ME5406 Deep Learning for Robotics

Project for part I: The Froze Lake Problem and Variations

Department: Mechanical Engineering

Student Number : A0263252L

Student Name: Shen Xiaoting

Email address: E1010610@u.nus.edu

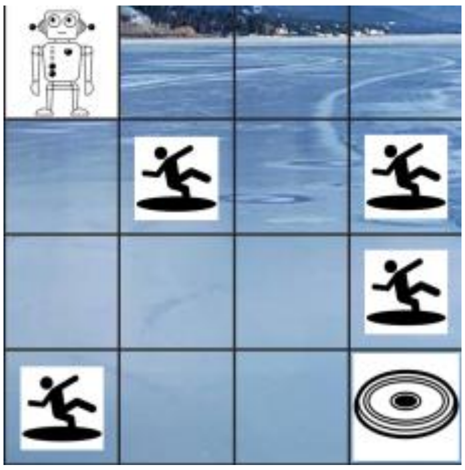
Contents

1 Problem Statement and Initialization

1.1 Problem statement

The robot intended to go through the frozen lake from one location (top left corner) to another (bottom right corner). However, there are four holes on the ice and if the robot steps into each of them, the task fails. The location of the holes, robot and target are shown in the figure1.1.

We use three reinforcement learning techniques to do find the best policy to reach the target, including Monte Carlo control, SARSA, Q-learning.



0 0	1 0	2 0	3 0
4 0	5 -1	6 0	7 -1
8 0	9 0	10 0	11 -1
12 -1	13 0	14 0	15 1

Figure 1.1.1 A robot moving on a frozen lake   Figure 1.2.1 The reward of each state

1.2 Parameter initialization

Agent follows the polices to move from the start point to the terminal state while collecting the rewards along the way. The 16 states of 4*4 grid are labelled by 0~15 shown in the figure 1.2.1. We assume that the reward for each step r=0 except the state action pair to the location of the holes as r=-1 and to the target r=1.The robot can take 4 actions at each state can be signed as 0~4 and the direction is shown in figure 1.2.2. If the robot go the state with the reward r=-1 or reach the destination r=1,it can not move forward and will start from the beginning for a new episode.

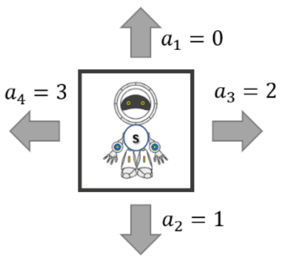


Figure 1.2.2 Actions the agent can take

We initialize the action value function $Q(s,a)$ at the beginning as 0 for all $s \in S\{0,1,2,...,15\}$ and $a \in A\{0,1,2,3\}$ and initialize the return G as 0 at the beginning of each episode. The discount rate $\gamma = 0.9$. We set the epochs as 10000 for each time of learning and the maximum steps in one episode is 100.

1.3 Implementation procedure

1.3.1 First-visit Monte Carlo control without exploring starts

1) Episode generating

At every beginning of the episode, we initialize the state of the agent as 0 (top left corner) due to the task command without exploring starts. We select an action with ϵ-greedy policy (mentioned in 4) part) to make the agent move forward and check whether the next station is the target or the trap and get the reward $\bar{\rho}(s, a, s')$. At the same time, we record the steps in one episode. Due to the limitation of the maximum steps in one episode, if the agent cannot reach the destination or get trapped, it will stop after 100 steps and start for a new episode.

We can judge whether an agent is successful or not by the last step of an episode. If the reward of the step is positive, the agent reach the terminal state and the episode end up with this step. If the reward of the step is negative, the agent is trapped and this episode is failed. We can record the time of success and failure.

2) Return calculation

As we get a sequence of $\{S_t,\ A_t,\ R_{t+1}\}$ from starting point S_0 and terminated at S_{T-1}, we can calculate the return G from the last step (t=T-1) and backward to t=0 by the equation

$$G_t = R_{t+1} + \gamma G_{t+1}$$

3) Update Q table

We just need to consider the condition of first-visit Monte Carlo, so we should check whether the step is shown in previous (state, action) pair to make sure that this step is the first visit. If the (state, action) pair is not shown and we update the Q(s,a) table with return G.

4) ϵ-greedy behavior policy

An agent can move in four directions at an state and every (s,a) pair have a non-zero probability of being visited. We observe the ϵ-greedy behavior policy to help the robot make decision which direction to move. The ϵ-greedy policy $(\epsilon = 0.1)$

$$\pi(a|s) = \begin{cases} 1 - \epsilon + \dfrac{\epsilon}{|A(s)|} = 0.925, & for\ a = A^* \\ \dfrac{\epsilon}{|A(s)|} = \dfrac{0.1}{4} = 0.025, & for\ a \neq A^* \end{cases}$$

The algorithm is show in table 1.1

Table 1.1 First-visit Monte Carlo control without exploring starts for estimating $\pi \approx \pi_*$

