PLAYING BLACKJACK WITH MARKOV CHAIN AND MONTE CARLO

PETER HE

1. Introduction

Some games have optimal playing strategies, while others do not. Blackjack, for example, has an optimized strategy has a well-known optimal strategy called "basic strategy." Other games, such as chess and Go, do not have a known optimal strategy. The critical difference between the two classes of games is that the number of possible states of the former is considerably lower than the number of possible states of the latter. There are 220 total states in blackjack; in contrast, the number of board configurations in Go is 10^{170}, a number that far exceeds the number of atoms in the known universe. As a result, the solutions for two classes of games are entirely different. While the first class of games typically can be solved under a Markov Decision Process framework, the solutions to the second class of games typically require Monte Carlo method. The first part of the paper aims to apply the Markov Decision Process to solve a simplified version of Blackjack. The second part of the paper introduces a more generalized Monte Carlo solution which will hopefully shed some light on finding an optimal strategy for games with much larger number of states.

2. Markov Decision Process – A Blackjack Application

2.1. **Blackjack.** There are two agents in Blackjack, the player and the dealer. The game is consisted of three stages. In the first stage, the player is dealt with two cards, both of which are known. The dealer is also dealt with two cards, one is known and the other is not. The player then decides to add an additional card (hit) as many times as they would like to. In this stage, if the card value of the player exceed 21, then the player goes bust and loses the game. Once the player stops hitting the card (stay), the game then proceeds to the second stage. In the second stage, the dealer reveals its second card and act according to the following policy. If the total card value of the dealer is strictly greater than 16, then the dealer takes another card (hit). Otherwise,

Date: November 2020.

Kind	Card Value
2, 3, 4, 5, 6, 7, 8, 9, 10	The corresponding kind
Jack, Queen, King	10
Ace	1 or 11

TABLE 1. Card Value Table

the dealer does not hit. Similar to the first stage, whenever the total card value of the dealer exceed 21, the dealer loses the game. Once the dealer stops hitting, the winner of the game is decided by their respective total card value. The card value of an individual card is a function of its kind, shown in Table 1.

2.2. **A Markov Chain Framework.** A Markov Decision Process is an extension of Markov Chains. In a Markov Decision Process, each state in the state space is associated with a reward, and its transitional probability function is parametrized a decision, or action, in addition to two arbitrary states. In this section, we describe Blackjack using the Markov Decision Process by defining the follow set of variables:

S : is a set of all possible states called the state space

A : is a set of all possible actions called the action space

$P_{a_t=k}(s_t = i, s_{t+1} = j)$ is the probability to transition from state $s_t = i$ to $s_{t+1} = j$ given action $a_t = k$

The state of Blackjack can be parametrized by the card value of the dealer and the player. If the card value of the player is greater than 21, then game ends, which means that card values greater than 21 can be ignored. The range of the player's card value is between 2 (two of Aces) and 21, inclusive. As for the dealer's card value, since only one of the two dealer's cards is known, the range of the dealer's observable card value is between 1 and 11, inclusive. Thus, the total possible number of states of Blackjack is $20 \cdot 11 = 220$. The state space can be described as a set of two-element tuples, where

$$S = \{(2, 2), \ldots, (2, 21), (3, 21), \ldots, (21, 21)\}.$$

There are two possible actions in the action space. The player either chooses to draw another card or not. Then,

$$A = \{1 : \text{hit}, 0 : \text{stay}\}.$$

Note that the dealer's policy is fixed and can be considered part of the game environment.

Formally, the transitional probability of a typical stochastic network is

$$P(s_t = i, s_{t+1} = j) = Pr(s_{t+1} = j | s_t = i).$$

Since there is a decision variable involved in the network, the transitional probability of reaching state s' is conditioned on the decision made, in addition to the previous state s,

$$P_{a_t=k}(s_t = i, s_{t+1} = j) = Pr(s_{t+1} = j | s_t = i, a_t = k).$$

In the case of Blackjack, $i, j \in S$ and $k \in A$.

