人工智能实验报告 第13周

姓名:刘卓逸 学号:21307303

一.实验内容

1.实验标题

hw9 强化学习：表格型Q-Learning算法

2.实验要求

在该环境上实现表格型Q-Learning算法. 要点如下:

- 将学习算法以及各种辅助方法集成到类QLAgent中, 自定义Q函数的数据格式(建议使用numpy数组)
- 在learn()方法中实现Q函数更新, 传入的参数自定; 在get_action()方法中实现基于Q-Learning的策略, 传入的参数必须为state, eval_mode且不可修改.

二.实验过程

1.实验原理

基于表格型Q-Learning算法

与策略迭代算法类似建立一个策略评估表，并且迭代更新这个表。然而由于环境关系无法没轮都遍历所有状态，通过多次采样来更新策略。

Q-learning方法

```
初始化q值函数，其中结束状态的每个动作的q值必须为0
重复以下训练过程(episode):
    重置游戏状态s
    进行一局游戏直到游戏结束或到最大步数:
        用ε-greedy方法通过策略选择一个动作A
```

执行动作A得到回报R与新状态S'
Q(S,A)←Q(S,A)+α[(R+γ* max_{a}(Q(S',a)))-Q(S,A)]
S←S'

Sarsa方法

初始化q值函数，其中结束状态的每个动作的q值必须为0
重复以下训练过程(episode):
    重置游戏状态s,抽取初始动作
    进行一局游戏直到游戏结束或到最大步数:
        执行动作A得到回报R与新状态S'
        用ε-greedy方法通过策略选择S'的一个动作A'
        Q(S,A)←Q(S,A)+α[(R+γ*Q(S',A'))-Q(S,A)]
        S←S',A←A'

2.实验代码

```
import random
import numpy as np
from office_world import Game
from rl_test import run_test

class LearningParams:
    def __init__(self, epsilon=0.3, lr=0.1, gamma=0.9):
        """
        :param epsilon: epsilon-greedy的随机探索权重
        :param lr: 学习率
        :param gamma: 折扣因子
        """
        self.epsilon = epsilon
        self.lr = lr
        self.gamma = gamma


class TestingParams:
    def __init__(self, test_freq=1, max_ep_len=100):
        """
        :param test_freq: 测试频率，即每测试一次要训练多少步
        :param max_ep_len: 单个episode的最大长度
        """
        self.test_freq = test_freq
        self.max_ep_len = max_ep_len


class QLAgent:
    def __init__(self, num_states, num_actions, learning_params):
        self.num_states = num_states
        self.num_actions = num_actions
        self.learning_params = learning_params
```

```
        self.qtable=np.zeros([num_states,num_actions])

    def learn(self, s1, a, r, s2, done):
        x=np.argmax(s1)
        y=np.argmax(s2)
        if done:
            self.qtable[x][a]+=self.learning_params.lr*(r-self.qtable[x][a])
        else:
            self.qtable[x][a]+=self.learning_params.lr*
(self.learning_params.gamma*np.max(self.qtable[y])+r-self.qtable[x][a])


    def get_action(self, state, eval_mode=False):
        if ((not eval_mode) and np.random.random()<self.learning_params.epsilon):
            return np.random.choice(self.num_actions)
        else:
            s=np.argmax(state)
            ac=np.argmax(self.qtable[s])
            return ac


def run_experiment(env, agent, testing_params, num_episode=1000):
    for i_ep in range(num_episode):
        s1 = env.reset()
        for t in range(testing_params.max_ep_len):
            a = agent.get_action(s1,False)
            s2, reward, done, _ = env.step(a)
            # learn()中自行添加参数
            agent.learn(s1,a,reward,s2,done)
            # 转移到下一步
            s1 = s2
            # 若环境已终止，则进入下一个episode的训练
            if done:
                break
        # 每训练test_freq个episode执行一次测试
        if i_ep % testing_params.test_freq == 0:
            # 测试是在env的副本上执行，不改变原来的env信息
            test_reward = run_test(env.task, testing_params.max_ep_len, agent)
            print("Training episodes: %d, reward: %.2f" % (i_ep, test_reward))
```

Sarsa

```
class QLAgent:
    def __init__(self, num_states, num_actions, learning_params):
        self.num_states = num_states
        self.num_actions = num_actions
        self.learning_params = learning_params
        self.qtable=np.zeros([num_states,num_actions])
    def learn(self, s1, a, r, s2, a2,done):
        x=np.argmax(s1)
        y=np.argmax(s2)
        if done:
            self.qtable[x][a]+=self.learning_params.lr*(r-self.qtable[x][a])
        else:
```

```
            self.qtable[x][a]+=self.learning_params.lr*
(self.learning_params.gamma*self.qtable[y][a2]+r-self.qtable[x][a])
        pass

    def get_action(self, state, eval_mode=False):
        if ((not eval_mode) and np.random.random()<self.learning_params.epsilon):
            return np.random.choice(self.num_actions)
        else:
            s=np.argmax(state)
            ac=np.argmax(self.qtable[s])
            return ac
        pass
def run_experiment(env, agent, testing_params, num_episode=20000):
    for i_ep in range(num_episode):
        s1 = env.reset()
        a1 = agent.get_action(s1,False)
        for t in range(testing_params.max_ep_len):
            s2, reward, done, _ = env.step(a1)
            a2 = agent.get_action(s2,False)
            # learn()中自行添加参数
            agent.learn(s1,a1,reward,s2,a2,done)
            # 转移到下一步
            s1 = s2
            a1 = a2
            # 若环境已终止，则进入下一个episode的训练
            if done:
                break
        # 每训练test_freq个episode执行一次测试
        if i_ep % testing_params.test_freq == 0:
            # 测试是在env的副本上执行，不改变原来的env信息
            test_reward = run_test(env.task, testing_params.max_ep_len, agent)
            print("Training episodes: %d, reward: %.2f" % (i_ep, test_reward))
```

