# Why the House Always Wins:

## Playing Card Games with Deep Learning and Reinforcement Learning

Final Report

COMP4801 Final Year Project

UID: 3035238565


| Student | Supervisor |
| --- | --- |
| Ish Handa | Professor Francis C.M. Lau |


April 2019

# Abstract

Artificial Intelligence has always had a long history with games. This project seeks to improve our understanding of popular card games played in Casinos using Reinforcement Learning and Deep Learning. The project aims to achieve so, to expand our knowledge of the strategies that are used in Blackjack and Texas Hold'em Poker. Ultimately, helping us understand how we can maximize our profit strategies, why the house always comes out on top and how machines and algorithms can perform complex strategic decisions that surpass human capabilities. The first game the project investigates is Blackjack. The aim is to study and explore various strategies learned by various reinforcement learning algorithms namely, Temporal Difference Learning, Monte Carlo methods, Deep Q Network, and its variants, to improve our understanding of the optimal strategies involved in order to maximize our overall earnings. The next game the paper will tackle, Texas Hold'em Poker, would take our understanding of the algorithms implemented previously further, to build a smart Poker AI, and perform an analysis to understand how to learns against different kinds of Poker players to improve our understanding of machines that can perform complex strategic decisions.

# Acknowledgments

I would like to offer my special thanks to my supervisor Professor Francis Lau from the Department of Computer Science for his guidance in this project. I also wish to Acknowledge Julian Chase from Center for Applied English Studies for suggestions on drafting this paper.

# Contents

# List of Figures



# List of Tables

# Abbreviations

| | |
|---|---|
| **AI** | Artificial Intelligence |
| **RL** | Reinforcement Learning |
| **MDP** | Markov Decision Process |
| **MC** | Monte Carlo |
| **TD** | Temporal Difference |
| **DQN** | Deep Q Networks |
| **DDQN** | Double Deep Q Network |
| **CNN** | Convolutional Neural Network |

# 1. Introduction

Reinforcement Learning is a form of Machine Learning and a branch of Artificial Intelligence- inspired by behavioral psychology. It originated from a science experiment studying how animals learn tasks by receiving rewards. In computer science, reinforcement learning is an approach to understand and automate goal-directed learning and decision making, therefore it is often referred to as the science of making optimal decisions. Formally, in RL, a software agent seeks to learn an optimal policy $\pi^*$ by interacting with the environment. At each time step t, the agent receives a representation $S_t$ of the environment' states, choosing an action $A_t$, to be executed in the environment, yielding the representation of the successor state **St** and a reward signal $R_{t+1}$. The optimal policy aims to maximize the cumulative reward of all-time steps $G_t$.

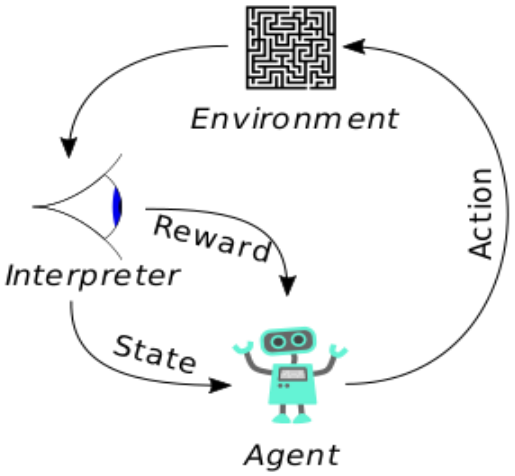



Figure 1: An illustration of a Reinforcement Learning

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum \gamma^{k-1} R_{t+k}$$

where the discount factor $0 \leq \gamma < 1$ is modeled to bound the total reward by exponentially decaying it. To further understand the agent-environment interactions, they are modeled as MDP Markov Decision Process which is explained in Section 3.1. Almost all RL problems can be formalized as MDPs. Various RL algorithms are used to solve such MDPs.

A popular application of RL algorithms is to play games. RL and games share a long beneficial saga. The game of Backgammon was the first success, where the RL algorithm surpassed the top human players. Since then, RL has been applied to many other games. Not all games have been cracked or solved but the subset is increasing as multiple variations of state of the art RL algorithms are being developed and tested.

There are two major reasons why such a history has formed. Firstly, it is relatively cheaper to train simulations as compared to the physical world. For example, teaching a robot to walk would face constant expensive maintenance costs while training. Thus, the simulations server as

a suitable platform for benchmarking and comparing various RL algorithms. Second, studying games have often been used to learn about human intelligence and the challenges faced by us. They have proven to a challenging task for RL algorithms as well as the games become more complex similar to us humans.

## 1.1 Aim and Structure

The goal of this project is to improve our understanding popular card games played in Casinos, a previously solved game, Blackjack and a complex game with present AIs at par human level, Texas Hold'em Poker.

For Blackjack, the project involves comparing and investigating the various state of the art reinforcement learning algorithms to see how they hold up against the proposed strategy by Edward Thorp. As a result, the paper explores, Monte Carlo simulation, on/off-policy learning, Deep Q Network, and its variants to simulate the game of Blackjack. The results of this paper could provide a solid basis to show the progress made by reinforcement learning algorithms as well as providing a better understanding of the previously explored methods and their pertinence.

For Texas Hold'em Poker, the project, learns from the results of Blackjack as well as other sources, to build a custom DQN based AI agent for the game and perform an analysis to understand how to learns against different kinds of Poker players, namely maniacs, rocks, calling stations and killers. The results of this paper could help improve our understanding of how machines and algorithms can perform complex strategic decisions that surpass human capabilities.

Section 2 of the report introduces the concept of MDPs and further introduces many RL techniques used to solve them such as Monte Carlo, on/offline policies and Deep Q Network and its variants which would be implemented in this project. Section 3 introduces both the game and their rules as well as the benchmarks/analysis reference points used for this report. Section 4 compares and evaluates the results of the various RL algorithms developed for Blackjack and gives a detailed description of the architecture used for Poker and its analysis. Section 5 details the phases of the project, along with the progress of the project, and finally, Section 6 concludes the report by a brief discussion of the casino card games and the project.

# 2. Background

## 2.1 MDP

The Markov property states that "the future is independent of the past given the present". Meaning that the state encapsulates all relevant information from the history and once a state is known, the history can be disregarded. Mathematically, the property states that a state $S_t$ is Markov if and only if $P [S_{t+1} | S_t ] = P [S_{t+1} | S_1, ..., S_t ]$. A Markov chain is a sequence of possible states satisfying the Markov property; further, Markov Decision Processes are extensions of Markov chains which involve multiple such decisions or actions. Figure 4 shows an example of such an MDP for a course, wherein a student could be in different states (class 1,2,3, Facebook, sleep, pub or pass) and the terminal state being to sleep. Each student can perform various actions to go to a different state of their choice (with a certain probability) and to achieve certain reward points. An episode (a series of events for a student) would end when they reach the sleep state with the total points, they would have with them at that moment. The map of all the states, actions, transition probabilities and rewards points can be referred to as an MDP.

