Blackjack: Deep Reinforcement Learning Methods

Muxin Tian
Department of Computer Science
University of Toronto
murphy.tian@mail.utoronto.ca

Yiteng Zhang
Department of Computer Science
University of Toronto
yiteng.zhang@mail.utoronto.ca

Yuyang Zhao
Department of Computer Science
University of Toronto
yyang.zhao@mail.utoronto.ca

Abstract

Traditional Reinforcement Learning algorithms such as Dynamic Programming and Temporal Difference Learning have been proven to be very powerful for tasks with limited state-action pairs, but tend to perform poorly in a more complex setting. So in this paper, we turn our attention to a more complex scene - blackjack, with larger action-state space and more diverse strategies. This research aims to leverage Deep RL techniques to optimize Blackjack playing strategies, with a focus on improving and enhancing current strategies in such a dynamic gaming environment. We apply Deep Q-Network and Double Deep Q-Network methods to blackjack to test how much better our improved algorithm can learn strategic decision-making in uncertain environments.

1 Introduction

We explores the application of two sophisticated reinforcement learning techniques: Deep Q-Network (DQN) and Double Deep Q-network (DDQN) to the domain of Blackjack, a popular casino card game.

Blackjack, with its blend of skill, strategy, and chance, presents a unique challenge that is emblematic of many real-world decision making scenarios. While traditional reinforcement learning methods like Q-Learning and Dynamic Programming have provided foundational insights into algorithmic decision making, they often fall short in more complex environments. Specifically, Q learning require us to learn the Q table, which is memory-costly in our case. We have infinitely many decks in game, so state can be different when every card is shown on table, and thus the total number of decks can be seen as infinite. According to this infinite number, we cannot expect our model to learn a higher-level strategy like card counting. Alternatively, we expect our model to learn a strategy based on current hands of player and dealer. The appearance of neural network could significantly applied on this case. The study benchmarks the performance of traditional algorithms, the agents will be trained and evaluated in a simulated blackjack environment. Moreover, regardless of its complex states, Blackjack has relatively simple rules that are easy to encode into a computer program. This simplicity allows for clear and controlled experimentation with DQN algorithms, making it easier to understand how changes in the algorithm or parameters affect performance.

2 Background and Related Work

2.1 Black Jack

The object of the popular casino card game of blackjack is to obtain cards the sum of whose numerical values is as great as possible without exceeding 21. All face cards count as 10, and an ace can count

Preprint. Under review.

either as 1 or as 11. We consider the version in which each player competes independently against the dealer. The game begins with two cards dealt to both dealer and player. One of the dealer's cards is face up and the other is face down. If the player has 21 immediately (an ace and a 10-card), it is called a natural. One then wins unless the dealer also has a natural, in which case the game is a draw. If the player does not have a natural, then he can request additional cards, one by one (hits), until he either stops (sticks) or exceeds 21 (goes bust). If he goes bust, he loses; if he sticks, then it becomes the dealer's turn. The dealer hits or sticks according to a fixed strategy without choice: he sticks on any sum of 17 or greater, and hits otherwise. If the dealer goes bust, then the player wins; otherwise, the outcome—win, lose, or draw—is determined by whose final sum is closer to 21.

2.2 Deep Q-Network (DQN)

The concept of Deep Q-Networks (DQN) was introduced by Mnih et al.[1], representing a significant advancement in the field of reinforcement learning. This approach extends the basic principles of Q-Learning, a method that estimates the value function $Q^*(s, a)$, by employing a deep neural network to approximate the Q-values for each possible action in a given state. Unlike traditional Q-Learning, which updates the Q-value of a specific state-action pair, DQN utilizes a deep neural network to handle larger and more complex state and action spaces. The key innovation in DQN is the use of a loss function that measures the discrepancy between the predicted Q-values and the target values, derived from the Bellman equation. Through gradient descent, the network's weights are iteratively adjusted to minimize this loss, enabling the model to approximate the optimal policy more effectively. This method has demonstrated remarkable success in various domains, especially in environments with high-dimensional state spaces, where traditional tabular methods are impractical [3].

2.3 Double Deep Q-Network (DDQN)

Double Deep Q-Network (DDQN) is an enhancement over the standard Deep Q-Network (DQN) architecture. In a DDQN, two networks with the same architecture are used: a target network and an online network. The online network makes the action selection (choosing which action to take), while the target network is used to evaluate the action (estimating the Q-value). The target network's parameters are updated less frequently, which helps to stabilize learning. This separation reduces the overestimation bias because it is less likely that the same network will both select and overly value an action. The introduction of DDQN has been a significant step forward in the field of reinforcement learning, leading to improved performance and stability across a range of environments.

