Playing blackjack using reinforcement learning

Qianying Wu

Stanford University
Stanford, CA 94305 USA
E-mail: wuqy@stanford.edu

Abstract

Blackjack is a classic Casino card game that involves competition between a player and a dealer. The winning criteria for the player is to have the sum of the numbers on their cards closer to but not over 21 than the sum of the numbers of the dealer's cards. In this project, we implement multiple reinforcement learning algorithms, including model-based value iteration and Q-learning, to find strategies that maximize the expected rewards of the player when deploying it in the blackjack game simulator environment. We examine the resulting policies in various scenarios, and evaluate the efficacy of the resulting policies by comparing the expected rewards with the benchmarks of random policy and a simple hit-and-hold policy.

I. Introduction

The rules of blackjack are as follows. The game starts with shuffled deck(s) of card. Only the number on the cards matters, not the suit of the cards. The Jack, Queen, and King are treated as 10. The Ace can be treated either as 11 or 1, depending on which is makes the sum of the cards closer to 21 but not exceeding 21. At the beginning of the game, the player places a bet. Both the player and the dealer are dealt with two cards. The player is able to observe their own cards and the first card of the dealer. If one of the two cards is an ace, and the other card is a ten, then it is a special winning case of "blackjack" where the player will win 150% of the bet. For other winning cases, the player will win 100% of the bet. Then, it is the player's round of action. The player chooses an action of 'hit' or 'hold', where hit means getting an extra card from the deck which adds to the the player's total sum of cards, and hold means staying still with the cards that the player already owns and actively choosing to end the player's round. The player can hit for as many times as they desire. Every time, the player can make the decision to hit or hold after observing the value of each new card. However, if the sum of the player's cards exceed 21, the player 'busts' and immediately loses the game. After the player's round, and if the player has not busted yet, it enters the dealer's round where the dealer takes actions to hit or hold. As is prevalent in most casinos, the dealer has a predefined strategy. The dealer will continue to hit until the sum of their cards is 17 or more. This is a single-agent game, since the strategy for the dealer is predefined and can be incorporated in the game environment.

II. Related work

As a popular single-agent Casino game, the optimum strategy of Blackjack based on probability theories in mathematics has been studied back in 1950s [1]. Recently, Blackjack has also been a popular environment for studying the efficacy of reinforcement learning, because of it is convenient to model as a Markov Decision Process (MDP), and it is also convenient to evaluate the policies using a large number of simulated games.

Blackjack has been modeled in the OpenAI Gym framework, which provides a standard method for defining the environment for researchers to experiment with different reinforcement learning algorithms. The framework also provides convenient APIs to define the states, actions, and rewards. The OpenAI Blackjack environment assumes an infinite card deck, where the probability of the cards to be dealt is not influenced by the cards that have been dealt. The OpenAI Blackjack framework also limits the action of the player to either hit or hold, and neglecting possible actions of doubling down and splitting, as often allowed in the casino Blackjack games.

Clifford Mao [2] expanded the OpanAI Blackjack framework to allow more actions for the player, and assumes a finite card deck. The BlackjackDoubleSplit-v0 environment allows the player to double down on the first two cards in the hand, and allows the player to split once on any matching pairs. It also assumes a finite number of decks of cards (one deck is referred to 52 cards of different numbers and suites). Based on this, the later developed BlackjackBet-v0 environment further accounts for the bet the the player places before each game, and defines an episode of the game as multiple rounds of blackjack using the same deck(s) of cards before shuffling.

III. Modeling approach

A. Formulation

Following the Blackjack environment modeling methodology of OpenAI Gym framework, we implemented the Blackjack environment in Julia. The player's options of actions are limited to only hold and hit for simplification of the problem.

The problem of playing Blackjack is formulated as a Markov Decision Process (MDP). The states are defined as tuples of the sum of the player's hand, the up card of the dealer's hand, and a logical value of whether there is at least one ace in the player's hand. The action space consists of hit and hold. The reward if the player wins is +1, with an exception of getting "blackjack" with the first two cards which gives a reward of +1.5. The reward is 0 if there is a tie, and -1 if the player loses. The MDP setup of the Blackjack problem is used for the model training and evaluations.

