TDQL In Finding Optimal Strategy For Simplified Single Player Gin Variant

Riley Roberts
COMP3106

Contents

1 Introduction 1
1.1 Background 1
1.2 Prior Work 2
1.3 Objective Statement 2

2 Methods 2
2.1 Reinforcement Learning Architecture 2
2.2 Validation Strategy 2

3 Results 3
3.1 Training 3

4 Discussion 4
4.1 Limitations 4
4.2 Future Work 4

1 Introduction

1.1 Background

Upon learning that there was the freedom to pick a topic for the final project, it was immediately apparent which direction I would go in. I grew up playing the card game Gin with my father and would try and apply what I have learned from the artificial intelligence course to make the most advanced gin actor I have the knowledge for. For those unaware of how the original game is played, the following is a brief synopsis of how the game is played. Gin rummy is a two-player adversarial card game with numerous variants depending on tradition and region. Where I come from in southern California (yes there are differences in northern California) one player starts as the dealer and deals the opponent ten cards and themselves ten in alternating order. The rest of the cards are placed between the players as the draw pile and the player with 11 cards is the first to discard. Then the players alternate getting to choose whether they want to pick up the top discarded card or pick from the draw pile and then picking a card to throw. A player wins when they fit all of their cards into a meld with none left over. A meld is when at least 3 cards of the same suit are in numeric order or when you have at least 3 cards of the same number. Commonly one wins with 3 melds; however, you can also win with 2 or even one extremely large meld. There is also a relatively complex scoring system between hands however for the purposes of this project we will not be touching on that.
After reading that synopsis on how to play one may come to the realization that this is an incredibly complex card game to solve algorithmically. This is most apparent when you calculate the number of states, 52! of which to be precise. In addition to this there is also the element of hidden information since the other player's hand is not known to you. This is why, as this report will get into in the next section, there has been no artificial intelligence algorithm to solve the game. In order to tackle this monumental problem I decided working with a simplified version of the original game. In order to curb the hidden information I made it a single player experience where the goal is simply to get gin before the draw deck runs out. This still leaves us with the state space that has more combinations than there are atoms on earth[4] so to cut down we will be limiting the deck to 3 suits (Spades, Hearts and Clubs) and only the cards 2-8 inclusively. Now we only need $\binom{24}{11} = 2,496,144$ different states with 11 actions for each. While computationally expen-

sive this is within the scope of this project.

1.2 Prior Work

As mentioned previously, there have been attempts to work on Gin rummy in various ways, most popularly Hao and Vayisberg achieved a 67% win-rate using separate dynamic algorithms for discarding and drawing in addition to estimating opponent hands[1]. Also in 2021 was the paper from Princeton University where they were able to achieve 61% win rate using random forests[2]. This project will be set apart from those projects as instead of trying to partially solve the entire game of gin, this project achieves a substantially higher win rate on a simplified game of gin, meaning that these are now two different problems that are being solved.

1.3 Objective Statement

To summarize above, the objective of this project is to use Q-learning in temporal difference to find the optimal strategy in a simplified single player variant of the classic card game Gin.

