Sarsa and Q-learning

516021910212
Liuyuxi

April 2019

1 Introduction

I did experiment with model-free control, including on-policy learning (Sarsa) and off-policy learning (Q-learning). And I use Cliff Walking as my example. I wrote an agent class and make it possible to interact with the existing grid world environment classes. Then I implement the SARSA and Q-learning algorithm and looked at its training effect in the Cliff Walking world.

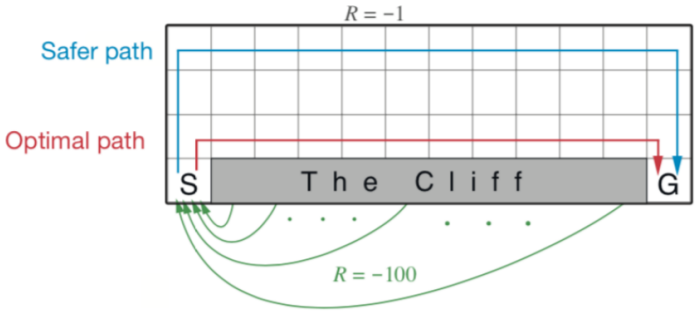

Figure 1: Cliff Walking

Consider the gridworld shown in the Figure 1. This is a standard undiscounted, episodic task, with start state (S), goal state (G), and the usual actions causing movement up, down, right, and left. Reward is -1 on all transitions except those into the region marked "The Cliff". Stepping into this region incurs a reward of -100 and sends the agent instantly back to the start.

2 Algorithm

2.1 Setup

This is a model in the Reinforcement Learning literature. In this particular case:

1. Cliff environment: The cliff has 12×4 girds, with start state (S), goal state (G)
2. Actions A: The agent can choose from up to 4 actions to move around.
3. Attenuation coefficient $\gamma = 0.9$
4. Learning rate $\alpha = 0.05$
5. Methods: Sarsa and Q-learning
6. $\epsilon = 0$ and $\epsilon = 0.1$

2.2 Algorithm

The model-free control replace the state value with the value $Q(s, a)$ of the state behavior.

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

Those algorithm start with an initial Q and policy π, first updating the q value of each state behavior pair according to this strategy, and then determining an improved greedy algorithm based on the updated Q.

2.3 ϵ-Greedy Exploration

The goal of greedy exploration is to make all possible behaviors in a certain state have a certain non-zero probability to be selected, thus ensuring continuous exploration, and choosing the current best behavior under the probability of 1-ϵ, and The probability of epsilon is chosen among all possible behaviors (including the best current behavior).

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

2.4 Sarsa

$$Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma Q(S', A') - Q(S, A))$$

2.5 Q-learning

$$Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma \max_{a'} -Q(S, A))$$

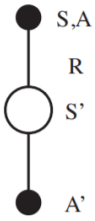

Figure 2: Sarsa

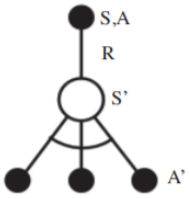

Figure 3: Q-learning

3 Code

3.1 Setup

the Cliff Walk Environment

```
def CliffWalk():
    env = GridWorldEnv(n_width=12,
                       n_height=4,
                       u_size=60,
                       default_reward=-1,
                       default_type=0,
                       windy=False)
    env.action_space = spaces.Discrete(4)  # left or right
    env.start = (0, 0)
    env.ends = [(11, 0)]
    # env.rewards=[]
    # env.types = [(5,1,1),(5,2,1)]
    for i in range(10):
        env.rewards.append((i + 1, 0, -100))
        env.ends.append((i + 1, 0))
    env.refresh_setting()
    return env
```

Define the agent with Sarsa and Q-learning process

```
class SasaAgent():
    def __init__(self, env: Env):
        self.env = env       # the current environment
        self.Q = {}          # Q(s,a)
        self.state = None    # the current observation

    def performPolicy(self, state): pass #  a Strategy

    def act(self, a):        # an action
        return self.env.step(a)

    def SasaLearning(self): pass    # learning process
    def QLearning(self): pass    # learning process
```

ϵ-strategy, the *use_ϵ* means the policy can be switched between Sarsa and Q-learning randomly.

```
    def performPolicy(self,s,episode_num,use_epsilon):
        #episilon=1.00/(episode_num+1)
        episilon=0

        Q_s=self.Q[s]
        str_act="unknown"
        rand_value=random()
        action=None
        if use_epsilon and rand_value<episilon:
            action=self.env.action_space.sample()
            #choose a random action
            #as the episode increases the unsure probability
                                                decreases
        else:
            str_act=max(Q_s,key=Q_s.get)
            action=int(str_act)
        return action
```

