ENGG6600 Reinforcement Learning Programming Assignment

Fall 2022

Yaowen Mei
1177855

Question 1. Description of the Imp of the Easy21

In order to implement the Easy21 environment, I have designed the following classes:

First, I have defined the Action, and Reward as Enum:

<<**Enum**>> **Action**
+ STICK + HIT

<<**Enum**>> **Reward**
+ LOOSE = -1 + DRAW = 0 + WIN = 1

Then, I have defined the Easy21 states as a class, **State**,

- the **dealerFirstCard** is the the field to store the dealer's fully observed card;
- the **playerSum** is the filed to store the player's card's accumulated sum;
- the **isFinished** is the a boolean filed to indicate if the game is finished or not
- the **tuple()** method will return a tuple of (dealer's first card, and player sum), this tuple can be used for hash purpose.

State
+ dealerFirstCard : Int + playerSum : Int + isFinished : Bool ...
+ tuple () ...

Finally, I have defined the Easy21 Environment as a class, **Easy21**, it only contains one public method, **step()**, which takes a instance of state and action as input, and return the next state and the immediate reward from the environment.

Easy21
+ step (state: State, action: Action) - playerHit (state: State) - dealerHit (state: State) - calculateReward (state: State)

- If the action is to STICK, then in the step() method, Easy21 will call dealerHit() method until dealer bust or dealer's cards add more than 17.
- If the action is to Hit, then in the step() method, Easy21 will call playerHit() method.
- If the game is finished, then reward is generated and returned, if the game is not finished, reward 0 is returned.

Question 2. Pseudo-code of the implemented algorithms

Pseudo-code of MC Control

I have used the every-time MC implementation for this assignment, the pseudo-code looks like the following (Note that I have set the MAX episode number to be a very large number, 50,000):

```
Algorithm 1 Every Time MC
initialization $N(s,a) = 0$, $q(s,a) = 0$ for each pair of $(s,a)$
set $\pi(s)$ to be $\frac{100}{100+N(s,\text{HIT})+N(s,\text{STICK})}$ to explore non-best action; otherwise, choose the best action;
while $episode_{num} < 50,000$ do
    Sample an episode i = S, A, R, S, A, R, ... S, A, R, Terminate
    $G = R_{Last}$ as $R_{internal} \equiv 0$ and $\gamma = 1$
    for each (s,a) visited in episode i do
        for Every time, state (s,a) is visited in episode i do
            $N(s,a)$ += 1
            $q(s,a)$ += $\frac{G-q(s,a)}{N(s,a)}$
        end
    end
    update policy $\pi$
    episode$_{num}$ += 1
end
```

Pseudo-code for SARSA Learning

Please note that as required by this assignment, the Max training episode for SARSA Learning is 1000 times. The pseudo code is showing below:

Algorithm 2 SARSA Learning

```
initialization $N(s,a) = 0$, $q(s,a) = 0$ for each pair of $(s,a)$
set $\pi(s)$ to be $\frac{100}{100+N(s,\text{HIT})+N(s,\text{STICK})}$ to explore non-best action; otherwise, choose the best action;
get $S_0$ from Random Generator,
$A_0 = \pi(S_0)$
i = 0
while $i < 1000$ do
    if $S_i$ is Terminated then
        get $S_i$ from Random Generator
        $A_i = \pi(S_i)$
    end
    $N(S_i, A_i)$ += 1
    $S_{i+1}, R_{i+1} = step(S_i, A_i)$
    $A_{i+1} = \pi(S_{i+1})$
    $q(S_i, A_i)$ += $\frac{R_{t+1}+q(S_{i+1},A_{i+1})-q(S_i,A_i)}{N(S_t,A_t)}$
    update policy $\pi$
    $i$ += 1
end
```

Pseudo-code for Q-Learning Learning

Please note that as required by this assignment, the Max training episode for Q-Learning is 1000 times. The pseudo code is showing below:

Algorithm 3 Q-Learning

```
initialization $N(s,a) = 0$, $q(s,a) = 0$ for each pair of $(s,a)$
set $\pi(s)$ to be $\frac{100}{100+N(s,\text{HIT})+N(s,\text{STICK})}$ to explore non-best action; otherwise, choose the best action;
get $S_0$ from Random Generator,
$A_0 = \pi(S_0)$
i = 0
while $i < 1000$ do
    if $S_i$ is Terminated then
        get $S_i$ from Random Generator
        $A_i = \pi(S_i)$
    end
    $N(S_i, A_i)$ += 1
    $S_{i+1}, R_{i+1} = step(S_i, A_i)$
    $A_{i+1} = \pi(S_{i+1})$
    $q(S_i, A_i)$ += $\frac{R_{t+1}+\mathbf{MAX}(q(S_{i+1},\text{HIT}),\ q(S_{i+1},\text{STICK}))-q(S_i,A_i)}{N(S_t,A_t)}$
    update policy $\pi$
    $i$ += 1
end
```