3 Data

We are using this dataset from Kaggle that contains the information of 900,000 hands of blackjack games of six players. Each record represents a unique hand made by a player. There are 900,000 rows and 21 columns in the dataset. Each column basically contains the following information: The values of each card for player and dealer, if anyone hits a blackjack, the one who wins, and the amount of money won.

Each row, which is a hand played by a player, can be seen as an episode in the experience replay. In each row, the values of each card for player and dealer determines the state on the table for each time-step. The one who wins and the amount of money won determines the immediate reward for a player. For each hand of a player, the action space is just $\{$Hit, Stick$\}$. Having these information, we can determine the state-action pair and calculate value function needed for training. Therefore the nature of data is appropriate for Deep RL algorithms like Deep Q Network.

Figure 1 is a bar plot of the distribution of each unique value of a hand for all six players, a bar plot of the distribution of each unique value of a hand for the dealer, a bar plot of the distribution of the sum of value of the first two cards dealt to a player, and a bar plot of the result of each hand.

We can see from the first two subplots that the distribution of the sum of values of each hand for a player and dealer are quite similar, especially for the number greater than 17. From the third subplot, we can see that the distribution of the sum of first two cards for a player is slightly left skewed, and the mean and median is around 14. This means that the player needs to be very careful when dealing with the third card. Also from the last subplot, we can see that the win rate of player is below 50%. The key insight of the above discussion is that it is challenging for players to deal with the third card and win in the general case. Therefore, we can conclude from the subplots that playing blackjack as a player is challenging, and win rate can be the primary metric for evaluating our model.

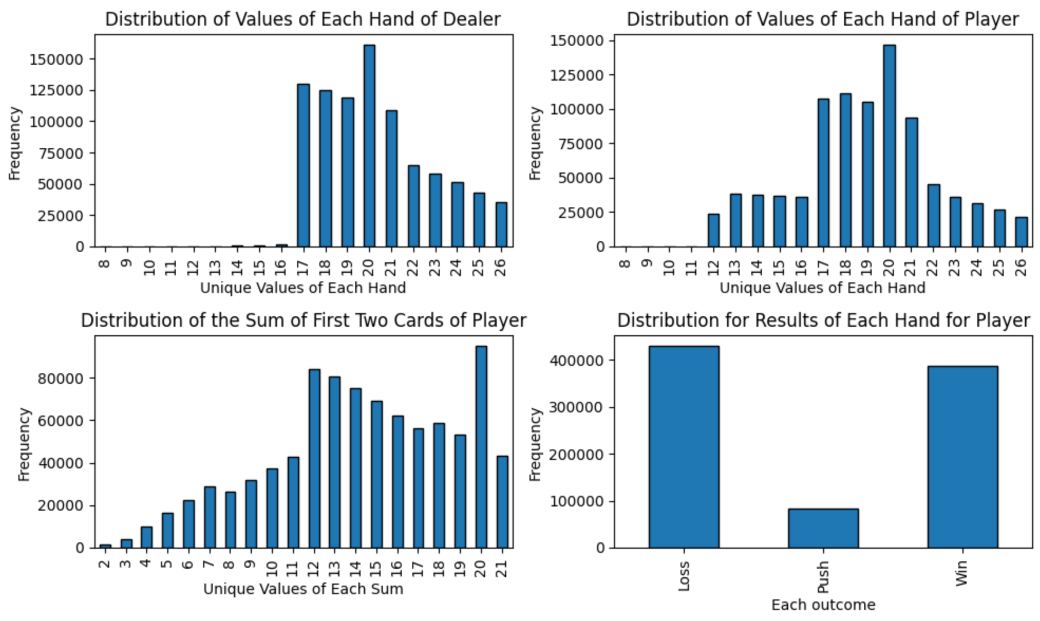

Figure 1: Exploratory Data Analysis

4 Model Architecture

We are using the Dynamic Programming Algorithm and the Monte Carlo Method as baseline models to evaluate the performance of our proposed algorithms in this paper.

Dynamic Programming (DP) in reinforcement learning aims to solve Markov Decision Processes (MDP) where the environment's dynamics (state transition probabilities and rewards) are known. We are using a simpler version of DP here: Policy Iteration. Specifically, the algorithm involves two steps – policy evaluation (estimating V^π for a policy π) and policy improvement (generating a better policy given V^π). During training, we have the following settings: 1. Iterative update of value functions. 2. Repeated application of the Bellman Equation for convergence. 3. Policy is updated based on the current value function.