2.3. **A Recursive Solution.** For each action we take, our goal is to maximize the probability of winning given the state of the game, which is the card value of the player and the dealer. One critical nuance of maximizing over the value of the decision d_t is that we assume future actions to also maximize the probability of winning. Since the current action can affect the future sequence of states and therefore the probability of winning conditioned on future states and future decisions, it is sensible to incorporate that impact in our maximization process too. Let $R(s_t = i, a_t = k)$ be the probability of winning when taking action k in state i given that future decisions $a_{t+1:T}$ maximize the probability of winning given future states. Our objective is to find the action in the current period t that maximizes the probability of winning,

$$\underset{a_t}{\textbf{argmax}}\ R(s_t, a_t).$$

If we can find $R(s_t = i, a_t = k)$ for all values of i and k, then we can solve Blackjack.

In order to derive the solution to find $R(s_t, a_t)$, it is quite helpful to examine the easy cases first to gain some intuition. Let $s_t = (21, 10)$, which indicates that the player has a card value of 21 and the dealer has a card value of 10. If the player hits another card, they are guaranteed to go bust. Hence, the probability of winning if hitting given $s_t = (21, 10)$ is 0. Furthermore, since there are no future transitions, we don't need to consider the critical nuance of maximizing the probability of winning conditioned that future decisions are also optimal. Thus,

$$R(s_t = (21, 10), a_t = 1) = 0.$$

Similarly, if the player stays, the game turns to the second stage, at which point the probability of winning is dependent on dealer's fixed policy of drawing cards and is easily calculable. Alternatively, we can use Monte Carlo method to estimate this probability. In the easy case, both decisions lead to the end of the game, which makes our calculation

quite simple,

$$0 = \underset{a_t}{\mathbf{argmax}}\ R(s_t = (21, 10), a_t).$$

Surprisingly, only a minor modification is needed to calculate the $R(s_t, a_t)$ in these cases. Let's consider the case of $s_t = (20, 10)$ and calculate

$$R((20, 10), 1), R((20, 10), 0)).$$

To begin, suppose that $a_t = 0$, then the game ends in a similar manner as the case examined above. The probability of winning can be determined by dealer's fixed policy, and $R((20, 10), 1)$ can be similarly determined using the Monte Carlo method. Now, suppose that $a_t = 1$. After the player hits a card, either the player goes bust or reaches to a new state of the game. The only possible state of the game is $(21, 10)$, at which point we should still take the action that leads to highest probability of winning,

$$\max_{a_{t+1}}\ R(s_{t+1} = (21, 10), a_{t+1}).$$

According to our previous analysis, since

$$0 = \underset{a_t}{\mathbf{argmax}}\ R((21, 10), a_{t+1}),$$

then

$$\max_{a_{t+1}}\ R((21, 10), a_{t+1}) = R((21, 10), 0).$$

In cases when the player's card value exceeds 21, the player loses the game. Thus, supose that $s_{t+1} = (l, 10)$ and $l > 21$,

$$\max_{a_{t+1}}\ R(s_{t+1} = (l, 10), a_{t+1}) = 0$$

By the law of total probability,

$$\begin{aligned}
&R(s_t = (20, 10), a_t = 1) \\
&= P_{a_t=1}(s_t = (20, 10), s_{t+1} = (21, 10)) \cdot \max_{a_{t+1}}\ R(s_{t+1} = (21, 10), a_{t+1}) \\
&+ \sum_{l>21} P_{a_t}(s_t = (20, 10), s_{t+1} = (l, 10)) \cdot \max_{a_{t+1}}\ R(s_{t+1} = (l, 10), a_{t+1}) \\
&= P_{a_t=1}(s_t = (20, 10), s_{t+1} = (21, 10)) \cdot \max_{a_{t+1}}\ R(s_{t+1} = (21, 10), a_{t+1}) \\
&+ \sum_{l>21} P_{a_t}(s_t = (20, 10), s_{t+1} = (l, 10)) \cdot 0 \\
&= P_{a_t=1}(s_t = (20, 10), s_{t+1} = (21, 10)) \cdot \max_{a_{t+1}}\ R(s_{t+1} = (21, 10), a_{t+1}).
\end{aligned}$$

