Group G - SDSC4001 Foundation of Reinforcement Learning

By CHAN Chak Yan. CHOY Ka Hei, LEE Chun Wing, YUEN Ho Man.

Topic : Playing BlackJack with Reinforcement Learning Algorithms

1.Introduction

1.1 Background

Around the world, BlackJack has been one of the most popular card games, whether for a party game, for competitions, or in the casino.It is commonly used for studies and research in reinforcement learning, and in other learning algorithms like neural networks. Not only is its system of rule well defined, there are also well developed strategies that were discovered by others.

This project aims to study and understand Blackjack by reinforcement learning, through the environment in OpenAI Gym.

1.2. Blackjack Game's Rule

The objective of the game is to obtain poker cards, excluding jokers, such that the total value of the cards is higher than the dealer's without exceeding 21. This game works with replacement of poker cards and starts with one dealer having one face-up and one face-down card, while the player has two face-up cards. There are two actions : the first one is that the player can then request an additional card, which is called a 'hit'. The other one is that the player can request a stop, which is called a 'stick'. After the player sticks, the dealer reveals their facedown card and draws until the dealer's card value sum is 17 or greater. For both the player and dealer, if the card value sum of the player's hand exceeds 21, then the player loses, and the dealer wins, which we can call a 'bust, vice versa'. If they are not busted after the player and dealer finishes all their action, then if the player's card value sum is greater than the dealer's card value sum, then the player wins, the dealer loses, vice versa. The card values are defined as follows : Face cards including Jack, Queen, King have a point value of 10 while numerical cards (2-9) are worth their face value, and an ace counts as 1 or 11 which is called a "usable ace".

To define a Blackjack game cleary, we describe it in pseudo-code format as follow:

```
Game episode loop:
        player's hand ← two face-up cards
        dealer's hand ← one face-up and one face-down card
        Player round loop:
                if player hits:
                        player's hand ← player's hand + one face up card(draw a card)
                        if player's card value sum > 21:
                                Player busted, dealer wins, player loses.
                else if player sticks:
```

```
                Leave the player round loop.

    Dealer reveals the face down card.
    Dealer round loop:
        dealer's hand ← dealer's hand + one face up card(draw a card)
        if dealer's card value sum > 21:
            Dealer busted, player wins, dealer loses.
        if 21 > dealer's cards' total value  > 17:
            Leave the Dealer round loop.
    if dealer's card value sum > playler's card value sum:
        Dealer wins and the player loses.
    else:
        Player wins and the dealer loses.
```

1.3. Motivation

Considering an optimal Blackjack strategy is a difficult challenge due to its stochastic nature. While a supervised learning algorithm is simple and may provide a viable solution, it does not take advantage of the inherent reward structure of the game. The alternation, reinforcement learning algorithms generally perform well in stochastic environments, and its goal is to maximize the problem of the financial return under a finite round. On the other hand, with the Markov property of the game, it can be easily formulated as a Markov Decision Process problem. The state and action can be limited and the rules and reward system are well-defined. These factors suggest that a reinforcement learning approach is appropriate for approximating an optimal blackjack strategy .

1.4 The main Idea

In this paper, its goal is to obtain the possible optimal strategy on playing Blackjack, which maximizes the rewards of the players in the game. In order to compare different approaches, it covers various Reinforcement learning Algorithms : Monte Carlo Control method , Temporal-Difference Learning , SARSA and Q-learning to verify the optimal policy for BlackJack.

2. Theory / Algorithm

2.1 Exploration-Exploitation Dilemma and ε-greedy policy

The basic problem encountered by Reinforcement Learning (RL) algorithms is Exploration versus Exploitation's Trade-off. Exploration in RL is a long-term benefit concept such that the agent aims to improve its knowledge about the environment and each action at any given time step. Therefore the agent may execute a less optimal policy or actions, which cause the loss of immediate rewards. Exploitation is a greedy approach in which agents aim to maximize the total rewards based on the current estimated value instead of the true value. It implies that the agent makes the best decision given information while the total rewards at the end may not be optimal. These two different strategies come up with a question "Should the algorithms spend more time exploring unknown states, actions for each state, associate

reward and the transition probability, or should it choose the best action on updating the Q-value with observable information from its history?". It indicates a trade-off between exploration and exploitation under the RL environment.

