Learning to play Blackjack with Deep Learning and Reinforcement Learning

Interim Report

COMP4801 Final Year Project

UID: 3035238565

Student

Ish Handa

Supervisor

Professor Francis C.M. Lau

January 2019

Abstract

Artificial Intelligence has always had a long history with games. This project seeks to compare and understand various reinforcement learning algorithms namely, Temporal Difference Learning, Monte Carlo methods, Deep Q Network and its variants on the game of Blackjack targeting to compete and potentially outperform Edward Thorp's strategy from his book "Beat the Dealer ". This report deals with the implementation of the mentioned algorithms and compares them based on their average payoff. From the initial findings, we were able to learn Q-Learning was outperforming the other algorithms and was able to achieve a payoff close to Thorp's strategy. However, the DQN model requires tuning to produce better results which would the next step of this project. This would be followed by the implementation of Double DQN and Dueling DQN algorithms for further comparison. Despite the challenges faced, the project is well on schedule with the on-going fine-tuning of the DQN model.

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
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 π^* 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.

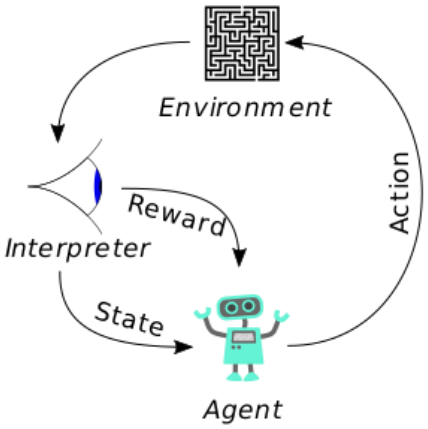


Figure 1: An illustration of a Reinforcement Learning

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

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, a 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 understand the game of Blackjack, a previously solved game and compare and investigate 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.

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 which would be implemented in this project. Section 3 introduces the game of Blackjack and its rules as well as the benchmark domain set by Edward Thorp's strategy. Section 4 compares and evaluates the results of the various RL algorithms developed. Section 5 details the phases of the project, along with the progress of the project, and finally, Section 6 concludes the report by discussion the work to be done.

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.

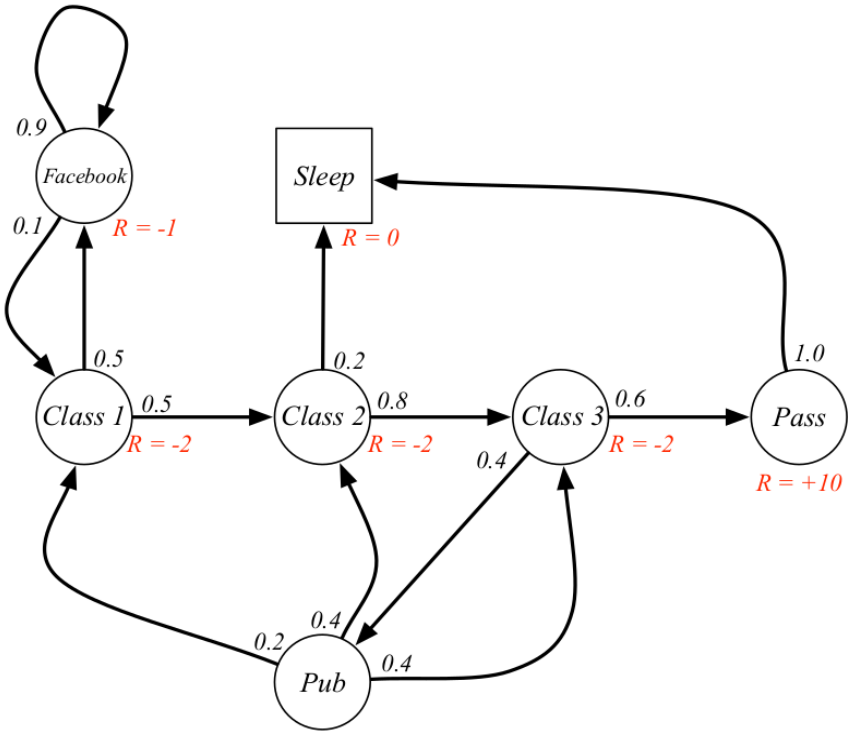


Figure 2: Student MRP from David Silver class

Formally, an MDP can be defined as a tuple ⟨ S, A, P, R_y ⟩ 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.