Parameter:	A small $\epsilon > 0$

```
Initialize:    $\epsilon$-greedy policy ($\epsilon = 0.1$)(as shown in 1.3.1)
               Q(s,a)=0 for all $s \in S$ and $a \in A(s)$
               Return Total_return(s,a) as an empty list for all epochs
               Time of visit of $(S_t, A_t)$ pair as N[(s,a)]=0 for all $s \in S$ and $a \in A(s)$
               Cost all_cost[] as an empty list for all epochs

               Loop forever(for each episode)
                 Generate an episode with T steps following $\pi$ by $\epsilon -$ greedy policy :
                 $[(S_0, A_0, R_1),(S_1, A_1, R_2),\ldots(S_{T-1}, A_{T-1}, R_T)]$
                 G = 0
                 Loop for each step of an episode, t=T-1, T-2,…,0
                   G $\leftarrow R_{t+1} + \gamma G$
                   If $(S_t, A_t)$ pair doesn't appear in $[(S_0, A_0),(S_1, A_1),\ldots(S_{t-1}, A_{t-1})]$
                     Total_return$(S_t, A_t)$ $\leftarrow$ Total_return$(S_t, A_t)$+G
                     Add the time visiting (S,A) pair : N(S,A) $\leftarrow$ N(S,A)+1
                     Q$(S_t, A_t)$ $\leftarrow$ Total_return$(S_t, A_t)$/N(S,A)
                     $A^* \leftarrow argmax_a Q(S_t, a)$
                   Cost $\leftarrow$ Cost + Q$(S_t, A_t)$
                 Append Cost to all_cost[] list
```

1.3.2 SARSA with an ϵ-greedy behavior policy

1) Action generating

SARSA algorithm is different from Monte Carlo which should generate a complete episode. In Monte Carlo method, it should check whether the (s,a) pair appearing in previous and make sure the first visit. SARSA method just generates an action to make the agent move forward. It can continuously generate action until the agent get trapped or reach the target which is similar to Q-learning

2) Update Q table

When the agent chose an action by ϵ-greedy policy $(\epsilon = 0.1)$ at the next state s' to move, it will update the Q table with the formula

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

We choose the learning rate $\alpha = \alpha_t = 1/t$, where t is a time step, with initial value of $\alpha_0 = 1$.

The algorithm is show in table 1.2

Table 1.2 SARSA for estimating Q≈ q_*

```
Parameter:     A small $\epsilon > 0$
               Step size $\alpha_0 = 1$ and $\alpha = \alpha_t = 1/t$
Initialize:    $\epsilon$-greedy policy ($\epsilon = 0.1$)(as shown in 1.3.1)
               Q(s,a)=0 for all $s \in S$ and $a \in A(s)$
               Cost all_cost[] as an empty list for all epochs
```

```
$\hat{S}$ being the set of terminal states

Loop forever (for each episode)
    Initialize S
    Choose action A from S using $\epsilon -$ greedy policy from Q
    Loop for each episode
        Take action A; Observe R,S'
        Choose action A' from S' using $\epsilon -$ greedy policy from Q
        Q(S,A) $\leftarrow$ Q(S,A) + $\alpha$ $[R + \gamma Q(S', A') - Q(S, A)]$
        S←S', A←A'
        Cost←Cost + Q($S_t, A_t$)
    Until S∈ $\hat{S}$
        Append Cost to all_cost[] list
```

1.3.3 Q-learning with an ϵ-greedy behavior policy

1) Update Q table

Q-learning is similar to SARSA and the most different part is temporal difference update formula which has the following specific form

$$\mathrm{Q(S,A)} \leftarrow \mathrm{Q(S,A)} + \alpha\ [\mathrm{R} + \gamma \mathrm{argmax_{a'}} \mathrm{Q}(\mathrm{S}', \mathrm{A}') - \mathrm{Q}(\mathrm{S}, \mathrm{A})]$$

It learns the optimal action value q_* of the optimal policy π_*.The aim of the Q learning is to find its target policy which is also the optimal policy for a reinforcement learning problem. It uses $\epsilon -$ greedy behavior policy to generate the data samples needed in the learning process. The algorithm is shown in table 1.3

Table 1.3 Q-learning for estimating $\pi \approx \pi_*$

```
Parameter:    A small $\epsilon > 0$
              Step size $\alpha_0 = 1$ and $\alpha = \alpha_t = 1/t$
Initialize:   $\epsilon$-greedy policy ($\epsilon = 0.1$)(as shown in 1.3.1)
              Q(s,a)=0 for all $s \in S$ and $a \in A(s)$
              Cost all_cost[] as an empty list for all epochs
              $\hat{S}$ being the set of terminal states

              Loop for each episode
                  Initialize S
                  Loop for each step of episode
                      Choose action A from S using $\epsilon -$ greedy policy from Q
                      Take action A; Receive R and observe S'
                      Q(S,A) $\leftarrow$ Q(S,A) + $\alpha$ $[R + \gamma argmax_{a'} Q(S', A') - Q(S, A)]$
                      S←S'
                      Cost←Cost + Q($S_t, A_t$)
                  Until S∈ $\hat{S}$
                      Append Cost to all_cost[] list
```

2 The optimal policy and Q-table analysis

2.1 Optimal policy results

After the learning through the algorithm of Monte Carlo, SARSA and Q-learning, we can get the routes of the agent as follows by the 4*4 grid and 10*10 grid.