Note that $P_{a_t}(s_t, s_{t+1})$ is the transition probability from s_t to s_{t+1} taking action a_t, as defined above. Even though we don't know $P_{a_t}(s_t, s_{t+1})$, it is easily obtainable using Monte Carlo method. The idea of law of total probability is quite generalizable to cases where s_{t+1} has several non-terminal states. Concretely, for any arbitrary state s_t,

$$R(s_t, a_t = 1) = \sum_{s_{t+1}} P_{a_t}(s_t, s_{t+1}) \cdot \max_{a_{t+1}} R(s_{t+1}, a_{t+1}),$$

where s_{t+1} represents all possible states that can be reached from state s_t taking the action a_t. Additionally, $R(s_{t+1}, a_{t+1})$ is defined recursively where the base cases occur when $a_{t+1} = 0$ or $s = (l, \cdot)$ where $l > 21$. In both cases, the value of R can be estimated using the Monte Carlo method. We can see that it is a function of transitional probabilities and the probabilities of winning at states where both decisions lead to the end of the game. The two sets of values can be approximated using Monte Carlo method as described above. Finally, the optimal decision at period t corresponds to the highest value of $R(s_t, a_t)$, in which case, we have found the optimal decision-making strategy in Blackjack,

$$a_t^* = \operatorname*{argmax}_{a_t} R(s_t, a_t).$$

3. A Monte Carlo Approximation

In this section, we will introduce a more generalized algorithm that solves the Blackjack problem without directly estimating transitional probabilities and explicitly calculating $R(s_t, a_t)$.

Our original definition of $R(s_t, a_t)$ is the probability winning conditioned on state s_t, action a_t, and assuming that future actions are determined by the same process of maximizing the probability of winning. The final condition gave us the recursive definition of $R(s_t, a_t)$ derived above. Then, $R(s_t, a_t)$ can be thought as a function that outputs the probability of winning conditioned on state s_t , a_t, and assuming that the actions are taken according to specific policy. The specific policy simply does not need to be the policy that maximizes the probability of winning. We can define a function $R_\pi(s_t, a_t)$ that outputs the probability of winning conditioned on state s_t, a_t, and a general policy π , where π is defined as a function that outputs an action given a state,

$$\pi(s_t) = a_t.$$

On a high level, $R_\pi(s_t, a_t)$ can also be thought as an evaluation of policy π in state s_t. After we evaluate the policy π at each state, we arguably have some information about π . Using this information, we

can improve the policy π . Perhaps over large number of iterations, we can reach the optimal policy. This intuition is the core of a more generalized Monte Carlo-based method. The following two sections address the implementation of policy evaluation and policy improvement.

3.1. **Evaluation.** Let $R_\pi(s_t, a_t)$ be the probability of winning given policy π and state s_t. We can use Monte Carlo method to find $R_\pi(s_t, a_t)$ by playing the game a large number of times following the same policy and recording the percentage of wins taking action a_t in state s_t. The detailed implementation is Algorithm 1.

Algorithm 1 Monte Carlo Policy Evaluation (π)

- **for** all possible state action pair (s, a) **do**:
  - $N(s, a) \leftarrow 0$: the number of times visited state s
  - $Q(s, a) \leftarrow 0$: the number of times winning the game after visiting state s
  - $R_\pi(s, a) \leftarrow 0$: estimated probability of winning after reaching state s following policy π.
- **for** $i = 1$ to Number of Monte Carlo Trials **do**:
  - episode$_i$ = generate an episode following the policy π
  - until the game ends
  - **for** state action pair (s, a) appearing in the episode **do**:
    - $G_i \leftarrow$ result of the game in episode$_i$
    - $Q(s, a) \leftarrow Q(s, a) + G_i$
    - $N(s, a) \leftarrow N(s, a) + 1$
    - $R_\pi(s, a) \leftarrow \frac{Q(s,a)}{N(s,a)}$

3.2. **Improvement.** A naive change to the current is simply choosing the action that maximize R_π given state s. Concretely, we can define a new policy function

$$\pi_2(s) = \underset{a}{\textbf{argmax}}\ R_\pi(s, a),$$

and simply updates the policy in each iteration of Monte Carlo trial. Other than experimentation, there is some intuition that justifies updating π to be π_2 . Action from π_2 yields the same or better probability of winning conditioned on that actions are taken according to the old policy π . Observe that for any state s,

$$\begin{aligned} R_\pi(s, \pi_2(s)) &= R_\pi(s, \underset{a}{\textbf{argmax}}\ R_\pi(s, a)) \\ &= \underset{a}{\textbf{max}}\ R_\pi(s, a) \\ &\geq R_\pi(s, \pi(s)). \end{aligned}$$