3.2 Sarsa Algorithm

```
    def SasaLearning(self,gamma,alpha,max_episode_num):
        total_time,time_in_episode,num_episode=0,0,0
        while num_episode<max_episode_num:#the terminal condition
            self.state=self.env.reset()
            s0=self.getStateName(self.state)
            self.assertStateInQ(s0, randomized=True)
            self.env.render()
            a0=self.performPolicy(s0,num_episode,use_epsilon=True)
            time_in_episode=0
            reward1=0
            is_Done=False
            while not is_Done:
                s1,r1,is_Done,info=self.act(a0)
                self.env.render()
                s1=self.getStateName(s1)
                self.assertStateInQ(s1,randomized=True)
                #get A'
                a1=self.performPolicy(s1,num_episode,use_epsilon=
                                                    True)
                old_q=self.getQ(s0,a0)
                new_q=self.getQ(s1,a1)
                td_target=r1+gamma*new_q
                new_q=old_q+alpha*(td_target-old_q)
                self.setQ(s0,a0,new_q)
                if(num_episode==max_episode_num):#the last episode
                    print("t:{0:>2}: s:{1}, a:{2:2}, s1:{3}". \
                          format(time_in_episode, s0, a0, s1))
                s0,a0=s1,a1
                time_in_episode+=1
                reward1+=r1
            print("SarsaEpisode {0} takes {1} steps.".format(
                num_episode, time_in_episode))
```

```
            #self.episode.append(num_episode)
            self.SarsaReward.append(reward1)
            total_time+=time_in_episode
            num_episode+=1
        return
```

3.3 Q-learning

```
    def QLearning(self,gamma,alpha,max_episode_num):
        total_time,time_in_episode,num_episode=0,0,0

        while num_episode<max_episode_num:#the terminal condition
            self.state=self.env.reset()
            s0=self.getStateName(self.state)
            self.assertStateInQ(s0, randomized=True)
            self.env.render()
            a0=self.performPolicy(s0,num_episode,use_epsilon=True)
            time_in_episode=0
            reward_in_episode=0
            is_Done=False
            while not is_Done:
                s1,r1,is_Done,info=self.act(a0)
                self.env.render()
                s1=self.getStateName(s1)
                self.assertStateInQ(s1,randomized=True)
                #get A'
                a1=self.performPolicy(s1,num_episode,use_epsilon=
                                                    False)
                old_q=self.getQ(s0,a0)
                new_q=self.getQ(s1,a1)
                td_target=r1+gamma*new_q
                new_q=old_q+alpha*(td_target-old_q)
                self.setQ(s0,a0,new_q)
                if(num_episode==max_episode_num):#the last episode
                    print("t:{0:>2}: s:{1}, a:{2:2}, s1:{3}". \
                          format(time_in_episode, s0, a0, s1))
                s0,a0=s1,a1
                time_in_episode+=1
                reward_in_episode+=r1
            print("QEpisode {0} takes {1} steps.".format(
                num_episode, time_in_episode))
            self.episode.append(num_episode)
            self.QReward.append(reward_in_episode)
            total_time+=time_in_episode
            num_episode+=1
        return
```

3.4 My print function

```
    def print(self):
        x=self.episode
        y=self.SarsaReward
        y1=self.QReward
```

```
        plt.plot(x,y,color="orange",label="Sarsa")
        plt.plot(x,y1, color="brown",label="Q-learning")
        plt.title("epsilon=0")
        plt.xlabel("Episode")
        plt.ylabel("reward")
        plt.legend()
        plt.savefig("0.png")
        plt.show()
```

4 Result

4.1 Comparison between Sarsa and Q-learning

In my test, I tried several ϵ, from $\epsilon = 0$ and $\epsilon = 0.1$ to $\epsilon = 1/(episode\_number + 1)$ and in all the test $num\_of\_Episodes = 800$. The result of $\epsilon = 0.1$ is shown in Figure 4. In the early stage of training, the individual cost is relatively large. As the number of training increases, the individual cost gradually converges to a certain value. And the ϵ causes the Curve is not smooth. Obviously, in the late training Q-learning has smaller fluctuations than Sarsa. This reflects the difference between the two algorithms obviously, that Q-learning is evaluate another strategy. The result of $\epsilon = 0$ is shown in Figure 5. When $\epsilon = 0$, it means that Sarsa and Q-learning always use greedy action selection. And Sarsa and Q-learning show no difference in this case. Another example I use $\epsilon = 1/(episode\_number + 1)$. It means with the episode_number goes, the ϵ will become smaller. This result is shown in Figure 6. And it is obvious that with the episode goes, the fluctuation is smaller.