In the following results, we can observe that all the algorithm can get the best route with 6 steps. Three different algorithms have the same optimal policy with 10000 learning episodes in the 4*4 grid.

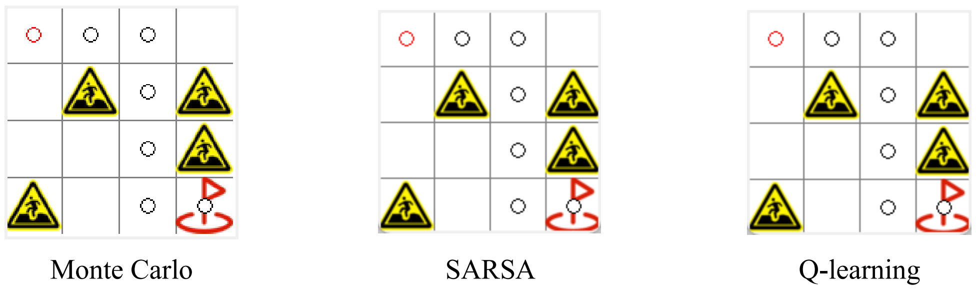


Figure 2.1.1 The optimal policy of 4*4 grid map with 10000 learning episodes

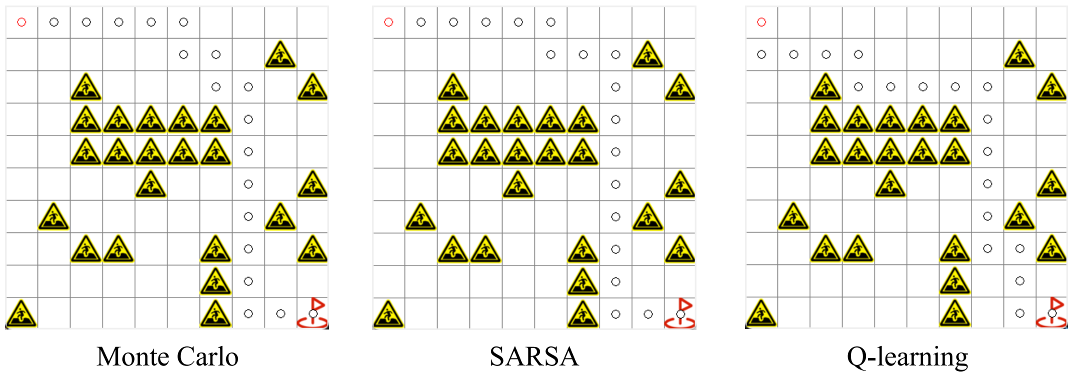


Figure 2.1.2 The optimal policy of 10*10 grid map with 50000 learning episodes

In the 10*10 grid environment, the best policy of three learning algorithm is different. We have placed many traps in the middle of the environment. It is clear that the optimal policy of Q-learning near the traps' edge, while SARSA learns the action values of a near-optimal policy that directs the agent to first move away from the traps, then turn to move towards to the target state.

2.2 Q-table results

After the implementation of the learning algorithm, we can get the Q table which will get the best policy for the task. The blue arrows show the action of the state with maximum q value which will direct to the target. The orange arrows indicate the maximum q value at that state. The Q table of the three learning algorithms a shown in figure 2.2.1.

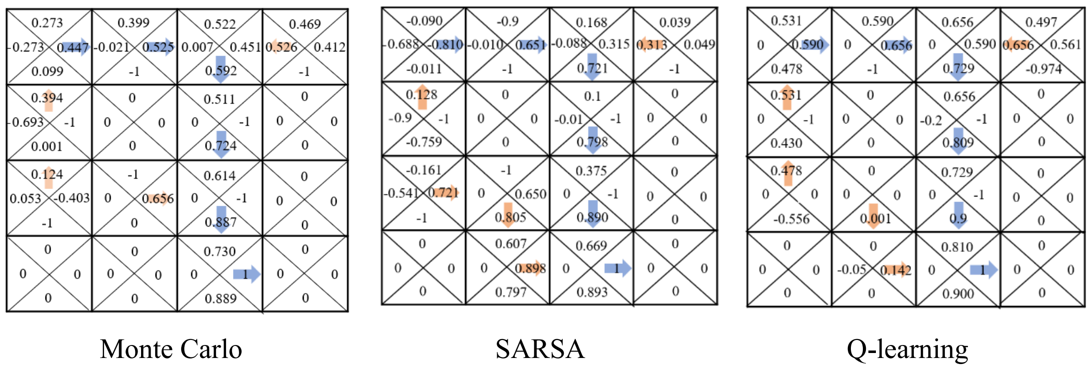


Figure 2.2.1 Q table of learning algorithm with 10000 episodes

The Q table only list the condition of 4*4 grid and the Q table of 10*10 grid is show in the file 'Q-table'.