In order to balance the trade-off between Exploration and Exploitation, we adapt an ε-greedy policy / exploration on our algorithms. The idea is to pick a greedy action with probability 1-ε , otherwise, choose a random action uniformly with probability ε.

$$\pi(a \mid s) = \begin{cases} \epsilon/A + 1 - \epsilon & \text{if } a = \arg\max_{a \in \mathcal{A}} Q(s,a) \\ \epsilon/A & \text{otherwise} \end{cases}$$

2.2 Monte Carlo Methods

Monte Carlo methods are ways of solving the reinforcement learning problem based on averaging sample returns. They require only sample sequences of states, actions, and rewards from actual or simulated interaction with an environment and are using randomness to solve problems that might be deterministic in nature. For the Blackjack problem here, we only consider the Monte Carlo Control Method. For the model-free policy iteration methods like Monte-Carlo control, we always consider two key steps, which are policy evaluation and policy improvement. In other words, our objective to apply model-free policy iteration is to first evaluate the action value function, then generate a better policy for the next state and action. As greedy policy improvement over Q(s,a) is model-free, we will apply greedy policy for Monte Carlo methods.

$$\pi'(s) = \arg\max_{a \in \mathcal{A}} Q(s,a)$$

We need to compute Q(s,a) instead of V(s) to avoid the use of the MDP model.

In this project, we will use the Monte Carlo First Visit Control method to simulate an environment for reinforcement learning. Below is the baseline algorithm:

Initialization: N(s,a) = 0, G(s,a) = 0, Qπ(s,a) = 0, ∀s ∈S, ∀a ∈A
For loop (looping over episodes i):
Get episode observations
Define return G in step t.
For every state-action pair visited in episodes i, and for the first time t that (s,a) is visited in episodes i.
N(s,a) = N(s,a)+1
G(s,a) = G(s,a)+1
Qπ(s,a) = G(s,a)/N(s,a)

Compared to Temporal-Difference Learning, which will be covered in the next section, Monte Carlo is having better convergence properties. Also, it is not very sensitive to initial value, and is very simple to use. Applying first visit Monte Carlo control method, we will have a relatively high variance but zero bias environment.

However, playing Blackjack only by applying Monte Carlo control methods will not be enough. Next up, we will come up with an important concept used in our project.

2.2.1 Greedy in the Limit of Infinite Exploration (GLIE) with ε-greedy policy

'Greedy in the Limit of Infinite Exploration' suggests that an exploration strategy is GLIE if all state-action pairs are explored infinitely many times and converges on a greedy policy. For example, for an ε-greedy policy, if epsilon reduces to zero at ε = i / k at time step k, then it is GLIE. Therefore, a Monte Carlo method can be guaranteed to converge by applying an ε-greedy policy. Below is an example of GLIE Monte Carlo Control algorithm:

Initialization: Q(s,a) = 0, N(s,a) = 0, $\forall s \in S$, $\forall a \in A$
For loop (looping over episodes i):
    Set epsilon <- 1/k, πk = epsilon-greedy(Q)
    Get episode observations
    Define return G in step t.
    For every state-action pair visited in episodes i, and for the first time t that (s,a) is visited in episodes i.
        N(s,a) = N(s,a)+1
        Q(s,a) = Q(s,a)+(1/N(s,a))*(G-Q(s,a))

However, for epsilon <- 1/k in each time step k, we discover that it may not be that efficient to make the learning algorithms converge through training. Therefore, instead of initializing epsilon, we apply a decay rate for the epsilon to decay. Below is the algorithm:

For loop (over episodes):
    epsilon_start=1.0
    epsilon_decay=0.99999
    epsilon_min=0
    epsilon = eps_start*(eps_decay^(episodes-1))

By applying this algorithm, we can control the decay speed of epsilon, to let the agent explore more or stay more safe.

2.3 Temporal-Difference Learning

Temporal Difference Learning focuses on predicting a variable's future value in a sequence of states. It is able to replace complicated reasoning with simple learning procedures and generate the same results. In such a case, it is able to predict the chance of winning based on different actions and tell whether we should hit or stick in order to have a larger sum of value while not being busted.

