
Applying Reinforcement Learning to Blackjack Using Q-Learning

Charles de Granville
University of Oklahoma

CHAZZ184@OU.EDU

Abstract

Blackjack is a popular card game played in many casinos. The objective of the game is to win money by obtaining a point total higher than the dealer's without exceeding 21. Determining an optimal blackjack strategy proves to be a difficult challenge due to the stochastic nature of the game. This presents an interesting opportunity for machine learning algorithms. Supervised learning techniques may provide a viable solution, but do not take advantage of the inherent reward structure of the game. Reinforcement learning algorithms generally perform well in stochastic environments, and could utilize blackjack's reward structure. This paper explores reinforcement learning as a means of approximating an optimal blackjack strategy using the Q-learning algorithm.

1 Introduction

1.1 The Blackjack Problem Domain

The objective of blackjack is to win money by obtaining a point total higher than the dealer's without exceeding 21. The game works by assigning each card a point value. Cards 2 through 10 are worth their face value, while Jacks, Queens, and Kings are worth 10 points. An ace is worth either 1 or 11 points, whichever is the most beneficial. Before the start of every hand, each player at the table places a bet. Next, the players are dealt two cards face up. The dealer also receives two cards: One face down, and one face up. The challenge is to choose the best action given your current hand, and the face up card of the dealer. Possible actions include hitting, standing, splitting, or doubling down.

Hitting refers to receiving another card from the dealer. Players may hit as many times as they wish, as long as they do not bust¹. If additional cards are not desirable, one may choose to stand. If a player receives two cards of the same value, it is possible to split, and play each card as a separate hand. For example, if two 7's are dealt, one may split, and play each 7 separately. The catch is that an additional bet of equal size to the original must be placed. At any given time a player may also choose to double down. This means doubling one's bet in the middle of a

hand, on the condition of receiving only one additional card from the dealer.

The stochastic nature of blackjack presents an interesting challenge for machine learning algorithms. A supervised learning algorithm constructs a model by learning on a labeled set of instances. One drawback with this approach is that it does not take the inherent reward structure of the game into account. An ideal blackjack strategy will maximize financial return in the long run. Reinforcement learning algorithms are particularly well suited for these types of problems. Also, the game can be easily modeled as a Markov Decision Process (MDP). These factors suggest that a reinforcement learning approach is appropriate for approximating an optimal blackjack strategy. The next section provides a brief overview of reinforcement learning, which is followed by a description of the Q-learning algorithm.

1.2 Reinforcement Learning

Reinforcement learning techniques rely on feedback from the environment in order to learn. Feedback takes the form of a numerical reward signal, and guides the agent in developing its policy. The environment is usually modeled as an MDP, which is defined by a set of states, actions, transition probabilities, and expected rewards (Sutton & Barto, 1998). Each action has a probability of being the action selected, as well as an associated value, which corresponds to the expected reward of taking the action. A greedy action is an action that has the greatest value. In order to learn, the agent must balance exploration and exploitation of the environment. During exploration, the agent tries non-greedy actions in hopes of improving its estimates of their values.

Value functions allow agent to compute how "good" it is to be in a given state. $V^\pi(s)$ is called the state-value function, and allows the agent to compute the expected reward of being in state s , and following policy π (Sutton & Barto 1998). $Q^\pi(s, a)$ is called the action-value function, and allows the agent to compute the expected reward of being in state s , taking action a , and thereafter following policy π (Sutton & Barto 1998). An optimal policy consists of the actions that lead to the greatest reward over time. The optimal state-value function is denoted by $V^*(s)$, and the optimal action-value function is denoted by $Q^*(s, a)$.

¹Busting refers to going over 21.

