Solving Blackjack with Q-Learning

Nathan Little and **Brad Shook**
{nalittle,brshook}@davidson.edu
Davidson College
Davidson, NC 28035
U.S.A.

Abstract

In this paper, we build a Blackjack AI using Q-learning, a model-free reinforcement learning algorithm. We explore different parameter settings and assumptions, tuning the Q-learning algorithm for Blackjack. We subsequently evaluated the performance of our Q-learning bot against a random player and a heuristic player. While our bot did not converge to an optimal or near optimal policy, it did clearly outperform the random player.

1 Introduction

We designed an AI which learns and subsequently utilizes a policy for Blackjack. Our Blackjack AI uses the model-free reinforcement learning algorithm Q-learning in order to develop a policy with which to play the game. We set out to determine whether Q-learning would yield a policy which maximizes returns, and we evaluated the efficacy of our reinforcement learning approach against the results of two different Blackjack strategies: a random player and a heuristic player, which utilizes an optimal static policy.

The goal of our Q-learning algorithm was to derive an optimal policy which minimized losses or, more optimistically, maximized winnings. Blackjack is typically understood to hold a four percent disadvantage to the player who adheres to an optimal strategy, so such players should expect to lose $4 for every $100 bet (Jensen 2014). In their 1956 paper, Baldwin et al. devise an analytic approach to Blackjack which yields a 0.62 percent disadvantage (Baldwin et al. 1956). Contemporary AI strategies often utilize Markov Chains since the game can be discretely modeled as a set of states, actions, transition probabilities and expected rewards, though such approaches often use card counting strategies to predict future states (Wakin and Rozell 2004). At a high level, reinforcement learning algorithms are appealing to solve Blackjack since Blackjack is episodic and provides clear rewards in the form of winnings and losses. Indeed, de Granville finds that a Q-learning approach to Blackjack converges to a near optimal policy, outperforming the random player by a large margin (de Granville 2017).

In this paper, we will describe the Q-learning algorithm used by our player, summarize the details of our experiments, and then evaluate the performance of our player against other strategies.

2 Background

Blackjack Domain

The objective of Blackjack is to obtain a higher number of points than the dealer without exceeding 21, where points are determined by cards held. Cards numbered 2 through 10 are worth their face value; face cards (Jack, Queen, King) are worth 10 points each; and aces are worth either 1 or 11 points. Our game consists of one player, the dealer, and one 52-card deck, shuffled prior to each game. The player bets $1 prior to each hand, and the player and dealer are dealt two cards each, where only the value of the dealer's first card is known to the player. The player can then choose to either hit, which entails being dealt an additional card, or stand with the current hand, ending the player's turn. If the player exceeds 21, the game is over and the dealer wins the player's $1. If not, the dealer hits until her points reach 17 or greater, in which case she must stand.

Blackjack players evaluate their hand relative to the dealer's card, often assuming that the dealer's hidden card is of value 10 (Jensen 2014). Since cards of value 10 account for 30 percent of cards in a 52 card deck, a plurality, they are the most likely point value for the dealer to hold. Such an assumption gives an approximate upper bound for the dealer's point total and is therefore a useful assumption to roughly estimate the actual state of the game. For instance, typical Blackjack wisdom holds that players ought to stand if the dealer shows a 6 and the player has some point total above 11. Since the dealer is likely to have 16 total points, she is likely to bust once she hits (which the dealer must do at sixteen). We explore the impact of the 10 point assumption in this paper.

Q-Learning

Q-learning is a model-free reinforcement learning algorithm. More specifically, Q-learning is a temporal difference algorithm which directly approximates the optimal state-action value function $Q^*(s,a)$, which is defined as the expected sum of discounted future rewards assuming that the agent takes action a in state s, acting optimally in

each later state (Russell and Norvig 2009). The optimal state-action value function is obtained by updating the state-action value function $Q(s, a)$ episodically, so in Blackjack, $Q(s, a)$ is updated after each game. The optimal policy is that which maximizes rewards over time. De Granville provides the following pseudocode for the Q-learning algorithm:

Initialize $Q(s, a)$ arbitrarily

For each episode:
  Choose a from s using policy derived from Q
  Take action a, observe reward r, and new state s'
  $Q(s, a) = Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
  $s = s'$
until s is terminal

Note that α refers to the learning rate, which determines the size of the update made each episode, and γ is the discount rate, which determines the weight of future rewards (de Granville 2017). For selecting an action to take from a given state, many Q-learning algorithms use the epsilon-greedy strategy. This strategy introduces randomness to the selection of actions which allows the algorithm to better train the policies and avoid local minima.

