Learning to Play Blackjack using Deep Reinforcement Learning

## Chris Durbin

## Abstract

Using deep reinforcement learning to learn to play games has been successful for many different types of games. However there has not been much research in using deep reinforcement learning for playing Blackjack, especially with regards to tracking cards in the deck. In this paper, I demonstrate using deep reinforcement learning to both play hands effectively as well as learn to track the cards in a deck to determine when to bet large or small bets and gain a positive expected return over the course of many hands.

## 1 Introduction

Blackjack is a popular game played at casinos where players attempt to have a better hand than the dealer in order to win. With standard rules and without taking bet sizes into account the dealer has a slight advantage over the player even with optimal play, and therefore has a higher expected value. However, when a player keeps careful track of the cards that are remaining in the deck (referred to as card counting, a player can achieve an advantage over the dealer by varying the bet sizes.

In this project I attempt to achieve two goals using deep reinforcement learning. First, I attempt to train an agent to learn the best actions to take for a given Blackjack hand. Based on experimentation I found that with random action selection the average winning percentage for a player is 31%. My hypothesis is that using a deep reinforcement learning agent I can achieve an average winning percentage of at least 40% rejecting the null hypothesis.

The second goal of the project is to learn when to bet the maximum bet for a hand and when to bet the minimum bet for a hand using another deep reinforcement learning agent. There are two potential ways to test the null hypothesis in this case. The first is to choose random bet sizes and then compare the final balance of the deep reinforcement learning agent to the final balance of the random agent. The second is to have an agent always choose the minimum bet size since the expected value in Blackjack is negative and so it should have the highest expected value if there is nothing to learn from tracking the remaining cards in a deck. My hypothesis is that the deep reinforcement learning agent beats both of those agents with a confidence of 90%.

Note that for trying to track the remaining cards in the deck there are more than 1.7 billion unique states[[1]](#footnote-1). Table based algorithms such as Q-learning are not reasonable choices for such a large state space, and they cannot generalize so even if it would be possible to track using a table, it would take an excessive number of hands to be able to learn. A neural network based approach that can generalize to never before seen inputs is a much better choice in this scenario, which makes the Deep Q-learning agent a reasonable choice.

## 2 Related Work

There are relevant publications for both optimal play for the game of blackjack as well as for deep reinforcement learning that I relied on to help design my research project.

Vidámi describes how card counting (tracking the cards that have been seen to know what cards are remaining) can be utilized to improve the odds such that there are situations based on the remaining cards in the deck that the player has odds better than the dealer (Vidámi, 2020). In this situation, the player can increase their bet size to increase their expected return and have a positive expected value.

There is a large amount of prior work on using reinforcement learning to learn how to play games without any built-in knowledge of optimal play for the game. In the past ten years there has also been a large focus on using deep reinforcement algorithms. In a ground-breaking paper for deep reinforcement learning, Minh introduced the Deep Q-Network (DQN) algorithm which used a neural network and experience replay to estimate future rewards for a given state and all possible actions (Minh, 2013). The paper demonstrated success on many Atari games.

It is not guaranteed that the same kind of approach using a DQN will work to learn how to play Blackjack and one of the largest concerns with the approach to use deep reinforcement learning is the “Deadly Triad” described by Sutton and Barto. When combining function approximation, off-policy learning, and bootstrapping (such as temporal difference methods), learning can diverge, and the value estimates become unbounded (Sutton, 2018). One example technique to try was provided by Wang with the concept of dueling DQN networks which utilize two different neural networks when performing learning and identifying targets for the error calculation. They freeze the target network for each minibatch of experience replay learning so that the target does not keep updating as the examples from the batch update the weights during backpropagation leading to more stable weight updates (Wang, 2016).

I utilized the approaches from both Minh and Wang in order to train DQN agents for learning how to play hands and to determine bet sizes for Blackjack.

## 3 Methods

The project utilizes standard casino blackjack rules as follows:

* The player (agent) places a bet that is at least the minimum bet size.
* Both the player (the agent) and the dealer will be dealt two cards. The player’s cards will both be dealt face up and the dealer will have one card dealt face up and the other face down.
* If the player has a natural 21, they will automatically win 1.5 times their bet unless the dealer also has 21 in which case it is a tie. If the dealer receives a natural 21 and the player does not the dealer automatically wins, and the player’s bet is lost.
* Otherwise play continues. The player can choose to hit or stay. If the player hits they receive another card. If they have less than 21, they can continue to hit.
* If the total value of the player’s cards is greater than 21, they lose the hand.
* If the player has 21 or fewer and they choose to stay it is then the dealer’s turn.
* The dealer will always stay with 17 or higher and always hit with 16 or less. If the dealer has a total greater than 21 or the dealer stays with fewer points than the player, the player wins the hand.
* If both the dealer and player have the same value for their hands it is a tie.
* If the dealer has a greater value than the player, the player loses the hand.