We then consider the second baseline model. The Monte Carlo method in reinforcement learning is used to estimate value functions and derive policies when the model of the environment is not known. We used both on-policy and off-policy sampling method in practice. During training, we have the following settings: 1. Learning from complete episodes; no bootstrapping. 2. Policy evaluation is done after episodes, and policy improvement can be on an episode-by-episode basis.

Next is our proposed algorithm DQN to apply. Though the data have several players, but we only focus on dealer and one player, other players are irrelevant of simulation. We have two actions: **Hit** and **Stick**. The agent would receive a positive reward $+1$ in the win state, a negative reward in the lose state. If tie, the agent would receive a 0 reward. Playing blackjack formulated as an episodic finite MDP. Each game of blackjack is an episode. All rewards within a game is zero. We assume that cards are dealt from an infinite deck so that there is no advantage to keeping track of the cards already dealt.

We also have a improved version of DQN algorithm – Double DQN, by making use of a target network. Specifically, we need to adjust the way the target value y_j is calculated during the training. In DQN, the target value is computed using the same network to both select and evaluate an action, leading to the overestimation of Q-values. This problem is mitigated in DDQN by using two networks: one for selecting the best action (the primary network) and another for evaluating that action (the target network).

Algorithm 1: Deep Q-Learning Algorithm

```
Initialization replay memory $\mathcal{D}$ to capacity N;
Initialization action-value function $Q$ with random weight $\theta$;
Initialization target action-value function $\hat{Q}$ with weight $\theta^- = \theta$;
for episode = 1, M do
    Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$;
    for t=1,T do
        With Probability $\epsilon$ select a random action $a_t$;
        otherwise, select $a_t = argmax_a Q(\phi(s_t), a; \theta)$;
        Execute $a_t$ and observe reward $r_t$ and $x_{t+1}$;
        Store transition $(\phi, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$;
        Sample random mini-batch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$;
        $set y_i \begin{cases} r_j & \text{(If episode terminates at step j+1)} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{(Otherwise)} \end{cases}$;
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network
          parameter $\theta$;
        Every C steps reset $\hat{Q} = Q$
    end
end
```

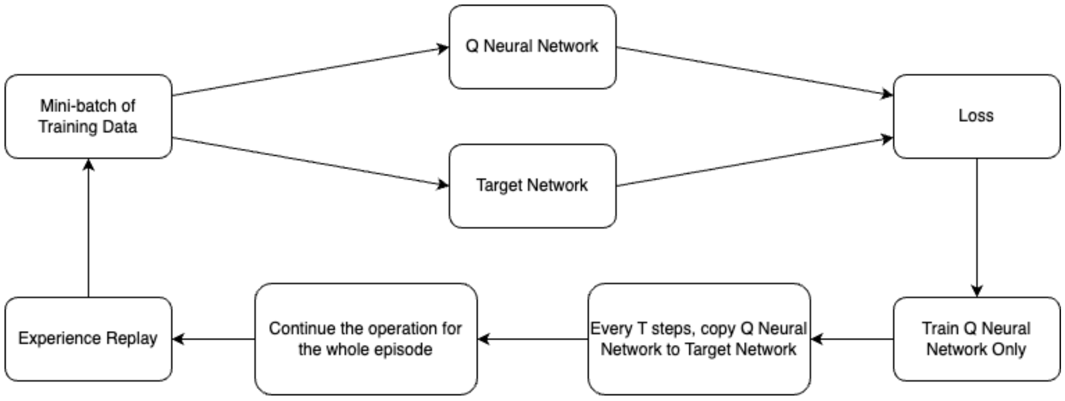
Q Neural Network
Mini-batch of Training Data
Loss
Target Network
Experience Replay
Continue the operation for the whole episode
Every T steps, copy Q Neural Network to Target Network
Train Q Neural Network Only

Figure 2: Workflow of DDQN

5 Results

We first implement the dp and mc as our baseline with episode equals to 100000.

Table 1: Comparison of model performances

model	10000 episodes win rate	100000 episodes win rate
DP		0.35
Monte Carlo		0.29
DQN	0.36	0.42
DDQN	0.38	0.45

6 Discussion

Traditional DQN algorithms tend to overestimate Q-values because the same network is used to select and evaluate an action. This overestimation can lead to suboptimal policies. Introduced to mitigate this issue, Double DQN uses two networks (one for selecting the best action and another for evaluating that action) to reduce overestimation bias.

