Assignment 3

Name：Shiqu Wu

Student ID：518021910665

1.

(1.1)

SARSA and Q-learning

SARSA stands for State-Action-Reward-State-Action. In SARSA, the agent starts in state 1, performs action 1, and gets a reward (reward 1). Now, it's in state 2 and performs another action (action 2) and gets the reward from this state (reward 2) before it goes back and updates the value of action 1 performed in state 1. The algorithm is described as below:

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(terminal\text{-}state, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Q-learning the agent starts in state 1, performs action 1 and gets a reward (reward 1), and then looks and sees what the maximum possible reward for an action is in state 2, and uses that to update the action value of performing action 1 in state 1. The algorithm is described as below:

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(terminal\text{-}state, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$
    $S \leftarrow S'$;
  until S is terminal

Comparison

The difference between SARSA and Q-learning is in the way to find the future reward. In Q-learning it's simply the highest possible action that can be taken from state 2, while in SARSA it's the value of the *actual* action that was taken. This means that SARSA takes into account the control policy by which the agent is moving, and incorporates that into its update of action values, where Q-learning simply assumes that an optimal policy is being followed.

(1.2)

Implementation

```
if self.sarsa:
    for s in reversed(self.states):
        pos, action, r = s[0], s[1], s[2]
        current_value = self.q_table[pos][action]
        reward = current_value + self.lr * (r + reward -
current_value)
        self.q_table[pos][action] = round(reward, 3)
else:
    for s in reversed(self.states):
        pos, action, r = s[0], s[1], s[2]
        current_value = self.q_table[pos][action]
        reward = current_value + self.lr * (r + reward -
current_value)
        self.q_table[pos][action] = round(reward, 3)
        reward = np.max(list(self.q_table[pos].values()))
```

From the code implementation we can clearly see that the difference between SARSA and Q-learning is how to calculate the `reward` variable, which represents the future Q value. SARSA use the actual action reward while Q-learning use the optimal action reward. In order to better show the differences between them, we do a series of experiments on `Cliff Walking` environment to intuitively present the idea.

2.

(2.1)

Experiment on reward effect

This experiment is aim to show the effect of falling cliff punishment. When the mouse falls from the cliff, it get a large negative reward. And we change the value of this negative reward to see how it effects the mouse choosing a path. In all the sub-experiments, we set the `exp_rate=0.1,round=1000` to exclude their effects. `R` characters in the result means "route" and `G` character means "goal". All other positions are set to `0`.

SARSA:

`falling cliff reward=-50`

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| 0 | 0 | 0 | 0 | R | R | R | R | 0 | 0 | 0 | 0 |

| R | R | R | R | R | 0 | 0 | R | R | R | R | R |

| R | * | * | * | * | * | * | * | * | * | * | G |

`falling cliff reward=-100`

| 0 | 0 | 0 | 0 | 0 | R | R | R | R | 0 | 0 | 0 |

| 0 | 0 | R | R | R | R | 0 | 0 | R | R | R | R |

| R | R | R | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | R |

| R | * | * | * | * | * | * | * | * | * | G |

`falling cliff reward >= -300`

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| R | R | R | R | R | R | R | R | R | R | R | R |

| R | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | R |

| R | * | * | * | * | * | * | * | * | * | G |

Q-learning:

`falling cliff reward= -50, -100, -300` all has the same result

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| R | R | R | R | R | R | R | R | R | R | R | R |

| R | * | * | * | * | * | * | * | * | * | * | G |

From the experiment results we can see that for SARSA if the falling cliff punishment goes down, the mouse will choose a route much closer to the cliff edge. When the falling cliff punishment increases, the mouse will be frightened by the punishment and dare not take the risk to get close to the cliff, because the cost of falling is too high. At this point, the mouse will avoid the risk-taking strategy. But for Q-learning, the mouse always takes the optimal route no matter how large the falling cliff punishment is.