When the player wins the hand, they receive a reward equal to their bet. If the player and dealer tie no reward is received, and on a loss the player loses their bet. The one exception is for a winning natural 21 on the first two cards the player receives 1.5 times their bet.

The project includes several components to implement the reinforcement learning environment:

* A blackjack environment that implements the mechanics and enforces the blackjack rules.
* A DQN agent to make the decisions for selecting actions for playing hands.
* A DQN agent to make the decision on the bet size to use for each hand.
* A driver to connect all the components together and run the experiments.

A diagram of a blackjacket

Description automatically generated

Figure 1: The workflow for a single episode showing the interaction between each component.

Both the bet size and the hand playing DQN agents use similar architectures. They use fully connected linear layers with an input layer, two hidden layers, and an output layer with the output layer producing the Q-values for each potential action. I use batch normalization for the layers and the Rectified Linear Unit (ReLU) activation function for each layer. When the network is used to select an action, it chooses the action that has the highest output value.

For action selection I use an epsilon greedy algorithm and decay the epsilon over time so that initially I perform a large amount of exploration selecting random actions and then by the end I am almost always exploiting. For this project the choice of epsilon is not as important as most reinforcement learning projects since there are exactly two actions which can be selected, and the rewards are always opposite for the actions meaning that agent will learn similar information with either selection.

I use experience replay for training my agents and use a separate target network with frozen weights that I update periodically to match the action selection network to increase stability during training. See table 1 below for the parameters used for training the DQN.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Min replay size | Minibatch size | Replay buffer size | Target network update frequency (steps) | Gamma | Learning Rate |
| 500 | 128 | 1,000,000 | 500 | 0.995 | 0.001 |

Table 1 – DQN learning parameters

To train my agents I play one million episodes using the workflow shown in figure 1 for each episode. For selecting a bet size, the agent can use one of two bet sizes. The minimum bet is represented by 10 and the maximum bet is represented by 500. I repeat each experiment using 10 agents and track performance metrics across the agents using both average and standard deviation.

The primary metrics I am tracking for the bet size agent are the final average balance for each of the experiments and the percentage of time the maximum bet is chosen.

For the hand playing agent I track the win rate over time.

## 4 Results

The results are presented for both the hand playing agent and the bet size selection agent.

### 4.1 Results for hand playing agents

For the hand playing agent I tracked the average win rate across all episodes for each of the agents. For the random agent the average win rate across the 10 runs was 0.3133. For the DQN agent the average win rate was 0.4465. See table 2 below.

|  |  |  |  |
| --- | --- | --- | --- |
| Agent type | Average win rate | Number of hands | Standard deviation |
| Random | 0.3133 | 10,000,000 | 0.0004 |
| DQN | 0.4465 | 10,000,000 | 0.0009 |

Table 2 – Average win rates for Blackjack hand playing agents

Based on the results and the minimal standard deviation, the DQN agent demonstrates clearly that it learned how to play hands much better than random and has an even better win rate than our hypothesis of 0.4.

### 4.2 Results for bet size agents

For the bet size selection agent, I performed experiments with three different types of agents. For each agent ten runs were performed with each run including one million episodes. I used the same DQN agent for playing the hands for each of the agents for these experiments. The two bet sizes used were a minimum bet of 10 and maximum bet of 500.

#### 4.2.2 Random bet size agent

The random bet size agent had an average final balance of -440,015 and standard deviation between runs of 379,227. Based on these results a 95% confidence interval range for the random agent is between -711,297 and -168,732. See figure 2 for the agent balance for each individual run over time.

A graph of different colored lines

Description automatically generated

Figure 2 – Random bet size agent balance over time

#### 4.2.3 Fixed minimum bet size agent

The fixed minimum bet size agent had an average final balance of -24,394 and standard deviation between runs of 10,595. Based on these results a 95% confidence interval range for the random agent is between -31,973 and -16,814. See figure 3 for the agent balance for each individual run over time.