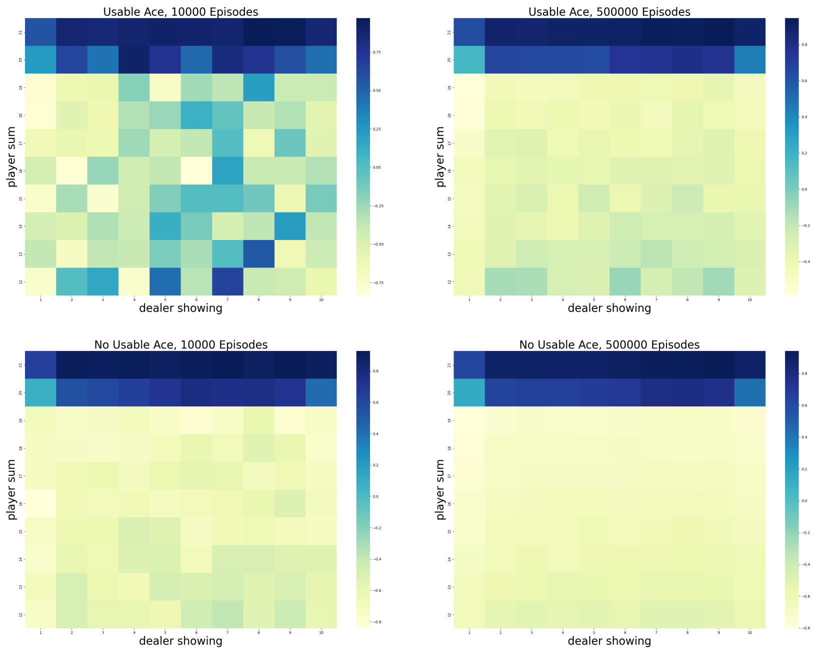

Figure 3: 5.1

7 Limitations

When using the Deep Q-Network (DDQN) and Double DQN algorithms to solve a game like Blackjack, there are several limitations to consider:

7.1 Generalization Across Different Game Rules

Blackjack can have varying rules (number of decks, dealer stands on soft 17, etc.). DQN and Double DQN might not generalize well across these variations without retraining or fine-tuning.

7.2 Stability and Convergence

DQN algorithms can be unstable and may not always converge. Techniques like experience replay and fixed Q-targets are used to improve stability, but they do not guarantee convergence in all scenarios.

7.3 Simplification of BlackJack

In our model, we assume that the number of cards is infinite. That is, the probability of drawing a new card is not related to the cards previously drawn. This is because we import the environment from gymnasium where the number of cards is assumed to be infinite. But this is not the case in real world and we believe there should be some better model considering the number of remaining cards. Also, for simplification, we only consider actions hit and stick while in the real world there are other actions like double and split. Thus, our model might not suit all the cases in the real world.

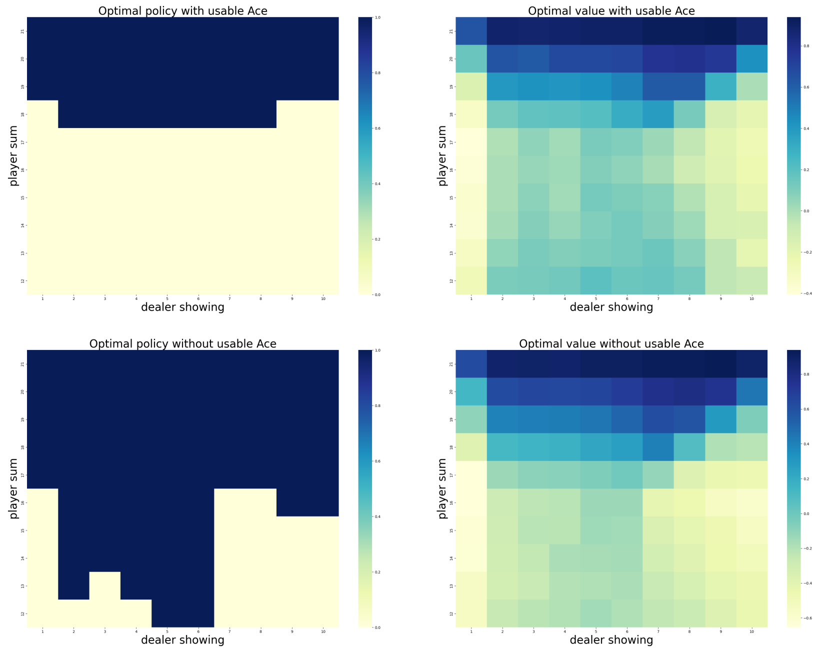

Figure 4: 5.2

8 Ethical Considerations

8.1 Fair Play and Cheating

Games of chance are designed with the premise that all players have an equal opportunity to win based on the odds of the game and the players' skills. Utilizing a tool that provides some players with an advantage violates the principle of fairness. This can be akin to cheating, as it disrupts the level playing field that is expected in gambling.