2 Methods

2.1 Reinforcement Learning Architecture

The chosen method for this project was Temporal Difference Q-Learning. The rationale for this is due to the relatively reduced state space, it is feasible for TDQL to converge on optimal strategy as discussed in class. There are multiple drawbacks to this approach however, mainly the computation required to visit each state action pair an appropriate number of times. With over 2 million states and 11 actions each that give us a Q-Table with around 20 million entries. Simply loading the Q-Table into memory once initiated takes on average 180 seconds.
The implemented reward function gave a reward of +10 for winning states and a reward equal to the number of melds - 4. This approach rewarded increasing the number of melds in your hand as the higher the number of melds the closer you are to a winning state. The reasoning for not simply rewarding number of leftover cards was due to the fact that large melds can be suboptimal for closing the final gap between close to a winning state and actually being a winning state. For example, say you have a long meld of 2,3,4,5,6 and a second meld of 2,3,4,5 of a different suit. In this example there are only 2 cards that will take you into a winning state while any combination of 3 melds would have at minimum 3 cards that could take you to a winning state. The policy used a combination of an exploration function as well as greedy epsilon. The exploration function was implemented in order to ensure each state action pair was visited a suitable number of times to begin setting q-values. However this method can be prone to local maxima so greedy epsilon was implemented as well to minimize this effect. While the number of times a state is visited is less than the specified amount, in this project 11 (once for each following action), the exploration function is used. Once the number is past 11 the policy visits the highest Q-Value action with 1-ϵ probability and a random action with ϵ probability. Due to the scale of the Q-Table a decently high ϵ was chosen at 0.1 to give a 10% chance of exploration.

2.2 Validation Strategy

There were two main validation strategies that were chosen for this project. The two strategies are total rewards and win percentage. In both cases these categories are tested against a random actor. In the battle function inside the arena file the random actor and the TDQL algorithm are both set to play a certain number of games and then the win rates and total rewards are outputted to the console.