Question 3. Results and Discussion

Compare the learned q values at 1000 episodes for SARSA and Q learning with the true values from MC

Firstly, we need to find the true values $q^*(s,a)$, as stated in the assignment handout, we train the Monte-Carlo Control agent 500,000 times until it fully converged, and this q value are used as the true values $q^*(s,a)$. As shown in Fig.1,

- The trend of the value function, $v*(s) = \mathbf{max}\ q^*(s,a)$, (z axis in the plot) indicates that the smaller the dealer's card is, the more advantage the player got;
- There is a local peak when player's sum is around 11 to 13, which means the player has advantage here ;
- When the player's sum is greater than 15, the greater the player's sum is, the higher the value function is (player are more likely to earn more money if in these states).

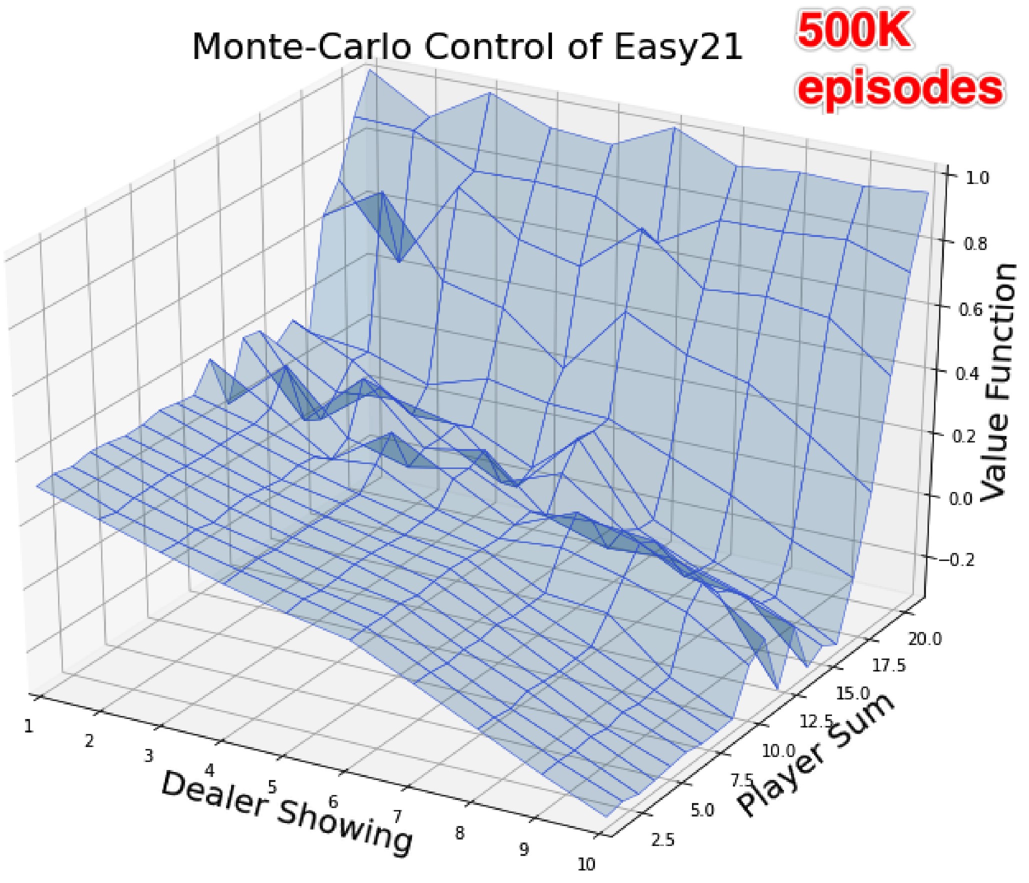
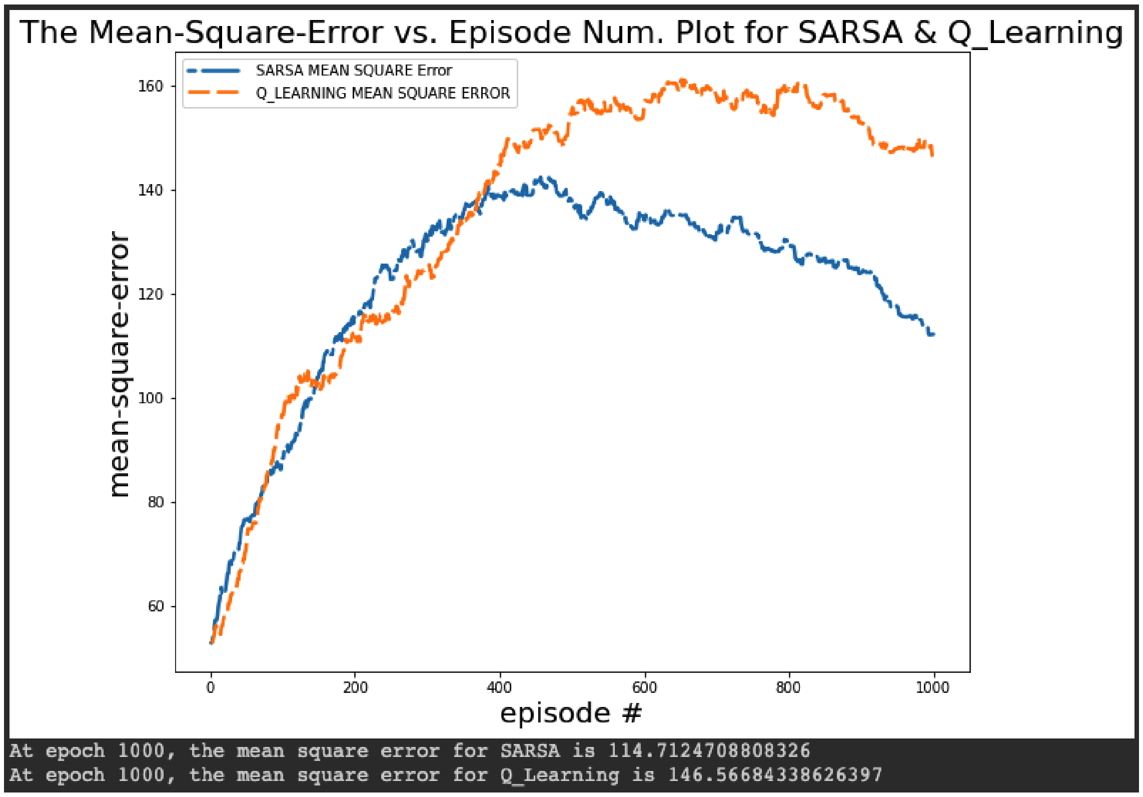