The formula is given as:

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$$

The value of [rt+1+γV(st+1)–V(st)] is called the TD Error. It is the difference between the discounted value estimate of Vt+1, current estimate for Vt and the actual reward gained from transitioning between st and st+1. The error in Vt will be corrected slowly during the long process.

As for α, it is a constant step-size parameter which affects the speed of how the Temporal Difference algorithm learns. It makes the algorithm most sufficient at the value of around and below 1, as the smaller the value is the smaller the changes made for each update, and thus the slower the convergence.

Instead of trying to calculate the total future reward, temporal difference learning predicts the combination of immediate reward and its own reward prediction at the next moment in time. At the next state, there will be new information, the new prediction will be compared with the expected one. If these two predictions are different from each other, it will then calculate the difference between the predictions and use this temporal difference to produce a new prediction by adjusting the old one.

As for the advantages of Temporal-Difference Learning, it has lower variance as it depends on one random action, transition and reward. It is also able to learn online and learn from incomplete sequences, allowing it to tackle continuous problems.

2.3.1 State–Action–Reward–State–Action (SARSA)

SARSA belongs to a on-policy temporal difference method. SARSA has 5 features in a current state(t), including the current state(St), action(At), the reward received by the agent(Rt+1), next state(St+1) and action in next state(At+1).

The formula of SARSA is given as:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \cdot Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$$

where alpha is the step size, gamma is the discount factor.

The value of alpha is between 0 to 1. For the value equal to 0, the agent will stop learning. For the value equal to 1, the agent will only focus on the closest information.

The value of gamma is between 0 to 1. For the value equal to 0, the agent will focus on the current reward received. For the value equal to 1, the agent will attempt for a future reward.

When compared with the Monte Carlo algorithm, The on-policy algorithm allows the SARSA model to update its policy depending on the actions taken.

The algorithm of SARSA is described as:

1. Initialize the Q value (Q(s,a))
2. Give a observation to the state (S0=s0)
3. Choose an action (At) based on ε-greedy policy π0

4. Take the action A0~π0(S0), and observe the reward, R1, also the new state,S1.
5. Repeat the following steps for each episode until terminate(t=0,1,2…):
    5.1. Take action At+1 ~πt(St+1) and observe (Rt+2,St+2)
    5.2. Update the Q-value for the state with the observed reward and expected reward for the next state.
    5.3. Update the policy πt+1 with εt+1-greedy(Q)

The method of SARSA is applied to play Blackjack, the environment is provided by OpenAI's gym module. In this application, we want to perform the convergence of SARSA in obtaining the result where Sarsa converges to the optimal action-value function, Q(s,a) →q*(s,a).

Therefore, we set the policy is satisfies the condition of GLIE and the stepsizes αt satisfy:

$$\sum_{t=1}^{\infty} a_t = \infty, \sum_{t=1}^{\infty} a_t^2 < \infty$$

With dealing with the tradeoff of explore and exploit, With this algorithm, we would like to explore more at the first and eventually stay on the best action. The value of ε will be decreasing during the training progress.

2.3.2 Q-learning (SARSAmax)

Q-learning, as known as SARSAmax, is a popular algorithm in reinforcement learning. It belongs to an off-policy temporal difference method. The target of Q-learning is to find the best action in the current state. With the idea of off-policy, the algorithm will learn from action which is not from the current policy.

The formula of Q-learning is given as:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha_t \cdot \left(R_{t+1} + \gamma \max_{a \subset \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t) \right)$$

where alpha is the step size, gamma is the discount factor.

For the formula, we can notice that there is a difference from SARSA. The updated algorithm of Q value is not the same. SARSA updates with the value provided by a ε-greedy policy. For Q-learning, the algorithm use the ε-greedy policy to update the actions, and try to evaluate the ε-greedy policy while following another policy. Under a ε-greedy policy, Q-learning only calculates the Q-value and the maximum action value.