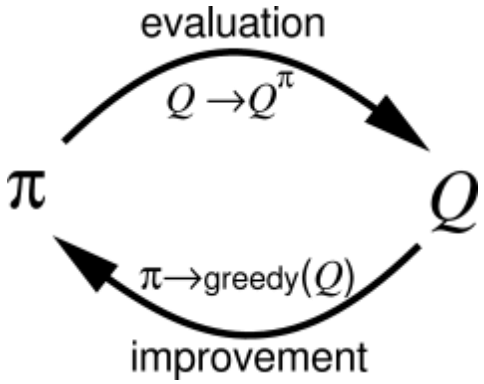


Figure 3: Monte Carlo iteration cycle

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 π_{k+1} is uniformly better than π_k, unless it is equal to π_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,

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

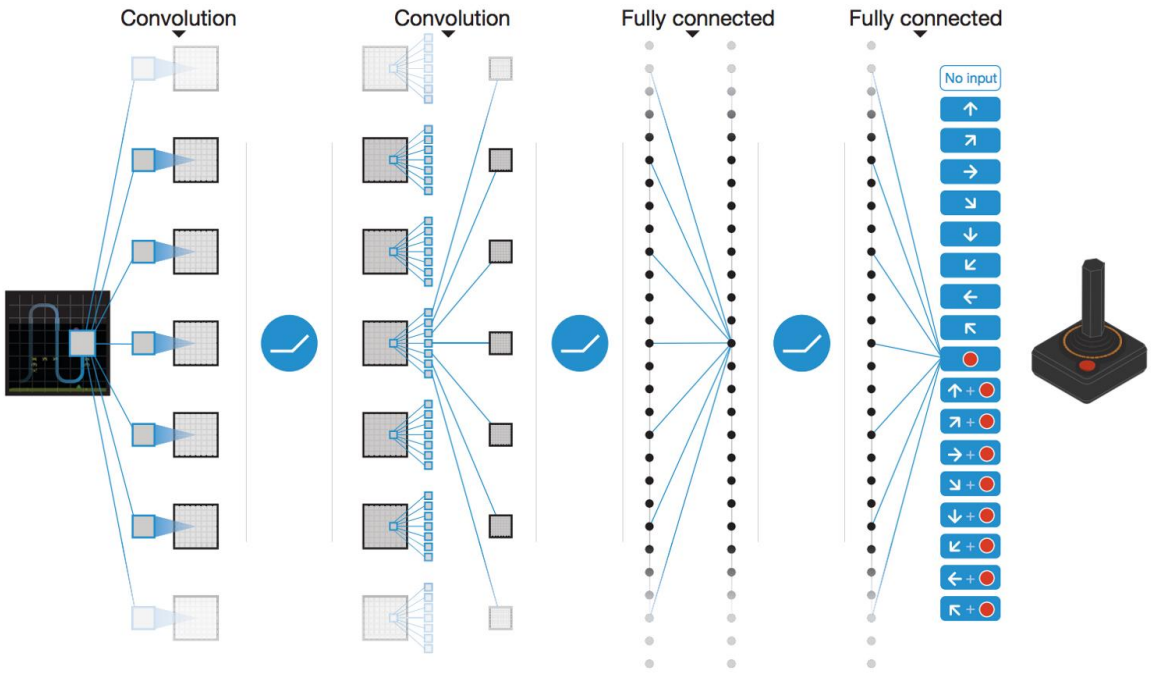


Figure 4: DQN network with a Convolution Neural Network illustrating Q function approximation

However, the introduction report recognized that by simply adding a DNN 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 Other Variations of DQN

The development of DQN in 2012, followed the release many variations of the algorithm such as Double DQN and Prioritized Experience Replay. Double DQN involved the use of two different DNNs. It has been proven to not improve the performance, however, it considerably improves the stability of learning to help solve more complicated tasks. Prioritized Experience Replay which is another popular variation played with the sampling distribution, enforcing the overfitting transitions to be explored again and again until the model shifts to fit it. This method was in fact proved to improve performance and increase the learning rate.

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. If time allows, such Deep RL algorithms may be covered for a deeper analysis.

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.

