes the action $a_t$ $\pi(s_t; \theta_P)$ sampled from the policy. $\theta_P$ is optimized to maximize the expected sum of rewards,

$$\max_{\theta_P} E_{\pi(s_t;\theta_P)}\big[\sum_t r_t\big]$$

The curious reward model can potentially be used with a range of policy learning methods, with the paper utilizing the asynchronous advantage actor critic policy gradient (A3C). [18]

## 2.10 Proximal Policy Optimization

Introduced by Schulman et al. in Proximimal Policy Optimization Algorithms[23], the proximal policy optimization algorithms are a family of policy gradient methods, which alternate between sampling data through interaction with the environment, and optimizing a "surrogate" objective function using stochastic gradient ascent. The standard policy gradient methods perform one gradient update per data sample, whereas PPO enables multiple epochs of minibatch updates. PPO has some of the benefints of trust region policy optimization (TRPO), but are much simpler to implement, are more general and have better sample complexity (empirically).

The main motivation for developing this algorithm was to have the reliable performance of trust region policy optimization (TRPO)[24], while only using first-order optimization. Let the probability ratio $r_t(\theta)$ denote the probability ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta\,old}(a_t|s_t)}$, so $r_t(\theta_{old}) = 1$. TRPO maximizes a "surrogate" objective:

$$L^{CPI}(\theta) = \hat{E}\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)}\hat{A}_t\right] = \hat{E}\left[r_t(\theta)\hat{A}_t\right]$$

Where CPI refers to a conservative policy iteration. If not constrained, maximization of $L^{CPI}$ would leaf to an excessively large policy update. PPO modifies the objective, to penalize changes to the policy that move $r_t(\theta)$ away from 1:

$$L^{CLIP}(\theta) = \hat{E}_t\left[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

PPO's clipped objective supports multiple SGD passes over the same batch of experiences. RLlib's multi-GPU optimizer pins that data in GPU memory to avoid unnecessary transfers from host memory, substantially improving performance over a naive implementation. PPO scales out using multiple workers for experience collection, and also to multiple GPUs for SGD. The clipped objective:

$$L_t(\theta) = min(r_t(\theta)\hat{A}_t, clip(r_t(\theta)), 1-\epsilon, 1+\epsilon)\hat{A}_t$$

## 2.11 Parameter Sharing

Parameter sharing in reinforcement learning is a type of centralized learning where all policies are represented by a single neural network with the same share parameters for each policy. Previously thought to only being able to handle agents with identical behavior in the environment, due to having only one policy network, it has been proved that parameter sharing allows for convergence to optimal policies[25].

## 2.12 Apriori

The Apriori algorithm is a technique for mining frequent itemsets that was proposed by R. Agrawal and R. Srikant in 1994[26]. The algorithm iteratively creates frequent k-itemsets for $k = 1, 2, 3..n$ by using prior knowledge of k-1-itemsets. First, it finds the frequency of 1-itemsets, then the 1-itemsets are used to find the 2-itemsets, then 2 itemsets are used to find 3-itemsets, and so on until there are no more k-itemsets. The support is found through taking the number of transactions containing the itemset, and divide it by the total amount of transactions.

## 2.13 Nash Equilibrium

The Nash equilibrium is the most commonly used solution concept in game theory and was firstly presented by John Nash[27]. It captures the steady state of the play of a strategic game in which each player holds the correct expectation about the other players' behaviour and acts rationally. It does not attempt to examine the process by which a steady state is reached.

The definition is as follows: A Nash equilibrium of a strategic game $\langle N, (A_i), (\succsim_i)\rangle$ is a profile $a^* \in A_i$ of actions with the property that for every player $i \in N$ we have

$$(a^*_{-i}, a^*) \succsim_i (a^*_{-i}, a^*) \text{ for all } a_i \in A_i$$

Thus for $a^*$ to be a Nash equilibrium it must be that no player $i$ has an action yielding an outcome that he prefers to that generated when he chooses $a^*_i$, given that every other player $j$ chooses his equilibrium action $a^*_j$. Which results in the case where no player can profitably deviate, given the actions of the other players. The definition can be restated as such: For any $a_{-i} \in A_{-i}$ define $B_i(A_{-i})$ to be the set of player $i$'s best actions given $a_{-i}$:

$$B_i(a_{-i}) = \{a_i \in A_i : (a_{-i}, a_i) \succsim_i (a_{-i}, a'_i) \text{ for all } a'_i \in A_i\}. \tag{2.1}$$

The set-valued function $B_i$ is called the **best-response function** of player $i$. A Nash equilibrium is a profile $a^*$ of actions for which

$$a_i^* \in B_i(a_{-i}^*) \text{ for all } i \in N. \tag{2.2}$$

The alternate formulation of the definition points to a method of finding Nash equilibria: first calculate the best response function of each player, then find a profile $a^*$ of actions for which $a_i^* \in B_i(a_{-i}^*$ for all $i \in N$. If the functions $B_i$ are singleton-valued then the second step entails solving $|N|$ equations in the $|N|$ unknowns $(a_i^*)_{i \in N}$.

Nash equilibrium has zero exploitability in two-player zero-sum games, which means that it is unbeatable.

When comparing two-player zero-sum results to multiplayer none of the two-player zero-sum results hold. There can exist multiple equilibria, each with different payoffs to the players. One player may follow one equilibirum while other players follow another equilibrium resulting in the overall profile not guaranteed to be an equilibrium. A player may do worse if the other players deviate from the equilibrium the player uses. Computing the equilibrium is rated PPAD in difficulty[28].

## 2.14 Equilibrium in multiplayer games

Due to the intractability of N-player games, the computation of the Nash equilibirum has been shown extremely time-consuming and memory demanding, especially for large N[28].

## 2.15 Learning Automata

Originally developed in the area of mathematical psychology, Learning Automata (LA) are adaptive decision making devices suited for operation in unknown environments[29].LA combine fast and accurate convergence with low computational complexity, have seen appliance to a broad range of modeling and control problems. The intuitive and analytically traceable concept of learning automata is suitable as a theoretical framework for MARL. La are updated strictly on the basis of the response of the environment, and not on the basis of any knowledge regarding other automata.

## 2.16 Tsetlin Automaton

The Tsetlin Automaton was the first Learning Automaton[29] and one of the first solutions to the well-known multi-armed bandit problem. In an environment, it performs actions $a_z, z \in \{1, 2\}$ in a sequential order, receiving a reward or penalty based on a probability $p_z$. The reward probabilities are unknown to the automaton, and are subject to change over time. The goal under these challenging conditions are to find the action with the highest reward probability in the fewest amounts of tries.

The mechanism driving the TA is simple, as it is a fixed, finite-state automaton. The state of the automaton decides what action to perform, the actions are divided in two "zones" of length $n$. If the current state is in one "zone" the automaton performs one of the two

actions, and the other action in the other. If the automaton is penalized, the automaton moves towards the opposite "zone"'s end, and if rewarded it will move toward the end of the "zone" it is currently in. This is designed to reinforce successful actions. Implementation-wise the Tsetlin Automaton simply maintains an integer and learning is perfomed through increment and decrement operations, according to the transitions.

## 2.17 The Tsetlin Machine

In the Tsetlin Machine (TM)[30], a collective of two-action TA are used for bandit-based learning with the goal of categorize input feature input vectors $\mathbf{x}$ into one of two classes, $y \in \{0, 1\}$. The vector $\mathbf{x}$ consists of $o$ propositional variables, $x_k \in \{0, 1\}^o$. Negations of the variables are also incorporated, which allows for the TM to capture more sophisticated patterns. Together the negations and variables are referred to as literals $L = [x_1, x_2, ..., x_0, x_1, x_2, ..., x_o, ] = [l_1, l_2, ..., l_{2o}]$.

At the core of the TM lies a set of $m$ conjunctive clause, which captures the sub-patterns associated with each output $\mathbf{y}$. All of the clauses in the TM receive the same input, the vector of literals $L$. The clauses each includes distinct literals. In the case of an empty clauses, it will be 1 during learning and 0 otherwise. The TA are the ones that assign literals to clauses. Each clause is equipped with $2 \times o$ TAs, which results in one per literal $k$, shown in *Clause-1* of Figure 2.1. The TA states in "zone" 1 - N map to the *exclude* action, which excludes the corresponding literal from the clause. For states in N + 1 to 2N, the decision is *include*, including the corresponding literal in the clause. The states of all the TAs in all the clauses are stored together in the matrix $\mathbf{A} = (a_{j,k}) \in \{1, ..., 2N\}^{m \times 2o}$, where $j$ refers to the clause and $k$ to the literal. Clause output can be produced when the decisions of the TA are known. As the clauses are conjunctive, they evaluate to 0 if any of the literals included are of value 0.

