实验二 RL 报告

江辉--171250506

2020-3-31

南京大学

软件学院

目录

一、 实验目的

本次实验使用 python 构建一个 Tabular Q-learning 算法完成完成寻宝游戏，了解 Q-learning 过程

二、 内容描述

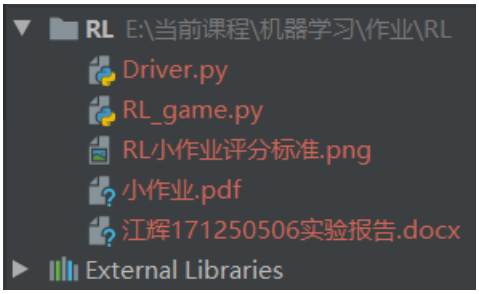
本程序用 python 编写，手写 Q-learning 算法的实现。预先定义好游戏规则和各个超参数，Q table 来记录并预测行动和回报，使用ε-greedy 策略来选择动作。

1、项目目录描述

本项目推荐使用 pycharm 打开

Driver.py 里写了 main 函数和几个对照试验的参数，我把对应的对照试验都封装成函数了，可以直接在 main 函数里调用运行

RL_game.py 里写了 RL model 的类，所有模型相关的函数和参数都以注释的形式写在这个文件里



三、 运行用例和截图

1、输出的 Q 表和学习过程

右图为游戏地图长度为 6，贪婪系数ε=0.1，α=0.1，γ=0.9 的结果。可以看到，基本上在第 5 次学习之后基本就得到了目标的 Q table，已经可以确定是往右走了，此后除了随机选择到往左走外，都会选择往右走。可以看到每一轮后 Q 表都有更新，且其中采取 right 动作的预估值远高于采取 left 动作的预估值(如果ε-贪婪策略没有随机到采取往左走就不会增加，而由于90%的概率选取最高估值动作，所以一旦学习到 right，就很少选择往左走)

```python
GAME_LENGTH = 6  # 长度
MAX_EPISODES = 15  # 学习轮数

game = RLGame(GAME_LENGTH)
for i in range(MAX_EPISODES):
    print("Episode %d :" % i)
    game.reset()
    q_table = game.learn(to_print=True)
    print(q_table)
```

```python
ACTIONS = ["left", "right"]  # 可选动作
EPSILON = 0.1  # 贪婪系数
ALPHA = 0.1  # 学习系数
GAMMA = 0.9  # 衰减系数
FRESH_TIME = 0.3  # 每次跳动时间(s)
```

```
Driver
Episode 0 :
Finish
Total Steps = 26
   left  right
0   0.0    0.0
1   0.0    0.0
2   0.0    0.0
3   0.0    0.0
4   0.0    0.1
5   0.0    0.0
Episode 1 :
Finish
Total Steps = 11
   left  right
0   0.0  0.000
1   0.0  0.000
2   0.0  0.000
3   0.0  0.009
4   0.0  0.190
5   0.0  0.000
Episode 2 :
Finish
Total Steps = 7
   left     right
0   0.0  0.00000
1   0.0  0.00000
2   0.0  0.00081
3   0.0  0.02520
4   0.0  0.27100
5   0.0  0.00000
Episode 3 :
Finish
```

```
Driver
5   0.0  0.000000
Episode 4 :
Finish
Total Steps = 6
   left     right
0   0.0  0.000007
1   0.0  0.000335
2   0.0  0.006934
3   0.0  0.073314
4   0.0  0.409510
5   0.0  0.000000
Episode 5 :
Finish
Total Steps = 5
   left     right
0   0.0  0.000036
1   0.0  0.000926
2   0.0  0.012839
3   0.0  0.102839
4   0.0  0.468559
5   0.0  0.000000
Episode 6 :
Finish
Total Steps = 5
   left     right
0   0.0  0.000116
1   0.0  0.001989
2   0.0  0.020810
3   0.0  0.134725
4   0.0  0.521703
5   0.0  0.000000
Episode 7 :
```

```
Driver
Episode 8 :
Finish
Total Steps = 5
   left     right
0   0.0  0.000585
1   0.0  0.006073
2   0.0  0.042907
3   0.0  0.202643
4   0.0  0.612580
5   0.0  0.000000
Episode 9 :
Finish
Total Steps = 7
       left     right
0  0.000000  0.001512
1  0.000097  0.009328
2  0.000000  0.056855
3  0.000000  0.237511
4  0.000000  0.651322
5  0.000000  0.000000
Episode 10 :
Finish
Total Steps = 5
       left     right
0  0.000000  0.002200
1  0.000097  0.013512
2  0.000000  0.072545
3  0.000000  0.272379
4  0.000000  0.686189
5  0.000000  0.000000
Episode 11 :
```