B. Model-based algorithm

Since the state space in this Blackjack problem is not too large, it is possible to use a model-based method to

generate a policy. In this method [3], we use off-line training to first generate the policy before policy evaluation. We first implement maximum likelihood method to approximate the transition function, which states

$$T(s'|s,a) \approx \begin{cases} N(s,a,s')/N(s,a) & N(s,a) > 0 \\ 0 & otherwise \end{cases} \quad (1)$$

where $N(s,a,s')$ is the occurrence of the transition from a state-action combination (s,a) to the next state s'. The reward associated with each state and action combination is estimated as

$$R(s,a) \approx \begin{cases} \rho(s,a)/N(s,a) & N(s,a) > 0 \\ 0 & otherwise \end{cases} \quad (2)$$

where $N(s,a) = \sum_{s'} N(s,a,s')$ is the number of occurrence of the state and action combination (s,a).

Then, the policy is found using value iteration using greedy approach, after which the policy is evaluated.

C. Q-learning algorithm

Q-learning is a model-free approach of reinforcement learning which is convenient to implement without explicitly formulating the transition functions of the MDP. Q-learning algorithm is used to learn to perform the optimal actions by approximating the optimal Q-functions, and is especially powerful when dealing with MDPs with state spaces which are not too large, such as in Blackjack. Q-function represents the expected reward for each state-action combinations. During training, it iteratively updates the estimates of the Q-values based on the rewards and transitions that can be observed. The Q-learning algorithm is summarized in Algorithm 1.

Algorithm 1 Q-learning algorithm

for i = 1:max˙iteration **do**
  0 ← Q(state, action) for all state-action combinations
  while The game is not over **do**
    action ← $\arg\max_a$ Q(state, action)
    reward ← reward(state, action)
    Q(state, action) ← Q(state, action) + α * (reward + γ * max([Q(next state, a) for a in actions]) - Q(state, action))
    state ← next state
  end while
  if Q-value convergence criteria is met **then** break
  end if
end for
  return Q-values

D. Model evaluation

The evaluation of the various policies are based on the criteria of expected reward. The expected reward (ER) per game is averaged over the reward (r) of 5 million games after training the algorithm, and is defined as

$$ER = \frac{1}{N}\sum_{i=1}^{N} r_i$$

where

$$r_i = \begin{cases} 1.5 & blackjack \\ 1 & other\ win \\ 0 & tie \\ -1 & lose \end{cases} \quad (3)$$

The ER criteria depicts the mean outcome of a certain policy, and 5 million hands are simulated to calculate the ER of policies in the following sections.

IV. Results and Discussion

A. Model-based parameters study

In model-based algorithm, the number of loops used in value iteration is a hyper-parameter that can be tuned (Fig. 1). However, we also note that the ERs resulted from various value iteration loops is stochastic and will change with different experiments. We parameterized the value iteration loops for 40 loops for the policy evaluation in the next section.

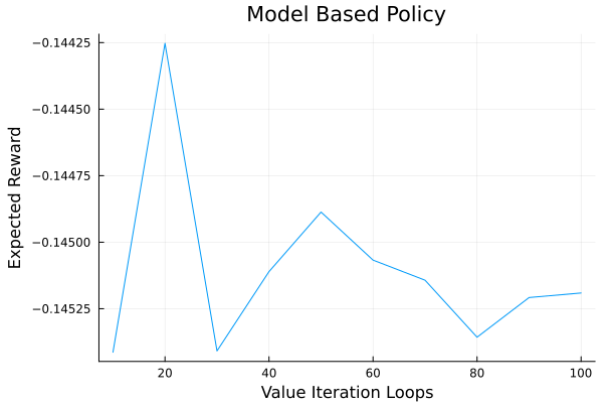

Fig. 1: Model-based expected reward (ER) as a function of the number of value iteration loops after exploration for 5 million hands and evaluated averaged over 5 million hands for evaluation.