1.3 The Q-Learning Algorithm

The Q-learning algorithm is a temporal difference (TD) method that approximates $Q^*(s, a)$ directly (Sutton & Barto, 1998). TD methods such as Q-learning are incremental algorithms that update the estimates of $Q(s, a)$ on each time-step. The Q-learning algorithm is outlined below.

```
Initialize  $Q(s, a)$  arbitrarily
Repeat (for each episode):
    Choose  $a$  from  $s$  using policy derived from  $Q$ 
    Take action  $a$ , observe  $r$ , and  $s'$ 
     $Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$ 
     $s \leftarrow s'$ 
until  $s$  is terminal
Figure 1: The Q-learning algorithm
```

The parameter α is referred to as the *learning rate*, and determines the size of the update made on each time-step. γ is referred to as the *discount rate*, which determines the value of future rewards. The Q-learning algorithm is guaranteed to converge to $Q^*(s, a)$ with probability 1, as long as each state-action pair is continually updated (Sutton & Barto, 1998). The algorithm is well suited for on-line applications, and generally performs well in practice.

2 Implementation Details

The Q-learning algorithm is an excellent method for approximating an optimal blackjack strategy because it allows learning to take place during play. This makes it a good choice for the blackjack problem domain. Blackjack is easily formulated as an episodic task, where the terminal state of an episode corresponds to the end of a hand. The state representation consists of the agent's current point total, the value of the dealer's face up card, whether or not the hand is soft², and whether or not the agent's hand may be split. Certainly, more robust state representations could be used, but I wanted to keep the size of the state space at a minimum. Actions include hit, stand, split, and double down. All of the possible actions were included in an attempt to stay true to the game. The reward structure is as follows: For each action that does not result in a transition to a terminal state, a reward of 0 is given. Once a terminal state has been reached, a reward is given based on the size of the agent's bet. For example, if the agent bets 5 dollars, and wins the hand, a reward of +5 is given. If the agent loses the hand, a reward of -5 is

²A soft hand refers to a hand that contains an ace that is being counted as 11 points.

given. A static betting strategy is used because the state representation does not allow predictions to be made concerning which cards may appear in the following hands. A complete list of the rules adopted for the blackjack simulator are listed in Figure 2.

1. 6 decks (5 in play, and 1 as a buffer)
2. Dealer stands on soft 17
3. Double down allowed after splitting
4. No limit on the number of re-splits
5. Insurance is not offered
6. No surrender
7. Natural blackjack pays 3:2

Figure 2: Rules Adopted by Blackjack Simulator

3 Experimental Evaluation

3.1 Methodology

To evaluate the performance of my learning agent, two hard coded players have been implemented. The first player takes completely random actions, while the second follows a strategy known as basic strategy. Basic strategy cuts the casino's edge down to less than one percent, and is the optimal policy for the state representation adopted (Wong 1994).

My experimental approach consists of five runs, where each run is a cycle of training hands, and test hands. During training, the agent plays with four other reinforcement learning agents, all of which share the same lookup tables. During the test hands, the agent plays with the random player, and the basic strategy player. Each player starts with \$1,000, and makes the same initial bet of \$5. Within each run, the number of training hands is continually increased to a total of 10 million, while the number of test hands remains constant at 100,000. This procedure is summarized in Figure 3.

For each run

- While the number of training hands < 10 million
- Train the RL agent
- Reset each player's money total
- Test on 100,000 hands
- Increase the number of training hands

Figure 3: Experimental Procedure

The reinforcement learning agent's initial policy is completely random. Thus, one would expect it to perform similarly to the random player initially. As the number of training hands increases, the learning agent's policy should converge to basic strategy because the Q-learning algorithm directly approximates $Q^*(s, a)$.