8.2 Addiction Prevention and Responsible Gambling

The model provides people with a much higher probability to win the game with little cost. If the model is integrated into an application intended for a real-world casino or online gambling environment, it might lead to the addiction of players using this model cause they could probably win the game and earn money for free. Thus, developers should consider implementing addiction prevention features to remind users to play responsibly and not become addicted to gambling.

8.3 Impact on the Gambling Industry

The widespread use of assistive tools can undermine trust in the gambling industry. If the players realize that they can not win against others that are using this model, they may lose faith in the fairness of the system and become unwilling to play the game. This might lead to a decrease in the number of players and result in a decrease in the profit of the gambling industry, which might even impair the economics.

9 Conclusion

References

[1] Mnih, V. & Kavukcuoglu, K. & Silver, D. & Graves, A. & Antonoglou, I. & Wierstra, D. & Riedmiller, M. (2013) *Playing Atari with Deep Reinforcement Learning.*

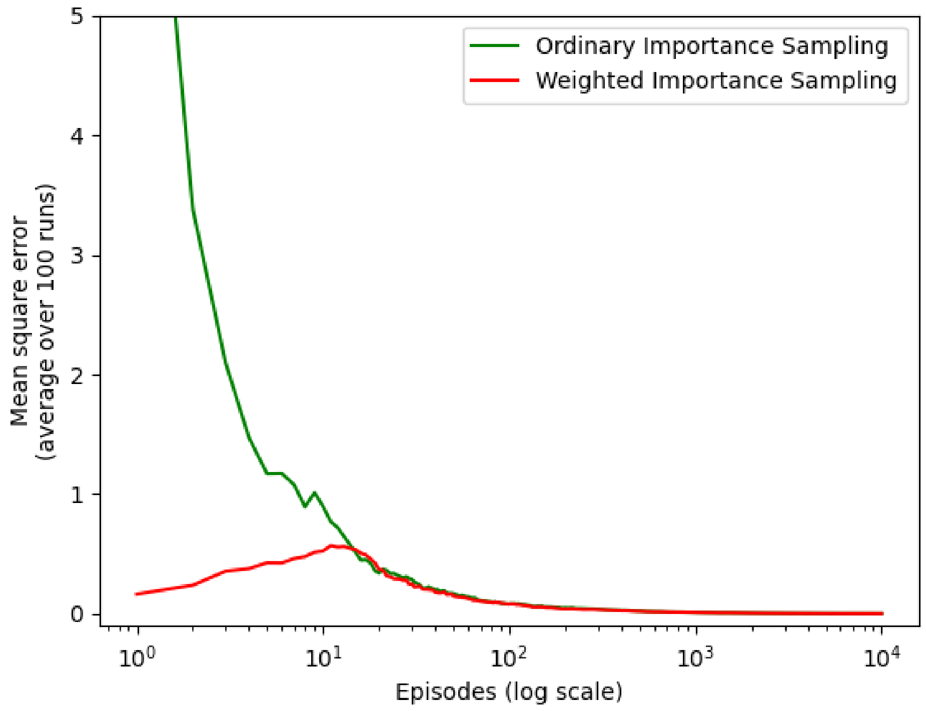

Figure 5: 5.3

[2] Schulman, J. & Wolski, F. & Dhariwal, P. & Radford, A. & Klimov, O. (2017) *Proximal policy optimization algorithms.* arXiv preprint arXiv:1707.06347.

[3] Sutton, R. & Barto, A. (1998) *Reinforcement Learning An introduction.* MIT PRESS.

[4] van Hasselt, H. & Guez, A. & Silver, D. (2016) *Deep Reinforcement Learning with Double Q-Learning.* Proceedings of the AAAI Conference on Artificial Intelligence, 30(1). https://doi.org/10.1609/aaai.v30i1.10295

[5] Wu, A. (2018) *Playing Blackjack with Deep Q-Learning.*

[6] OpenAI. (2023). ChatGPT conversation on 2023, December 8. https://openai.com/chatgpt/