(2.2)

Experiment on exploration effect

This experiment is aim to show the effect of exploration rate. The exploration rate is the rate that mouse randomly chooses an action instead of greedy choose using $\epsilon - greedy$ policy improvement. The mouse will tend to explore the environment for large exploration rate. So we change the value of `exp_rate` to see how it effects the mouse choosing a path. In all the sub-experiments, we set the `falling cliff reward=-100,round=1000` to exclude their effects. `R` characters in the result means "route" and `G` character means "goal". All other positions are set to `0`.

SARSA:

`exp_rate=0.1`

| 0 | 0 | 0 | 0 | 0 | R | R | R | R | 0 | 0 | 0 |

| 0 | 0 | R | R | R | R | 0 | 0 | R | R | R | R |

| R | R | R | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | R |

| R | * | * | * | * | * | * | * | * | * | * | G |

`exp_rate=0.15`

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| R | R | R | R | R | R | R | R | R | R | R | R |

| R | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | R |

| R | * | * | * | * | * | * | * | * | * | * | G |

`exp_rate >= 0.2`

| R | R | R | R | R | R | R | R | R | R | R | R |

| R | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | R |

| R | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | R |

| R | * | * | * | * | * | * | * | * | * | G |

Q-learning:

`exp_rate=0.1, 0.15, 0.2` all has the same result

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| R | R | R | R | R | R | R | R | R | R | R | R |

| R | * | * | * | * | * | * | * | * | * | G |

From the experiment results we can see that for SARSA if the exploration rate goes down, the mouse will choose a route much closer to the cliff edge. When the exploration rate increases, the mouse dare not take the risk to get close to the cliff because the mouse knows that "Sometimes I'll make stupid decisions (randomly choose the action), and at that moment I'd better stay in a safe position to prevent killing myself ". At this point, the mouse will avoid the risk-taking strategy. But for Q-learning, the mouse always takes the optimal route no matter how large the falling cliff punishment is.

(2.3)

Discussion in extreme cases

This experiment is aim to discuss the behavior of the mouse in extreme environment settings: **no exploration setting** and **high randomness setting**. In these settings, mouse seems to act abnormally: either stay in the start state without moving or permanent stuck in part of the environment after several steps. Here are the results:

SARSA (No exploration setting):

`exp_rate=0, reward=-100 (for cliff), -1 (for other position)`

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| R | * | * | * | * | * | * | * | * | * | G |

Q-learning(high randomness setting):

`exp_rate=0.8 , reward=-100 (for cliff), -1 (for other position), round=1000`

- permanent stuck in (2,6), (2,7) these two positions (no ending).

`exp_rate=0.9 , reward=-100 (for cliff), -1 (for other position), round=1000`

- permanent stuck in (2,9), (2,10) these two positions (no ending).

`exp_rate=0.8, 0.9 , reward=-100 (for cliff), -1 (for other position), round=5000`

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| R | R | R | R | R | R | R | R | R | R | R | R |

| R | * | * | * | * | * | * | * | * | * | * | G |

From the results we can see that in high randomness environment setting, it need more round (from 1000 --> 5000) for Q-learning to converge to the optimal solution. While in no exploration setting, as we've learned in the lecture, some part of Q-table will never be reached. So the policy may never be improved, which results in the result of SARSA above. However, this strategy (stay in the start position without any movement) is the best strategy for the mouse in current situation, for the fact the even it reach the goal state, it have the same reward (-1) as in the other state. But each movement in the approaching route will take a cost of -1. So just stay in the start position without moving is the most reasonable choice for the mouse.

3.

(3.1)

Further Discussions:

From the above experiments we've stated the effect of exploration on choosing a path. And we can turn off the exploration to have a better strategy. However, whether it is beneficial to stop exploration depends on the problem we are trying to solve. If we wanted to teach a robot to bring first aid to wounded humans and avoid mines along the way, we can have it learn and experiment for as long as we can in a controlled, non-explosive environment. But once it is actually used in action we prevent it from exploring because we don’t want it to randomly roll into a mine in the name of science. It is a common practice in Reinforcement Learning to train up the system with ingenious exploration methods and then just turn the exploration off entirely in testing.