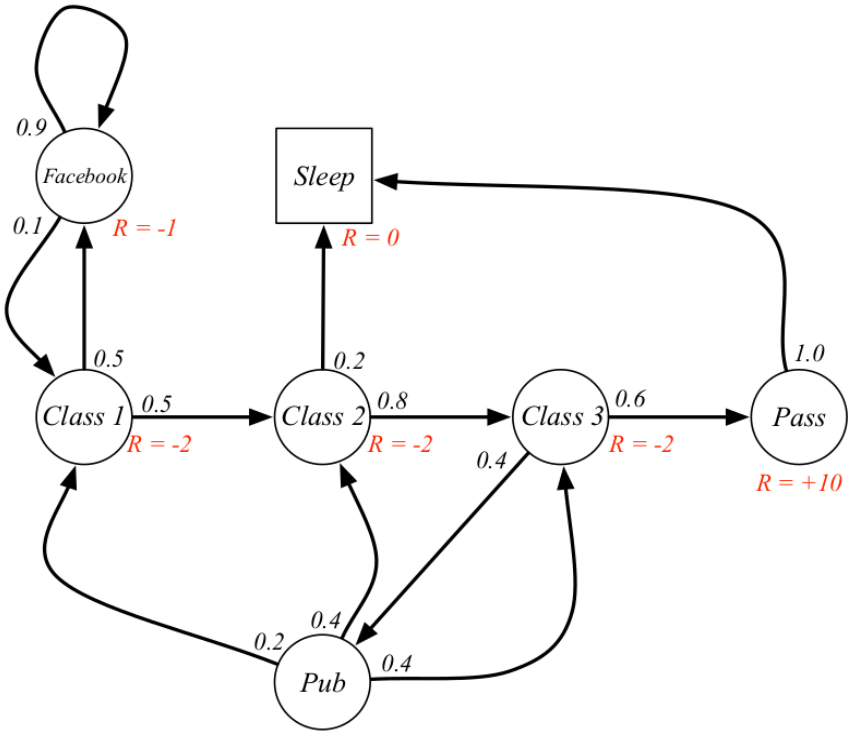



Figure 2: Student MRP from David Silver class

Formally, an MDP can be defined as a tuple $\langle S, A, P, R_y \rangle$ where

1. S is a finite set of states
2. A is a finite set of action
3. P is a state transition probability matrix, $P_{a,s} = P[S_{t+1} = s_0 \mid S_t = s, A_t = a]$
4. R is a reward function, $R_{a,s} = E[R_{t+1} \mid S_t = s, A_t = a]$
5. γ is a discount factor $\gamma \in [0, 1]$

To solve MDPs, there many RL fundamental techniques such as value and policy-based iteration algorithms. These algorithms do not always assume that the agent knows the MDP model beforehand and rely on experience to determine so.

Before we discuss some of the techniques that will be implemented in this report, we need understand the concept of Q-value of a state-action pair (s,a), denoted by Q*(s,a). It is the sum of discounted future rewards the agent can expect on average after reaching state s and choosing action a, before seeing the outcome of this action and It can be formulated as follows,

$$\mathbf{Q^*(s,a) = \max \mathbb{E}\,[r_t + \gamma\, r_{t+1} + \gamma^2\, r_{t+2} + \gamma^3\, r_{t+3} + \ldots.\,]}$$

## 2.2 Monte Carlo

Monte Carlo methods use the concept of averaging sample returns. The basic principle behind this algorithm is using randomness to solve problems that might be deterministic in nature. In RL, MC can be used to approximate policies to solve MDPs.

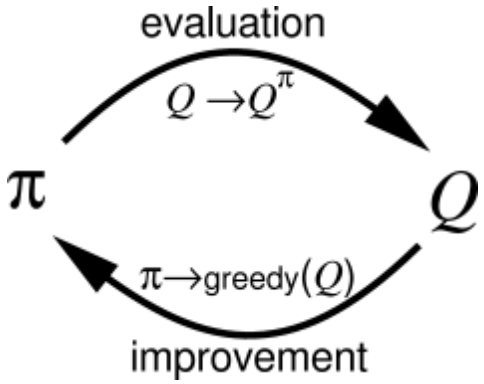



Figure 3: Monte Carlo iteration cycle

As shown in figure 3, MC is a two-step cycle, the first step being the MC policy evaluation, which updates the approximate Q-value function to approach the true function asymptotically using random sampling of multiple experiences of the simulation. The second step is the MC policy improvement, where the algorithm, starting from an arbitrary policy improves to converge to an optimal policy by making the policy greedy with respect to the current Q-value function. To say mathematically, $\boldsymbol{\pi(s) = \arg\max Q(s,a)}$.

The following is known to converge by the simple theorem stating each $\pi_{k+1}$ is uniformly better than $\pi_k$, unless it is equal to $\pi_k$, which implies both are optimal policies. Thus, ensuring that MC will converge to an optimal policy and an optimal Q-value function by sampling.

## 2.3 Temporal Difference Learning

Similar to MC methods, TD methods also learn directly from raw experiences without a model or any previous knowledge of the environment's dynamics. There are two approaches to learn a policy: on and off-policy learning. To illustrate the difference between the two, let us discuss the biggest and the most basic problem faced by RL algorithms: exploitation vs exploration problem.

### 2.3.1 Exploitation vs Exploration

The problem faced by RL algorithms is to update the Q-value function to reach its reach value in as fewer updates as possible. The question of exploitation and exploration rises to achieve the above; Should the algorithm spend more time exploring, unknown regions of the MDP or should it focus on updating the Q-values faster with the information it already holds.

To solve the exploration-exploitation dilemma, most algorithms adopt an epsilon-greedy methodology. In particular, the agent picks actions uniformly at random with probability ε and greedily with probability 1 - ε. To encourage exploration during the start of training and exploitation near the end of training, epsilon is set to 1 and linearly annealed to 0.1 over the training.

### 2.3.2 Q-Learning: Off-policy

In Offline Learning Q-value function directly approximates to optimal by acting randomly and independent of the policy being followed as long as it focuses on exploring the environment thoroughly. The Q-value function is updated according to,

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[ r_{t+1} + \gamma \max Q(s_{t+1}, a_t) \right]$$

An off-policy learning algorithm is proven to learn the optimal policy even if it is acting randomly although it might take more time to do so.

### 2.3.3 SARSA: On-policy

State-Action-Reward-State-Action involves learning online, implying the agent learns the value of the policy the agent is actually carrying out, i.e. also during the exploring stage. The online policy updates the Q-value function after each step using the following,

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[ r_{t+1} + \gamma\, Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

SARSA will find a policy that is optimal, by taking into account the exploration step that is inherited in the policy.