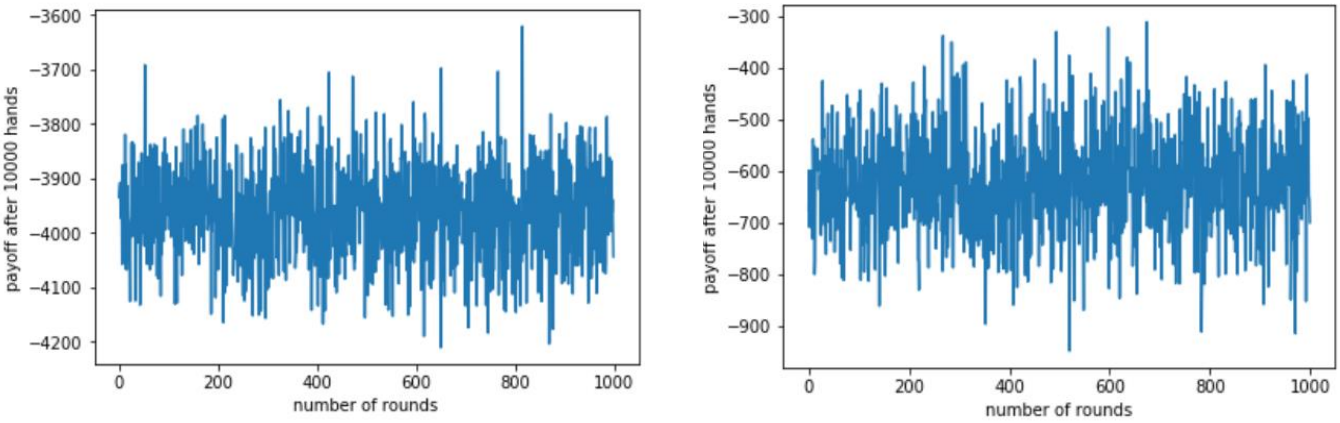
Figure 5: Game of Blackjack

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 Random Strategy

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 6 (a).



(a) using random strategy

(b) using Edward Thorp's strategy

Figure 6: Average Payoff after 10000 hands per round to estimate range

3.3 Edward Thorp's Strategy

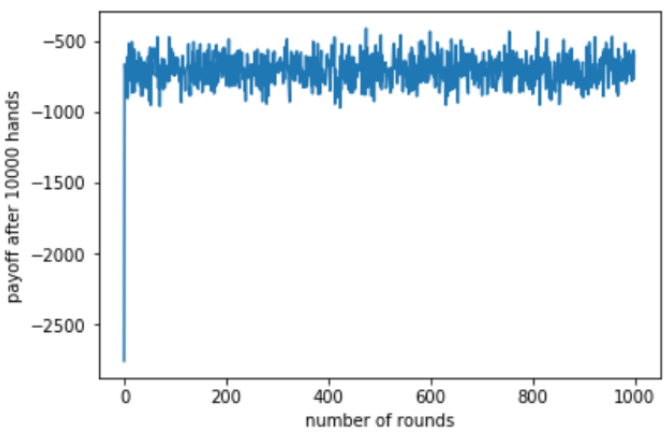
In 1962 Edwards Thorp wrote a book "Beat the Dealer" which explored the philosophy and mathematics behind Blackjack, defining a set of instructions that gives any players the best

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 our RL algorithms to compete and achieve. Figure 6 (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.

