BlackJack Reinforcement Learning

In this project I will be using OpenAIs' Gym[1] and a common Reinforcement Learning technique called Monte Carlo to learn how to play BlackJack. One of the highest win rates you can have when playing BlackJack at a Casino is 40%.[2] So my expectations for this project is not to create an AI agent that will guarantee you win money the next time you visit a Casino; as that is impossible. However, I want to see if by implementing the Monte Carlo method, can this AI learn to play BlackJack at an optimal level. What I define as the "optimal level" is if it can reach or get close to the 40% win rate. Before continuing with the project, I will quickly explain the rules of BlackJack and a common optimal strategy players use.

What is BlackJack?
The objective of the game is simple. Each player wants to beat the dealer by getting as close to 21 as possible without going over.[3] Every card has a value amount it represents. Numbered cards are simply the amount they say that are. Picture cards (Jack, Queen, King) are 10s. Ace's are 11 or 1 (the player can decide the value between these two options). Players are given two cards to begin with and the dealer is given one. The player can then decide to either hit - get another card, or stand - stop receiving cards. The dealer must keep hitting until they reach 17 or higher.

Now that we know the rules. The most common optimal strategy is as follows:
- Hit on 16, 15, 14 and 13, but stand against a dealer showing two to six.
- Hit on 12, but stand against a dealer showing four to six.
- Hit on anything 11 or lower

There are some other rules recommended to follow. But these three are the main rules that would increase your chances the most. These three rules are what I'm hoping the AI agent can learn through Reinforcement Learning.

What is Monte Carlo and Why am I using it?
Monte Carlo is a simple Reinforcement Learning algorithm that is commonly used when there is no model. In other words, the agent learns about the states and rewards when it interacts with the environment[4]. This method of RL learns state values under a policy π by using the average return from all sampled episodes. If we included card counting and a lower deck count, BlackJack could be argued as a modeled environment. However, most Casinos nowadays play with 8+ decks that are continuously shuffled, and the gym environment I am using "is played with an infinite deck (or with replacement)". With these two points, we can say that our version of BlackJack is a model free environment, which makes Monte Carlo RL a good option for this problem.

$$V_\pi(s) = \sum_a \pi(a|s) \sum_{s' \in S} \sum_{r \in R} \underbrace{\boxed{p(s', r \mid s, a)}}_{\text{Unknown!}} (r + \gamma V_\pi(s'))$$

[1] https://www.gymlibrary.ml/
[2] https://www.playusa.com/blackjack/odds/
[3] https://bicyclecards.com/how-to-play/blackjack/
[4] https://towardsdatascience.com/monte-carlo-learning-b83f75233f92