The TM classifies data into two classes, meaning that sub-patterns associated with both classes must be identified. This is done by dividing clauses into two groups. Odd indexed clauses are assigned positive polarity and seek sub-patterns of output $y = 1$, and even indexed clauses are assigned negative polarity and seek sub-patters of output $y = 0$

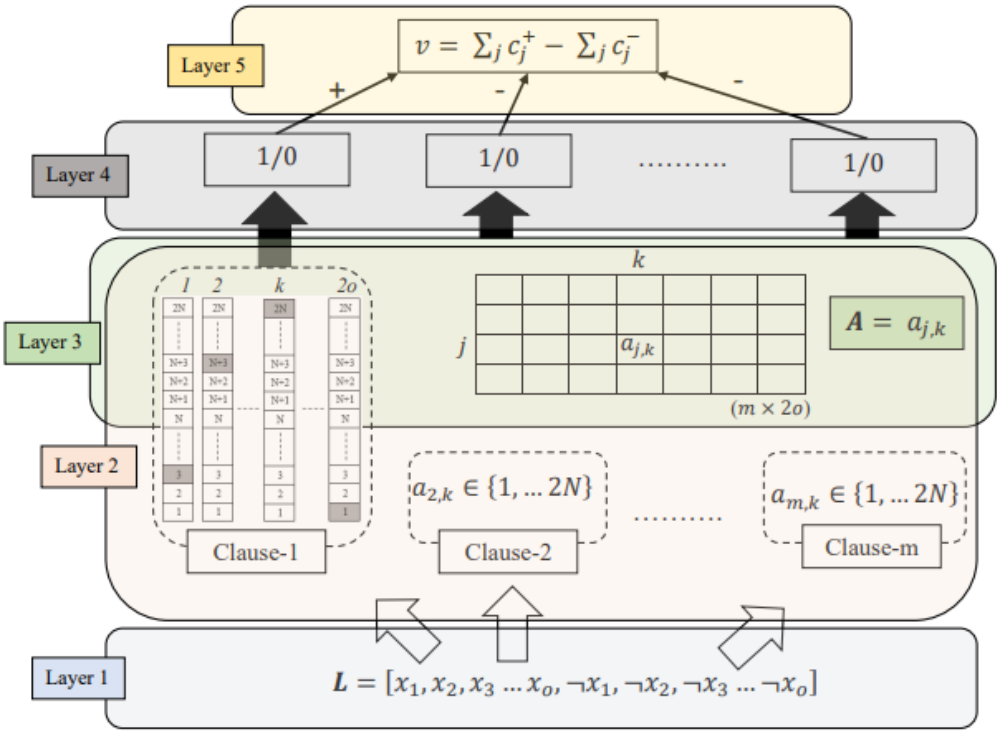



Figure 2.1: Tsetlin Machine structure[31]

## 2.18 Coalesced Tsetlin Machine

The Coalesced Tsetlin Machine (CoTM) consists of four elements[32]:

- The input space $X$, consisting of vectors $\mathbf{x}$ of $o$ propositional inputs: $\mathbf{x} = [x_1, ..., x_o] \in X, X = \{0, 1\}^o$
- The output space $Y$, The output space contain vectors $\mathbf{y}$ of $m$ propositional outputs: $y = [y^1, y^2, ..., y^m] \in Y, Y = \{0, 1\}^m$ each signaling if their assigned class is active or not.
- The memory matrix $C = \{1, 2, ..., 2N\}^{n \times 2o}$. Each of the $n$ rows represents the pattern memory of a single clause. The columns, represent the $o$ inputs in $\mathbf{x}$ and their negations. Together the inputs and their negations are referred to as literals.
- The weight space $W = \{..., -2, -1, 1, 3, ...\}^{m \times n}$ consists of weight matrices $\mathbf{W}$. A weight matrix relates each clause in the memory matrices to an output in $\mathbf{y}$. A positive weight assigns the clause to output value 1. A negative weight assigns it to output value 0. The magnitude of the weight decides the impact of the assignment. A pattern matrix combined with the weight matrix $\mathbf{W}$ configure a complete CoTM.

Based on the above information, the output $\mathbf{y}$ can be predicted form input $\mathbf{x}$ using propositional and linear algebra.

The memory matrix is updated based on a training example ($\mathbf{x}$, $\mathbf{y}$) using three kinds of feedback matrices:

- Type Ia feedback concerning output $y_i$ of output vector $\mathbf{y}$. It operates on clauses that both match the input $\mathbf{x}$ and that are assigned to output value $y_i$. It only affects

literals that are True and tunes the clauses to represent the current input $\mathbf{x}$ more finely, reinforcing *Include* actions.

- Type Ib feedback concerning yet again $y_i$. It operates on all the clauses that are assigned to utput value $y_i$ by the clause-output weights. For those clauses that does not match the input $\mathbf{x}$ it operates on all the literals. For clauses that match the input it only affects False literals. The clause is coarsened by being forced to forget literals, reinforcing *Exclude* actions.

- Type II feedback concerns yet again $y_i$. It operates on clauses that both match the input $\mathbf{x}$ and that are assigned to the negated output value, **not** $y_i$. Only False literals are affected. This feedback increases the discrimination power of the matching clauses by introducing literals that invalidate the matching.

# Chapter 3

# Reinforcement learning

This chapter will present how a reinforcement learning environment for a desired game can be created with the help of PettingZoo[15] and trained through the use of an already implemented algorithm from Ray RLlib[22] with a custom backbone network.

## 3.1 Creating the environment

The first step was to implement an environment that let the agent play Stefan Dorra's For Sale. At the time of writing, there is no openly available implementation of this environment, and as such, it had to be implemented. The objective is to generate an agent playing at near optimal winning trajectories, to help with further investigation.

### 3.1.1 Framework

Initially, the environment and agent were created fully from scratch, but due to implementational difficulties when attempting to train a model with multiple different algorithms, the environment was migrated to PettingZoo[15] due to its compatibility with Ray RLlib. Ray RLlib has several algorithms already implemented, this combined with plug and play abilities of the library made the task possible, and the work put in to solve the Ray RLlib dependencies worth it.

### 3.1.2 Action masking

After the correct game logic had been programmed, the simulator also needed a way of representing which actions are available to an agent in any given state. Stefan Dorra's For Sale has an action space with a variety of different actions that an agent can play, not all of which are possible depending on the current state of the game. In fact, in a typical state, a vast majority of all possible actions an agent could take will not be allowed, in both the first and second round, as there are 30 cards to choose between in each round, and each agent will at most be able to play one of eight different cards in the three player version. There is also the betting of coins during the first phase, where it is not possible to bid the same or lower as the current highest bid. As such, to train the agent efficiently it is crucially important to mask out invalid actions. The alternative - i.e. allowing the agent to take invalid actions but ensuring they don't have any effect on the environment - is incredibly inefficient, since the

agent would waste a huge amount of time trying out actions that would amount to nothing. This led to utilizing one-hot encoding as the action mask. Each available action would be encoded to a 1, and each unavailable action would be encoded to 0.

### 3.1.3 Observational space

The observational space is the information the agent receives regarding the current state of the game. Parameters were found through playing games of For Sale with other people, and each player evaluated what parts of the game made the player do what action. This is divided between agent-related parameters and board-related parameters. These parameters together result in the betting players' possibility of making an informed choice regarding the action to perform. A visualized example state of gameplay can be seen in Figure 3.1 and a simple example of an environmentally rendered example can be seen in 3.1

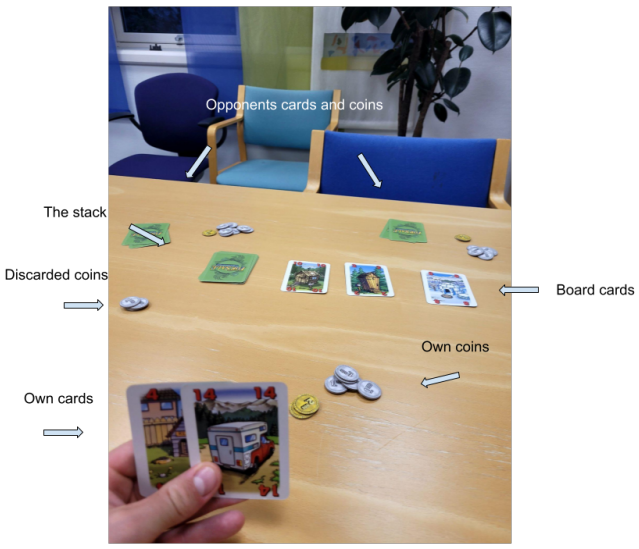



Figure 3.1: A visualized example state.

For each agent these parameters were displayed (here with value range):

- Current coins [0 - 18].
- Current acquired cards (max amount dependant on the number of rounds).
- If can bid currently [0-1].
- This current bid [0-18].
- Total coins on pass [0-18].
- Card received on pass (max dependant on the number of rounds).
- Next round order if all passes from now.

The board related parameters are used:

- The current cards on the board.
- The current stack (possible cards that can appear on board).
- The current standard deviation of the board.

| Active clause's reduced literals |
|---|
| P1 coins 15 |
| P1 cards [7, 2, 0, 0] |
| P1 can bid 1 |
| P1 current bid 0 |
| P1 coins on pass 15 |
| P1 card on pass 8 |
| P1 next round order if pass 3 |
| ⋮ |
| P2 ... |
| ⋮ |
| P3 ... |
| ⋮ |
| board [12, 10, 8] |
| stack [12, 11, 10, 0, 8, 0, 6, 0, 0, 3, 0, 0 |
| current standard deviation 1.63 |

Table 3.1: An example state from the 3 round version, only showing player one for simplicity.

#### Agent Related Parameters

The total agent-related parameters is the list of agent-related parameters, repeated for each of the three players. The visible parameters were directly taken from For Sale, such as player coins, acquired cards, etc. were included to make the game follow the rule-set of the original game. In addition, some "hidden" parameters were included, parameters that are in the game, but not immediately visible. One of these parameters is if each player can bid. Although not blatantly visible, it is an important part of the game and may change actions taken, if the next player cannot bid, and the current one can, bidding will prove to win the highest card. If however the final player also can bid and has more money than the current player, the current player may be prompted to pass instead. The current bid is also one of these "hidden" parameters, as in the game when played with people, this is only conveyed orally. The current bid is also connected strongly to the current coins of each player and the can bid parameter, as it is only possible to bid if a player has a larger amount of coins compared to the largest current bid. Mainly dependent on the current bid are the coins the player has after a fold at the current state of the game. If the player wins the round, it will pay the full amount of their current bid, otherwise, it will pay the current bid divided by two rounded up. The final two agent-related parameters are the card the player will receive if it folds, and the expected betting order for the next sub-round if the player folds. These two parameters are quite self-explanatory, and the order depends mainly on if any agents have folded already.

#### Board Related Parameters

The board related parameters can be argued to have the most impact per slot. These parameters describe the status of what the agent is currently bidding on, and what has yet to appear on the board. The first parameter is the current cards on the board, which have been included to let the agent be able to see what they are betting on. Followed by the stack, which is all the possible cards that can appear on the board, giving the agent the option of possible planning and knowledge of future cards. It may choose to wait for certain high cards that have yet to come, or bid on the current card available now. The possibility

of utilizing the agents memory and innate risk-taking is also granted through the use of the stack. Finally the standard deviation is used so that the agent may differentiate the spread of the boards from each other. The agent will hopefully learn that getting a high cards is good, but a board consisting of the cards 25, 26 and 27 and a board consisting of 5, 6, and 27 should be played very differently. In the first board not bidding may actually be better than using a lot of resources to get the 27, the standard deviation on the first board is low resulting in a low spread of values, while on the second board it is high. In the first board it is much less rewarding compared to the default reward to use resources. Difference between the numbers could also have proved to be effective, but the standard deviation provides some of the same benefits, by only taking up one numbered spot as opposed to the difference which would have needed up to three spots.