B. Q-learning parameters study

In Q-learning algorithms, there are hyper-parameters that can be optimized to increase the expected reward per game. We choose to train the algorithm for 8000 epochs, and optimize the learning rate (α) for highest expected reward per game.

As depicted in Fig. 2, the expected reward (ER) stabilizes when the learning rate ranges from 0.0001 to 0.005, and significantly decreases when the learning rate is larger than 0.01, since large learning rate is associated with higher instability during the training. We will use a training rate of 0.0002 to generate the final Q-learning algorithm as used in the next section.

We also studied the influence of training epochs on the expected reward (ER). As shown in Fig. 3, with the training rate of 0.0002, when the training epochs is too high, the ER

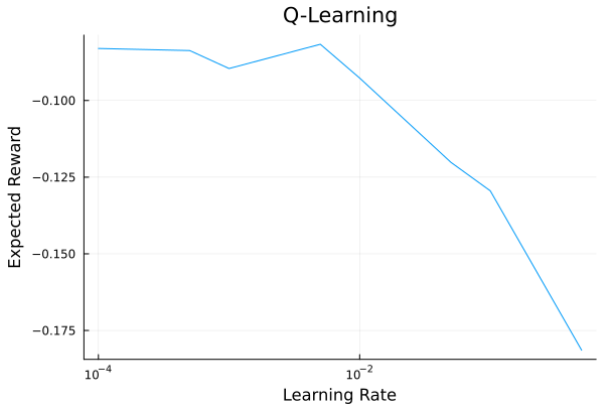

Fig. 2: Q-learning expected reward (ER) as a function of learning rate (α) after training for 8,000 epochs and averaged over 5 million hands for evaluation. The learning rate evaluated ranges from 0.0001 to 0.5.

will potentially decrease. This can be attributed to model over fitting to the training data. We will use training epochs of 8000 to generate the final Q-learning algorithm as used in the next section.

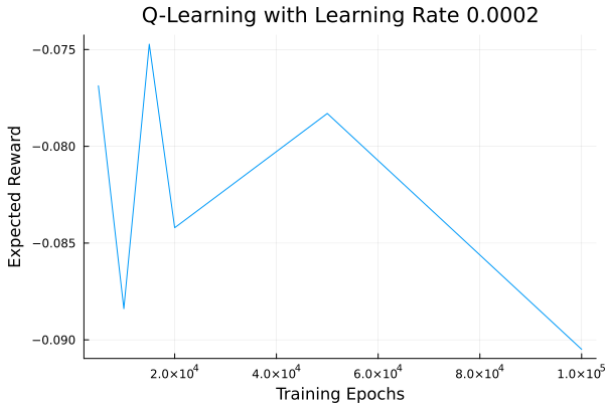

Fig. 3: Q-learning expected reward (ER) as a function of training epochs ranging from 5,000 to 100,000.

C. Resulting policy

The policies of the Model-based algorithm and the Q-learning algorithm are compared with two benchmark strategies. Random strategy is one of the benchmarks which states that the player randomly chooses actions of hold or hit regardless of the player's hand, or the dealer's up card. This results in a policy map shown in Fig. 4. Hold-and-hit strategy is another simple benchmark which states that the player will always hit when the sum of the player's hand is not more than 11, and hold when the player's hand is equal to or exceeds 12. The policy map of the hold-and-hit strategy is shown in Fig. 5.

The policy map of the model-based policy is shown in Fig. 6. The algorithm has learnt the basic trend of hitting when the sum of the player's hand is low, and holding when the sum of the player's hand is high and has the risk of busting. However, the threshold of the sum when the player should switch from

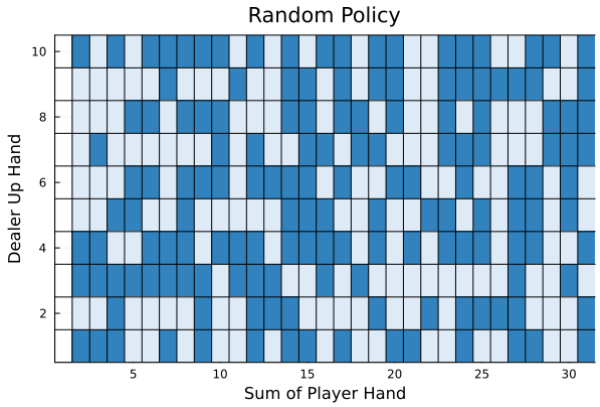

Fig. 4: Random policy map as a function of the dealer's up card and the sum of the player's hand. Dark blue corresponds to hit, and light blue corresponds to hold.