## 2.4 Deep Q-Networks

With the introduction of Deep Learning, neural networks help us approximate the Q-value function, leading to the introduction of DQN algorithm proposed by Google DeepMind as modeled in figure 5 for an Atari game. Approximating Q-value function help learn different state-action pairs together, improving the speed at which Q-value table is learned. As a result, the replacement of the Q-value table by a Neural Network. This nonlinear function approximator aims to minimize the loss function,

$$L_i(\theta_i) = E_{(s,a,r,s0)\sim U(D)} \left[ \left( r + \gamma \max Q(s', a' ; \theta'_i \right) - Q(s, a ; \theta_i) \right)^2 \right]$$

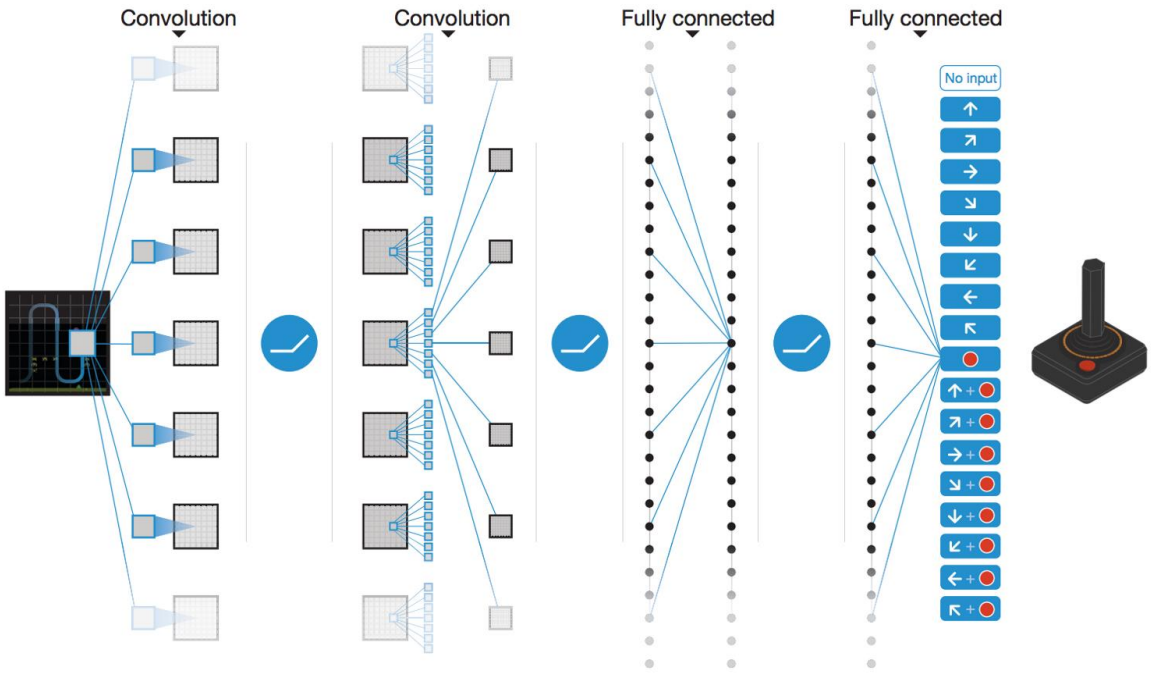



Figure 4: DQN network with a Convolution Neural Network illustrating Q function approximation

However, the introduction report recognized that by simply adding a Deep neural net based function approximator lead to its instability in converging to the optimum solution. They then went ahead to describe 3 techniques to overcome this problem.

### 2.4.1 Experience Replay

Experience Replay was first proposed by long-ji in 1993 which mentioned that DNN was often overfitting and therefore unable to produce various experiences. The paper then suggested the concept of storing past experiences which included state transitions, actions, and rewards i.e. the building blocks of Q Learning to overcome this problem. This helped reduce the correlation between experiences while updating the DNN and involved the reuse of older transitions to avoid catastrophic forgetting, a problem commonly faced by neural nets. Finally, Experience Replay also involved updating the DNN in mini-batches which helped increase the learning speed.

### 2.4.2 Target Network

The algorithm updates as $Q(s,a) \rightarrow r + \gamma \max_a Q(s',a)$ on each step. We can observe that the target depends on the last update and changes after every update, like a dog chasing its tail that every time the dogs moves so does the position of the target i.e. the tail. The dog eventually starts spinning in circles unable to catch its tail, indicating that it is never able to reach the target. Similarly, DNNs with updating targets often fail to converge, leading to oscillations around the optimal solution. To overcome this problem, the DeepMind's paper suggested the use of separate target networks. A target network is a previous network that is fixed (in time) and is updated after certain updates (generally thousand steps). This gives the DNN enough time to converge to the saved target network. The new update state becomes $Q(s,a) \rightarrow r + \gamma \max_a \hat{Q}(s',a)$ where $\hat{Q}(s',a)$ represents the target network.

### 2.4.3 Reward Clipping

Different Environments have different reward scales. for example, in a game of pong, upon winning the round the player is awarded 1 point, whereas in a paceman, the points are the sum of pellets collected before being caught by a ghost or completing the level. This variation in the scoring could make the training unstable. The DeepMind's paper suggested a very simple re-scaling solution i.e. to clip rewards to +1 for positive rewards and -1 for negative rewards which improved the stability of the algorithm.

## 2.5 Variations of DQN

### 2.5.1 Prioritized Action Replay

One of the improvements mentioned in [9] was to understand the way experience is used. [10] introduced PER, a strategy that adjusts its samplings distribution, by treating all samples differently. It does so, as some samples are not the same, we can learn or extract more information from some as compared to others. Priority is given to those samples which are of more use, being the ones that do not fit well into the current Q function estimations. In simple terms, it can be explained by saying when we encounter a new scenario, we think about it again

and again and adjust the model accordingly until it fits. Results from the [10] showed that DQN with PER outperformed DQN in 41 out of 49 Atari 2600 games.

### 2.5.2 Double DQN

One limitation of the DQN was the overestimation of Q values during training by DQN due to the max used to set targets. This resulted in overly optimistic policies. [8] introduced Double Learning, where two Q functions are learned independently. One is used to determine maximizing action and other for estimating its value. The paper showed that this method was able to eliminate the maximization bias. The new target formula for Double DQN is

$$\mathbf{Q(s,a) \rightarrow\ r + \gamma\ \bar{Q}(s', \max_a Q(s', a))}$$

On testing Double Learning on 49 Atari games, it produced two to four times the average scores of the standard DQN.

### 2.5.3 Dueling DQN

Another limitation of DQN is that the Q-values are learned separately for actions for the same state. To tackle this issue, a generalization of state-action pairs with the same state can be done. A simple approach would be the update the Q-value as following (where V is the value function and A is the Advantage function)

$$\mathbf{Q(s,a;\ \theta,\alpha,\beta) \rightarrow\ V(s;\ \theta,\beta)\ + A(s,a;\ \theta,\alpha)}$$

However, [12] labeled the use of such a method inefficient, since the value and advantage functions generated are not unique to the problem.