Figure 1: The plot of value function for Monte Carlo control at episode 500,000 (the q value at this episode is considered as the TRUE q value for this Easy21 Game)

Secondly, after we have obtained the true q value from the 500,000 episodes of MC training, we can calculate the mean square error for SARSA and Q-Learning now.

As we can see from the last plot in Fig. 2, the mean square error are very high for both Q-Learning and SARSA at episode 1000 (way larger than 1)

- Between episode 1 to 400, the mean square error between Q-Learning and SARSA are very close to each other, and both of them are increasing. This increasing is due to the fact that we have initialized the q values to be 0 at the very beginning, and we are updating the q values via random sampling, and q values are not converged yet.
- Between episode 400 to 1000, we can see that the mean square error of both SARSA and Q-Learning start decreasing in this range, and we also noticed that SARSA is decreasing more rapidly than Q-Learning, which indicates that SARSA has better performance than Q-Learning.
- At episode 1000, the mean square error for SARSA is 114.7 and the mean square error for Q-Learning is 146.6, this indicates that if we set true value to be the MC q value, SARSA has lower error than Q-Learning by 21% ((146.6 − 114.7)/(146.7) = 21%)
- We also noticed that since the mean square error for both SARSA and Q-Learning are very high, these training's are not fully converged yet, more episodes are need to get the training converge. This fact can also be seen from Fig.3, we noticed that the q-value plots for both SARSA and Q-Learning are nothing but noise, therefore we need to train more episode to see where they finally get converged to.



```
At epoch 1000, the mean square error for SARSA is 114.7124708808326
At epoch 1000, the mean square error for Q_Learning is 146.56684338626397
```

Figure 2: The plot of mean square error vs. episode number for the first 1000 episodes.)

