Blackjack Strategy Optimization: A Study of Reinforcement Learning Approaches

Adam Garay
McGill University
Montreal, Quebec
adam.garay@mail.mcgill.ca

Abstract

The paper investigates the use of reinforcement learning algorithms to optimize blackjack strategies. The author explores the effectiveness of Q-learning, SARSA, and DQN algorithms for learning optimal strategies for different variations of the game. Extensive simulations and experiments are conducted to evaluate the performance of these algorithms to study reinforcement learning techniques for solving complex decision-making problems.

1 Introduction

Blackjack is a popular casino game that mathematicians and computer scientists have played for centuries. The objective of the game is to beat the dealer by having a hand that is worth more points than the dealer's, without going over 21 points. Players are dealt two cards and can request additional cards (hit) to improve their hand or stand with their current hand. The dealer follows specific rules for taking additional cards, and the game ends when either the player or the dealer goes over 21 points or when both sides stand.

The game involves complex decision-making under uncertainty, making it an ideal domain for studying the effectiveness of reinforcement learning algorithms. In this paper, we investigate the use of reinforcement learning approaches to optimize blackjack strategies. Specifically, we explore how different algorithms, such as Q-learning and policy gradient methods, can be applied to learn optimal strategies for different variations of the game. The study includes extensive simulations and experiments to evaluate the performance of these algorithms. The goal of our research is to provide insights into the effectiveness of reinforcement learning techniques for solving complex decision-making problems and to develop strategies that can outperform existing approaches in blackjack.

2 Blackjack Environments

To simulate the game of blackjack, I created multiple environments that allow agents to interact with the game and learn optimal strategies through reinforcement learning. The environment includes a deck of cards, which is shuffled at the beginning of each game, and hands for the player and the dealer. The player is able to observe the first card of the dealer's hand, and can request additional cards to improve their hand or stand with their current hand. The dealer must hit until their hand value is 17 or greater, and the game ends when either the player or the dealer goes over 21 points or when both sides stand.

Although pre-existing environments such as OpenAI's Gym can provide a convenient starting point for developing reinforcement learning agents, I decided to write my own blackjack environment. By doing so, I had greater freedom to customize and manipulate various aspects of the game, facilitating

COMP 579: Reinforcement Learning, April 2023, McGill University (Formatted for NeurIPS 2023).

experimentation with different variations and rules. As well, I enjoyed the opportunity to engage in a programming exercise that enabled me to gain a deeper understanding of the game's mechanics.

2.1 BlackjackEnv

The most basic implementation of a blackjack environment involes only three state values and two actions. The environment keeps track of the player's score and the dealer's face-up card, which are the two primary pieces of information that a player can use to make decisions. As well, the environment also includes a feature for checking whether the player has a usable ace, which can be counted as either 1 or 11 points depending on the player's hand. The player can either hit (draw another card) or stand (end their turn).

2.2 BlackjackCountEnv

Skilled human players of blackjack often use a technique called card counting to gain an advantage over the house by keeping track of the number of high-value cards remaining in the deck. To emulate this strategy, we can add functionality to our environment to keep track of the number of aces and tens remaining in the deck. This can be accomplished by extending the state space to include variables for the number of aces seen and number of cards seen that have a value of ten (including face cards).

2.3 BlackjackMoreEnv

In addition to the standard "hit" and "stand" actions, a more complex blackjack environment may include options for "splitting" and "doubling". To implement these actions, the player's available actions must change depending on the state of their hand. For example, a player cannot choose to split their hand if they have already split once or if their two cards have different values. Similarly, a player cannot double down if they have already taken a hit. Adding these actions increases the complexity of the environment and can make it more difficult for a reinforcement learning agent to learn optimal strategies. However, it also more closely mirrors the options available to human players and can lead to more interesting and nuanced gameplay.

3 Algorithms Implemented

Using reinforcement learning algorithms to learn blackjack involves training an agent through repeated gameplay interactions to make optimal decisions in selecting actions that maximize the expected long-term reward. There are a plethora of algorithms that exist in reinforcement learning literature, and an exhaustive study should compare several state-of-the-art techniques. For the scope of this project, we explore three different algorithms that have been successfully applied to the game of Blackjack: Q-learning, SARSA, and DQN.