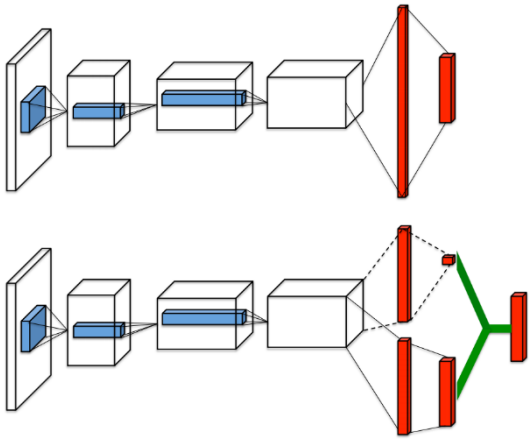


Figure 5: CNN architecture used in DQN (first) and Dueling DQN (second). The convolutional layers split into two part by separate fully connected layers. The Q function is formed from the value function and advantage function at the end of the network.

To solve the problem, the authors of [12] suggested an updated definition of the Q value function.

$$Q(s,a;\theta,\alpha,\beta) \rightarrow V(s;\theta,\beta) + \{ A(s,a;\theta,\alpha) - \max_a A(s,a';\theta,\alpha) \}$$

The updated Q-value function lead to the convolutional layers split the network into two parts as shown in figure 5, highlighting the difference between the original CNN model and the modified. The Q function is formed from the value function and advantage function at the end of the network. According to [2] the dueling CNN DQN model significantly outperformed DDQN.

### 2.5.4 Other Variants

The development of DQN in 2012, followed the release of many variations of the algorithm such as Double DQN and Prioritized Experience Replay as mentioned. Others include Multi-step DQN, Noisy networks, Distributional DQN, and Rainbow DQN. Multi-step DQN use multistep targets as compared to single targets used by previously mentioned DQNs. Noisy networks aim to improve the epsilon-greedy strategy used for exploration by adding noise to the CNN model. Distributional DQN treats the Q-values as a distribution rather than an individual value. Finally, Rainbow DQN seeks to combine the mentioned complimentary algorithms. Noisy networks and Multistep DQN of these algorithms have been proven to not improve the performance, however, they considerably improve the stability of learning to help solve more complicated task sand. Whereas Distributional DQN considers the sampling distribution, enforcing the overfitting transitions to be explored. This method was in fact proved to improve performance and increase the learning rate. Finally, Rainbow DQN combining all these features resulted in significantly higher performance, learning and stability when compared to any individual variants.

## 2.6 Other Deep Reinforcement Learning Algorithms

Apart from DQN and its variants, other deep reinforcement algorithms have also been developed such as deep deterministic policy gradients and proximal policy optimization algorithms which output action probabilities instead of Q-value table. However, for the purpose of this report, we would only be exploring the earlier mentioned algorithms as they cover a broader range of variation of types of RL Algorithms

# 3. Methodology

## 3.1 Blackjack

The game of Blackjack is a well-known casino card game. The objective of the game is to obtain cards whose total values are as high as possible without exceeding 21 and beat the dealer by either achieving a sum greater than the dealers or letting the deal draw additional cards until their hand exceeds 21. The standard deck of 52 cards is used, where all face cards count as 10, and an ace can count either as 1 or as 11.

Figure 6: Game of Blackjack

For this report, we consider the following version of the game, where the game starts off by dealing two cards to both the player and the dealer. One of the dealer's cards is shown to the player i.e. is face up and the other is face down or unknown to the player. If the player has an ace and a face card, the player has a natural and will win the round, unless the dealer also has a natural, then it would be a tie. Generally, this is not the case, the player can then hit or pass. Hit implies that additional cards are requested, until a stick or bust (stop or exceed 21). A bust implies an automatic loss. If the player either passes or sticks, it becomes the dealers turn. The dealer can also hit or stick, however, is restricted to fixed strategy; stick on a sum of 17 or greater, else hit. Similar to the player going bust, if the dealer gets a bust, the player wins. If none of the cases happen, the outcome is determined by whose final value is closer to 21.

Luckily, Blackjack can easily be formulated as a finite MDP. where the rewards allocated are +1, −1 or 0 to the player/agent for winning, losing or drawing, respectively. Each hand is considered as a round and the total number of rounds is the total training time. All players start with a reward of 0 at the beginning of a game.

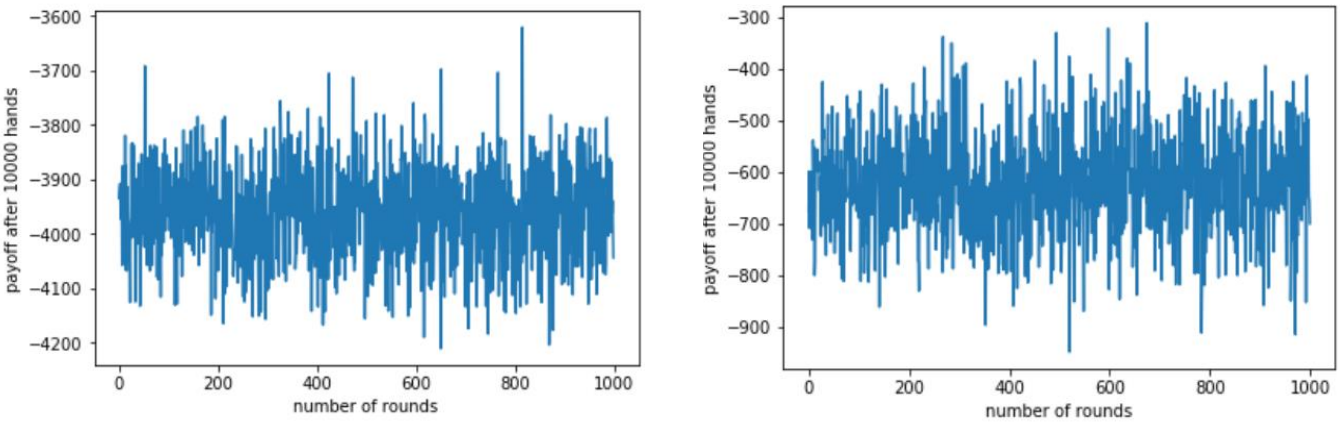
## 3.2 Edward Thorp's Strategy and a Random Strategy

To benchmark, the Blackjack algorithms Edwards Thorp's strategy was used as a best case or target to beat strategy and a random strategy was considered to define the worst case.

In 1962 Edwards Thorp wrote a book “Beat the Dealer” which explored the philosophy and mathematics behind Blackjack, defining a set of instructions that gives any players the best possible odds of long-term success. The book further introduces the concept of card counting, a popular technique that is still used by gamblers to maximize their profits.

The algorithm running Edwards Thorp’s Strategy is used as the pinnacle reference point for the RL algorithms to compete and achieve. Figure 7 (b) shows the implemented algorithm following this strategy achieves an approximate total reward of -623.4 when the number of hands and rounds are 10000 and 1000 respectively.

A random strategy is used to get an idea of the range of total reward that can be achieved by a player who is unaware of the rules yet knows their possible actions. The implemented algorithm randomly hits or passes and achieves an approximate total reward of -3960 when the number of hands and rounds are 10000 and 1000 respectively as shown in figure 7 (a).