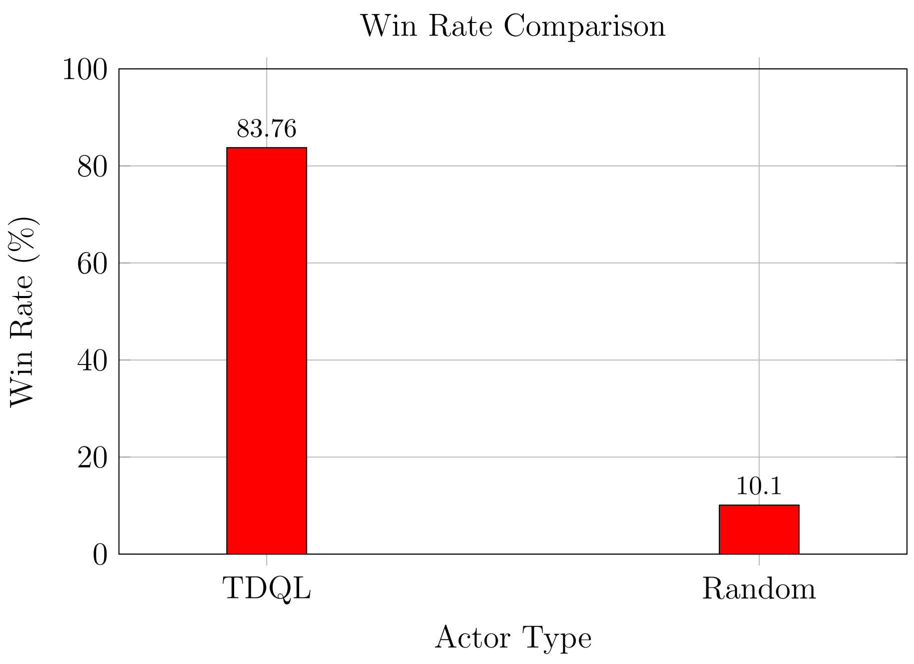
3 Results

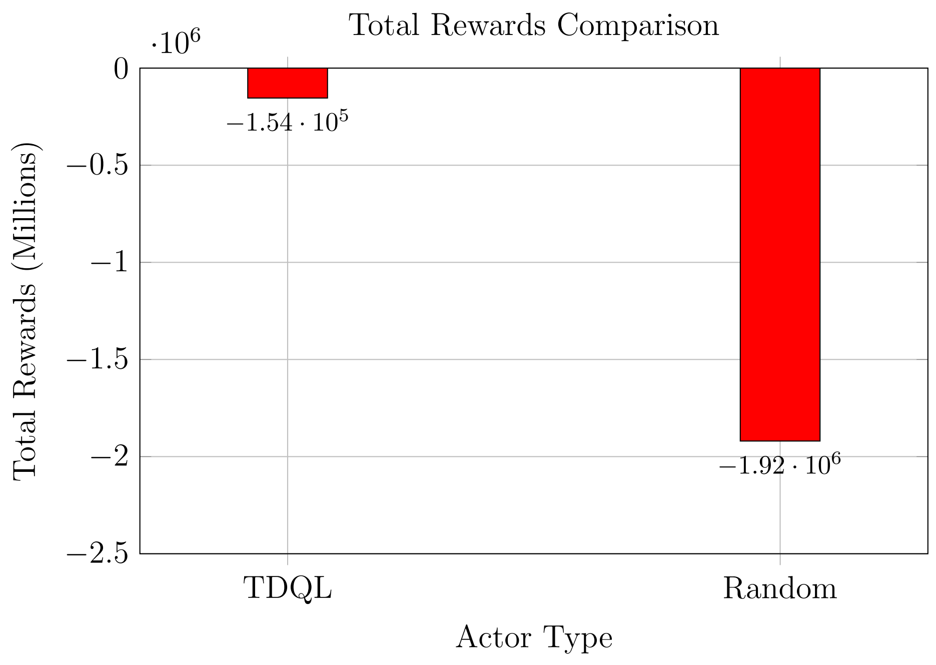
3.1 Training

The model was trained for 2,100,030,000 epochs. After the training, an arena function was implemented in order to test the algorithm against a random actor. For validation purposes the total number of episodes for each actor was 10,000. Below is the table showcasing the results.

Method	**TDQL**	**Random Actor**
Total Rewards	-154,029	-1,919,721
Win Rate	83.76%	10.098%

Table 1: TDQL vs. Random Actor in Rewards and Win Rate





As seen above, by implementing temporal difference Q-learning we achieve a substantially higher win rate over taking random actions, winning over 4/5 games that it plays in. The random actor on the other hand wins only 1/10 games. Moreover, this drastic improvement can also be seen in the total rewards obtained by both actors. While the random actor receives almost 2 million negative rewards by the end of the ten thousand episodes, the TDQL algorithm achieves a value over ten times better at -154,000 meaning that it visit states closer to winning at a significantly higher rate.

4 Discussion

4.1 Limitations

While the results are significant, there are a multitude of drawbacks to the current implementation. The most apparent drawback is that we are working with a partial deck due to the computing power and storage that would be necessary to hold 52 choose 11 different states. This is unideal for a project originally designed to find optimal strategy in the classic version of gin. Additionally, the current implementation also does not contain the second player and therefore would most likely perform poorly in a true gin environment.

4.2 Future Work

The next course of action for progressing optimal strategy in gin performance and downsizing the limitations of this project will be to implement a form of counterfactual regret minimization. Counterfactual regret minimization, or CFR for short, recently been shown to be particularly affective in another two-player adversarial game with hidden information, poker. Specifically, Monte Carlo CFR was recently shown to converge efficiently in large state spaces such as poker and bluff[3]. If this avenue is to be followed, more research should be done into the advantages and disadvantages between Monte Carlo CFR vs Deep CFR.

References

[1] Y. Hao and M. Vaysiberg. Dynamic strategies and opponent hands estimation for reinforcement learning in gin rummy game. In K. Arai, editor, *Intelligent Systems and Applications*, volume 294 of *Lecture Notes in Networks and Systems*. Springer, Cham, 2022. IntelliSys 2021.

[2] A. Hein, M. Jiang, V. Thiyageswaran, and M. Guerzhoy. Random forests for opponent hand estimation in gin rummy. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(17):15545–15550, 2021.

[3] Michael Johanson, Nolan Bard, Marc Lanctot, Richard G. Gibson, and Michael Bowling. Efficient nash equilibrium approximation through monte carlo counterfactual regret minimization. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 837–846, February 2012.

[4] Dennis Weisenberger. Questions and answers - how many atoms are there in the world? n.d.