3.1 Q-Learning

Q-learning is a model-free, off-policy reinforcement learning algorithm that learns to approximate the optimal action-value function of a Markov decision process by iteratively updating the Q-values of state-action pairs using the Bellman equation, which allows for policy improvement without knowledge of the environment's dynamics.

3.2 SARSA

SARSA (State-Action-Reward-State-Action) is a model-free, on-policy reinforcement learning algorithm that learns to approximate the optimal action-value function of a Markov decision process by iteratively updating the Q-values of state-action pairs based on the observed rewards and the next state and action, using an epsilon-greedy exploration strategy to balance exploration and exploitation.

3.3 DQN

Deep Q-Network (DQN) is a model-free, off-policy reinforcement learning algorithm that extends Q-learning by using a deep neural network to approximate the Q-values of state-action pairs, enabling

it to handle high-dimensional and continuous state spaces; it uses experience replay to decorrelate and reuse experience tuples, and a target network to stabilize the learning process by fixing the target Q-values for a certain number of iterations, but may still suffer from overestimation bias and instability due to the non-stationarity of the target values.

4 Results

The code provided contains several hyperparameters used in reinforcement learning algorithms. The annealing epsilon parameter is used in the epsilon-greedy exploration strategy to balance the exploration and exploitation trade-off, where the value of epsilon is gradually reduced over time to prioritize exploitation over exploration as the agent becomes more experienced.

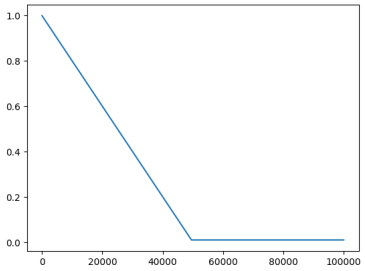

Figure 1: Linearly annealing epsilon over 100000 episodes.

The Q-table is randomly initialized to values in $[-0.001, 0.001]$ to provide a starting point for the agent to learn the optimal action-value function. A learning rate parameter of 0.01 means that the updates will be relatively small to avoid overfitting to noisy data. Finally, the DQN model architecture consists of two fully connected layers with 128 units each, which will be trained to approximate the optimal action-value function using a deep neural network.

We found that all three algorithms performed comparably, with final average returns all between -0.1 and -0.05, significantly outperforming the random average of about -0.4. However, the simplest algorithms and environments showed a small edge in terms of computation time and ease of implementation. It is worth noting that more complex environments and algorithms may demonstrate larger differences in performance. Further exploration of these more complex scenarios may yield additional insights into the strengths and weaknesses of each algorithm..

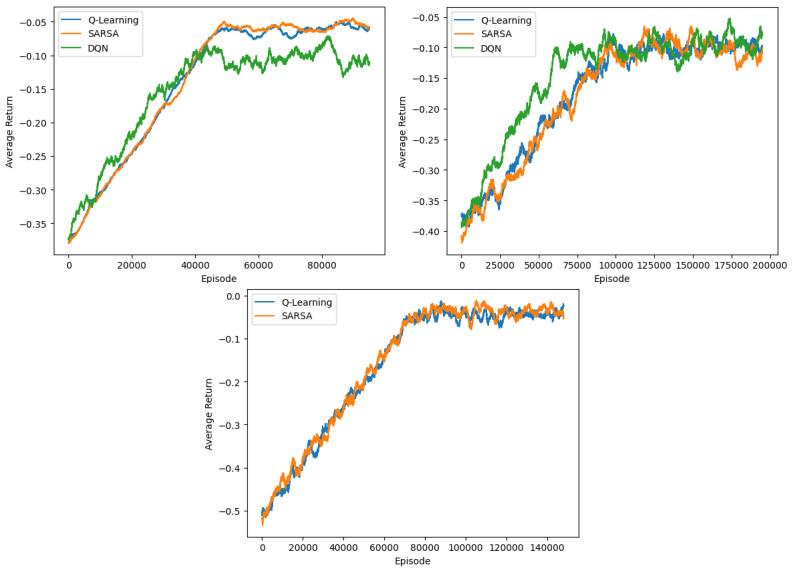

Figure 2: Learning curves for BlackjackEnv, BlackjackCountEnv, BlackjackMoreEnv.