(a) using random strategy

(b) using Edward Thorp’s strategy

Figure 7: Average Payoff after 10000 hands per round to estimate range

## 3. 3 Texas Hold'em Poker

Figure 8: Game of Poker

Texas Hold'em Poker is an equally popular casino card game with many variations. This report would be dealing with is No Limit Texas Hold'em variation of the game. The objective of the game is to maximize your stack in hand. The winner of each hand is the one who has the best hand at the showdown (after 5th round) or is the last person remaining on the table. The winner then receives all the pot value on the table. However, the second implies the best hand always doesn't win; signifying the importance of bluffing in the game. At the beginning of each hand, all players are given two cards. Two players need to bid a fixed initial amount to get the round started. Once the bidding is over, 3 cards are displayed on the table followed by another round of betting occurs. This is repeated until there are 5 cards on the table. After the final round of bidding, the showdown takes place, where all players reveal their hand and compare who has the highest combination. Figure 7 lists the rank of hands from highest to lowest. At each round, the player can check (when is there is no initial bet), call (when they want to bid the same amount as the previous player, to remain in the game), raise (to increase the stake of the game) or fold (leaving the game with the loss of the amount already invested in the pot). Unlike Blackjack, Poker cannot be constructed into finite MDP easily, as the reward from the pot is a discrete variable and hence harder to estimate the reward function as well as the probability function at each stage. Another major reason is that the information available is imperfect, i.e. a player does not have all the information in-hand (they do not the cards of the other players). For this report, we refer to the stack in the player's hand as its final earnings. Each hand consists of 5 rounds and the total number of hands is the total training time. All players start with a reward of 10,000 units amount at the beginning of a game.


Figure 9: The rank of hands from highest to lowest. Retrieved from [13].

## 3. 4 Poker Strategies

There are different kinds of poker players, each playing abiding by their own strategies.

### 3.4.1 Tight vs Loose

The first thing that one can observe a player's strategy is how frequently he plays or folds. If someone tends to play only good hands and folds often, they are a tight player. On the other hand, if a person takes part in a lot of hands, they are considered to be a loose player.

### 3.4.2 Passive vs Aggressive

The next observation one can make is how often does the player bets. A player who calls or check more often is referred to as a passive player and a player who raises or bets more is an aggressive player.

### 3.4.3 Killers-Maniacs-Rocks-Calling Stations

With the help of the above classifications, we can categorize most play styles as either killers, maniacs, rocks or calling stations.

Killers are tight and aggressive. They strategize to play and bet big but do not particularly play many hands. They tend to play and raise with a strong hand to win big with confidence.

Maniacs are loose and aggressive. As the name suggests, these players always aim to bet high as well go infrequently. These players are well, just maniacs and they often start placing bets from the starting hands, in order to steal blinds.

Rocks are tight and passive, being the opposite of Maniacs. These players play less and prefer to check or call or fold. They tend to wait for a good hand but are often timid and cautious about raising the pot.

Calling stations are loose and passive. These kinds of players play a lot of hands but rarely raise. They tend to hope their hand will improve and then they can win.

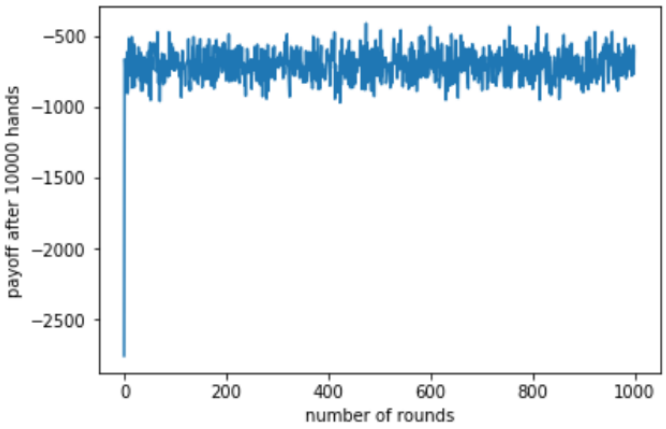
# 4. Results and Discussion

## 4.1 Blackjack

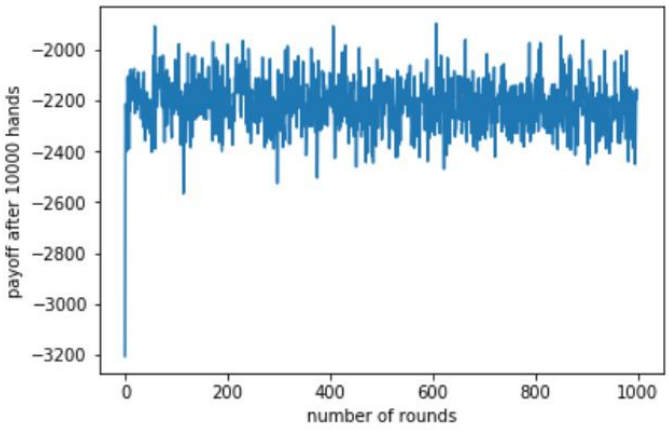
Figure 7 sets a target to beat and defines a worst-case boundary set by Edward Thorp's strategy and the random strategy respectively. The offline and online policy-based TD Learning algorithms, Q-Learning and SARSA, MC, DQN and its variants were implemented. Figure 10(a) and (b) indicates that when the number of hands was set to 10000 and the number of rounds to 1000, the average payoff obtained was -710.5 by Q-Learning and -2133.7 by SARSA; indicating that the Q-Learning algorithm outperformed SARSA by significant margin, in terms of achieving of learning the correct strategy which could potentially because the game of Blackjack only contains two actions, hit and pass at each step and hence, updating the policy after each action would not be improving learning.

Similarly, MC, as seen from figure 10(c), achieved an average payoff -1866.3, with 10000 hands and 1000 rounds, performing slightly better than SARSA, but not as well as Q-Learning. This shows that the mechanism of random sampling might not be the best strategy to tackle this problem and further research on Q-Learning and its variants would be more useful.

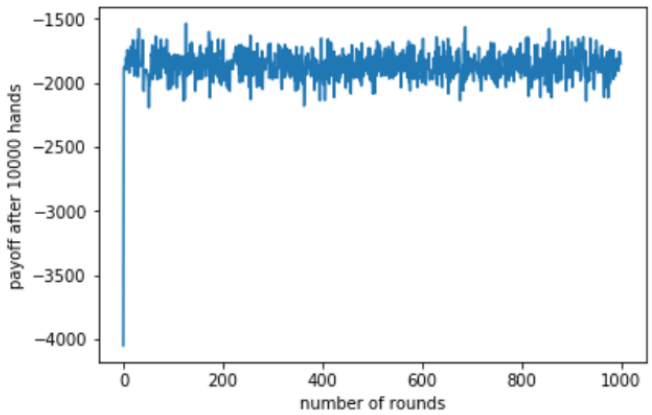
Finally, from figure 10(d) DQN again in 10000 hands and 1000 round of training was able to get an average payoff -1864.475, performing worse than Q-Learning. This was a surprising result as it was expected to perform like Q-Learning. This was potentially due to the simplicity of the game, as the MDP is formulated easily and can be stored in a 2D matrix, neural nets which perform estimation to store these values, are not as accurate as the hard-values stored in the matrix. Similarly, upon testing Double DQN and Dueling DQN (figure 10(e & f)), it performed as expected i.e. was not an improvement over the DQN model, also achieving an average payoff of 2180, 2137, illustrating the complex we made the model, there seemed to be a slight decrease an accuracy in the results.