3 Steps needed via episodes analysis

We do the learning process with 10000 episodes and record the success and fail episodes and their steps in the 4*4 grid. The left side is the bar chart which shows the comparison of success and failure times in 10000 episodes and the right figure shows the steps of each episode needed (limited by 100 steps in Monte Carlo control of one episode). The blue points show the number of steps of a failure episode and the orange points shows the number of steps of a successful episode. We only extract part of the results to show the detail of the points.

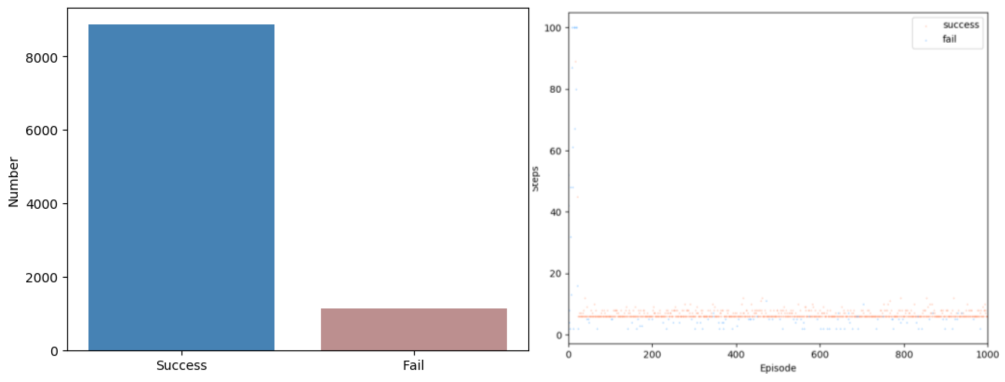


Figure 3.1 Monte-Carlo training process with steps of success and failure episodes

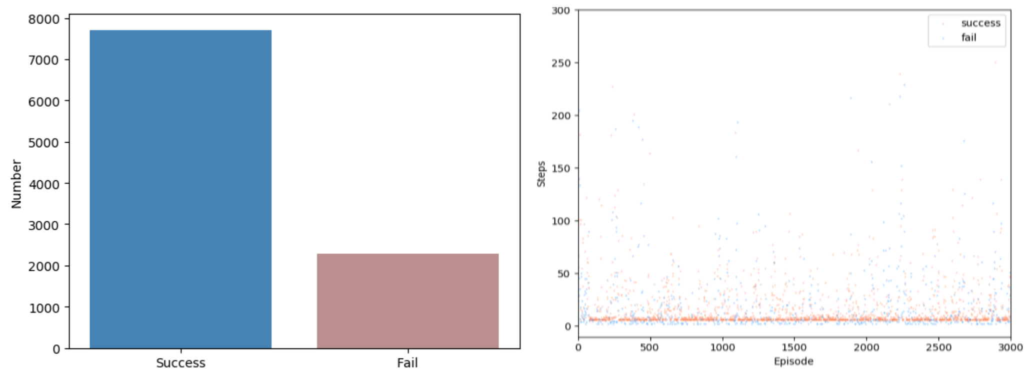


Figure 3.2 SARSA training process with steps of success and failure episodes in 4*4 grid

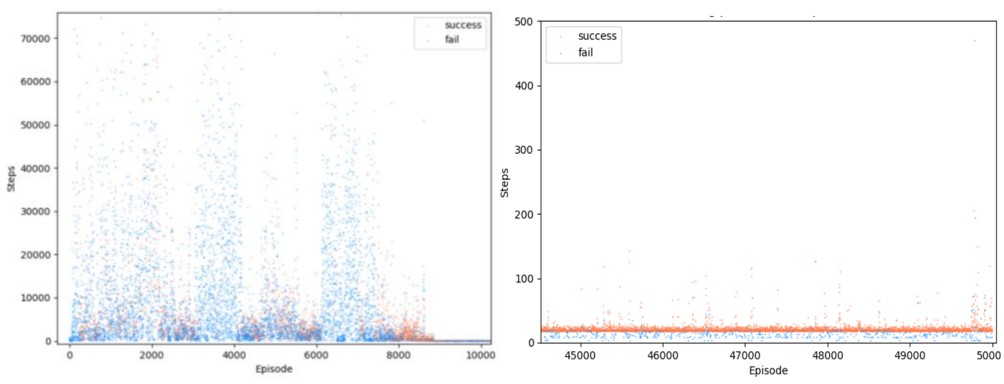


Figure 3.3 SARSA training process with steps of success and failure episodes in 10*10 grid

From figure 3.2, we can observe that the failure times with SARSA in 10000 episodes in more than Monte-Carlo and Q-learning algorithm. SARSA is an algorithm which does exploration. It will explore other places which will results in more failure trail and more steps to reach the target. The steps of an episode to end are more decentralized compared with the learning process using the algorithm of Monte-Carlo and Q-learning. The steps in an episode range from 0 to 10 of Monte-Carlo and Q-learning method. It is also clear in figure 3.3

We can also observe in figure 3.3 that there are many failures at first 10000 episodes with thousands of steps. From the episodes 4500 to 5000, the learning process converge. Compared with the Q learning method in figure 3.4 and figure 3.5, it is clear that at first 500 episodes, there are many failure episodes with hundreds of steps. Then the learning process will converge at a higher speed and the steps of an episode will decrease within 40 steps.