The algorithm of Q-learning is described as:

1. Initialize the Q value (Q(s,a))
2. Give a observation to the state (S0=s0)
3. Initialize ε-greedy policy π'0
4. Repeat the following step for each episode until terminate(t=0,1,2…):
    4.1. Take action At ~π′t(St) and observe (Rt+1,St+1)
    4.2. Update the Q-value for the state using the observed reward and the maximum reward for the next state.
    4.3. Update π′t+1 with ε-greedy(Q)

In applying Q-learning algorithm on Blackjack, we also set the policy to be satisfying the condition of GLIE and the stepsizes αt satisfy:

$$\sum_{t=1}^{\infty} a_t = \infty, \sum_{t=1}^{\infty} a_t^2 < \infty$$

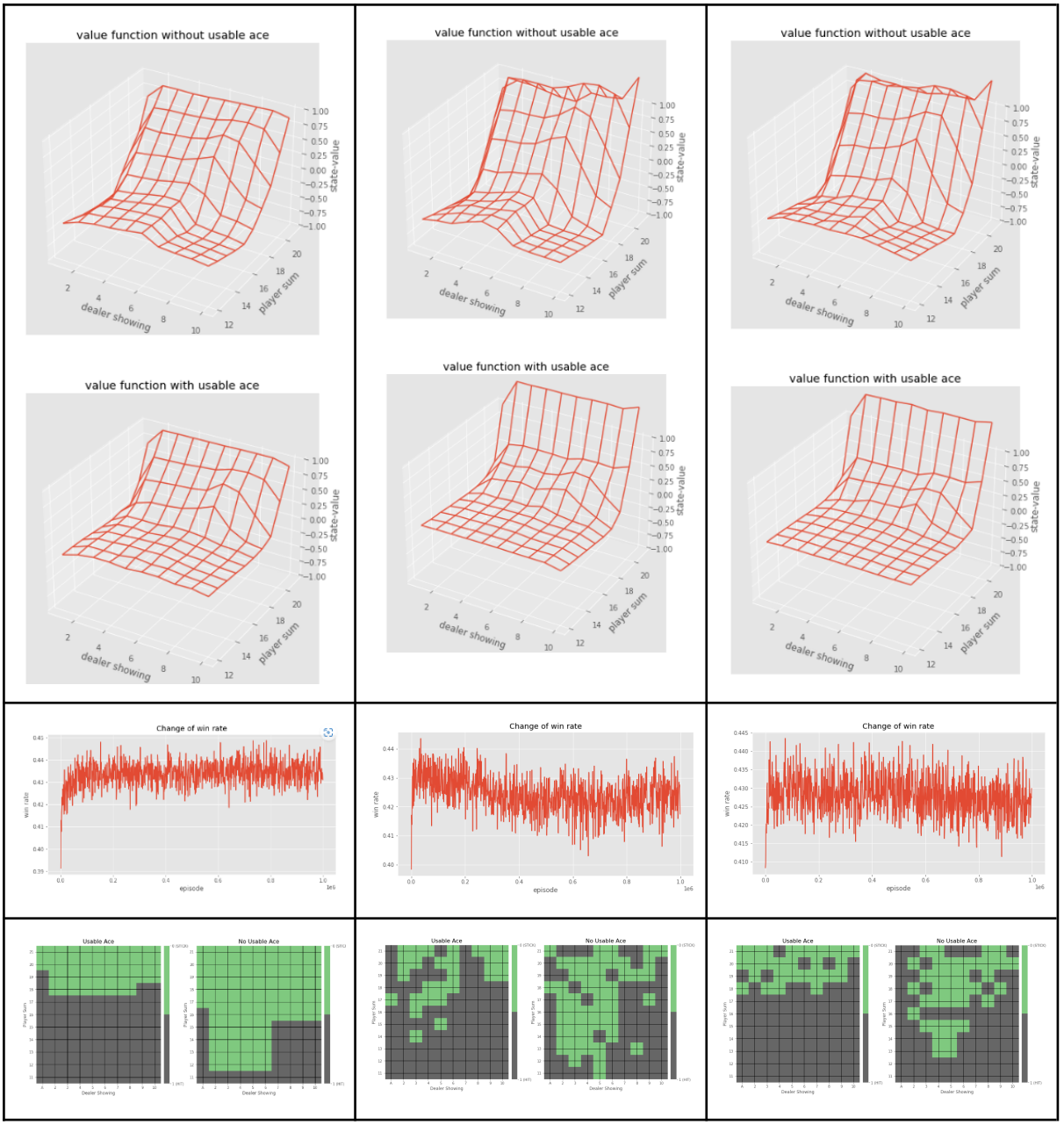
3. Result

For GLIE Monte Carlo control method, we set the number of episodes= 1000000, gamma = 0.4 and evaluating episodes = 10000. For GLIE SARSA and GLIE SARSAMAX, we set the number of episodes = 1000000, alpha = 0.001, gamma = 0.4 and evaluating episodes= 10000. The gamma = 0.4 was selected for the sake of the short horizon needed (1-3 turns) for Blackjack. Also, as mentioned above, for the epsilon-greedy policy, we initialize epsilon at the starting time step to be 1.0 to encourage exploration at the beginning, and then choosing the decay rate of epsilon to be 0.99999, and finally the minimum value of epsilon to be 0, to ensure the convergence of the algorithm.