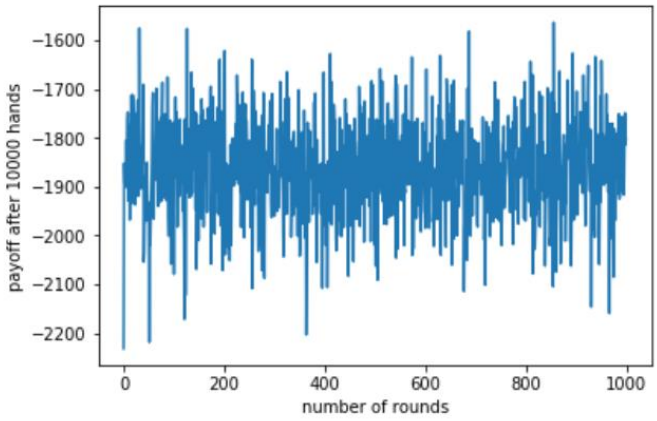

(a) TD: Q-Learning



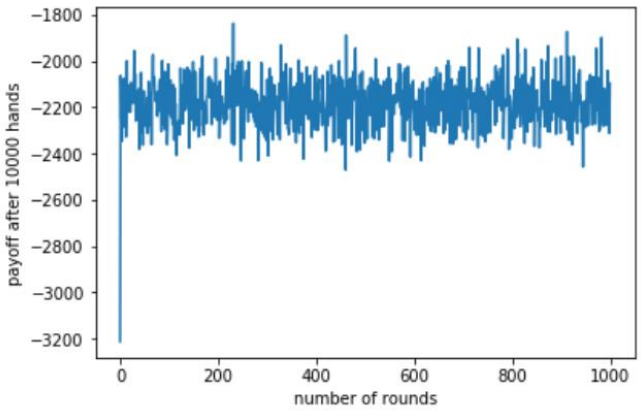

(b) TD: SARSA



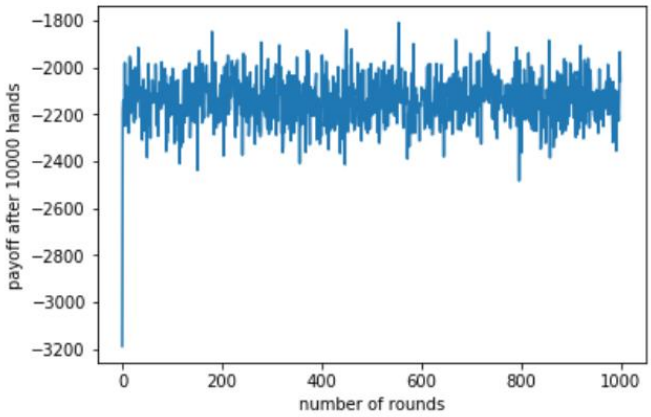

(c) Monte Carlo




(d) Deep Q Network




(e) Double Deep Q Network




(f) Dueling Deep Q Network

Figure 10: Average Payoff after 10000 hands per round using RL algorithms

## 4.2 Texas Hold'em Poker

As compared to Blackjack, Poker is more complex as well as an imperfect game meaning that the player does not have all the information available to them. Therefore, the algorithms that performed the best for blackjack might not be applicable for poker. However, from our understanding of these algorithms and with the help of neural networks for approximation, the project builds a customized Deep Q-network (section 2.4).

### 4.2.1 Architecture

The Poker AI implements a custom Deep Q-network, bring together many aspects of already tried and tested improvements and variations of DQN (as mentioned in section 2.5).
The neural net is a standard feedforwarding network with three full connected hidden layers and 32,16,4 rectified linear units respectively for the output, utilizing the relu activation function. The duel embedding forces the architecture to split into two parts by separate fully connected layers, forcing a zero-sum game between the two networks (section 2.5.3). The AI also implements various improvements (section 2.4.x) and strategies such as prioritized action replay (section 2.5.1), double DQN (section 2.5.2) and Multi-step mechanism (mentioned in section 2.5.4). Manual hyperparameter tuning was performed to optimize the algorithm.

### 4.2.2 Analysis

The algorithm is trained over 2000 rounds (or hands), with an initial stack of 10,000 units given to each player. For the analysis, the algorithm plays against 3 other players at a time where all players are either of the mentioned four kinds of players maniacs, killers, rocks and calling-stations (section 3.4.2). The blue line in Figures 11 to 14 represent the AI.
Figure 11 shows the AI against Killers who are tight and aggressive. They are the type of players that only go in with their good hands and are always betting big, which explain the multiple vertical slopes in the graph. As they are folding often, it harder to constantly win against them. Hence why once the player represented by orange line was able to win big at the beginning, it was hard for the AI to win against it. After around 1000 hands, the AI against the orange player seems to be oscillating, as the AI is not letting the player win nor is player is constantly folding upon receiving weak hands.

Like figure 11, figure 12 shows the AI against three maniacs. Since they are aggressive players as well, we see the vertical slopes present in the graph. However, over time when the AI starts to learn their methodology and it is successfully able to profit off them as unlike rocks, maniacs are loose players and do not fold so often.

In both cases, since the players are aggressive, the AI is able to win big and get a stack of around 16k, up by more than 50% of the starting amount.

Figure 13 displays how the AI learns against Calling Stations. Since these players are more passive, the slopes are more gradual. After around 1000 hands, the AI learns about this detail and tries to use it to its advantage by trying to bid more, hence the rise of vertical slopes towards the end.

Finally, Figure 14 shows the AI playing against Rocks, who like calling stations are also passive and thus do not bet huge, in-turning producing gradual slopes. However, since they are tight players, the AI is not able to take advantage of them even after multi rounds of training to extract profits.

In the last two cases, the tightness of the players made a huge difference as although the AI was eventually able to make huge profits from calling stations, around 18k after 1000 hands, would eventually be able to cash out all the players from the game, the AI was unable to extract any kind profit off from the tight rocks, only being able to achieve around 13k, but similar to its behavior to calling stations, the AI was slowly and steadily gaining profit without suffering from multiple losses.