A graph of different colored lines

Description automatically generated

Figure 3 – Fixed minimum bet size agent balance over time

#### 4.2.4 DQN bet size agent

The DQN bet size agent had an average final balance of 912,878 and standard deviation between runs of 910,698. Based on these results a 95% confidence interval range for the DQN agent is between 261,404 and 1,564,353. See figure 4 for the agent balance for each individual run over time.

A graph of different colored lines

Description automatically generated

Figure 4 – DQN bet size agent balance over time

#### 4.2.5 Bet size agent comparison

As expected both the random and fixed minimum bet size agents had negative average balances on average, and furthermore, with 95% confidence both agents are expected to have negative balances after one million episodes. The DQN agent on the other hand had a large positive average final balance of 912,878 and the minimum range of the 95% confidence interval was also positive at 261,404. Therefore with the maximum ranges of the random and fixed agents negative and the minimum range of the DQN agent positive, our hypothesis that the DQN agent would perform better than both the random and fixed minimum bet size agents with 95% confidence was correct. Figure 5 below shows the average final balance across the ten runs over time for each agent for easy comparison.

A graph of a graph with numbers and a line

Description automatically generated

Figure 5 – Average agent balances over time for each type of agent across runs

Finally, see table 3 below for a summary of the results of the experiments on the bet size agents.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Agent type | Average final balance | Standard deviation | 95% confidence interval min | 95% confidence interval max |
| Random | -440,015 | 379,227 | -711,297 | -168,732 |
| Fixed minimum bet size | -24,394 | 10,595 | -31,973 | -16,814 |
| DQN | 912,878 | 910,698 | 261,404 | 1,564,353 |

Table 3 – Average final balance and statistics for Blackjack bet size agents

## 5 Conclusions

Overall the project was interesting for me to work on and extremely challenging, especially trying to use deep reinforcement learning to learn when to choose the maximum bet size. It took many weeks of cycles of implementing, tuning, and testing to get it to reliably learn. In the end it was exciting to see that across ten million episodes over ten runs the average balance was positive indicating that the DQN agent was able to learn and appear to generalize over such a large state space.

An interesting next project to further the research would be to attempt to start with a balance and choose bet sizes to maximize return while ensuring the balance was always positive. The project would be setup to determine if a reinforcement learning agent could learn the Kelly criterion for choosing bet sizes.

Another avenue for additional research would be to see if the results for this project still hold if attempting to use multiple decks instead of counting cards with just one deck. Perhaps run experiments using two, four, and eight card decks to determine if an agent can still learn when to use maximum size bets over an even larger state space.

Finally, a similar architecture could be used to attempt to learn how to play and choose optimal bet sizes for other card games beyond Blackjack.

## References

1. Kelly, John L. "A new interpretation of information rate." *the bell system technical journal* 35.4 (1956): 917-926.
2. Riedmiller, Martin. "Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method." *Machine Learning: ECML 2005: 16th European Conference on Machine Learning, Porto, Portugal, October 3-7, 2005. Proceedings 16*. Springer Berlin Heidelberg, 2005.
3. Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." *arXiv preprint arXiv:1312.5602* (2013).
4. Schaul, Tom, et al. "Prioritized experience replay." *arXiv preprint arXiv:1511.05952* (2015).
5. Wang, Ziyu, et al. "Dueling network architectures for deep reinforcement learning." *International conference on machine learning*. PMLR, 2016.
6. Sutton, Richard S., and Andrew G. Barto. *Reinforcement Learning, second edition: An Introduction*. MIT Press, 2018.
7. van Hasselt, Hado, et al. "Deep Reinforcement Learning and the Deadly Triad." *arXiv preprint arXiv:1812.02648* (2018).
8. Vidámi, M., Szilágyi, L., and Iclanzan, D. "Real Valued Card Counting Strategies for the Game of Blackjack." *Neural Information Processing*, edited by H. Yang et al., vol. 12533, Springer, Cham, 2020, pp. 1-10. doi:10.1007/978-3-030-63833-7\_6.

## Appendix

The code is available in the GitHub repository: <https://github.com/cdurbin/jhu-aaml-final>

Examples of running experiments are shown in the README.

In addition to the agents referenced in the paper, I implemented two other agents as a baseline comparison for selecting actions for playing hands. One agent implements the first-visit Monte Carlo algorithm, and the other agent implements the Q-learning algorithm. Using those helped me verify the performance of the Deep RL agent for playing hands met my expectations.

1. There are 5 potential values each for the number of remaining aces, twos, threes, fours, fives, sixes, sevens, eights, and nines. There are 17 potential values for the number of remaining value 10 cards. There are 52 possible deck state dealt percentages. [↑](#footnote-ref-1)