Whether

$$R_{\pi_2}(s, \pi_2(s)) \geq R_{\pi}(s, \pi(s)),$$

or policy π_2 provides overall the same or better probability of winning than the old policy π is unclear to me and most likely not true. In experimentation, while most of the time Algorithm 2 can reach the same $R(s,t)$ as in Section 2.3, it typically takes much more Monte Carlo trials and sometimes converge far from the optimal policy. This behavior suggests there could be some sort of local maxima trapping the searching process, which points to my suspicion that π_2 does not necessarily provides overall the same or better probability of winning. The specific implementation is presented in Algorithm 2.

Algorithm 2 Monte Carlo Policy Improvement (π)

- **for** $i = 1$ to Number of Monte Carlo Trials **do**:
  - episode$_i$ = generate an episode following the policy π
  - until the game ends
  - **for** state action pair (s, a) appearing in the episode **do**:
    - $G_i \leftarrow$ result of the game in episode$_i$
    - $Q(s,a) \leftarrow Q(s,a) + G_i$
    - $N(s,a) \leftarrow N(s,a) + 1$
    - $R_{\pi}(s,a) \leftarrow \frac{Q(s,a)}{N(s,a)}$
    - $\pi(s) \leftarrow \underset{a}{\textbf{argmax}}\ R_{\pi}(s,a)$

We can do better! In Professor Bowman's seminar, we talk about methods to escape local extremum, one of which is simulated annealing. The heart of simulated annealing is to use random exploration to escape local extremum. Specifically, simulated annealing has a cooling schedule that gradually decreases the level of random exploration throughout the Monte Carlo trials. Borrowing the idea from simulated annealing, we modify Algorithm 3 to allow the algorithm to sometimes randomly choose an action during episode generation, rather than strictly following the current policy π. Concretely, let B be a uniform variable between 0 and 1. During episode playout, the algorithm follow policy π^* such that

$$\pi^*(s) = \begin{cases} \pi(s), B > \epsilon \\ \text{A random action from the action space}, B < \epsilon \end{cases},$$

where ϵ is an arbitrary between 0 and 1.

Algorithm 3 Monte Carlo Policy Improvement - Random Exploration (π)

- **for** $i = 1$ to Number of Monte Carlo Trials **do**:
  - episode$_i$ = generate an episode following the policy π^*
  - until the game ends
  - **for** state action pair (s, a) appearing in the episode **do**:
    - $G_i \leftarrow$ result of the game in episode$_i$
    - $Q(s, a) \leftarrow Q(s, a) + G_i$
    - $N(s, a) \leftarrow N(s, a) + 1$
    - $R_\pi(s, a) \leftarrow \frac{Q(s,a)}{N(s,a)}$
    - $\pi(s) \leftarrow \underset{a}{\textbf{argmax}}\ R_\pi(s, a)$

By varying the threshold ϵ in policy π^*, we can adjust the degree to which the algorithm explores, disregarding what it already knows about the probability of winning.

4. Results

After running the Algorithm 2, 3, as well as the Markov Decision Process, Figure 1-4 in the appendix presents the optimal decision under each policy. In each run, our initial policy is always to randomly choose an action. The x-axis represents the card value of the player. The y-axis represents the card value of the dealer. The yellow color indicates that stay is the optimal decision where as blue color indicates that hit is the optimal decision. In mathematical terms, Figure 1-4 show

$$\underset{a}{\textbf{argmax}}\ R_\pi(s, a),$$

for all s given π. In Figure 4, we vary the threshold ϵ in Algorithm 3 by decreasing it with the number of Monte Carlo trials that have been run so far. In essence, the algorithm tend to explore a lot early during simulations. As it learns more about the game through simulations, it explore much less later on. Figure 5 to 8 presents probability of winning at each state given that the action taken is the most optimal under the respective policy. In mathematical terms, it is plotting

$$\underset{a}{\textbf{max}}\ R_\pi(s, a),$$

for all s given π.