3.实验结果

任务	QL结果	稳定episode数	Sarsa结果	稳定episode数
get_coffee	0.92	117	0.92	不稳定
to_office	0.86	368	0.86	极不稳定
get_mail	0.81	765	-1	没稳定
to_a	1.00	1	1.00	1
to_b	0.89	208	0.89	没稳定
to_c	0.81	660	-1	没稳定
to_d	0.92	104	0.92	没稳定

对**to_office**的策略：

qlearning:

```
↓ → ↓ ↓ ← ↓ ↓ ← ← → → →
→ → → ↓ ↑ ↓ ↓ ↑ ↓ ← ← ↓
→ ↑ ↑ → ↓ ← ↑ → → ↓ ← →
→ ↑ ← → ↓ ← ↑ ↑ ↑ → → ↓
↑ ↑ ↑ ↑ ↑ ← ↑ ↑ ↑ ↑ ↑ →
→ → ↑ ↑ ↑ → ↑ ↑ ↑ ↓ → ↓
→ ↑ ← ↑ → ↑ ← ← ↑ → ↓ ↓
→ ↑ ← ← ↑ → ← ↑ → ↑ ↑ ←
↑ ↑ ↑ ↑ ↓ ↑ ← ← ↓ ← ↑ →
```

Sarsa:

```
↑ ↓ ↓ ↓ ← → ↑ → ↓ → ← ←
→ → → ↓ ↑ ↓ ↓ ↑ ↓ ↓ → ↑
→ ↑ ↑ → ↓ ← ↓ → → ← ↑ →
→ ↑ ← → ↓ ← ↑ ↑ ↑ ← → →
↑ ↑ ↑ → ↑ ↓ ↑ ↑ ↑ ↑ ↑ ↑
↑ ← ↑ ↑ ↑ ↑ ↑ ↑ ↑ → → →
↓ ↑ ← ↑ → ↑ → → ↑ → → →
↓ ↑ ← ↑ ↑ ↑ ↑ ↑ ↓ ↓ ↑ ↑
→ ↑ ← ↓ → ↑ ← ↓ ↓ ↑ → ↑
```

to_c的策略

qlearning

```
→ ↑ ↓ → → ↓ → → ↓ ↓ ↓ ↑
← → → ↓ ↑ → ↑ ↑ → → ↑ ←
↓ ↑ ↑ → → ↑ → → ↑ ↑ ↑ ←
↑ ↑ ↓ ← ← ← ↓ ↓ ↓ → ↑ ←
← ↑ ↑ ↓ ← ↓ ← ↑ ↓ ↑ ↑ ↑
← → ↑ ↓ → → ← ↓ → ↑ → ↑
↑ ↑ ↓ ↓ → ↓ ↓ → ↓ → ↑ ←
→ → → ↓ ↑ → ↓ ↑ → → ↑ ←
→ → ↑ → → ↑ → → ↑ → ↑ ←
```

Srasa

```
↑ ↓ ↓ → ← ↑ ↓ ← ← → ↑ ↑
↑ → → ↓ ↑ ↓ ↓ ↑ → → ↑ ↑
↓ → ↑ ↓ ← ↑ → ← ← ↑ ↑ ↑
```

→ ← ← → ↑ ↓ ← ↑ ↓ ↓ ← ↑
↑ ↑ ↑ ↑ ← ↓ ↓ ↓ ↓ ↓ ↑ ↓
← → ← ← ↑ ↓ ↑ ↑ ↑ → ← →
↓ ← → ↓ → ↑ → ← ↑ → → →
↓ ← ↓ ↓ ↑ ↑ ↑ ↑ → ↓ ↑ ↓
← ← ← ↓ ← ↑ ↑ → ↑ → → ↑

4.结果分析

从to_c看出Sarsa开头就没指对。

尝试更改Sarsa：更新上一步时候的抽取动作A'没有随机直接argmax,但是真执行A'是另外抽取的带随机的A'，这样sarsa就和qlearning效果一样了，但是经过分析，这修改后的sarsa就是qlearning啊.但也可以发现sarsa的问题：更新价值函数的时候，原本应该好的值更新，但是却突然的随机使得更新的值有问题。

四.实验总结

Q-learning实现效果挺好的，但Sarsa的效果却不太好.