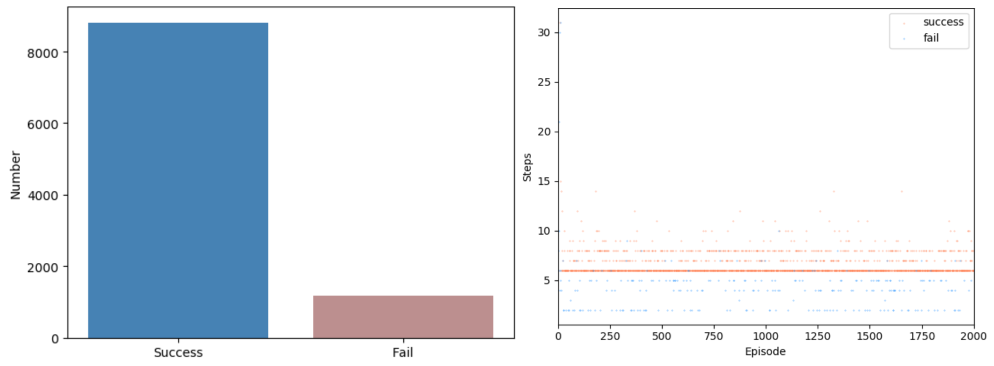


Figure 3.4 Q-learning training process with steps of success and failure episodes

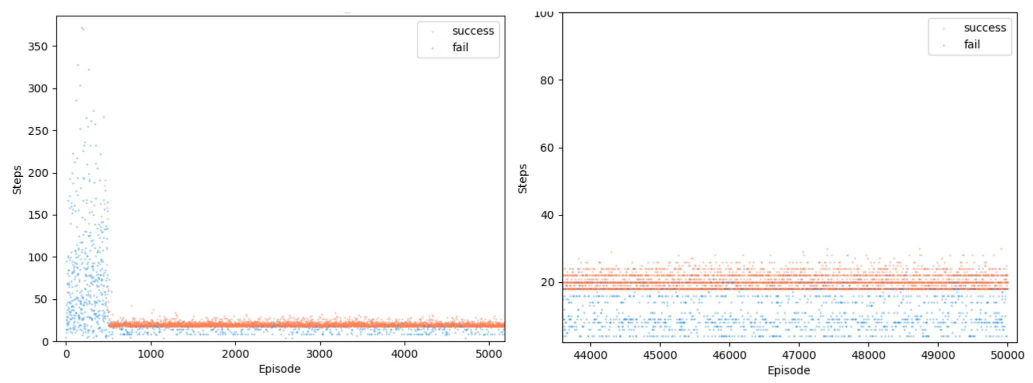


Figure 3.5 Q-learning training process with steps of success and failure episodes in 10*10 grid

4 Accuracy via episodes comparison and average Q value analysis

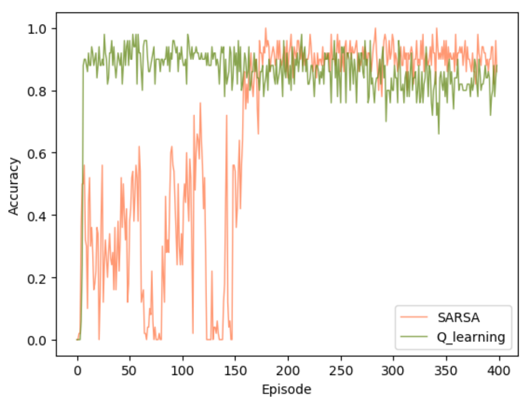


Figure 4.1 Accuracy via episodes of 10*10 grid (calculate every 50 episodes)

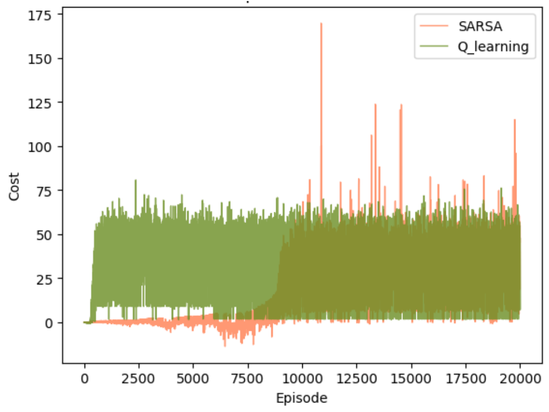


Figure 4.2 Sum of Q value of each episode via episodes of 10*10 grid

The accuracy of the learning process of SARSA and Q-learning algorithm is calculated by the successful episodes divided by the number of episodes. We get the figure 4.1 for 20000 episodes. It is obvious that the accuracy will reach to 90% at a high speed with Q-learning algorithm compared with the SARSA algorithm. The efficiency of Q-learning algorithm is higher than SARSA.