Overall, the AI was able to achieve eventually able to gain profit and beat all four kinds of players behaving differently in each situation. However, in all cases, the AI was learning to be more and more aggressive, regardless of the opponent being aggressive or passive. The opponent's tightness was its restriction factor, affecting the rate at it was able to gain profits, wherein against tight players, it was slow but eventually making money. However, against loose players, its profitability was increasing a linear rate, and would eventually be able to cash out all other players over time. From all the observations made, we can conclude that the custom DQN AI ultimately started to behave like Maniac i.e. became aggressive and loose.

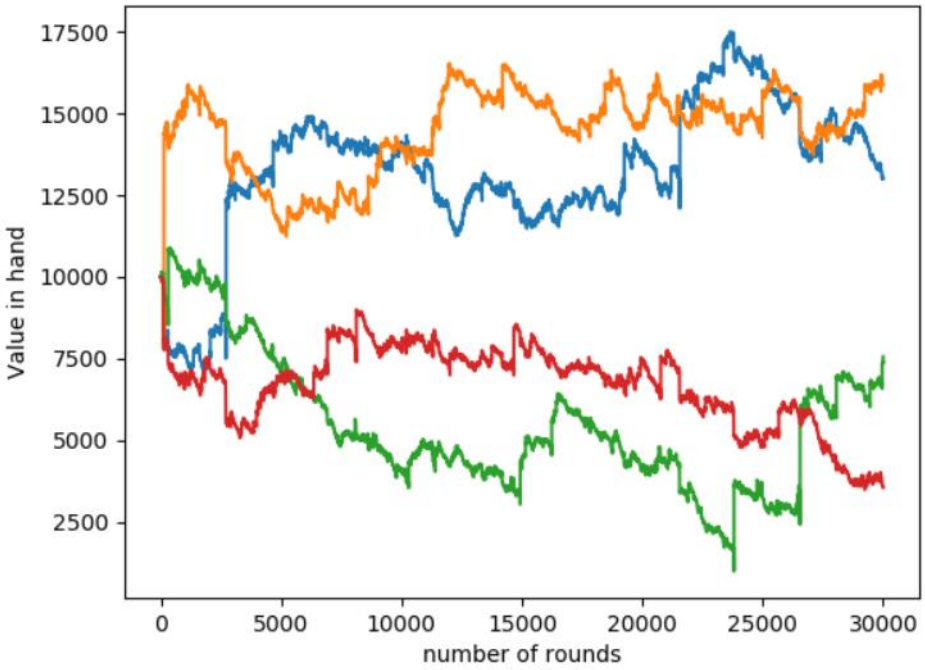



Figure 11: AI vs Killers

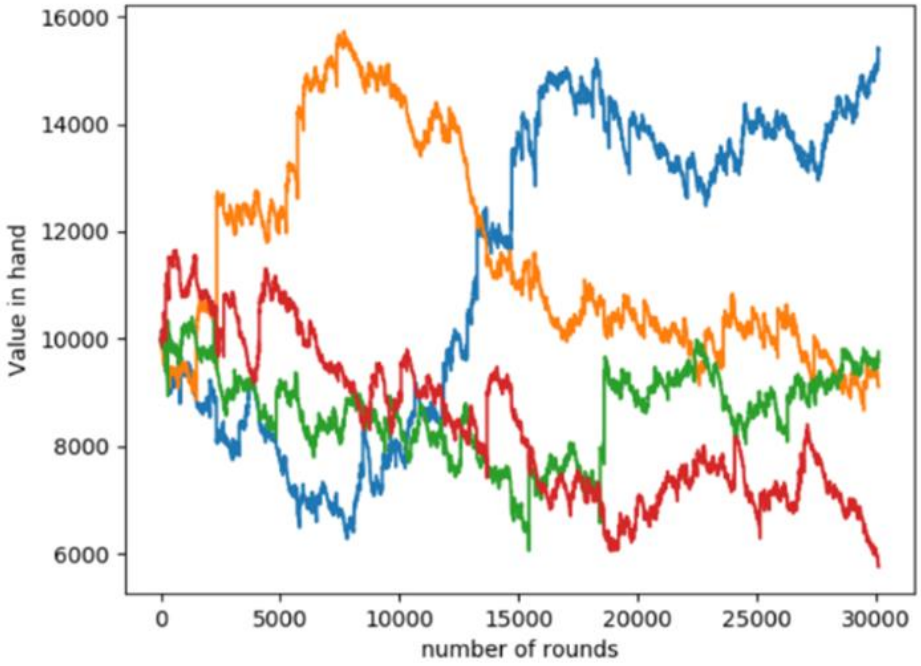



Figure 12: AI vs Maniacs

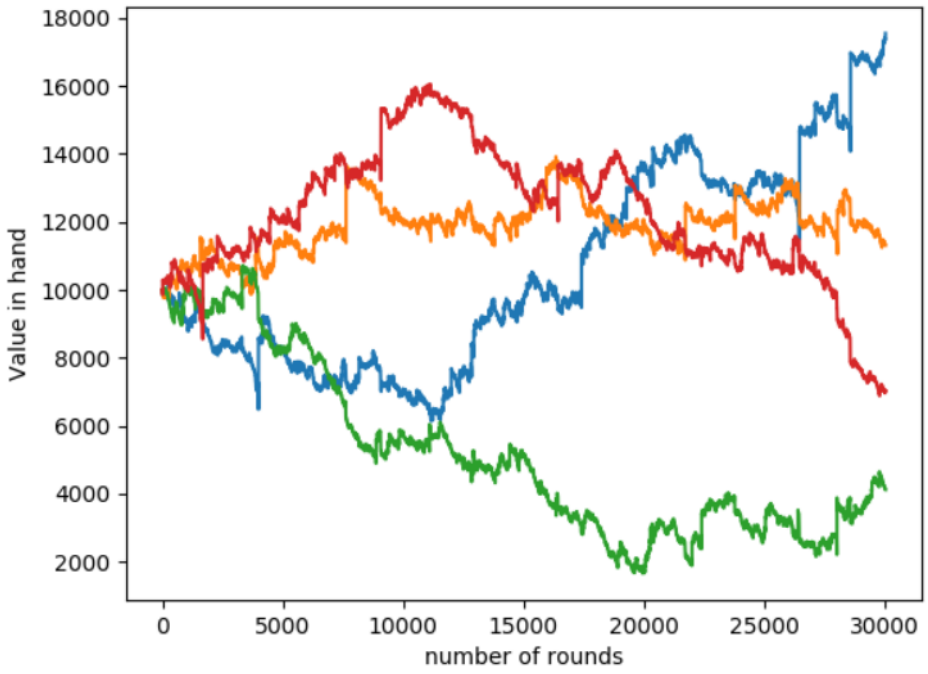



Figure 13: AI vs Calling Stations

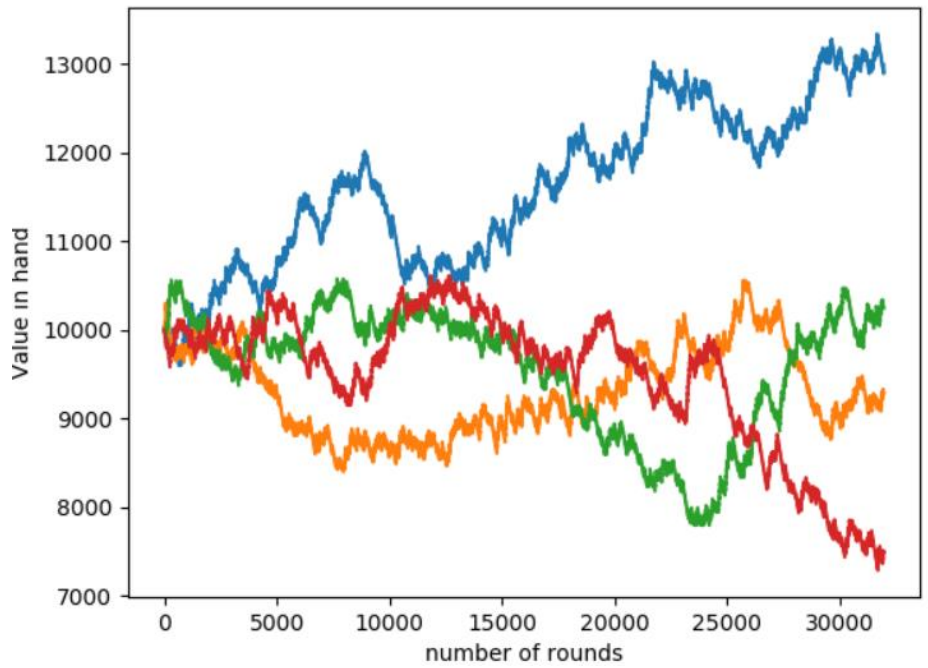



Figure 14: AI vs Rocks

## 4.3 Limitations and Difficulties – hard to find benchmarks

Several difficulties were encountered to optimize and tune the RL algorithms since there is no quick or formalized mechanism of optimizing the hyperparameters for the models. This led to slower progress of work since the manual optimization of hyperparameters is expensive and time-consuming. Further, different results could be produced than the ones currently achieved through more aggressive tuning, which could potentially affect the observations.

Another factor causing delay were the resource constraints. The RL algorithms were taking a long time for training, due to the lack of a powerful computer and proper use of the cores of the computer.

Two problem specific to the Poker AI, firstly the difficulty in benchmarking the results. The analysis was provided to understand the algorithm and its working. However, due to lack of time, the report was unable to benchmark the AI against other AIs and human players. Second, graphical data could not be reproduced regarding when the AI was able to cash out different players due to the restriction of the environment, where the game would end if one player cashed out.

# 5. Future Work

The project has successfully been completed. However, this does not mean there are no improvements that can be made. For starters, as these algorithms were written from scratch, they could be made more efficient and undergo further hyperparameter tuning. This is could produce slightly different results but will not change the conclusion.

As mentioned as one of the difficulties (section 4.3), the Poker AI, lacked benchmarking. Hence, the report could investigate that by comparing and playing it against other AIs. Further, A GUI could also be built for the game and the AI, for humans to play with it for benchmarking against human players.

For this report, all the previous stages of the project have been summarized in table 1.

Table 1: Project Schedule

| Date | Task |
|---|---|
| September 2018 | Preliminary Research<br>● Study Reinforcement Learning, Game Theory, and Multi-agent Systems<br>● Read Research papers and organize Ideas |
| 1st October 2018 | Phase 1 Deliverables (Inception): Project Scheme, Detailed Project Plan |
| October 2018 | ● Implement Random strategy for Blackjack<br>● Implement Edward Thorp's strategy for Blackjack |
| November 2018 | ● Implement MC for Blackjack<br>● Implement SARSA for Blackjack |
