Reinforcement Learning Assignment3 Report

516030910373 XueyanLi

Part 1: Sarsa Algorithm

✧ Introduction

In model-free policy improvement, the common algorithm is to determine the action under one state by value of Q. There are two methods in choosing Q values, one is greedy algorithm and the other is ε-greedy algorithm. The difference between the two algorithms is that the greedy algorithm takes the action with the largest Q value as the target action, while the ε-greedy algorithm gives the non-maximum action a probability of ε/m, which is shown in the following formula.

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

Formula 1. greedy Algotithm

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a^* = \underset{a \in \mathcal{A}}{\arg\max} Q(s, a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

Formula 2. ε-greedy Algorithm

Therefore, Sarsa algorithm applys ε-greedy method to select the action, and then updates the Q value, pseudo code of which can be shown as follow:

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
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

✧ Code

My code is shown as follow:

```
def sarsa(trans, Q, reward, alpha, gamma, e):
    action = random.randint(0, 3)
    for times in range(1000):
        state = [3, 0]
        newstate = [1, 1]
        if (times > 0):
            maxa = -100000
            for a in range(4):
                if (Q[state[0]][state[1]][a] > maxa):
                    maxa = Q[state[0]][state[1]][a]
                    action = a
            p = []
            for a in range(4):
                if (a == action):
                    p.append(e / 4 + 1 - e)
                else:
                    p.append(e / 4)
            p = np.array(p)
            action = np.random.choice([0, 1, 2, 3], p=p.ravel())
        while (newstate[0] != 3 or newstate[1] != 11):
            newstate = trans[state[0]][state[1]][action]
            #if (newstate[0] == 3 and newstate[1] == 11): # reach the terminal
                #break
            maxa = -100000
            for a in range(4):
                if (Q[newstate[0]][newstate[1]][a] > maxa):
                    maxa = Q[newstate[0]][newstate[1]][a]
                    newaction = a
            p = []
            for a in range(4):
                if (a == newaction):
                    p.append(e / 4 + 1 - e)
                else:
                    p.append(e / 4)
            p = np.array(p)
            newaction = np.random.choice([0, 1, 2, 3], p = p.ravel())
            Q[state[0]][state[1]][action] = Q[state[0]][state[1]][action] + \
                                            alpha * (reward[state[0]][state[1]][action] + \
                                                     gamma * Q[newstate[0]][newstate[1]][newaction] - \
                                                     Q[state[0]][state[1]][action])

            state = newstate
            action = newaction

    return
```

Part 2: Q-learning Algorithm

✧ Introduction

The main difference between Q-learning Algorithm and Sarsa Algorithm is the Q-value updating. In Q-learning, greedy algorithm is used to choose an action and update Q-value while the real action to execute is still choosed by ε-greedy algorithm. Contrarily, Sarsa choose the same action in updating Q-value and execution, which determines by ε-greedy algorithm. Therefore, Q-learning is an off-policy algorithm, of which the pseudo code is shown as follow:

> Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
> 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'
> $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
> $S \leftarrow S'$;
>   until S is terminal

✧ Code

```python
def qlearning(trans, Q, reward, alpha, gamma, e):
    action = random.randint(0, 3)
    for times in range(1000):
        state = [3, 0]
        newstate = [1, 1]

        while (newstate[0] != 3 or newstate[1] != 11):
            maxa = -100000
            for a in range(4):
                if (Q[state[0]][state[1]][a] > maxa):
                    maxa = Q[state[0]][state[1]][a]
                    action = a
            p = []
            for a in range(4):
                if (a == action):
                    p.append(e / 4 + 1 - e)
                else:
                    p.append(e / 4)
            p = np.array(p)
            action = np.random.choice([0, 1, 2, 3], p=p.ravel())
            if (times == 0 and newstate[0] == 3 and newstate[1] == 0):
                action = random.randint(0, 3)
            newstate = trans[state[0]][state[1]][action]
            # if (newstate[0] == 3 and newstate[1] == 11): # reach the terminal
            # break
            maxa = -100000
            for a in range(4):
                if (Q[newstate[0]][newstate[1]][a] > maxa):
                    maxa = Q[newstate[0]][newstate[1]][a]
                    newaction = a
            Q[state[0]][state[1]][action] = Q[state[0]][state[1]][action] + \
                                            alpha * (reward[state[0]][state[1]][action] + \
                                                     gamma * Q[newstate[0]][newstate[1]][newaction] - \
                                                     Q[state[0]][state[1]][action])

            state = newstate
    return
```

Part 3: Comparative Experiment

✧ Experiment setup

In this experiment, I set γ to 1 and α to 0.5. What's more, I set the times for the algorithm to take a step as 5000.

✧ Result

To visualize the result, I use some symbols to show the route in the grid world. The action of going west, north, east, south is represented by '<-', '^', '->', '.' so that the route can be seen directly.

When we choose the ε as 0.5, the result is shown as follow:

```
The route of Sarsa
->   ->   ->   ->   ->   ->   ->   ->   ->   ->   ->   .
^    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

The route of Q-learning
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
```

It can be seen that Sarsa choose the upper route while Q-learning choose the route nearest to the cliff.

When we choose the ε as 0.005, which is a quite small value, the result is shown as follow:

```
The route of Sarsa
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

The route of Q-learning
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
```

It can be seen that Sarsa and Q-learning choose the same route which is close to the cliff.

Therefore, when the ε is large, Sarsa tends to choose a route safer while Q-learning tends to choose the optimal route, because Sarsa has an possibility to choose the non-optimal action which will lead to a huge punish near the cliff.

When the ε is small, Sarsa and Q-learning will all choose the optimal route because the possibility to choose the non-optimal action is small which allows Sarsa to go near the cliff without dropping down.