Below are heatmaps describing the learned actions after 100000 episodes of Q-learning for BlackjackEnv, where 0 means the player should hit and 1 means stand. For the lower player values (values less than 12), there appears to be some randomness in the learned actions: in particular, the lower values with usable ace are not valid states. Without usable ace, this may be due to the limited number of episodes used for training. Overall, these heatmaps provide a useful visual representation of the learned policies and are a helpful tool for evaluating the performance of the trained agents.

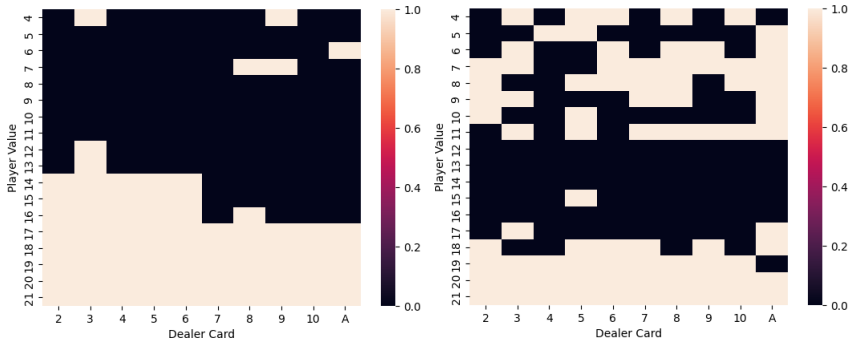

Figure 3: Sample heatmaps without usable, with usable ace.

5 Future Work

There are several areas that could be explored to improve the performance and robustness of the implemented algorithms. Different epsilon annealing schedules could be tested to determine if they yield better results. For example, rather than a linear annealing schedule, an exponential or logarithmic schedule could be compared. As well, various hyperparameters such as the learning rate, discount factor, and replay buffer size could be adjusted to see how they affect the performance of the algorithms. Additionally, different neural network architectures could be tested to see if they improve learning speed or stability. Regarding varying the number of decks, future work could involve exploring the effects of increasing or decreasing the number of decks used in the game, which can impact the optimal strategy for the agent. Additionally, extending the state space further could involve incorporating additional features such as the current count of cards or the number of cards remaining in the shoe.

It would be interesting to explore the use of different reinforcement learning algorithms beyond the ones implemented in this project. For instance, Proximal Policy Optimization (PPO) and Trust Region Policy Optimization (TRPO) are popular policy gradient methods that could be implemented and compared with the current algorithms. Finally, running experiments on more powerful hardware would allow for the use of larger neural networks and training for more episodes. This would allow for more complex environments to be tackled and potentially improve the performance of the agents.

6 Conclusion

The paper demonstrates the effectiveness of reinforcement learning algorithms in optimizing blackjack strategies. The authors show that the Q-learning, SARSA, and DQN algorithms can be applied to learn optimal strategies for different variations of the game. By creating my own blackjack environments, I was able to customize and manipulate various aspects of the game, facilitating experimentation with different variations and rules. The experiments showed that the reinforcement learning algorithms performed significantly better than random, highlighting the effectiveness of reinforcement learning techniques for solving complex decision-making problems.

References

[1] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing Atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

[2] Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

[3] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ... & Dieleman, S. (2016). Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587), 484-489.

[4] Lan, T., Liu, Y., Hu, J., Zhao, Y., & Wu, Y. (2019). RLCard: A Toolkit for Reinforcement Learning in Card Games. arXiv preprint arXiv:1910.04376.

[5] Liu, Z. & Spil, G. Learning Explainable Policy For Playing Blackjack Using Deep Reinforcement Learning. [Online]. Available: http://cs230.stanford.edu/projects_fall_2021/reports/103066753.pdf