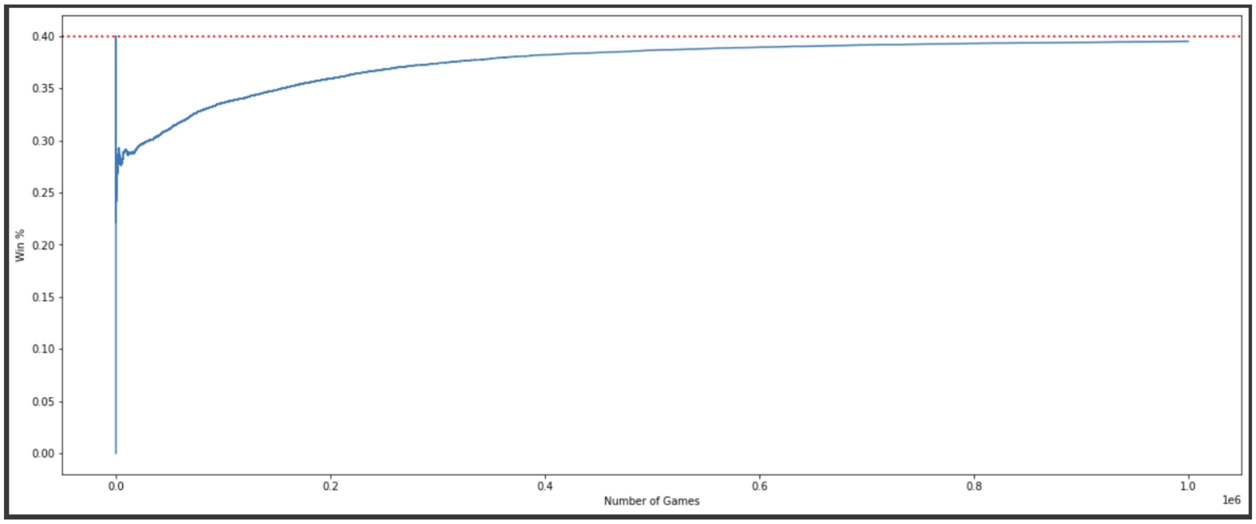
Decisions and Results

A lot of the smaller parameters such as the alpha and gamma values I was able to fine tune by trial and error. Once I was happy with these, I ran the algorithm on 10,000 episodes. The results were not very good and I was confused at first. But after adjusting the epsilon min value to 0.05 and changing the number of episodes to a much higher number, the results started to look more as expected. One big decision that I had to figure out when creating this project was what I was going to use as my Policy. I decided to go with a simple Epsilon-Greedy Algorithm[5].

$$\pi(a|s) \longleftarrow \begin{cases} 1-\epsilon+\frac{\epsilon}{|A(s)|} & \text{if a maximizes} Q(s,a) \\ \frac{\epsilon}{|A(s)|} & \text{else} \end{cases}$$

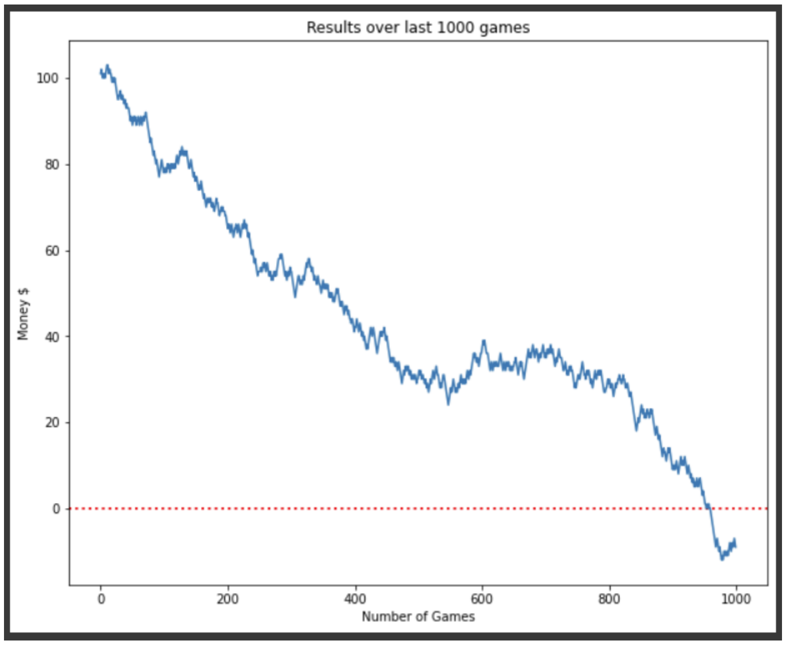
This seemed like a good fit for my agent and the game of BlackJack. Epsilon Greedy is a nice blend between exploration and exploitation which seemed to be a good option for this project. When epsilon is large we explore. When epsilon is small we exploit it. However, after looking at the results I am not too sure if this was the best option. I was able to receive results close to the range that I was expecting, but not 40% or higher.

This is the result of running through 1 million games! We can see that the agent's win rate just nearly reaches 40% which is the target I was aiming for.



Just for fun, I also plotted the results of the final 1000 games to see how our $100 would go in Vegas!!

[5] https://www.geeksforgeeks.org/epsilon-greedy-algorithm-in-reinforcement-learning/



We can see that we would sadly lose our money…as expected.

Conclusion

Overall though, I am very happy with the final results. I was never expecting this project to return a win rate over 50%, or really even over 40%. But looking at the results it's fun to see how the agent was able to figure out a way of playing to achieve an optimal outcome. I would be curious to see what would've happened if I created a policy that more closely follows the BlackJack strategy, similar to this article here[6]. In the future when I have some more time, I would like to experiment with other policy options and maybe even a model based option where I create a limited number of decks that the agent could use as information to count cards!!

Video:

https://youtu.be/RbyIOQ2IE_w

[6] https://towardsdatascience.com/cracking-blackjack-part-1-31da28aeb4bb