| December 2018 | ● Implement Q-network for Blackjack<br>● Tune the implemented algorithms and establish results |
| Mid-January 2019 | Phase 2 Deliverables (Elaboration): Presentation and Interim Report |
| January - February 2019 | ● Implement DQN for Blackjack<br>● Implement other variations of DQN for Blackjack<br>● Implement DQN for Poker |
| March - April 2019 | ● Implement additions features on the basic DQN model<br>● Tune the implemented algorithms<br>● Perform Analysis on the custom DQN<br>● Establish Justifications for results |
| Mid-April 2019 | Phase 3 (Construction): Final Presentation and Final Report |

# 6. Conclusion

This project sought out to improve our understanding of popular card games played in Casinos using Reinforcement Learning and Deep Learning. The first part of the project compared various reinforcement learning algorithms on the game of Blackjack and was benchmarked against Edward Thorp's strategy from the book "Beat the Dealer ". RL algorithms namely, Q-Learning, SARSA, and Monte Carlo along with Deep RL algorithms namely DQN, Double DQN, and Dueling DQN were implemented. The results showcased that Q-Learning was able to achieve the highest payoff and was nearly at the level of Thorp's strategy and concluded that for the simple game of blackjack which can be formalized as an MDP, state approximation with the help of neural nets only complicated the matter. The second part of the report involved the development of a custom DQN model on the game of Texas Hold'em Poker, which was then trained and analyzed against four different kinds of players, namely, killers, maniacs, rocks and calling stations. The results concluded that the after around 500 rounds of training, the AI took an aggressive approach regardless of the type of player it was facing, eventually behaving like a maniac (aggressive and loose) and was easily able to outplay those players.

# 7. References

[1] Sutton, Richard S., Andrew G. Barto, and Francis Bach. Reinforcement learning: An introduction. MIT press, 1998

[2] Kaelbling, L. P., Littman, M. L., and Moore, A. W. Reinforcement learning: a survey. Journal of Artificial Intelligence Research (JAIR), 4:237–285, 1996

[3] Watkins Ch. Learning from delayed rewards, PhD. thesis, Cambridge University, 1989

[4] Watkins Ch. and Dayan P. Q-learning, Machine Learning 8, 1992

[5] Mnih et al.– Human-level control through deep reinforcement learning, Nature 518, 2015

[6] Long-ji Lin. Reinforcement Learning for Robots using Neural Networks. Carnegie Mellon University, 1993

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning, in NIPS Deep Learning Work 21

shop, 2013

[8] Hado van Hasselt, Arthur Guez, David Silver. Deep Reinforcement Learning with Double Q-learning, arXiv:1509.06461, 2016

[9] Hado van Hasselt. Double Q-learning, Advances in Neural Information Processing Systems, 2010

[10] Schaul et al. Prioritized Experience Replay, arXiv:1511.05952, 2015

[11] McCloskey, M. & Cohen, N. Catastrophic interference in connectionist networks: The sequential learning problem. In G. H. Bower (ed.) The Psychology of Learning and Motivation,24, 109-164, 1989

[12] Z. Wang, N. de Freitas, and M. Lanctot, Dueling network architectures for deep reinforcement learning, CoRR, vol. abs/1511.06581, 2015

[13] Caesars Entertainment. World Series of Poker. From www.wsop.com/poker-hands

[14] Alons k. Analysis of poker strategies in heads-up poker, VU University Amsterdam, 2007