For evaluation of the training result, we recorded the state-value function plot and the optimal policy table to evaluate the training algorithms.

As we train all the algorithms for 1000000 episodes, we set every thousand of episodes that the algorithms will get into an evaluate function for 10000 episodes. The function will then play 10000 times of Blackjack to let us record the win-lose rate.By applying this evaluation function, we finally come up with a win rate graph for 1000000 episodes of Blackjack for every algorithm.

GLIE Monte Carlo control	**GLIE SARSA**	**GLIE Q-Learning (SARSAMAX)**



3.1 State value function plot

In the above state value functions, we can see that the highest state values correspond to when the player sum is something like 20 or 21. It is an anticipated outcome as we are most likely to win when we have a high card value without busted.

Compared to the state value function without a usable ace, players are more likely to win a game if the dealer is getting a showing between 2 to 6 in Monte Carlo methods and SARSA. However, for SARSAMAX or for the cases with usable ace, the player showings are having similar level of state value.

For the state value function with usable ace, the state value is generally higher than the state value function without usable ace. It may imply that the strategy for having a usable ace is more aggressive.

3.2 Win rate curve

The above Win rate curves show how the win rate variates when the episode increases.

For GLIE Monte Carlo control, it has increased trend and becomes stable at the value between 0.43 to 0.44 after 100000 episodes.

For GLIE SARSA, the win rate is first increased to around 0.43. However, after around 200000 episodes, the win rate starts to decrease and become steady at the value between 0.42 to 0.44.

For GLIE Q-Learning (SARSAMAX), the value varies a lot between 0.415 to 0.44 and therefore no trend is observed from the graph.

3.3 Optimal policy table

In the optimal policy table, black color region represents the hit action, and the green color region represents the stick action.

Comparing among 3 policy tables, we can see that the Monte Carlo algorithm established a well-rounded and clear strategy for playing Blackjack. It converged in a perfect way by sight. Therefore, we will do a simple comparative analysis based on the Blackjack strategy table in the next session.

Coming back to SARSA and SARSAMAX, we can see that the strategy tables are relatively scattered compared to Monte Carlo control methods. It may imply that the policy tables are not yet ready to be an output. In other words, surprisingly, SARSA and SARSAMAX may still need time to converge.

4. Discussion

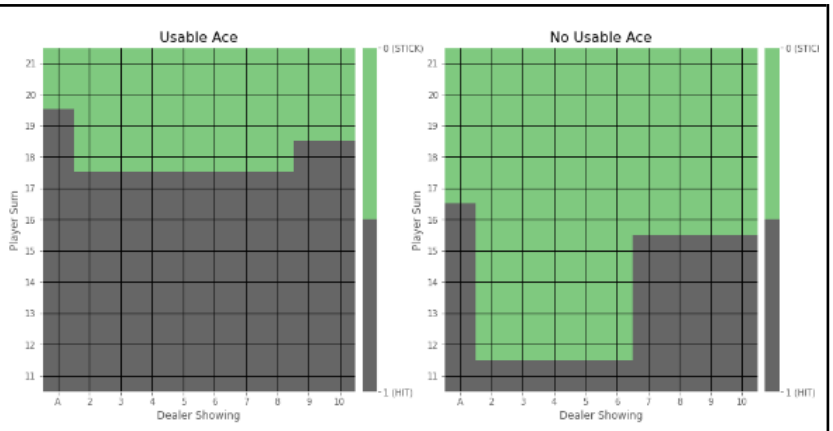
4.1 Performance of Method

If we look through the win rate curve and the optimal policy table, it should be easy to draw a conclusion that GLIE Monte Carlo Control is the best method in this project.