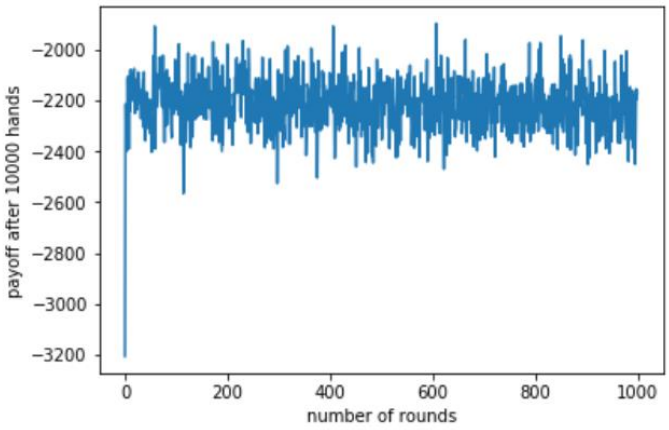
4. Results and Discussion

4.1 Initial Results

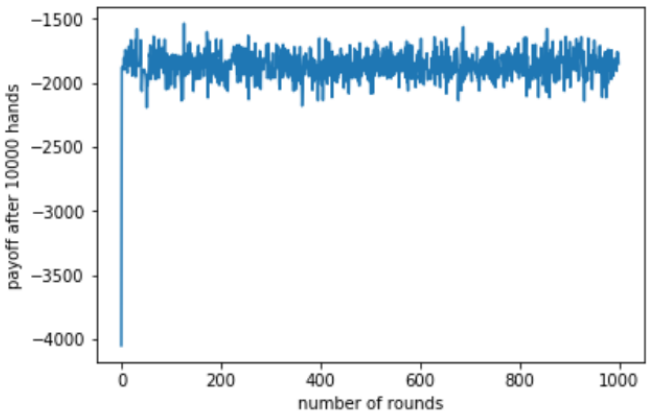
Figure 6 sets a target to beat and defines a worst-case boundary set by Edward Thorp's strategy and the random strategy respectively. The offline and online policy-based TD Learning algorithms, Q-Learning and SARSA, MC and DQN were implemented and tuned in the current stage of the project. Figure 7(a) 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 7(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 7(d) DQN 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. Further research is required to understand on why DQN was unable to perform so well. This means rebuilding and tuning the model and then finding academic articles to back up a claim to understand its failure.



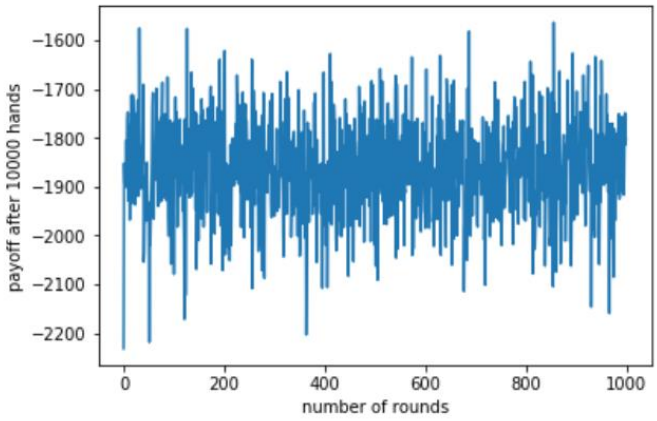
(a) TD: Q-Learning



(b) TD: SARSA



(c) Monte Carlo



(d) Deep Q Network

Figure 7: Average Payoff after 10000 hands per round using RL algorithms

4.2 Limitations and Difficulties

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 lead to slower progress of work since 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 our observations.

Another factor causing delay was 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. To reduce the time taken, further investigations regarding would be required to aiming to make the algorithms more efficient.

5. Future Work

With the current progress of the project, implementation and tuning of MC methods, TD Learning Q-network and SARSA, and DQN have been complemented and tuned, with the exception of DQN which requires further tuning. The next step involves studying and implementing variations of the DQN algorithm namely the Double DQN and the Dueling DQN, further if time allows looking into other deep learning RL algorithms. All the stages of the project have been summarized in table 1.

Table 1: Project Schedule

Date	Task
September 2018	Preliminary Research ● Study Reinforcement Learning, Game Theory, and Multi-agent Systems ● Read Research papers and organize Ideas
1st October 2018	Phase 1 Deliverables (Inception): Project Scheme, Detailed Project Plan
October 2018	● Implement Random strategy ● Implement Edward Thorp's strategy
November 2018	● Implement MC ● Implement SARSA
December 2018	● Implement Q-network ● Tune the implemented algorithms and establish results
Mid-January 2019	Phase 2 Deliverables (Elaboration): Presentation and Interim Report

January - February 2019	● Implement DQN ● Implement other variations of DQN
March - April 2019	● Tune the implemented algorithms ● Compare the algorithm, establish results ● Establish Justifications for results
Mid-April 2019	Phase 3 (Construction): Final Presentation and Final Report

6. Conclusion

This project seeks to compare and understand various reinforcement learning algorithms on the game of Blackjack and to compete and potentially outperform Edward Thorp's strategy from the book "Beat the Dealer ". This report first introduced the background studies and theory behind various reinforcement algorithms that were implemented at this stage of the project. The report then introduced the game of Blackjack, along with the implementation of a random algorithm to identify the worst-case scenario and of Edwards Thorp's strategy. The report further discussed our findings from the implementation of the four reinforcement learning algorithms namely, Q-Learning, SARSA, Monte Carlo and DQN. These results showcased that Q-Learning was able to achieve the highest payoff and was nearly at the level of Thorp's strategy. However, the DQN model still requires further to tuning for it to potentially provide better results. Finally, the next stage of the project is to understand and implement Double DQN and Dueling DQN algorithms to analyze their payoffs with the previously implemented algorithms and Thorp's strategy.

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.
https://pdfs.semanticscholar.org/54c4/cf3a8168c1b70f91cf78a3dc98b671935492.pdf

[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. (1989) Catastrophic interference in connectionist networks: The sequential learning problem. In G. H. Bower (ed.) *The Psychology of Learning and Motivation,24*, 109-164