5. Conclusion

The Monte Carlo Approximation is by no means as perfect as the first one. Additionally, it is much more computationally intensive to run

Monte Carlo simulations. That is not say that Markov Decision Process is always preferred in solving these problems. One of its critical advantage is that it doesn't require any knowledge of the transitional probabilities. In a game like Go with 10^{170} possible states, constructing a $10^{170} \times 10^{170}$ matrix is a challenging task. On the contrary, Monte Carlo Approximation can be applied to any games that can be described by a set of discrete states and actions. Admittedly, as the number of game states increases, the approximation can become more and more inaccurate. Nevertheless, modifications of this Monte Carlo approximation such as Monte Carlo Tree search is capable of leveraging modern advances in deep learning to parametrize, evaluate, and improve policies. To reduce the dimension of the search space, tt selects specific states to run simulations from rather than exhaustively iterating through all possible states. Using these modifications and more, Google's AlphaGo uses a Monte Carlo Tree Search algorithm and neural networks that beat the world Champion in Go, Lee Sedol. Hopefully, through a simple game of Blackjack, this paper is able to shed light on the two approaches in solving games.

6. Appendix

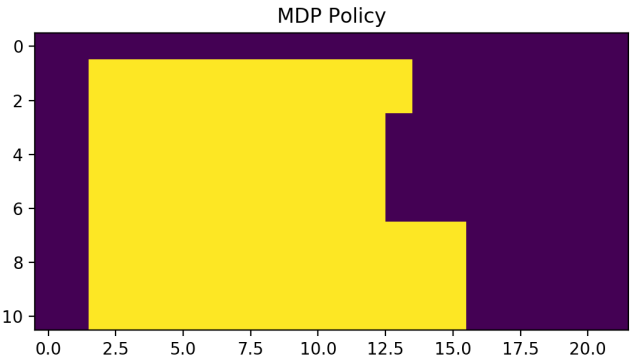

Figure 1. Optimal Decision from Markov Decision Process

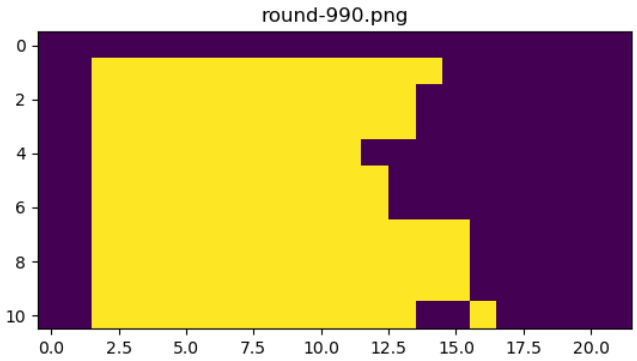

FIGURE 2. Optimal Decision from Algorithm 2

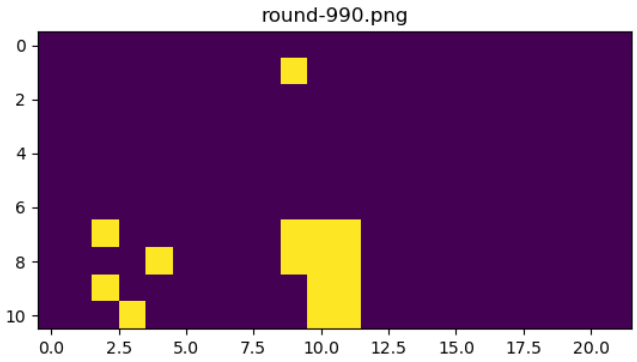

FIGURE 3. Optimal Decision from Algorithm 3, $\epsilon = 0$

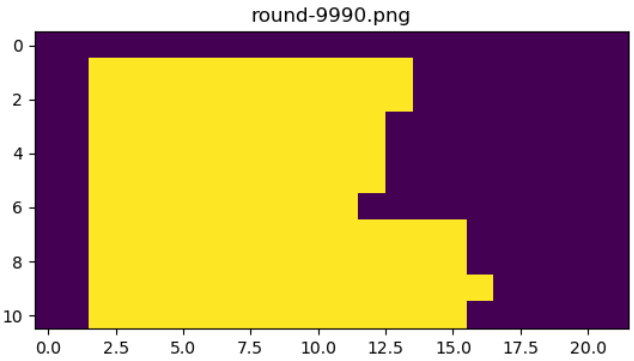

FIGURE 4. Optimal Decision from Algorithm 3, ϵ decreases with i

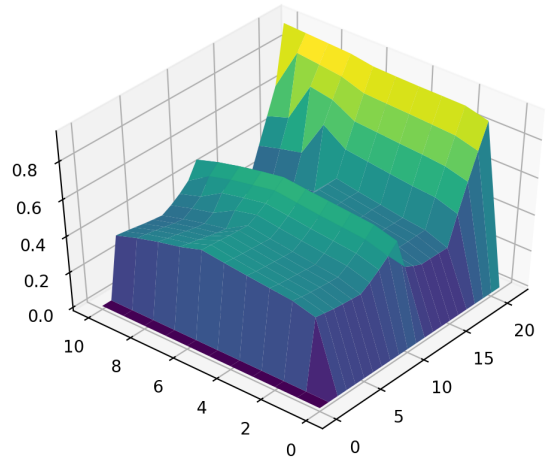

FIGURE 5. Optimal Decision from Markov Decision Process

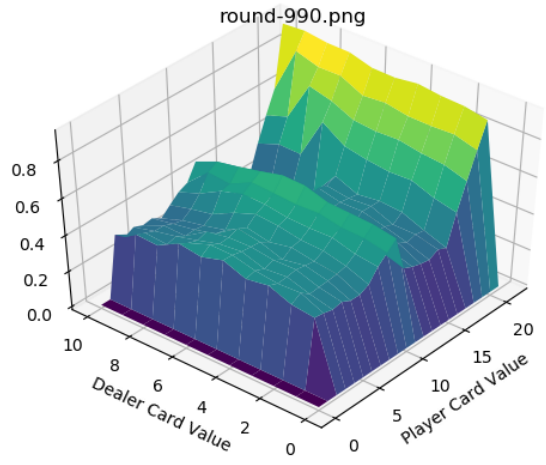

FIGURE 6. Optimal Decision from Algorithm 2

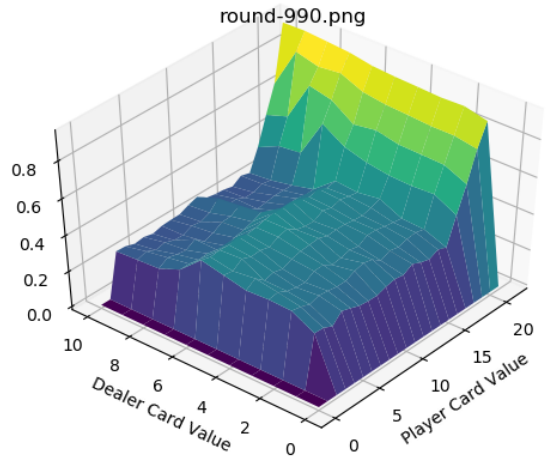

FIGURE 7. Optimal Decision from Algorithm 3, $\epsilon = 0$

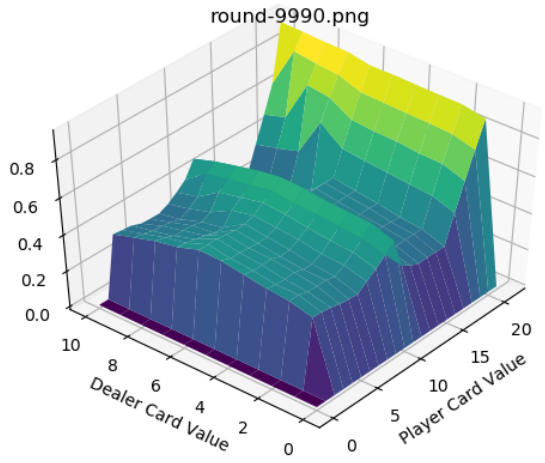

FIGURE 8. Optimal Decision from Algorithm 3, ϵ decreases with i

REFERENCES

[1] Fu, *Monte Carlo Tree Search: A Tutorial*
[2] Silver, *Introduction to Reinforcement learning with David Silver*
[3] Sutton, Barto, *Reinforcement Learning: An Introduction*