### 3.1.4 Rewards

The amount of rewards generated by the environment is sparse. A player is rewarded 0 for each action until the game is done. Depending on how many sub-rounds will be played, the total moves will increase or decrease, making the rewards more or less sparse. When reaching the terminal state the players will receive 1, 0, or -1 as a reward, depending on where the player is placed in the end game result. 1 for victory, 0 for second place, and -1 for last place, in the case of a draw, the winning players all receive a 1.

## 3.2 Generating Winning games

In this sections we explain how to construct the database of optimal plays for the game For Sale.

### 3.2.1 Black Box Solving the First Phase

The first phase, as previously mentioned consists of standard English auction bidding, with the goal of accumulating properties of the highest value possible. In this part of the game, there are at most 30 possible property cards available which the board can be created from, and each player has at most 18 coins that can be bid. The aim is to get into a position at the end of the first round, where the player has the best outlook towards winning the second round. Meaning the start with the highest possible value. The player with the highest total value, will be considered the winner. The training is done through parameter sharing of a single algorithm policy.

Initially the games was supposed to consist of 3 players, where at each game the amount of intelligent players was chosen randomly between 1, 2 and 3 intelligent players. Then the winning moves of each game would be saved together with the observation. After some analysis it was determined that the games against the random agents were of a low quality. and as such the creation of the winning games database was done through the agent playing against two copies of itself. The winning agent's moves was then recorded and saved with each corresponding observation.

#### ICM vs PPO

The main two algorithms compared with each other is ICM and PPO, more information regarding these two algorithms can be found in Chapter 2. PPO with clipped objective was chosen due to its performance being comparable or better than state-of-the-art approaches, in addition to being simpler to implement and tune. While ICM was chosen due to it being specialized in sparse environments, and the For Sale environment is sparse, as all bids give a reward of 0, until the terminating bid, where all agents get their positional reward. Each model will also be compared to a random agents, due to the nature of the game, a random agent should lose the great majority of the time. This is due to the great action space in the first sub-round, the random agent has a much greater chance of exhausting their funding early in the game, compared to the intelligent agent. Still it might be possible for the random agent to win games, but again, this should be rare.

#### Backbone network

To use the PPO and ICM algorithms with Ray RLlib, a backbone network is needed. In this project a neural network with fully connected ReLu layers was used, with two layers of 256 nodes and three layers of 256 nodes were used for the 3 rounds and 4 rounds respectively. The number of nodes was found through trial and error modeling for quick convergence and equilibrium.

### 3.2.2 Experiments

The experiments were of different sizes and were reduced in size compared to the original game, to reduce complexity, as well as being limited to the first round version of For Sale. Moreover the addition and focus on the 4-round version as opposed to the 3-round version was due to the 3-round ICM getting fixated on betting 9 every round it could. Resulting in the actual ability of the environment and the agent came into question, in addition to the comparability between the 3-round version and the 8 rounds used in the regular game. The main game has an even amount of rounds, which could be the design choice for a reason. The main purpose of these experiments is to create enough high-quality data that the Tsetlin Machine can use as training material.

#### 3 Sub-rounds of the First Round

The experiments with the 3-round version of the first round consists of 11 possible cards, 18 starting coins, 3-rounds and 3 players. The algorithms used are PPO and ICM. The ratio between the amount of additional cards to the cards given out is derived from how the original game is set up with 3 players, 25% of the total amount (rounds * players) floored cards are added as extra cards, making not all cards guarantied to make an entrance during play. In the For Sale environment all cards, including the excluded ones are added to the stack. This is due to the players not knowing what cards will be in play until the last sub-round of betting in the first round.

### 4 Sub-rounds of the First Round

The experiments consisting of the the 4-round version of the first round had 15 possible cards, 18 starting coins, 4 rounds and 3 players. The algorithms are PPO and ICM. The ratio between cards are the same as in the 3-round version.

# Chapter 4

# Training TM

This chapter will describe the different choices and what emphasis was put on.

## 4.1 Pipeline

Initially, the way of training the Tsetlin Machine to play For Sale had to be revised multiple times. The first iteration tried to train the Tsetlin Machine from scratch on the environment, only learning from winning games. At the start of the experiment, the TM played against 2 copies of itself, similarly to how AlphaZero works, the winning moves and states were saved, and the TMs were trained on this data at the end of each epoch. After the TM had trained for several timesteps, it was evaluated against 2 random players, where it did not outperform the random agents, but it consistently had an equal percentage win rate against itself where the spread was 35.5%, 35% and 34.5% and had reached a nash equilibrium. The next course of action was to train against these 2 random players, and train against itself when it beat the 2 random players consistently. After several games played, the TM did learn how to play For Sale similarly to the random agents, but could not outperform the random agents with the setup used at the time. After attempting to fix this issue, it was decided to approach the problem differently. If the TM would not learn from random, and would not learn from itself, then a black box model would train itself to a superhuman stage, and then the TM would learn the steps taken at the different states. This resulted in the general pipeline seen below:

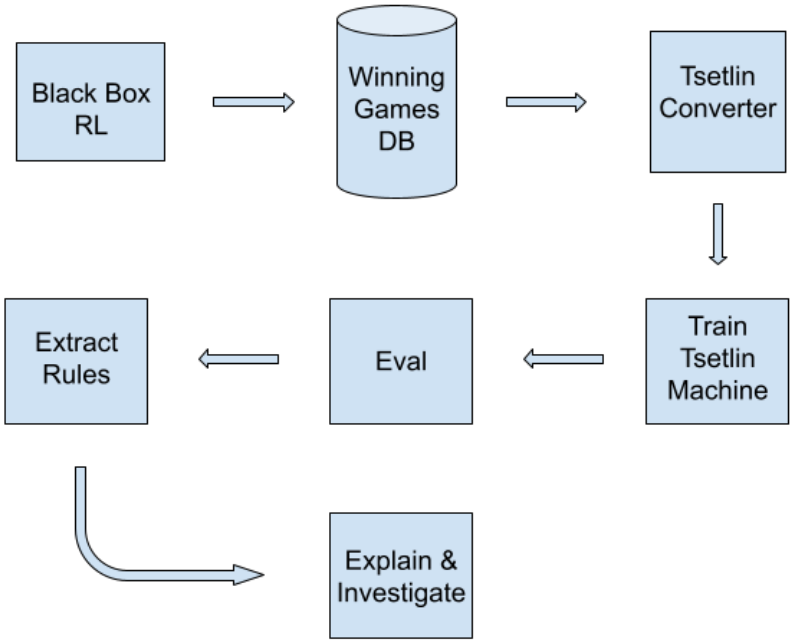



Figure 4.1: The general pipeline utilized in the project

First, a black box model is trained on the environment. After training the model sufficiently so that it acquires peak performance, the model plays against two clones of itself. Then the winning moves are saved in a database together with the observations as states. The entries in the database are then converted from the observations into a format that the Tsetlin Machine can learn correctly from. The Tsetlin machine is then trained on the winning games and evaluated. After training the rules can be extracted from the clauses, and analysis can begin. Finally, the rules are explained and different scenarios are investigated.

## 4.2 Comparison Between other Algorithms for Interpretability and Accuracy

The resulting Tsetlin Machine will be compared to other machine learning architectures, in regarding prediction of the next move. Additionally, a test of interpretability will be done. Following a setup inspired by[10] the algorithms are as follows:

- Decision Tree.
- Neural Network.
- KNN.

## 4.3 Encoding of the Parameters

For the CoTM to be able to learn correctly from the states and moves the winning agent generates, the parameters need to be encoded in a format that the CoTM understands. The CoTM only takes propositional input, as in the previous versions of the TMs, therefore all the parameters need to be encoded either into one-hot, cumulative encoding, or a binary representation of the current number. Initially, the idea was to reduce the number of features

used during the conversion from observations to TM compatible observations, to reduce the time spent on each epoch. This resulted in binary encoding for all the numbers apart from the numbers in the stack which were encoded to one-hot. The highest resulting amount of bits per number was 5, while the highest represented number was 18. After further analysis, the focus switched to reducing the number of features to a minimum, to create features that will be able to give the most explainability in regards to feature extraction.

The reason for the change of focus was due to the information gained from a single bit of the encoding is often dependent on the context. For example, information regarding a number from 0 - 7 is to be extracted from binary, and a rule shall be made. It is relevant to know the status of all the bits, as the other options could be either on or off, changing the number. If only one bit's location and status are known, it is difficult to know what is represented. Say the number is 4, and we receive the status of the second bit, which is off. The information known would only be that the bit representing the decimal number 2 is switched off, the number could be 0, 1, 4, or 5. This would not make it possible to accurately get information regarding the number, it is either a high number or a low one. A way of circumventing this would be to use cumulative encoding instead. In the cumulative example, if the number still is 4, and the information is to be received from the second bit, the number 2 would be active. The information received from this one bit would be that the number is either 2, or higher.

By utilizing the cumulative encoding for the features, it will result in when analyzing the literals in each clause, that the only number to care about for each feature is the literal of the largest magnitude in that clause. In a setting where feature negating is inactive, the result per clause would at most have the same amount of literals as the original features used to create the cumulative encoding. The TM will not at first have learned the connection between all the cumulative features, but after sufficient training, the connection will be learned, and rule extraction can be made easier.

## 4.4 Tsetlin Machine Rule Extraction

After training, the CoTM can predict a class by transforming the input consisting of 0's and 1's into an array of clauses being active (1) or inactive (0). If the clause is active the weights will be added to the total of that class, and the class with the highest sum, is the resulting classification. By utilizing the known weights for each class it is possible to learn what clauses are the most important for each desired output. The input space varied during testing, but ended at 556 features, after the cumulative encoding.

To extract the rules several different methods will be tried and evaluated.