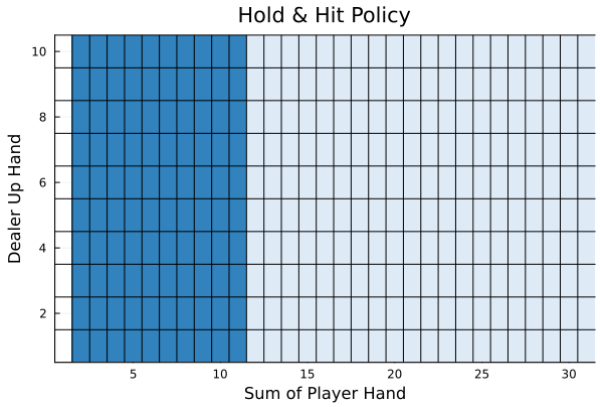

Fig. 5: Hold-and-hit policy map as a function of the dealer's up card and the sum of the player's hand. Dark blue corresponds to hit, and light blue corresponds to hold.

hit to hold is higher than the hit-and-hold policy, which could be overly aggressive. Meanwhile, we also note that when the sum of the player's hand is 2 or 3 and when the player does not have an ace, the policy suggests to hold, which will almost certainly result in a loss in the game (unless the dealer busts). And at this point it is 100% safe to hit which will increase the sum of the player's hand without any risk of busting. The reason that this aspect has not been learnt by the model-based algorithm is likely due to low probability of occurrence in the training process.

The policy map for the optimized Q-learning policy is shown in Fig. 7. The Q-learning policy is more conservative about hitting when the sum of the player's hand is on the large end, which is a wise move to prevent busting. However, the Q-learning policy also shows more discontinuity of hit or hold for adjacent cases (where there is only slight change in the dealer's up card, or a slight change in the sum of the player's hand).

D. Expected reward (ER)

The expected reward (ER) of the various algorithms are compared in Table I. The model-based policy increases the ER over the randomized policy, and the Q-learning policy further

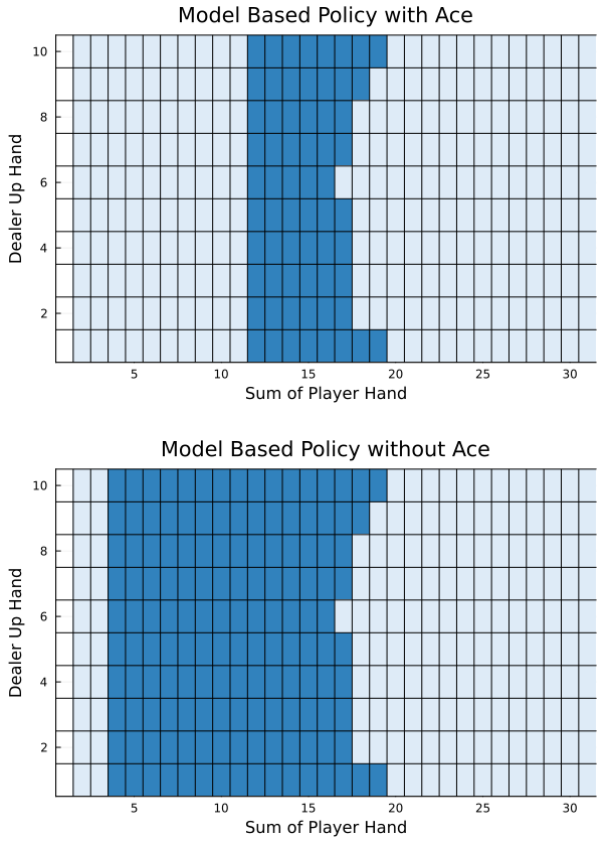

Fig. 6: Model-based policy map as a function of the dealer's up card and the sum of the player's hand. Dark blue corresponds to hit, and light blue corresponds to hold. Sub-figures depict scenarios with and without ace for the player's hand.