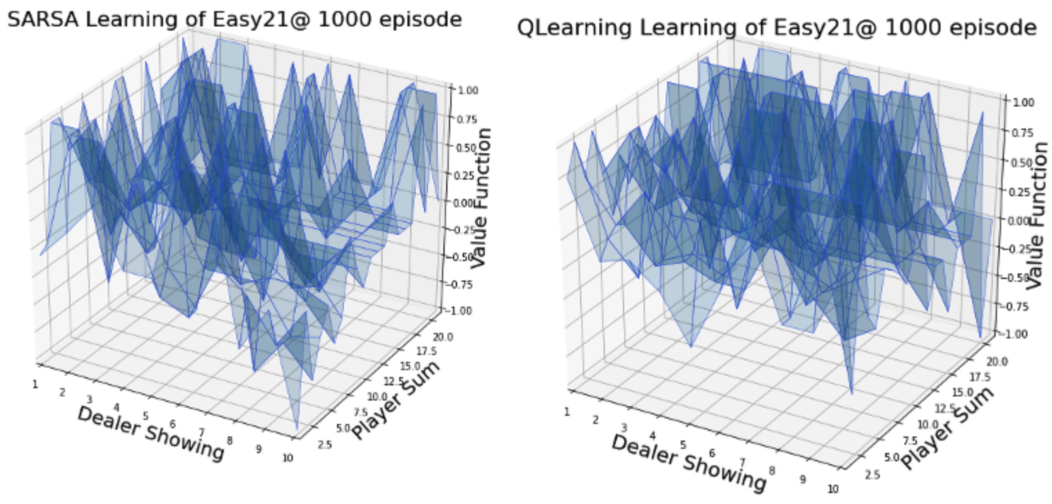


Figure 3: The plot of value function at episode 1000 for SARSA and Q-Learning)

Finally, let's train both SARSA and Q-Learning to 50,000 episodes. The results are shown in Fig.4. There are a few things we noticed from this Figure:

- The top two sub plots show that SARA and Q-Learning's value functions got more smooth after 50,000 episode of training (comparing to the non-smooth noisy plot in Fig.3)
- The right bottom sub plot shows that over all episodes, SARSA always has a lower mean square error than Q-Learning, which indicates that SARSA performs better than Q-Learning for this problem in terms of convergence speed.
- The right bottom sub plot also shows that at episode 50,000, the mean square error for SARSA and Q-Learning are very close, which means after a large number of training, the accuracy of SARSA and Q-Learning will be very close to each other.

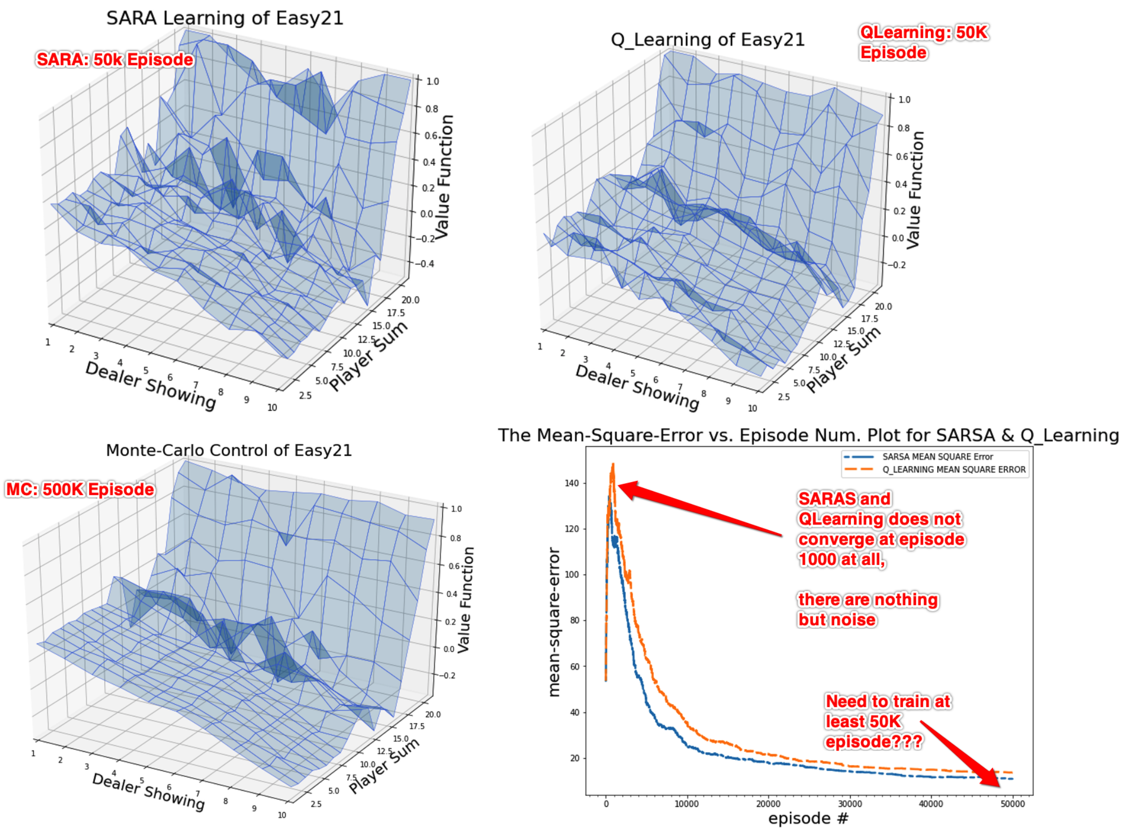


Figure 4: The value function of MC (trained 500K times), SARSA (trained 50K times, and Q Learning (trained 50K times) and the plot of mean square error vs. episode number.)