The sum of Q value of each episode is recorded by the list of cost. Figure 4.2 shows the cost change via episodes of SARSA and Q-learning algorithm. The cost reach to a average level at a fast speed with in about 500 episode of Q-learning and has a has a small oscillation. For the learning with SARSA algorithm, the cost will be negative at first 8000 episodes and it has a large cost after 10000 episodes which means that it will take many meaningless steps from time to time.

The results show the advantage of Q-learning with high efficiency compared with SARSA. With the same parameter, the Monte Carlo method can't get one successful episode in 20000 episodes. The tuning of parameters of Monte Carlo method is mentioned in the later part.

5 Problems and solutions of doing this project

5.1 Fail to find the optimal policy in 10*10 grid of SARSA algorithm

Modify the reward function. The reward function is a critical component of the reinforcement learning problem. If we use the same model of 4*4 grid to do the task in 10*10 grid, only the Q-learning algorithm can converge to the optimal policy. The Monte-Carlo and SARSA algorithms can not get the optimal policy over 50000 episodes. Then we change the reward of (state, action) pair when reaching the target to be 5. Then we can get the optimal policy. The comparison figure is shown in figure 5.1.1-5.1.4.

It is also obvious that the converge rate of the learning process using Q-learning is much faster than using SARSA and the change of the reward does not have much difference to converge rate.

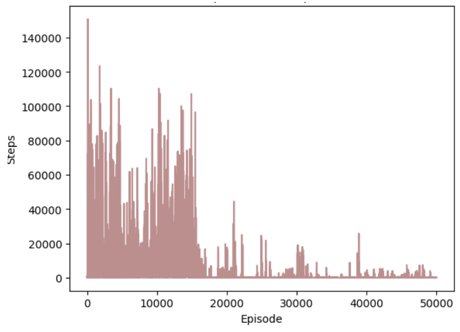


Figure 5.1.1 Steps via episodes with reward = 1 at the target state (SARSA)

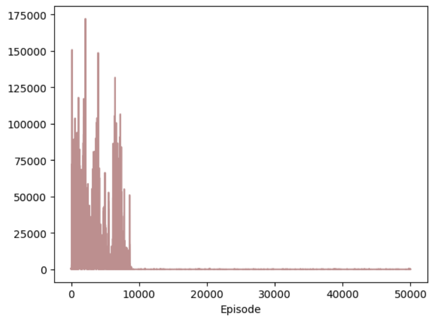


Figure 5.1.2 Steps via episodes with reward = 5 at the target state (SARSA)

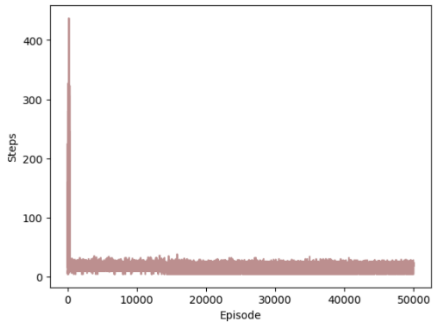


Figure 5.1.3 Steps via episodes with reward = 1 at the target state (QL)

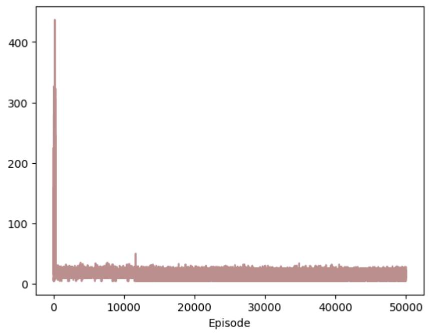


Figure 5.1.4 Steps via episodes with reward = 5 at the target state (QL)

5.2 Fail to find the optimal policy in 10*10 grid of Monte-Carlo algorithm

5.2.1 Always exploring in a small area

If we apply use the same parameter of exploring 4*4 grid, we find that the agent can only explore the 3*3 grid and trapped in this area (the orange box in figure 5.2.1.1). The training speed is too low. Then we tried the solutions as follows:

1) **Increase the learning episodes.** We increase the learning episodes from 10000 to 100000, the robot still trapped in a small area and even trapped in a 2*2 grid with increasing episodes. All the trails are failure. This method doesn't work.