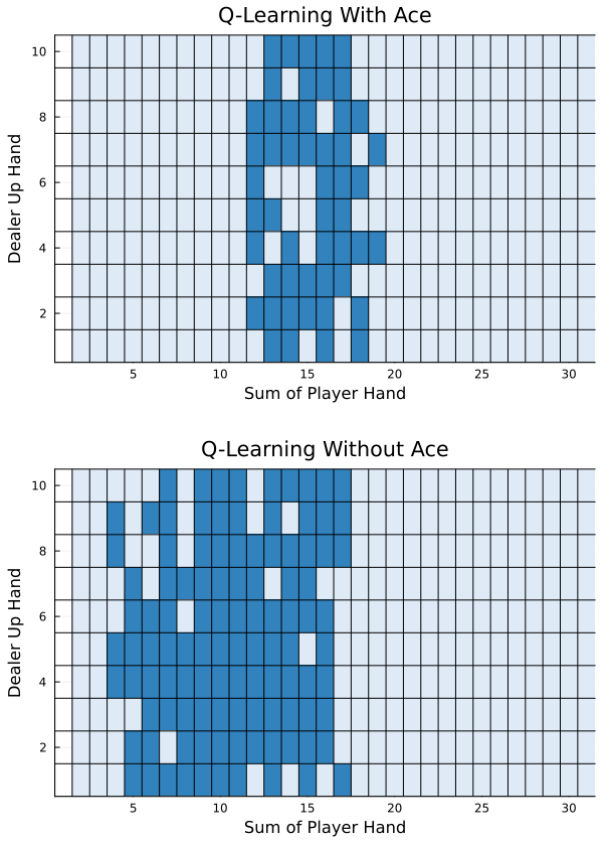

Fig. 7: Q-learning policy map as a function of the dealer's up card and the sum of the player's hand. Dark blue corresponds to hit, and light blue corresponds to hold. Sub-figures depict scenarios with and without ace for the player's hand.

increases the ER over the model-based policy. However, the ERs for these two policies generated by reinforcement learning are not as high as the simple hold-and-hit algorithm.

TABLE I: Expected reward (ER) of different algorithms

Expected reward (ER)	
Random	-0.2897
Hit-and-hold	-0.075
Model-based	-0.145
Q-learning	-0.076

V. Conclusion

With model-based algorithms and Q-learning algorithms, we are able to generate Blackjack policies that outperform a random policy, which shows the efficacy of reinforcement learning. However, it is surprising that both trained algorithms result in lower expected reward (ER) than the simple hold-and-hit strategy. Future work can be done in extending the Blackjack environment to consider wider action space including splitting and double-down. The definition of the state can be modified to include the number of aces in the player's hand, instead of a logical value of whether the player has any aces. This can better describe scenarios when the player has got two or more aces and could result in different action.

VI. Acknowledgments

Thanks to Prof. Mykel Kochenderfer and the teaching team for a fun quarter with MDPs and POMDPs.

References

[1] R. R. Baldwin, W. E. Cantey, H. Maisel, and J. P. McDermott, "The optimum strategy in blackjack," *Journal of the American Statistical Association*, vol. 51, no. 275, pp. 429–439, 1956. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/01621459.1956.10501334

[2] C. Mao, "Reinforcement learning with blackjack," *Californica State University Masters Thesis*, 2019.

[3] M. J. Kochenderfer, *Decision Making Under Uncertainty: Theory and Application*. The MIT Press, 07 2015. [Online]. Available: https://doi.org/10.7551/mitpress/10187.001.0001