3.2 Results

Figure 4 shows average performance results for each player over the five runs. The number of test hands remains constant at 100,000, while the number of training hands is continually increased to ten million. Notice that all of the players lose money. This is true even of the optimal policy, which demonstrates the difficulty of actually winning money playing blackjack. A comparison between the random player, and the learning agent reveals a significantly better performance by the learning agent. It is apparent that the agent is learning useful information during the training process. Also notice that the learning agent's performance asymptotically approaches the performance of the basic strategy player. This is to be expected because the Q-learning algorithm directly approximates the optimal action-value function $Q^*(s, a)$. Upon examination of the standard deviations between the performance of the basic strategy player, and the

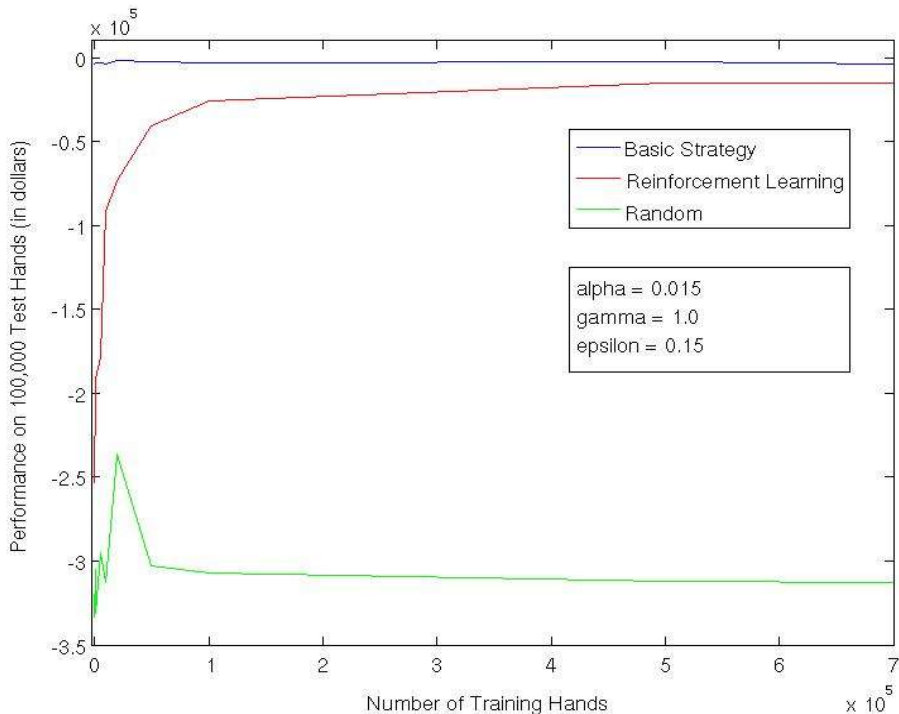
reinforcement learning player, it was revealed that they did not overlap. This means that the learning agent did not converge to the optimal policy, but rather a near optimal policy.

4 Future Work

While the results of the previous section indicate that the reinforcement learning agent is learning a near optimal policy, there is room for improvement. Currently, my learning algorithm uses an epsilon-greedy exploration strategy. An algorithm with a more sophisticated exploration strategy, such as the Bayesian Q-learning algorithm described by Dearden, Friedman, and Russel (1998), may yield better results. The Bayesian Q-learning algorithm maintains probability distributions over Q-values, in an attempt balance exploration and exploitation by maximizing information gain. Including more information in the state representation may increase performance as well. One could imagine the addition of a hi-low count. This, in conjunction with a dynamic betting strategy, may allow performance to surpass that of the basic strategy player.

5 Conclusion

This paper explored the use of reinforcement learning as a means of determining an optimal blackjack strategy using the Q-learning algorithm. It has been shown that the learning agent performed considerably better than random, and converged to a near optimal policy. These results are encouraging, but there remains room for improvement. A better exploration strategy, such as the Bayesian Q-learning algorithm, may lead to an improvement in the results. Also, a more robust state representation, together with a dynamic betting strategy, may lead to a policy that is more successful than basic strategy.



References

- Dearden, R., Friedman, N., & Russell, S. (1999).
Bayesian Q-learning. *American Association for
Artificial Intelligence*
- Sutton, R., & Barto, A. (1998). *Reinforcement Learning
An introduction*. MIT Press.
- Wong, S. (1994). *Professional Blackjack*. Pi Yee Press.