If each state-action pair is continually updated, Q-learning is expected to converge to the optimal state-action value function with probability 1 (de Granville 2017). Because Q-learning begins with an arbitrary model, it is expected to perform similarly to the random player at first. It should subsequently tend towards the performance of the heuristic player as it learns, developing a policy which approaches optimality.

Other Policies

In order to evaluate the success of our Q-learning algorithm, we compare its results to that of a random player and a heuristic player, which utilizes a basic, static policy.

- **Random Player**: randomly chooses an action, either hit or stand, at each given state where each action has an equal probability of being chosen.
- **Heuristic Player**: leverages a near-optimal static policy proposed by Howard (Howard 2016). The policy is given by the following set of rules:

Dealer shows	Player stands on (hard)
2,3	≥ 13
4,5,6	≥ 12
Otherwise	≥ 17

Dealer shows	Player stands on (soft)
9,10	≥ 19
Otherwise	≥ 18

Figure 1: Static Blackjack policy. Image courtesy of Howard 2016.

We call the heuristic policy static since the player adheres to the strict set of rules in order to make decisions at each given state. A typical optimal policy gives the player a 4 percent disadvantage, so we consider this policy near optimal.

The random and heuristic players set realistic lower and upper bounds, respectively, on the performance of our player. Our goal for the Q-learning algorithm was to devise a policy which tends towards, and ideally would exceed, the performance of the heuristic policy.

3 Experiments

In order to determine the efficacy of our Q-learning Blackjack player, we experimented with different parameter settings, testing each on 100,000 hands, and compared the results of 100,000 hands played by the Q-learning bot, the random bot, and the heuristic bot. Prior to playing the 100,000 round game, the Q-learning bot was trained on 1,000,000 hands. The lone deck was shuffled prior to each hand played, and each game consisted only of the dealer and a single player.

Experiment 1: Parameter Settings

In Experiment 1, we attempted to determine how two parameters, alpha and gamma, affected the performance of the Q-learning bot in terms of average rewards. The effects of alpha, the learning rate, were found by training 9 Q-learning policies, each with a different alpha value ranging from 0.1 to 0.9. Each of these policies had a gamma of 0.9 and an epsilon value of 0.1. Moreover, the effects of gamma, the discount factor, were derived by training 9 additional Q-learning policies, each with a different gamma ranging from 0.1 to 0.9. Each gamma test had alpha and epsilon values of 0.1. Each of these aforementioned policies were trained for 1,000,000 episodes. After training, 100,000 matches of Blackjack were played with each policy and the win differential was updated after each game. This experiment allowed us to determined if there were optimal values for these two parameters.

Experiment 2: Player Performance

The goal of Experiment 2 was to evaluate the performance of the Q-learning bot relative to the random bot and the heuristic bot. First, the Q-learning bot was given a set of parameters (.1 for epsilon, .1 for alpha, and .9 for gamma) and trained on 1,000,000 hands. Then, each bot played 100,000 hands of Blackjack. The average reward was updated after each match, and we used these reward averages to determine the varying performance among the bots. We also experimented with the assumption that the dealer's hidden card is a 10, again running 100,000 hands for policies trained with and without that assumption.

4 Results

Results of Experiment 1: Parameter Settings

Since we had a limited number of future states in each game, the first and last states in a game were close to each other. Thus, changes in gamma minimally impacted the performance of the Q-learning bot. Since our algorithm was not particularly expensive in terms of either space or time complexity, we were able to train our policy on a large number

of hands. Thus, alpha had minimal impact on the performance of our algorithm because we trained the policy on 1,000,000 hands. The lack of correlation between the alpha and gamma settings with average rewards is shown in Figure 2. This graph shows that these parameters had little to no effect on the average rewards of the Q-learning bot.

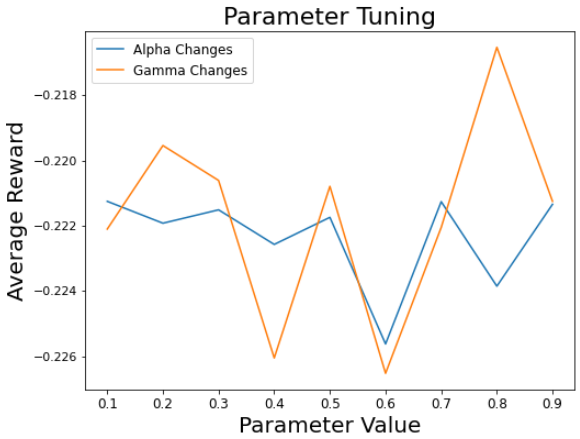

Figure 2: Comparison of average rewards for policies trained with varying alpha and gamma values.