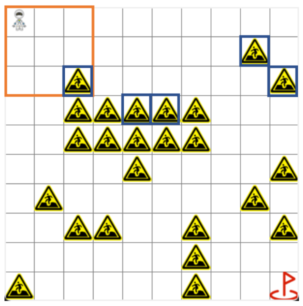


Figure 5.2.1.1 Trap area of an agent

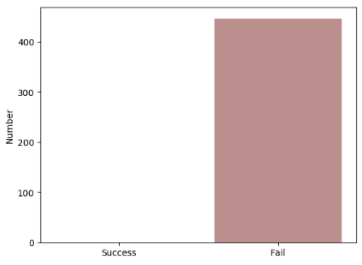


Figure 5.2.1.2 Success and fail bar

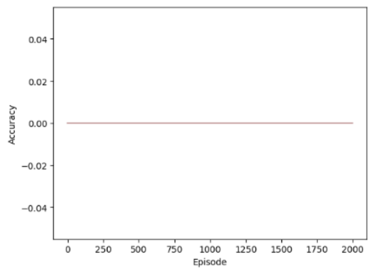


Figure 5.2.1.3 Accuracy via episodes

2) **Increasing the maximum steps of an episode.** In the 4*4 grid, we set the maximum steps of an episode are 100. Then we enlarge the maximum steps of an episode as 5000. At first few episodes, the agent can explore in a large area. With increasing episodes, the agent is still trapped in the 3*3 grid and can't reach the target. The trail results in all failure episodes and 0 accuracy as shown in figure 5.2.1.2 and figure 5.2.1.3.

3) **Give higher penalty of a trap and higher reward of the target.** I change the penalty of the trap as -5 and reward of the target as 5. The other states are zero. Then there is another problem is that the agent at the state near the traps (as shown in figure 5.2.1.4) will have higher Q value to go backward which will prevent moving forward to get the target and it can't get the target state. The problem can be solver in the next part 5.5.

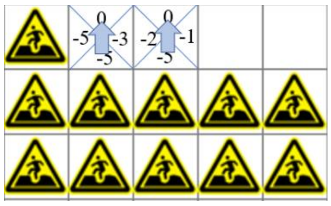


Figure 5.2.1.4 The condition of reward of trapper = -5

4) **Change the value of ϵ in ϵ-greedy behavior policy.** At the beginning, we set the value of $\epsilon = 0.1$. Then we set the value to be 0.5 which means that the agent will take greedy action at the probability of 62.5% and the probability of taking other actions as 37.5%. The effect of ϵ is very obvious that the agent can explore at a large area and closer to the target. However, the ϵ can't be too large which will contradicted with the greedy behavior policy. The value of the ϵ will ensure that the agent will take the greedy action at a higher probability.

5.2.2 Always trapped at the same point

After regulating of the parameter of step size and parameter ϵ, we found that the agent can explore at a larger area and be closer to the target. However, the agent is trapped in the same point (the blue boxes in shown in figure 5.2.1.1) with so many times and doesn't explore other areas. The solution of this problem is shown as follows:

Give a penalty to the agent if it reaches the same trap. We initialize a list to record the time of the ending (state, action) pair and give the penalty of -1*times to the agent. The more times the agent visit this trap, the more penalty it will receive. Then the effect is clear that the agent can explore more different areas and the ending at different state.

5.2.3 Always stay in the same state

Even though we enlarge the steps of an episode to 5000, we find that the agent will end at some point near the beginning. We get the detail of an episode and find that the agent remains in the same state for many times in one episode because the states at the edge of the grid can only move in some actions. The solution of this problem is shown as follows:

Give a penalty if the agent $(\boldsymbol{state, action})_t = (\boldsymbol{state, action})_{t+1}$. For the loop of each step in an episode, we check whether the (state, action) is same as the last step. If $(state, action)_t = (state, action)_{t+1}$, we give a penalty -3. Then the agent will explore more different states in other area.

5.2.4 Always move back and forth

When I change the penalty of a trap, the agent will go back and have less probability of getting closer to the target as mentioned in 5.2.1 3). The solution is shown as follows:

Give rewards when the agent moves down or right. The start state of an agent is fixed at left top and the target state is at right bottom point. If the agent moves right or down, it will get closer to the target, then we will give a reward to the agent with 4. The result shows that the agent can explore more space and have less action to go back.

5.2.5 The optimal policy lack of the last steps

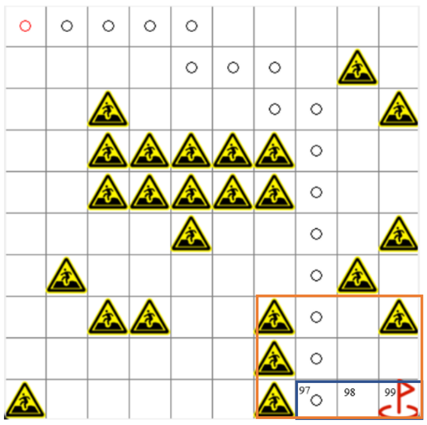


Figure 5.2.5.1 The route lack of last steps

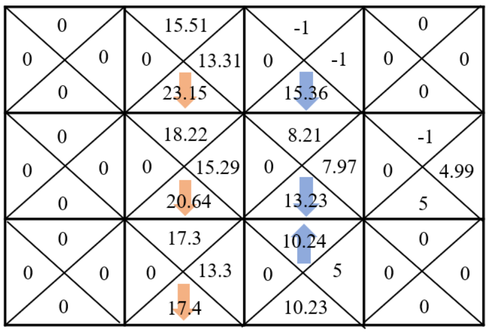


Figure 5.2.5.1 Q table of the bottom left corner

After tuning the parameters and add penalty rules mentioned above, the agent can reach the target point successfully and find the optimal policy lack of the last step. Then we get the q table at around the target is shown in figure 5.2.5.1. It is clear that the agent has the maximum reward when it moves down compared with the action to the target. The reward for the (state, action) pair to get the target should far more then other steps.