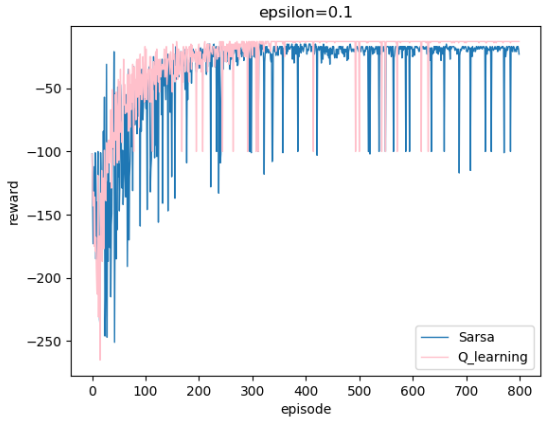

Figure 4: $\epsilon = 0.1$

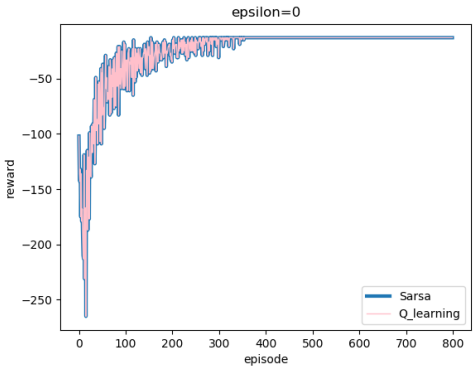

Figure 5: $\epsilon = 0$

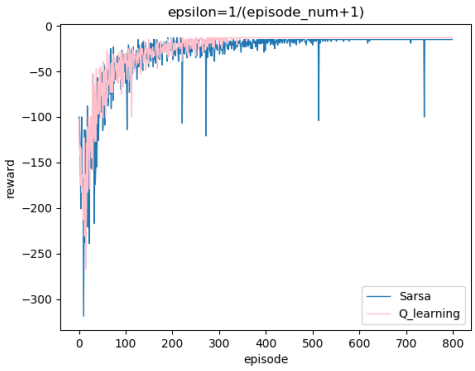

Figure 6: $\epsilon = 1/(episode\_num+1)$

4.2 Comparison between epsilon

The result is shown in Figure 7 and Figure 8. And we can get that ϵ influences the degree of fluctuation of the curve. And the learning algorithm may sometimes not converge to the optimal path. And when $\epsilon = 0$, Q-learning will degenerate to Sarsa.

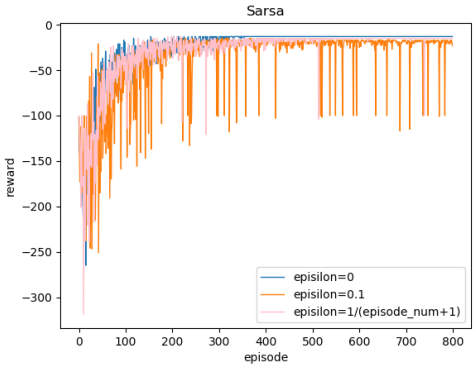

Figure 7: Comparison between Sarsa

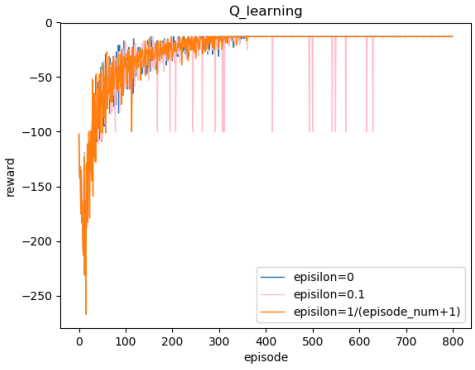

Figure 8: Comparison between Q-learning

5 Conclusion

Thanks to this project that I have a deep understanding of Sarsa and Q-learning. I also meet with some obstacles during the period. At the beginning, I imple-

ment the Sarsa and Q-learning on the same agent, and this cause the latter algorithm can always find the best path. So I implement them on two agent after adjustment. In this test I use others' girdworld environment and I implement Sarsa and Q-learning myself in the environment and test my hypothesis. Thanks for the guidance of the teacher and our teaching assistants.

References

https://github.com/qqiang00/reinforce/blob/master/reinforce/gridworld.py