1. A naive example of clause output with reduced literals.
2. The heaviest weighted clauses for each class for regular CoTM.
3. The frequent item mining algorithm Apriori applied to regular CoTM.
4. The frequent item mining algorithm Apriori applied to CoTM ensamble.
5. Exctraction of graphically displayed intervals through the cumulative encoding.

In this case the CoTM ensamble will consist of 3 different CoTM trained for the same amount of epochs, that all will be used together, and the results from each of the CoTM in the ensamble will be analysed as a whole.

### 4.4.1 Features and Feature Extraction

Using the feature transform function the propositional input is transformed into the clause output for that state through using the memory space. By multiplying the clause output with the weights it is possible to find what class the Coalesced Tsetlin machine has assigned to the input, which is decided by a majority vote, the highest vote decides what class it is. Investigating further into the clause output and the weights for each class, it is possible to see the reasoning for the choice and to see the runner up/s regarding what classes have gotten the most votes. Looking at the weights alone it is possible to determine what clauses are of most importance for each class, denoted by how positive or negative the number is. If the number is large it strongly suggests high importance, either in favor or against respectively. The most polarizing clauses for each class are the most important when extracting impactful rules during the investigation.

Through the different methods mentioned above the most important clauses can then be investigated , with the knowledge regarding the input features and what they represent. As each clause captures one particular feature interaction, it is possible to map out the most important feature rules. This is done by converting each feature to its "human readable" name, and then looking at each of the impactful clauses, and then reading out what that specific rule is.

The input features are of utmost importance in this part, as deciding on the right features to include in the analysis will greatly impact how much information it is possible to get out of the CoTM. This could include extra parameters created from the data, in relevance to the project in question for instance using the mean, how much a player is leading, if a player is leading, etc. Including smart features makes it much easier to extract good rules from the TM and greatly improves interpretability.

### 4.4.2 Graphical interval mining

The graphical interval will be shown in a game example found in Figure 5.8. It is found by getting all the activated clauses for a game state. Then the weights of the selected class is used to find either all positively or negatively weighted clauses that are active, depending on what is desired. The intervals are furthermore found through identifying the largest cumulative numerical literal of each type in each clause. Consequently the the largest numerical literals are added to the reduced clause together with other non-numerical activated clauses. Finally positive and negated clauses are plotted against each other to visualize the intervals.

### 4.4.3 Game scenario

Finally a game scenario will be presented, and the reasoning behind the CoTMs actions will be explained through graphically extracting the different value intervals.

# Chapter 5

# Results

This chapter will present all the findings and results throughout the thesis.

## 5.1 Training BlackBox AI player

During the training of the AI player, 3 different models were trained for 100.000.000 timesteps, one ICM, and two PPO. Information regarding the backbone network for the ICM and PPO can be found in chapter 3. These then played against each other for 1000 games, in every order possible, to decide the best model. The results can be seen below in Figure 5.1 and 5.2:

| Order | Model | Wins |
|---|---|---|
| 1. | ICMv0 | 18 |
| 2. | PPOv0 | 420 |
| 3. | PPOv1 | **642** |
| 1. | ICMv0 | 13 |
| 2. | PPOv1 | 261 |
| 3. | PPOv0 | **820** |
| 1. | PPOv0 | **770** |
| 2. | ICMv0 | 10 |
| 3. | PPOv1 | 279 |
| 1. | PPOv0 | 466 |
| 2. | PPOv1 | **612** |
| 3. | ICMv0 | 9 |
| 1. | PPOv1 | **586** |
| 2. | ICMv0 | 14 |
| 3. | PPOv0 | 468 |
| 1. | PPOv1 | 315 |
| 2. | PPOv0 | **746** |
| 3. | ICMv0 | 12 |

Table 5.1: All possible orders of play with the three models, with results.

This result shows that PPO outperform ICM significantly. Another test with the two PPOs was performed, but the last player was in this experiment a random agent. The results from this test show PPO outperformed ICM to almost the same degree as to how it outperformed the random playing agent, which signals that ICM was a poor choice for this specific environment.

| Model | Total Wins |
|---|---|
| PPOv1 | 2695 |
| PPOv0 | **3690** |
| ICMv0 | 76 |
| Random | 59 |

Table 5.2: The total wins of the models, with results from random agent vs PPOv0 and PPOv1

An additional reason for choosing PPO over ICM is that ICM currently lacks support for multiple workers in Ray RLlib. This makes it not possible to utilize GPU computing when training, and results in far longer training times for the same amount of timesteps. The PPO algorithm completed the same amount of timesteps in 1/3. of the time.

From the result of PPO*2 vs ICM, a pattern can be seen, as the winner is consistently the player that is two turns away from the worst player. Consequent analysis leads to the discovery that the best location for a player is 2 turns after the least skilled player.

| Player | Total Wins |
|---|---|
| player_0 | **403117** |
| player_1 | 345117 |
| player_2 | 353493 |

Table 5.3: The total wins per player position after PPOv0 played 1 million games against itself.

The result from 1 million games generated from the winning PPO playing against itself can be seen in Table 5.3 above. From the 1 million games, another pattern can be discovered, if all players are of equal skill, the player that makes the first move, will win more. This makes sense, as this is an auction game, and the type of auction when bidding for the highest card is an English auction[13], where the one betting the expected value is winning. Therefore the player that first gets to bet the expected value will come out slightly ahead. But as this also is an auction where all players will get an item, this offsets some of the winnings from the first player as the other players will also receive an item.

## 5.2 Creating the training data

During the training of the models and the comparison between them , the first approach of having 1, 2, or 3 intelligent players (where the unintelligent players played randomly) was scrapped. This was due to the extreme difference in winrate between the unintelligent players and the intelligent players, which can be seen below.

The agent reached Nash equilibrium with itself, and as the intelligent model consistently beat the random players the games were argued to be games of low quality. This was due to the possibility of the intelligent agent winning even with non-optimal moves. The agent played 100.000 full games of both 3 rounds and 4 rounds which resulted in over 500.000 and

| Order | Model | Wins |
|---|---|---|
| 1 | PPOv0 | **96763** |
| 2 | Random | 2176 |
| 3 | Random | 2533 |

| Order | Model | Wins |
|---|---|---|
| 1 | Random | 2649 |
| 2 | PPOv0 | **96032** |
| 3 | Random | 3011 |

| Order | Model | Wins |
|---|---|---|
| 1 | Random | 1560 |
| 2 | Random | 1794 |
| 3 | PPOv0 | **97722** |

Table 5.4: The PPOv0 model playing 100.000 games against 2 random agemts.

over 700.000 states for each amount of rounds previous to data preprosessing, respectively.

## 5.3 Data Preprocessing

As opposed to hex that has a natural binary representation, the multi faceted board representation of For Sale makes both the representation and the following analysis significantly more difficult. For this reason the games played by the black box agent require preprocessing. Ideally, the same amount of winning games for each class are represented, making it possible for the rules from the Coalesced Tsetlin Machine to be more decisive. Due to this operations will be carried out to smoothen the training data from the black box player into a more uniform dataset. The raw data will have an over-representation of 0 bets, as this is the passing bet, and at some point in the game, unless the player is winning the bet of that round, the player will have to pass. All games where the player has no choice but to bet 0 were removed, as there is no interest in seeing what the AI wants to do if it has no say in the matter. Duplicate games/states are also of no interest as the most frequent classes will have more appearances here and would need smoothing, and as such will be filtered out as well. To smoothen the data further, an oversampling is carried out, where the classes with less than 2000 games will have their games duplicated randomly to meet this requirement. The entire process can be seen for one example in Figure 5.1. Finally, the test set and the training set will be stratified, so that an equal percentage representation of each class will be used during training. In addition, the self can bid feature will also be omitted, as again, if the AI has no choice, it is not interesting to see what it chooses to do nor why the action was done.

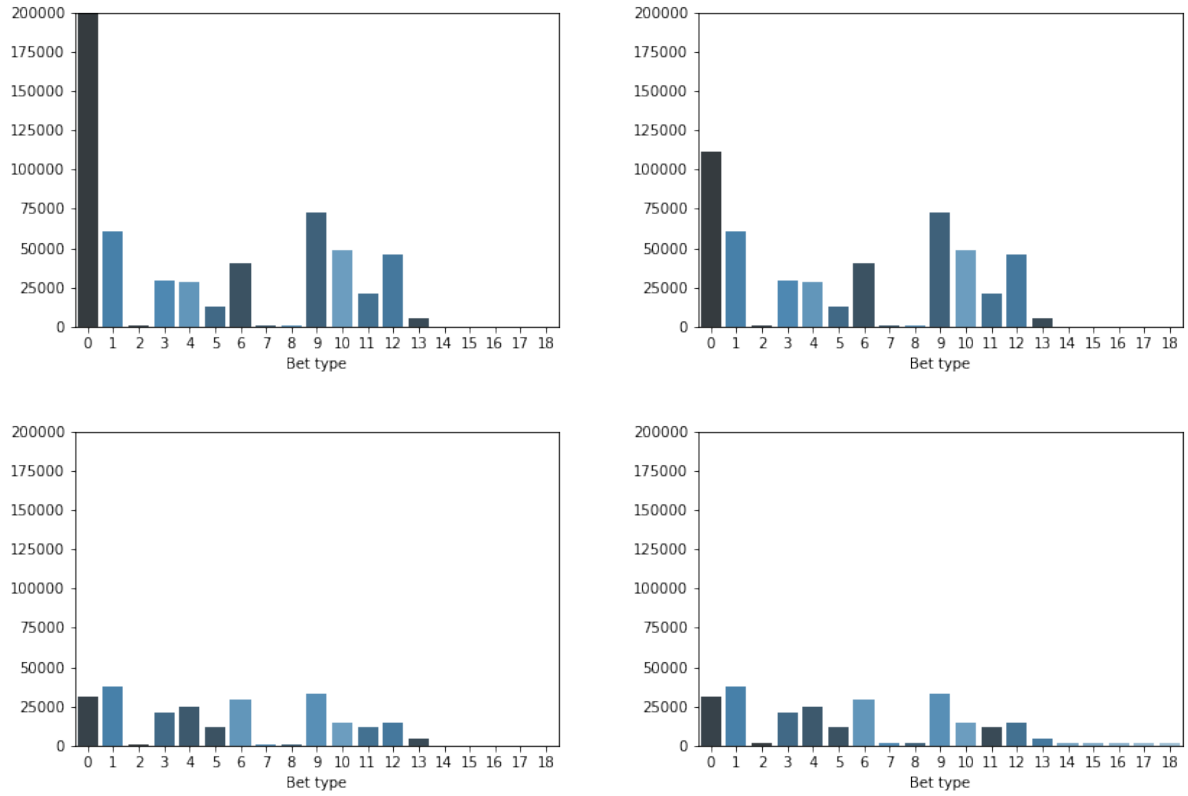



Figure 5.1: Visualization of the data prepossessing process, here with the results from 100000 games with ICM 3-rounds. Top left is the raw data, top right all games where the agent had no choice other than betting 0, bottom left is the data after duplicates have been removed and finally bottom right is the data after oversampling has been applied.