Set the reward of reaching the target to be 50. The reward to reach the target is much more than other reward and emphasize the final task is to reach the target. Then the agent can get the right left as expected. However, we can get the result of steps via episodes (figure 5.2.5.1), accuracy via episodes (figure 5.2.5.2), average Q value via episodes (figure 5.2.5.3), and training process via episodes (figure 5.2.5.4). The steps of an episode range from 0 to 160 and will not converge to the optimal policy of 18 steps. The accuracy of the Monte Carlo learning is very low and is about 20% which can also be observed in figure 5.2.5.4. The blue points (failure) are far more than the red points (success). Due to the increasing reward of the target point and some actions, the cost is much larger than other algorithms.

If we do the testing for evaluating the learning effect. The accuracy of Monte Carlo algorithm is only 50% while other algorithms like Q-learning and SARSA can be tested by 100% accuracy.

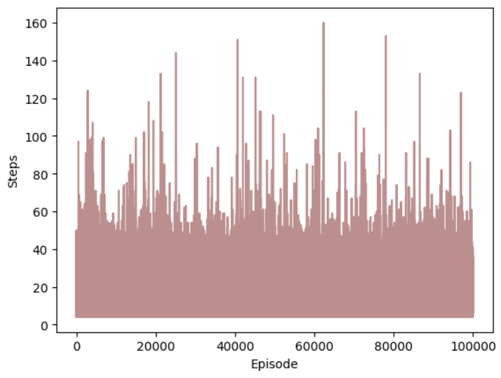


Figure 5.2.5.1 Steps via episodes

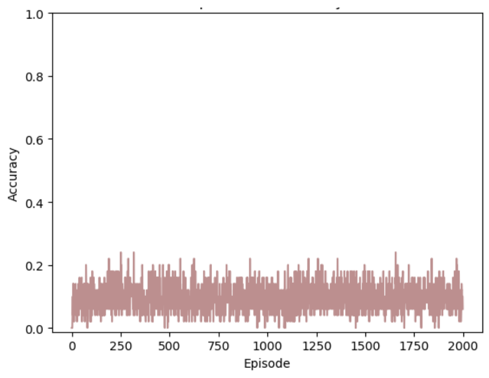


Figure 5.2.5.2 Accuracy via episodes

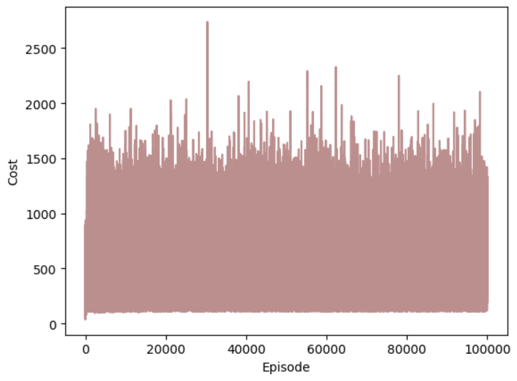


Figure 5.2.5.3 Average Q value via episodes

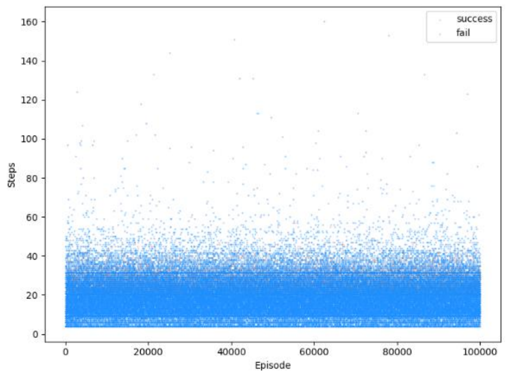


Figure 5.2.5.4 Training process via episodes

6 Conclusion

Monte Carlo methods learns from complete episodes and update the value function based on the total return from each episode. It is model-free which means that they don't require knowledge of the transition probabilities. However, Monte Carlo methods can be inefficient because they require a complete episode to finish before the value function can be updated.

SARSA learns from incomplete episodes and updates the value function based on the rewards obtained from the next state and action. SARSA has the advantage of being more sample-efficient than Monte Carlo methods.

Q-learning algorithm learns the optimal action-value function by using the maximum expected reward from the next state. Q-learning has the advantage of being model-free and converging to the optimal policy under certain conditions.

Through the learning and testing process discussed above, we find that the Q-learning algorithm is the best method to do this task with high efficiency. If the environment is complicated, the Monte Carlo method will be invalid.