Results of Experiment 2: Player Performance

As shown in Figure 3, our bot quickly outperformed the random player, reaching higher average rewards after only a few thousand hands. Subsequently, our bot failed to improve its performance, however. Figure 3 shows the Q-learning bot's early plateau: failing to improve at all after about 20,000 hands. In our experiment, the Q-learning bot failed to converge to an optimal policy or anything remotely close to optimal. Our bot averaged a reward of −22 percent whereas an optimal policy should make returns close to −4 percent. The random bot realized approximately −25 percent returns, just 3 percentage points worse than our player. These results suggest that our Q-learning bot was not able to effectively learn optimal policies that mirrored those used in the heuristic bot.

Furthermore, the assumption that the dealer's hidden card is of value 10 proved to have no impact on the performance of our player. We experimented with and without the assumption on different parameter settings, but the Q-learning bot realized average returns which were 99 percent similar, both converging to approximately −22 percent.

5 Conclusions

We set out to study the performance of a Q-learning algorithm applied to Blackjack. We determined that while Q-learning outperforms a random player, it failed to converge to an optimal strategy, which contradicts the results of de Granville. This result could be due to the fact that de Granville also used actions such as double-down, insurance,

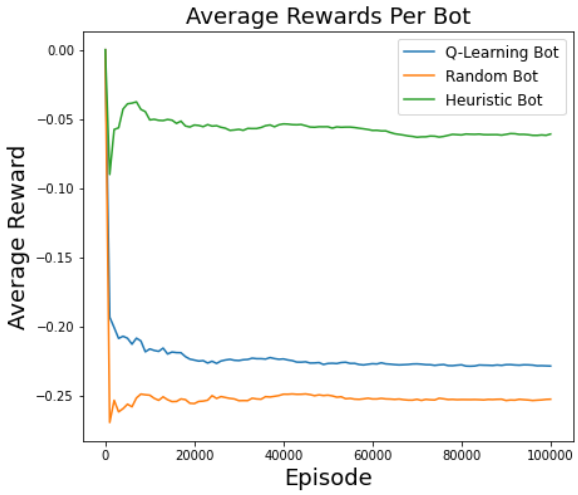

Figure 3: Comparison of average rewards for the Q-learning bot, random bot, and heuristic bot.

and splitting a hand, each of which allow for higher average returns in certain scenarios. In the future, we would expand our algorithm to include these additional actions. We might also define states more specifically, using the actual cards held instead of just the point value of the hand. This would provide our bot more information concerning the current state of the game, thus informing decision making.

6 Contributions

BS and NL contributed equally to the project. We did almost all of the coding collaboratively over VS Code LiveShare. On the paper, NL primarily focused on the Introduction and Background while BS spent more time on the Experiments and Results sections. That said, both BS and NL worked on every section of the paper. Both NL and BS collected data for the paper, but BS, in particular, collected most data and made the graphs.

7 Acknowledgements

We would like to thank Dr. Raghu Ramanujan for his insight and guidance throughout the process.

References

Baldwin, R. R.; Cantey, W. E.; Maisel, H.; and McDermott, J. P. 1956. The optimum strategy in blackjack. *Journal of the American Statistical Association* 51(275):429–439.

de Granville, C. 2017. Applying reinforcement learning to blackjack using q-learning. `https://www.cs.ou.edu/~granville/paper.pdf`. Retrieved on Nov. 10, 2020.

Howard, P. 2016. Optimal blackjack strategy. `https://www.math.tamu.edu/~phoward/m442/strategy21.pdf`. Retrieved on Dec. 9, 2020.

Jensen, K. 2014. The expected value of an advantage blackjack player. `https://digitalcommons.usu.`

edu/cgi/viewcontent.cgi?article=1528&context=gradreports. Retrieved on Dec. 9, 2020.

Russell, S. J., and Norvig, P. 2009. *Artificial Intelligence: a Modern Approach*. Pearson, 3 edition.

Wakin, M. B., and Rozell, C. J. 2004. A markov chain analysis of blackjack strategy. http://inside.mines.edu/~mwakin/papers/mcbj.pdf. Retrieved on Dec. 9, 2020.