## 5.4 Black Box Model Y Distributions

### 5.4.1 3-rounds ICM trained

The 3-round ICM model, where the data prepossessing can be seen in 5.1, initially had a fixation with betting either 9 or 0. Additionally, there seems to be no clear pattern in the data distribution, other than a divide seen at the 7 and 8 vote mark, as well as 2, 14, 15, and 16 not being used much. The bets 17 and 18 are possible bets that have not been used at all and has therefore been omitted from the plot in Figure 5.2below.

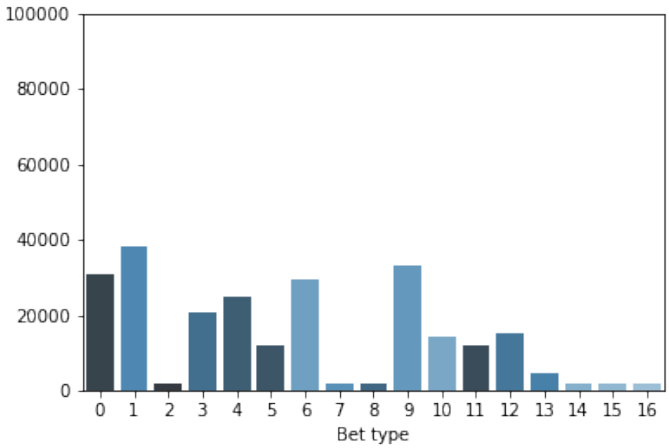



Figure 5.2: The y distribution after cleaning 100000 ICM 3-round games.

Due to the 9 being as prevalent as it was in Figure **??**, the entire game seemed to be too different from the 8 rounds of the original For Sale setup, and as such, the emphasis was put more on the 4 rounds instead as an even number of rounds could have been a design choice in the original game.

### 5.4.2 3-rounds PPO trained

The resulting data from the 3 rounds of PPO, can be seen in figure 5.3 below. The data did take on a more expected shape, as it is closer to a normal distribution, and more continuous than the ICM version. It can also be noted that almost all (apart from 10, and the oversampled bet numbers) even numbers of bets have a higher representation than their odd counterpart. This makes sense when taking into account how the game interacts with the different bets. The player betting an odd number, will "pay" equal amounts when folding, as the same odd number + 1. This is due to the returning coins being the bet / 2 rounded down. It also makes sense that there are substantially more 0 bets, as this is the pass bet, and for a sub-round to be over, 2 passes need to happen, one from each of the players losing the bid.

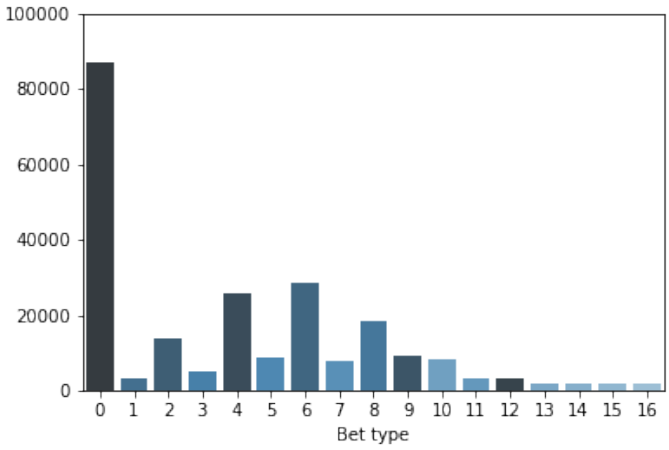



Figure 5.3: The y distribution after cleaning 100000 PPO 3-round games.

### 5.4.3 4-rounds ICM trained

The resulting plot from 4-rounds with ICM is skewed more toward a lower number and can be seen in 5.4. This is due to the game having more rounds, and a large bet will more likely contribute to the player exhausting their funds. Having exhausted funds means that it is impossible to bet and compete with the other players regarding the most valuable cards each round on average. The ICM still insists on using odd numbers more than even numbers in general.

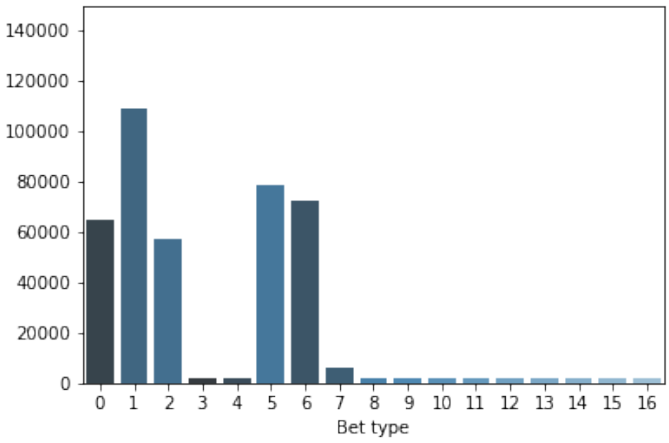



Figure 5.4: The y distribution after cleaning 100000 ICM 4-round games.

### 5.4.4 4-rounds PPO trained

The resulting plot from 4-rounds with PPO is also skewed more towards bidding lower values, and has a steeper curve at the start than the 3-rounds PPO. There are more even bids than odd bids, in the same manner as the 3-round version above. The result from 4-rounds can be seen below in Figure 5.5.

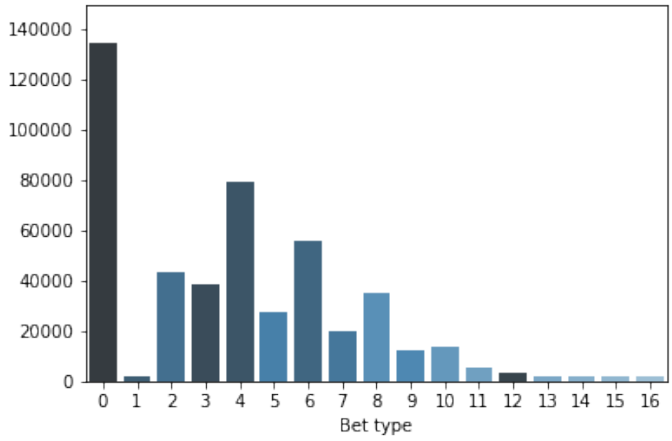



Figure 5.5: The y distribution after cleaning 100000 PPO 4-round games.

## 5.5 Comparing CoTM accuracy to other methods applied to the data

The experimental setup was as follows. The CoTM has two hyper-parameters $s$ and $T$, along with the number of clauses $n$. For the experiments the general rule from [10] of T being 80% of the total number of clauses were implemented, leaving s to be tuned. Resulting in s with a value of 40.4, 8000 clauses, 100 epochs, and a T value of 6400. As the TM used in this thesis is the CoTM instead of the standard TM, this may not be the optimal solution, regardless s was found through a limited random hyper-parameter search in the range between 0.1 and 100. Performance may increase following an extensive automated grid search. 80% of the data was used for training and 20% for evaluating prediction accuracy. Additionally, the data was stratified so that a more equal percentage of the classes was represented in both training and testing.

Table 5.5 contains the results from the CoTM training, including a comparison with three popular machine learning algorithms. K-Nearest Neighbours was trained with default sklearn parameters, and the Decision Tree was also trained with default parameters, apart from setting a limit on the max depth of 50. The neural network consisted of 5 layers with Sigmoid activation functions, trained with the Adam optimizer. The CoTM managed an average testing accuracy of 84.5%, beating all the algorithms, and outperforming the other interpretable method, the Decision Tree by roughly 7%.

| Architechture | Hyperparameter | Accuracy |
|---|---|---|
| Decicion Tree | Max Depth = 50 | 78.42 |
| K-Nearest Neighbour | K=5 | 74.79 |
| Neural Network | | 79.05 |
| Coalesced Tsetlin Machine | Clauses = 8000, T = 6400, S = 40.4 | **84.55** |

Table 5.5: Results from training on the generated dataset.

## 5.6 Interpretability of other methods

Decision trees are most often associated with interpretability, but some paths in the decision tree may include many literals that are irrelevant for an explanation[33]. In addition to possibly including literals that are irrelevant, the decision tree loses interpretability rapidly with growing size. Furthermore, the state space of For Sale is complex, which further helps reduce the interpretability with possible noise. As seen in Figure 5.6, no useful information can realistically be gained from this tree with a depth of 50 trained on the winning game data.

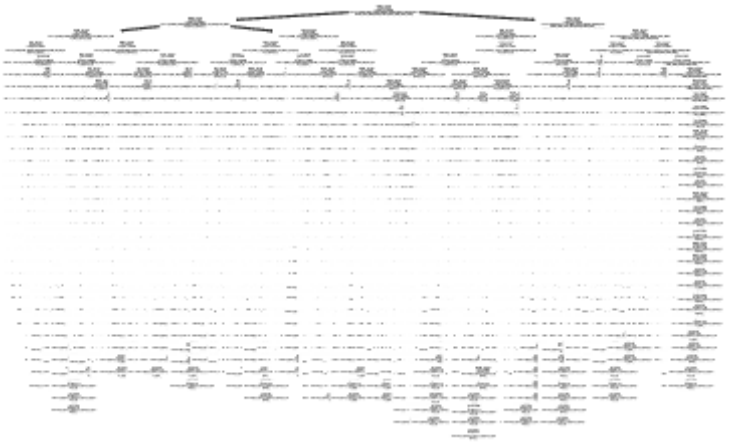


Figure 5.6: Plotted DecicionTree after training.