4.1.1 Blackjack strategy table comparative analysis with strategy table by Monte Carlo control

Hard Totals

Your Hand	Dealer 2	3	4	5	6	7	8	9	T	A
Hard 5	H	H	H	H	H	H	H	H	H	H
Hard 6	H	H	H	H	H	H	H	H	H	H
Hard 7	H	H	H	H	H	H	H	H	H	H
Hard 8	H	H	H	H	H	H	H	H	H	H
Hard 9	H	D	D	D	D	H	H	H	H	H
Hard 10	D	D	D	D	D	D	D	D	H	H
Hard 11	D	D	D	D	D	D	D	D	D	D
Hard 12	H	H	S	S	S	H	H	H	H	H
Hard 13	S	S	S	S	S	H	H	H	H	H
Hard 14	S	S	S	S	S	H	H	H	H	H
Hard 15	S	S	S	S	S	H	H	H	H	H
Hard 16	S	S	S	S	S	H	H	H	H	H
Hard 17	S	S	S	S	S	S	S	S	S	S
Hard 18+	S	S	S	S	S	S	S	S	S	S

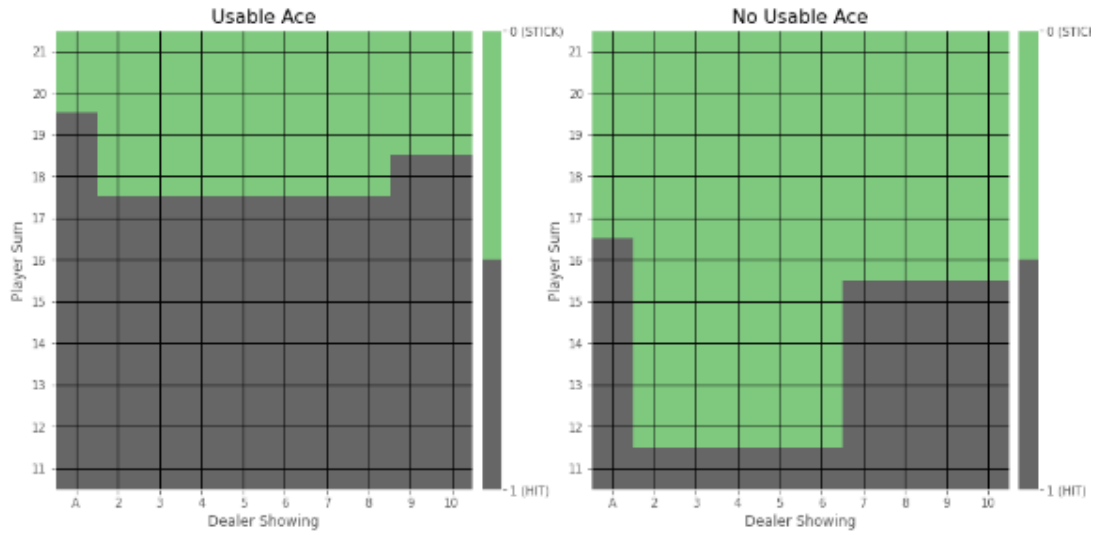


As mentioned above, as the Monte Carlo algorithm established a well-rounded and clear strategy for playing Blackjack, we will do a comparative analysis between some basic strategy tables. In the table above, the left strategy table, which is retrieved from www.blackjackinfo.com, is a hard total Blackjack strategy table, meaning that there would not be any usable ace. In this table, the green colour region is hit, blue is double (hitting with a double bet), and red is stick. We consider double as an action hit.