K-Nearest Neighbors is hard to interpret due to the complexity of the game state and Neural Networks are a black box model, due to this very little interpretable information can be extracted from these methods[34, 35].

## 5.7 Explaining CoTM output

Here the CoTM output will be explained as well as investigate how the input features have been engineered for global interpretation[36]. All literals have been reduced from the original cumulative encoding. The way reduction has been carried out is through finding the highest number following each literal name, for each clause. Then as the input has been cumulatively encoded, all numbers lower than the highest number can be ignored. The one-hot encoded inputs and the strictly binary inputs have been kept as they were.

### 5.7.1 Naive example

The naive example shows the reduced literals for one clause in Table5.6. Alone, this clause does not provide much interpretation, but analyzing all activated clauses from a transformed input would also be overwhelming and not feasible.

| Active clause's reduced literals |
|---|
| ¬std bit |
| self tot score 14 |
| ¬self tot score 58 |
| self pass score 24 |
| ¬next coins 15 |
| ¬next tot score 43 |
| next pass score 22 |
| ¬next pass score 40 |
| final coins 7 |
| ¬final coins 12 |
| final tot score 10 |
| final pass score 12 |
| ¬final pass score 43 |
| ¬2. card 11 |

Table 5.6: Example reduced naive output clause.

### 5.7.2 Top Weighted

An example of the Top 5 weighted clauses for a class with single CoTM can be seen in Figure 5.7. Top 5 weighted clauses provides more insight than a single activated reduced clause, as more clauses are represented and all clauses advocates for the same output. In the example below the top 5 heaviest clauses pushing for betting six can be seen. It is still difficult to clearly obtain why six should be bid, as there are more than 20 literals in the example. From the reduced literals some interpretation that makes sense can be extracted. From the first clause, it can be seen that it registers that the lowerst card is 8 or higher, and that the next player to bid has bid 3 or more as well as the final bid being lower than 5. The negated final bid 5 literal appears in some of the other clauses as well as does the next bid 3 literal. It may also be noted that the clauses does not convey much information alone, and needs to be seen as a collective.

| Active clause's reduced literals ranked | | | | |
|---|---|---|---|---|
| 1. | 2. | 3. | 4. | 5. |
| self pass score 4<br>¬next tot score 43<br>next bid 3<br>¬final bid 5<br>3. card 8 | ¬std bit<br>self coins 12<br>self bid 1<br>¬self bid 17<br>¬self tot score 53<br>self pass score 11<br>¬self pass score 60<br>next coins 3<br>next tot score 17<br><br>¬next tot score 41<br>next bid 3<br>¬next bid 13<br>next pass score 20<br>¬next pass score 57<br>final coins 5<br>final tot score 11<br>¬final tot score 60<br>¬final bid 5<br>final pass score 10<br>¬final pass score 45<br>1. card 6<br>¬2. card 15<br>¬3. card 12 | ¬self tot score 45<br>self pass score 15<br>¬self pass score 55<br>next bid 3<br>¬next bid 5<br>¬final tot score 40<br>¬final bid 5<br>¬3. card 6 | ¬std bit<br>self coins 11<br>¬self pass score 38<br>¬next coins 17<br>¬next tot score 37<br>next bid 3<br>¬final bid 5<br>final pass score 10<br>¬final pass score 41<br>¬3. card 11 | ¬next pass score 5 |

Table 5.7: Top 5 weighted clauses for class 6 for single CoTM.

### 5.7.3 Frequent item mining

Through using the Apriori algorithm the active reduced clauses that weighted towards the specific class was grouped together in k-itemsets. In Table 5.8 all the best represented groups with more than 1 % support can be seen. More details on Apriori and support can be found in Section 2.13. This way of representing the data was not found to contribute much interpretability for a single CoTM, as many of the literals are general. Almost all the negated score literals depict close to the maximum score theorized and set in this version of For Sale, which only tells the reader that the game is not over yet.

| Support | Itemsets | Length |
|---|---|---|
| 0.014851 | (final coins 8, self tot score 19, final tot score 19) | 3 |
| 0.014851 | (final tot score 15, self pass score 22, ¬self bid 10) | 3 |
| 0.014851 | (¬self pass score 60, final tot score 15, ¬final bid 17) | 3 |
| 0.014851 | (¬final bid 16, final tot score 19, ¬final pass score 60) | 3 |
| 0.014851 | (self bid 1, next coins 6, ¬final bid 18) | 3 |
| 0.014851 | (¬next tot score 62, ¬2. card 15, ¬final bid 17) | 3 |
| 0.014851 | (¬next pass score 53, ¬final bid 16, ¬self tot score 56) | 3 |
| 0.014851 | (¬self tot score 61, ¬next bid 17, ¬final bid 17) | 3 |
| 0.014851 | (¬self pass score 60, ¬next can bid, ¬final pass score 60) | 3 |

Table 5.8: Example apriori result from single CoTM with support over 0.01 and length more than 1.

## 5.8 Ensamble CoTM

The ensamble CoTM consists, as mentioned earlier of 3 CoTM, and will be analyzed in a similar way as the CoTM in the above section..

### 5.8.1 Frequent item mining

Apriori was again used as a way to get the most important clauses from the CoTM ensemble. The results for most support and the most supported groups can be seen in Table 5.9 and Table 5.10. Apriori from the ensemble has the same drawbacks as the single CoTM version, more example clauses still pose close to the same result. From the biggest support entries, it can again be seen that the most supported entries contain the most common negations, such as negated self pass score 62. This again means that the player does not have the maximum score if the player chooses to fold.

| support | itemsets | length |
|---|---|---|
| 0.116866 | (¬final bid 17) | 1 |
| 0.104914 | (¬self pass score 62) | 1 |
| 0.118194 | (¬self tot score 62) | 1 |

Table 5.9: Biggest support entries from ensamble CoTM.

| support | itemsets | length |
|---|---|---|
| 0.011952 | (self bid 1, ¬self pass score 62, ¬cards to aquire 2) | 3 |
| 0.011952 | (self bid 1, ¬final pass score 62, ¬final bid 17) | 3 |
| 0.011952 | (self bid 1, ¬self tot score 62, ¬final bid 17) | 3 |
| 0.011952 | (self bid 1, ¬next tot score 62, ¬self pass score 62) | 3 |
| 0.011952 | (¬next pass score 62, ¬final pass score 62, ¬final bid 17) | 3 |
| 0.013280 | (¬next can bid , ¬final tot score 62, ¬final bid 17) | 3 |
| 0.011952 | (¬self bid 18, ¬next bid 18, ¬final bid 17) | 3 |
| 0.013280 | (¬self tot score 62, ¬next bid 18, ¬final bid 17) | 3 |
| 0.014608 | (¬self bid 18, ¬self tot score 62, ¬final bid 17) | 3 |
| 0.011952 | (¬self tot score 62, ¬self pass score 61, ¬final bid 17) | 3 |

Table 5.10: Example apriori result from ensamble CoTM with support over 0.01 and length more than 1.

## 5.9 Game scenario

Presented below in 5.7 is an example game where the CoTM decided what to bid. The AI is the 3. player to act, and the two other players have already bid. The current leading bid is 2. The previous round the "Next player" won the bid, and received the 10 card by paying 3 coins. The "Final player" has the same amount of coins as the AI, the 18 starting coins. The current board state has a 14, an 8 and a 7.

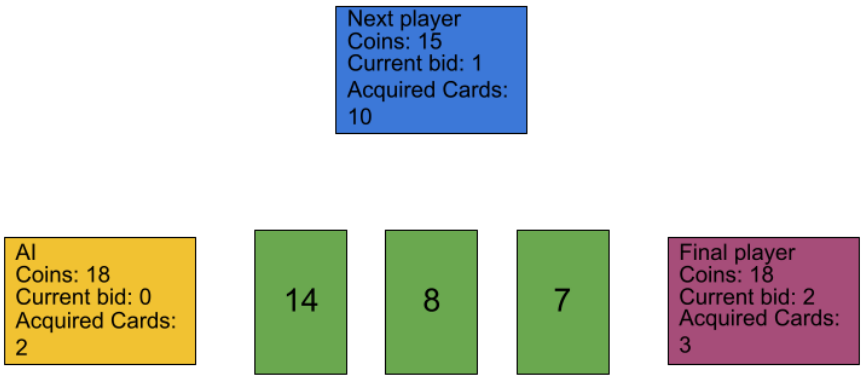



Figure 5.7: The game scenario, AI players turn to bid.

### 5.9.1 Graphical Interval Mining

Both from single CoTM and ensamble CoTM are the outputs similar, though ensamble TM output has been the chosen format due to being composed of more CoTMs and therefore giving a more accurate representation. In the paper regarding Hex, the board states already has a natural binary representation, For Sale does not have a such a natural representation, and as such this Graphical Interval Mining had to be created to easier interpret the CoTM ensamble's choices.

From Figure 5.8 the different relevant intervals can be seen.

How the graphs are analysed is as follows. The point with both the highest y value, and the highest x value on the green graphs marks where the interval starts, and the red ones with both the highest y value, and the highest x value where it stops. Not all literals have been included in the plots, as some were simply not included in the clause literals enough. From the graphs below in Figure 5.8 some of the most important details can be extracted. The highest card is registered to be value 8 or higher, but lower than 14. Next the second highest card is of value 8 or more and less than 15. Finally the last card is of at least value 7, but less than 12. Subsequently next player bid is of at least 1, the final players bid is registered to be of at least 3. Afterwards total score of each player is also displayed, where the players total score is at least 18, the next players score is at least 20, and the final player at least 19. The following pass scores are self at 23 to 49, next at 14 to 46 and final at 27 to 56. As a result the following intervals signals that the agent will not win the round if it and all subsequent players pass from this point on. To circumvent this the AI makes a choice, which is to up the current registered leading bid by 1, and bet 4.

Humans usually do not ask why a prediction was made, but would rather know why another prediction was *not* made[36]. Due to this the reasoning for not betting 6 is also shown below in Figure 5.9. One of the more striking points against betting 6 is according to the graph that the current final bid is between 2 and 16.

Through analysing the intervals, it is apparent that the encoding of the parameters / the input literals has not yet been pushed to the limits. Received output is heavily dependant on what literal names is put in. Another apparent change that needs to be made, is updating the scoring system, so that the agents focuses more on acquiring high value cards, rather than pass. Additionally the current state of the method struggles with pinpoint accuracy regarding the current game state, as can be easily seen by comparing the input state in Figure 5.7 with the two collections of graphs below.


Figure 5.8: Visualization of selected ranges after item mining the literals and negated literals from the clauses, for the example game scenario for CoTM ensemble in favor of betting 4.

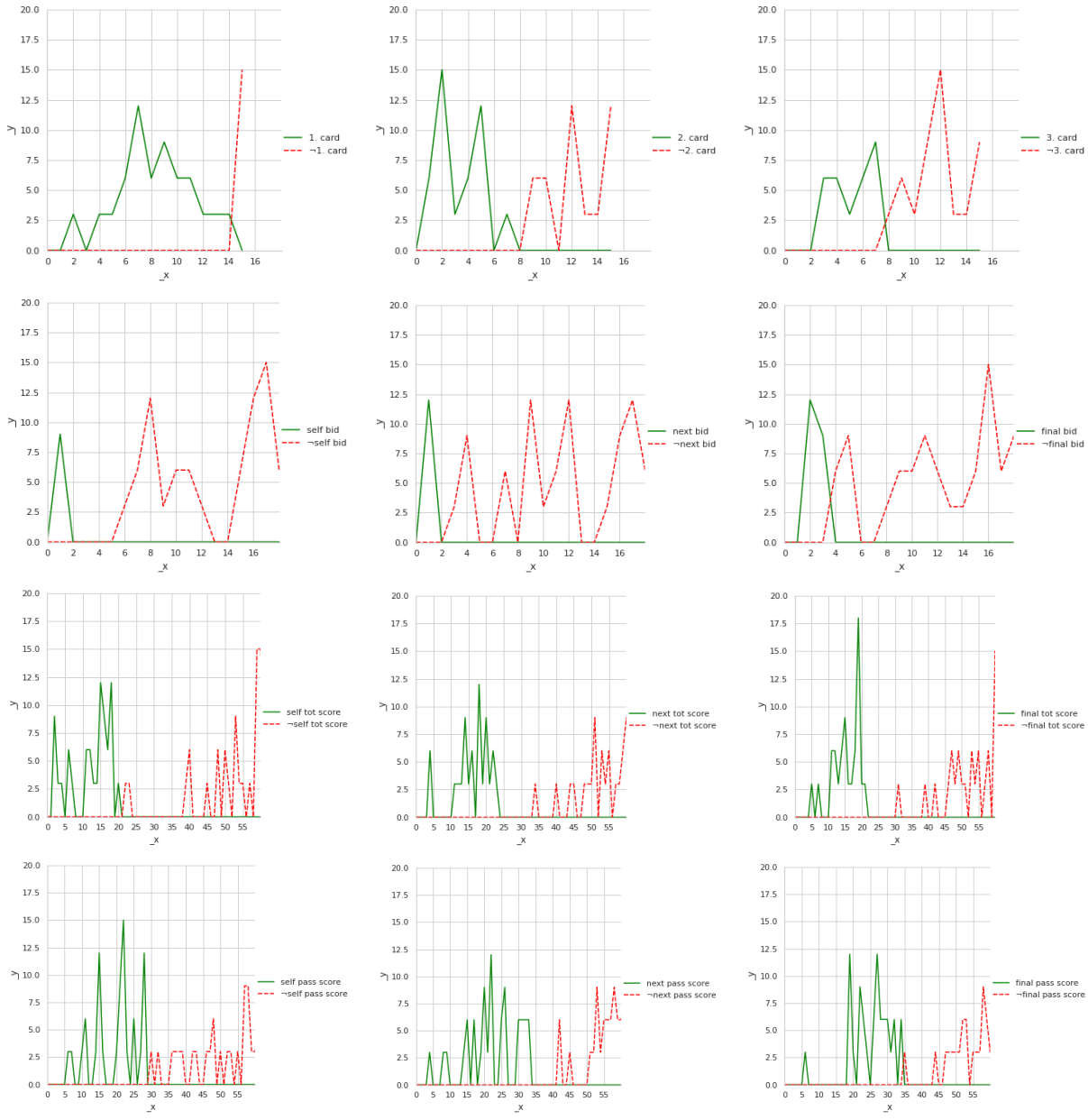



Figure 5.9: Visualization of selected ranges after item mining the literals and negated literals from the clauses, for the example game scenario for CoTM ensemble against betting 6.

# Chapter 6

# Discussions and conclusion

Chapter 6 consists of the final discussions and the conclusion of this thesis

## 6.1 Environment Related Improvements

Although the environment fully works for both the first and the second phase, it is still not optimal. Some minor tweaks and improvements may be needed.

### 6.1.1 Subsequent pass feature vs Actual pass feature

In the current version subsequent passing is used in the first round env. This could be changed to show the actual expected turn-order. As the players never move places, the one who passes first may find themselves to have either the second turn, or the third turn depending on the betting outcome of the two last players.

### 6.1.2 Scaling starting coins

Introducing scaling of the starting coins could have been a change to the environment. In turn making the experiment more similar to the original game and as the stack already scales with round size, but this was not included simply due to the sizing not having a large impact on training scaling. The starting coins for the 8 rounds is 18 coins, which would translate to roughly 3 coins per 5 cards, as there are 30 total possible cards, the cards/stack were reduced to 11 for the smallest example of 3 rounds, which would result in a starting amount of $11 * 3/5 \approx 7$ coins. The difference would at most be 2 less features (5 bits for 32 to 3 bits for 7), while the stack reduction was reduced by 1 for each number, so at the most part it was a difference of 19 features, almost 10 times the reduction.

### 6.1.3 Hidden vs not Hidden Coins and/or Cards

Information regarding the opponents total coins and cards were given to the agent by the current environment. It would have been of interest to see how changing this would affect the results, as the model used would need to be able to remember what the other players have at each state and/or find the optimal betting strategy.

## 6.2 PPO outperforming ICM to a large extent

PPO outperformed ICM with almost the same degree as how it outperformed random. Signaling poor performance from ICM. This may be credited to the "noisy-TV" problem being a challenge of a prediction-based curiosity method. And previous testing shows ICM exhaust its curiosity very quickly and stalls exploration when learning from scratch in a highly stochastic environment[37]. As the algorithm played and trained against two versions of itself, all three models started exploring at the same time, which resulted in an environment that could be classified as a random environment. In hindsight, a way to circumvent this would be to train the ICM module on already trained agents, resulting in developing a curiosity model based on "ideal" moves.

## 6.3 Extensive Hyperparameter Search

As mentioned earlier in Section 5, only a random limited hyper-parameter search has been done in the range 0.1 to 100. It would have been of interest to find out if there is a different general rule for the CoTm, than the rule from [10] where $T$ being 80% of total clauses. It would also have been of interest to find an optimal $s$ value and see the resulting accuracy after training.

## 6.4 Parameter encoding/ features

The current encoding has not yet been perfected for the game For Sale, as well as the method having issues with correctly conveying the entirely correct game state. By choosing input features that can be converted into more accurate descriptions of the game state the CoTMs would be able to give an explanation that could convey more information as well as raise the degree of description. Exploring other ways of purposefully encoding the data into propositional input could also increase the perceived correct game state and/or reasoning behind moves.

### 6.4.1 Changing Standard Deviation bit encoding to large, medium and low

In the current version some of the literals were not utilized as much as initially theorized. The standard deviation bit is one of the most surprising ones. There can be many reasons for this, but one of the main culprits are the way it is encoded into input. As of now, the current way of encoding was done though comparing the standard deviation of the board, and compare it to the theoretically largest standard deviation possible. If the standard deviation of the board is in the upper 50% then it is deemed large and active. An approach that has the potential to work better, is by dividing the standard deviation into 3 different gravities, large, medium and low. Resulting in an encoding divided into 33% segments, as opposed to only two segments.

## 6.5 Readability and state accuracy of the intervals of the plots

Interval readability could use some work, as they do not convey the result in the most accurate or readable way possible. However they do convey the information in a slightly more human-readable way than the apriori, top weighted and naive examples presented earlier does for this complex space. The readability and accuracy are linked heavily to the features put into the CoTM, and again, using features that represent the desired output format better, are highly likely to increase the readability.

## 6.6 Revisiting research questions and goals

In section 1.4 a set of research questions was presented. This section will review those questions in relation to the results.

### 6.6.1 Can we through the use of the CoTM explain how to play For Sale?

A complete explanation of how to play For Sale is not yet possible, but some reasoning for game states can be extracted.

### 6.6.2 Can we explain the behavior of the self learned algorithm?

How the self learned algorithm behaves, can be explained to some extent, through the use of game states. Although not presenting pinpoint accuracy, and still in need of more refinement.

### 6.6.3 How does the CoTM compare to other interpretable algorithms?

Considering the game state complexity of For Sale, the CoTM performed better than the interpretable algorithms CoTM was tested against. The interpretability of the CoTM through the methods described in this thesis have not been perfected, but all things considered the performance was somewhat satisfactory.

## 6.7 Future work

Even though the thesis has come to and end, there is still more work to be done, some ideas for ways to improve the project will be presented.

### 6.7.1 Further experiments with different input features

As previously mentioned the input features utilized can be improved though change of representation or by removing or adding other variables that could prove to increase the interpretability of the output. Tweaking features that did not work as intended, such as the std bit, or the stack cards could increase the readability of the array of graphs. Adding new features such as if the player is leading by a lot, is even with players or behind could also

make it easier to interpret the results and get a better explaination of why the CoTM chose an action over another.

### 6.7.2 Bigger TM Ensemble

Currently the ensemble in this thesis consists of 3 CoTMs, and different size has not been experimented on yet. Using an increasing amount of CoTMs could possibly increase the readability by properly filtering out more of the noise.