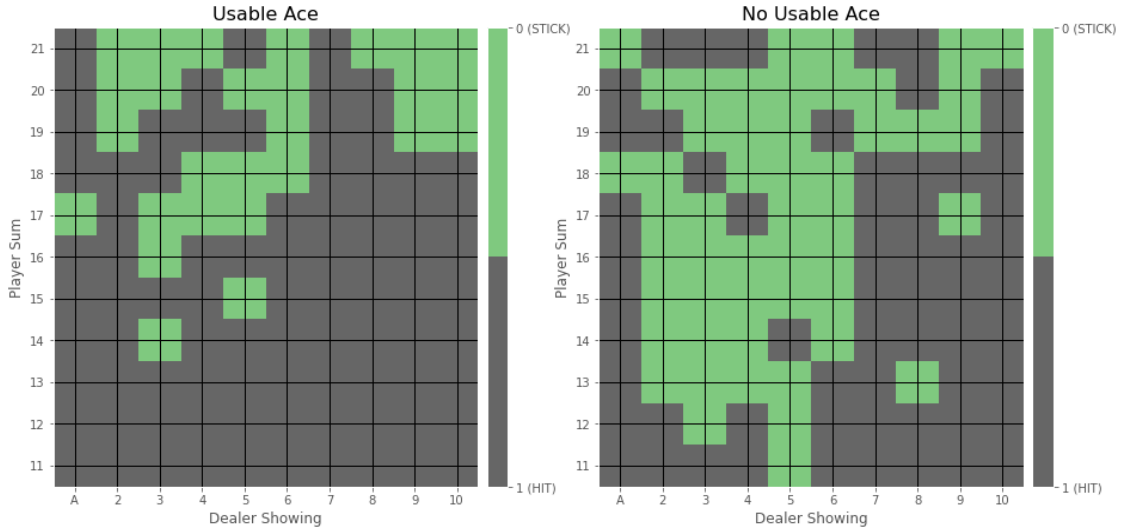
For the left strategy table, we can see that if the dealer is showing 7, 8, 9, T (having a value of 10) or an Ace, the player should take a risk for a high value of card hand, even though you are holding a card sum that will let you bust very easily. At the same time, for the Monte Carlo strategy table without a usable ace, players are also more willing to risk for a high value if dealer showing is 7, 8, 9, T or an Ace, regardless the total hand value of the player. In addition, for the table with usable ace, the player also tend to risk for a win if the dealer showing are 9, T or an ace. Indeed, the strategies in the policy table of Monte Carlo with usable ace are already very aggressive. However, it is still delighted to see the reinforcement learning AI start to learn something which is rational.

4.1.2 Potentials of the other model

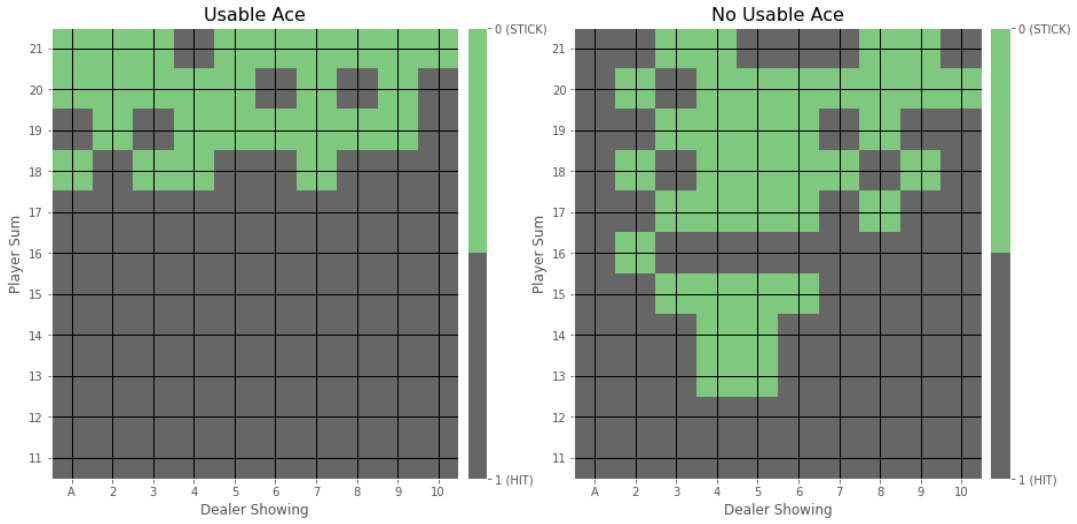
Although the performance of the SARSA and SARSAMAX algorithm may not be satisfactory, we can still see some correlation between the optimal policy table between three algorithms after doing the blackjack strategy table comparative analysis.



MONTE CARLO CONTROL



SARSA



SARSAMAX

Even the optimal policy table for SARSA and SARSAMAX is very scattered, we can see that sticking patterns of SARSA and SARSAMAX are very similar to Monte Carlo Control's sticking patterns. Therefore, we can see the possibilities that SARSA and SARSAMAX can converge to our desired result, which is very encouraging to us.