### 6.7.3 Other Potentional Black Box Algorithms

The current black box algorithm used showed great variability in terms of performance as the ICM algorithm performed very much so less than optimal compared to PPO algorithm. Comparing PPO with other black box algorithms could prove to give a better pointer on how it performed. In addition giving more credibility to the data that the CoTM was trained on. Other algorithms that were considered for parameter sharing together with the backbone network and not tried was DQN and A3C[38][39]. This was due to more time and focus being directed towards explainability from the CoTM, rather than a potential increase or decrease in the training data.

### 6.7.4 Using other Multi-Agent Methods

In this thesis parameter sharing was used as a means of training the agents in the multi-agent environment. Another considered way of training was through fully independent methods, where each agent have their own policy. Combining other multi-agent methods with a larger array of tried algorithms could prove to increase the ability of the agent playing For Sale.

### 6.7.5 Training on the Full 8 Rounds

This thesis only trained the agent on the 3-round and 4-round version. The original game consists of 8 rounds, and as such the next steps would be an analysis of the full 8 rounds of the game after a more descriptive interpretation of why certain moves should be made and when is made possible.

### 6.7.6 Train a black box model on the second round

The next step after being able to play the first round, is to train a model on the second round of For Sale as well as using the CoTM to describe why the moves have been made with the improved methods of interpretation resulting in the steps above.

### 6.7.7 Finish the whole For Sale environment with trained black box agent

Finally the For Sale environment would need to be completed by tying together the two now separate rounds to a completed game. Then either train an AI on each round, and then couple the AI of each round together in a pipeline, or train a single AI on the entire game.

## 6.8 Conclusion

Concluding this thesis, an environment for the first round of the game For Sale has been created. Compared to each other, the best of the trained black box models generated winning game data for both the 3-round and the 4-round version. The winning game data was then encoded in such a way that the Coalesced Tsetlin Machine was able to be trained, and presented better accuracy than other popular machine learning algorithms. Using a naive example, top weighted clauses, frequent item mining and a graphical view, the interpretability of the Coalesced Tsetlin Machine was tested, though far from perfect, the graphical view provides a small step towards a more interpretable AI trained on complex games.

# Bibliography


[1] David Silver et al. "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm." In: *CoRR* abs/1712.01815 (2017). arXiv: 1712.01815. URL: http://arxiv.org/abs/1712.01815.

[2] Matej Moravcık et al. "DeepStack: Expert-Level Artificial Intelligence in No-Limit Poker." In: *CoRR* abs/1701.01724 (2017). arXiv: 1701.01724. URL: http://arxiv.org/abs/1701.01724.

[3] David Silver et al. "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm." In: *CoRR* abs/1712.01815 (2017). arXiv: 1712.01815. URL: http://arxiv.org/abs/1712.01815.

[4] Christopher Berner et al. "Dota 2 with Large Scale Deep Reinforcement Learning." In: *CoRR* abs/1912.06680 (2019). arXiv: 1912.06680. URL: http://arxiv.org/abs/1912.06680.

[5] Isabela Granic, Adam Lobel, and Rutger C. M. E. Engels. "The benefits of playing video games." In: *American Psychologist* 69.1 (Jan. 2014), pp. 66–78. DOI: 10.1037/a0034857. URL: https://doi.org/10.1037/a0034857.

[6] Tara Kost Scanlan and Rebecca Lewthwaite. "Social Psychological Aspects of Competition for Male Youth Sport Participants: IV. Predictors of Enjoyment." In: *The Journal of Sport Psychology* 8 (1986), pp. 25–35.

[7] Marco Túlio Ribeiro et al. ""Why Should I Trust You?": Explaining the Predictions of Any Classifier." In: *CoRR* abs/1602.04938 (2016). arXiv: 1602.04938. URL: http://arxiv.org/abs/1602.04938.

[8] Scott M. Lundberg and Su-In Lee. "A unified approach to interpreting model predictions." In: *CoRR* abs/1705.07874 (2017). arXiv: 1705.07874. URL: http://arxiv.org/abs/1705.07874.

[9] Thomas McGrath et al. "Acquisition of Chess Knowledge in AlphaZero." In: *CoRR* abs/2111.09259 (2021). arXiv: 2111.09259. URL: https://arxiv.org/abs/2111.09259.

[10] Charul et al. Giri. *Logic-based AI for Interpretable Board Game Winner Prediction with Tsetlin Machine*. 2022. DOI: 10.48550/ARXIV.2203.04378. URL: https://arxiv.org/abs/2203.04378.

[11] Cynthia Rudin. "Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead." In: (2018). DOI: 10.48550/ARXIV.1811.10154. URL: https://arxiv.org/abs/1811.10154.

[12] Stefan Dorra. *Stefan Dorra's For Sale The Game of Property and Prosperity*. Published by Eagle-Gryphon games. 2008.

[13] Vijay Krishna. *Auction Theory*. Academic Press, 2010. ISBN: 9780123745071.

[14] Jing Tan, Ramin Khalili, and Holger Karl. *Learning to Bid Long-Term: Multi-Agent Reinforcement Learning with Long-Term and Sparse Reward in Repeated Auction Games*. 2022. DOI: 10.48550/ARXIV.2204.02268. URL: https://arxiv.org/abs/2204.02268.

[15] Justin K. Terry et al. "PettingZoo: Gym for Multi-Agent Reinforcement Learning." In: *CoRR* abs/2009.14471 (2020). arXiv: 2009.14471. URL: https://arxiv.org/abs/2009.14471.

[16] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. The MIT Press, 2020.

[17] Lucian Busoniu, Robert Babuska, and Bart De Schutter. "A Comprehensive Survey of Multiagent Reinforcement Learning." In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38.2 (Mar. 2008), pp. 156–172. DOI: 10.1109/tsmcc.2007.913919. URL: https://doi.org/10.1109/tsmcc.2007.913919.

[18] Deepak Pathak et al. "Curiosity-driven Exploration by Self-supervised Prediction." In: *CoRR* abs/1705.05363 (2017). arXiv: 1705.05363. URL: http://arxiv.org/abs/1705.05363.

[19] Rich Caruana and Alexandru Niculescu-Mizil. "An empirical comparison of supervised learning algorithms." In: *Proceedings of the 23rd international conference on Machine learning*. 2006, pp. 161–168.

[20] Greg Brockman et al. "OpenAI Gym." In: *CoRR* abs/1606.01540 (2016). arXiv: 1606.01540. URL: http://arxiv.org/abs/1606.01540.

[21] Philipp Moritz et al. "Ray: A Distributed Framework for Emerging AI Applications." In: *CoRR* abs/1712.05889 (2017). arXiv: 1712.05889. URL: http://arxiv.org/abs/1712.05889.

[22] Eric Liang et al. "Ray RLLib: A Composable and Scalable Reinforcement Learning Library." In: *CoRR* abs/1712.09381 (2017). arXiv: 1712.09381. URL: http://arxiv.org/abs/1712.09381.

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

[24] John Schulman et al. "Trust Region Policy Optimization." In: *CoRR* abs/1502.05477 (2015). arXiv: 1502.05477. URL: http://arxiv.org/abs/1502.05477.

[25] Justin K. Terry et al. "Parameter Sharing For Heterogeneous Agents in Multi-Agent Reinforcement Learning." In: *CoRR* abs/2005.13625 (2020). arXiv: 2005.13625. URL: https://arxiv.org/abs/2005.13625.

[26] Ramakrishnan Srikant. *Fast algorithms for mining association rules and sequential patterns*. The University of Wisconsin-Madison, 1996.

[27] John Nash. "Non-cooperative Games." In: *Annals of Mathematics* 54 (2 1951), pp. 286–295.

[28] Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. "The Complexity of Computing a Nash Equilibrium." In: *SIAM Journal on Computing* 39.1 (2009), pp. 195–259. DOI: 10.1137/070699652. eprint: https://doi.org/10.1137/070699652. URL: https://doi.org/10.1137/070699652.

[29] K. S. Narendra and M. A. L. Thathachar. *Learning Automata – An Introduction*. Englewood Cliffs, NJ: Prentice Hall, 1989.

[30] Ole-Christoffer Granmo. "The Tsetlin Machine - A Game Theoretic Bandit Driven Approach to Optimal Pattern Recognition with Propositional Logic." In: *CoRR* abs/1804.01508 (2018). arXiv: 1804.01508. URL: http://arxiv.org/abs/1804.01508.

[31] K. Darshana Abeyrathna. "Novel Tsetlin Machine Mechanisms for Logic-based Regression and Classification with Support for Continuous Input, Clause Weighting, Confidence Assessment, Deterministic Learning, and Convolution." PhD thesis. University of Agder, 2022.

[32] Sondre Glimsdal and Ole-Christoffer Granmo. "Coalesced Multi-Output Tsetlin Machines with Clause Sharing." In: *CoRR* abs/2108.07594 (2021). arXiv: 2108.07594. URL: https://arxiv.org/abs/2108.07594.

[33] Yacine Izza, Alexey Ignatiev, and João Marques-Silva. "On Explaining Decision Trees." In: *CoRR* abs/2010.11034 (2020). arXiv: 2010.11034. URL: https://arxiv.org/abs/2010.11034.

[34] Ana Elsa Hinojosa Herrera, Chris Walshaw, and Chris Bailey. "Improving Black Box Classification Model Veracity for Electronics Anomaly Detection." In: *2020 15th IEEE Conference on Industrial Electronics and Applications (ICIEA)*. 2020, pp. 1092–1097. DOI: 10.1109/ICIEA48937.2020.9248258.

[35] Vanessa Buhrmester, David Münch, and Michael Arens. "Analysis of Explainers of Black Box Deep Neural Networks for Computer Vision: A Survey." In: *CoRR* abs/1911.12116 (2019). arXiv: 1911.12116. URL: http://arxiv.org/abs/1911.12116.

[36] Christoph Molnar. *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable*. 2nd ed. 2022. URL: https://christophm.github.io/interpretable-ml-book.

[37] Xiaogang Ruan et al. "End-to-end autonomous exploration with deep reinforcement learning and intrinsic motivation." en. In: *Comput. Intell. Neurosci.* 2021 (Dec. 2021), p. 9945044.

[38] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning." In: *CoRR* abs/1312.5602 (2013). arXiv: 1312.5602. URL: http://arxiv.org/abs/1312.5602.

[39] 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.