4.2 Advantages and disadvantages of the methods

In this project, typically we implement all the algorithms with Monte Carlo methods or Temporal-Difference learning. Therefore, we will do a comparison between these two.

Monte Carlo methods have high variance, zero bias. It has good convergence properties, and is not very sensitive to initial value (initial state). It is also very simple to understand and use. Temporal-Difference learning has low variance, some bias. It is usually more efficient than Monte Carlo methods. It is more sensitive to initial value.

Temporal-Difference learning can learn online after every step, but Monte Carlo methods must wait until the end of the episode before return is known. Temporal-Difference learning can learn without the final outcome. Temporal-Difference learning can learn from incomplete sequences, but Monte Carlo methods can only learn from complete sequences. Temporal-Difference learning works in continuing (non-terminating) environments, but Monte Carlo methods only works for episodic (terminating) environments.

4.3 Limitation and difficulties

There are several limitations regarding the computation, the optimization and strategy on deciding the policy. Since the RL algorithms require a long time for training , it is difficult to take a complete list of actions : “Hit” , “Stand”, “Double down”, “Split”, “Surrender” into our algorithm, which increases the model complexity exponentially and the time used for computation. On the other hand, the algorithms might be improved by following another strategy called “Edward Thorp’s Strategy”, which allows players to gain the best possible return in long-term success under a set of instructions.

5. Conclusion

This project seeks to compare and understand three RL algorithms by obtaining the possible best Hit-Stand policies on the game of Blackjack. GLIE Monte Carlo , GLIE Sarsa, and Q-learning were implemented. In the state value functions for each algorithms, we can see a description of how the algorithms converged. For each of these algorithms, their produced policies in winning rate varies from 0.43 to 0.44, 0.42 to 0.44 and 0.415 to 0.44, respectively. However, we observe that from the optimal policy table, GLIE Monte Carlo is very clear to see the rules, while GLIE Sarsa and Q-learning (SARSAMAX) are scattered. We observe that GLIE Monte Carlo control is the best method applied in our project.

For future works, it would be interesting to explore the effect of different strategies that already hold together with some extensions of RL algorithms such as Deep Q-network and Bayesian Q-learning algorithm.

6. Reference

1. Gautam, A. (2019, March 15). *Introduction to reinforcement learning (coding sarsa) - part 4*. Medium. Retrieved November 17, 2022, from https://medium.com/swlh/introduction-to-reinforcement-learning-coding-sarsa-part-4-2d64d6e37617
2. *Simple reinforcement learning: Q-learning - towards data science*. (n.d.). Retrieved November 16, 2022, from https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56
3. *Reinforcement learning*. Reinforcement Learning - Algorithms. (n.d.). Retrieved November 17, 2022, from https://www.cse.unsw.edu.au/~cs9417ml/RL1/algorithms.html
4. Thoma, M. *What are the advantages / disadvantages of off-policy RL vs on-policy RL?* Data Science Stack Exchange. Retrieved November 17, 2022, from https://datascience.stackexchange.com/questions/13029/what-are-the-advantages-disadvantages-of-off-policy-rl-vs-on-policy-rl
5. *Temporal difference learning (TD learning)*. Engati. (n.d.). Retrieved November 17, 2022, from https://www.engati.com/glossary/temporal-difference-learning
6. Jordan J Hood. (2021, May 10). *Reinforcement learning: Temporal difference (TD) learning*. Jordan J Hood. Retrieved November 17, 2022, from

https://www.lancaster.ac.uk/stor-i-student-sites/jordan-j-hood/2021/04/12/reinforcement-learning-temporal-difference-td-learning/
7. *Blackjack - learn the rules, strategy and more at blackjackinfo*. BlackjackInfo.com. (2017, December 7). Retrieved November 17, 2022, from https://www.blackjackinfo.com/
8. *Beating Blackjack - A Reinforcement Learning Approach.* (n.d.). Retrieved November 17, 2022, from https://web.stanford.edu/class/aa228/reports/2020/final117.pdf
9. *Learning to play Blackjack with Deep Learning and Reinforcement Learning* (January 2019). Retrieved November 17, 2022, from https://i.cs.hku.hk/fyp/2018/fyp18013/reports/InterimReport_3035238565.pdf
10. Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.