四、 Q-LEARNING 伪代码描述

Initialize Q(s,a) arbitrarily：即初始化 Q 表(state-action 的估值表)
Repeat (for each episode)：每一回合从起点开始，到终点结束
    Initialize stete s
    Repeat (for each step of episode)：回合中每一步都要进行学习
        // 在当前状态和环境下先通过ε-贪婪策略从 Q table 中选取一个动作
        Choose action a from s using policy derived from Q (e.g. ε-greedy)
        // 预先计算采取动作后的回报和下一状态
        Take action a, observe reward r, next state s'
        // 下式右边的 Q(s,a)是当前 Q table 中当前状态下采取动作 a 的预估价值
        // maxQ(s',a')是采取动作 a 之后的新状态下的最优动作的价值，即预判一步
        // γ是代表预判的价值是不确定的，需要打折扣
        // r 是采取动作 a 获得的实际价值
        // r + maxQ(s',a')得到采取动作 a 的实际价值，减去 Q(s,a)获得矫正值

        $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$

        //上面以α的速度更新了 Q(s, a)后，下面就是切换状态并准备进行下一步
        s ← s'
    until s is terminal state

五、 过程分析描述

1、超参数选择

本次实验中画布的长度=6，最大学习轮数=15，
贪婪系数ε=0.1，学习速率α=0.1，衰减系数γ=0.9，每一步状态切换用时 0.3s
这些超参数都可以在代码中手动设置切换

Driver.py

```
if __name__ == "__main__":
    # custom()

    GAME_LENGTH = 6  # 长度
    MAX_EPISODES = 15  # 学习轮数

    game = RLGame(GAME_LENGTH)
    for i in range(MAX_EPISODES):
        print("Episode %d :" % i)
        game.reset()
        q_table = game.learn(to_print=True)
        print(q_table)
```

RL_game.py

```
ACTIONS = ["left", "right"]  # 可选动作
EPSILON = 0.1  # 贪婪系数
ALPHA = 0.1  # 学习系数
GAMMA = 0.9  # 衰减系数
FRESH_TIME = 0.3  # 每次跳动时间(s)
```

2、Q TABLE 设计

本次实验有 6 个状态，每个状态都有 left 和 right 两个动作可选，因此 Q table 为如下的形式

game.QTable

	left	right
0	0.00003	0.00481
1	0.00000	0.02719
2	0.00000	0.11226
3	0.00187	0.34079
4	0.00000	0.74581
5	0.00000	0.00000

相应的初始化语句如下：

```
class RLGame:
    def __init__(self, length: int, actions=None,
                 epsilon=EPSILON, alpha=ALPHA, gamma=GAMMA, fresh_time=FRESH_TIME):
        """
        :param length: 游戏长度
        :param actions: 动作列表，默认使用["left", "right"]
        :param epsilon: 贪婪系数
        :param alpha: 学习系数
        :param gamma: 衰减系数
        :param fresh_time: 每次跳动时间(s)
        """
        if actions is None:
            actions = ["left", "right"]
        self.QTable = pd.DataFrame(
            np.zeros((length, len(actions))),
            columns=actions
        )
```

3、EPSILON-贪婪策略

下面ε=0.1，首先生成随机数，判断是否小于ε，小于时，采取随机动作
大于时，选择最高预估值的动作进行选择，为了避免在多个最高价值动作时只选择前面的，我打乱了 index 来随机选取最大值标签

Driver.py RL_game.py

```
75      def choose_action(self):
76          """
77          :return: 当前状态下通过ε-贪婪策略选择动作
78          """
79          if np.random.uniform() < self.epsilon:
80              action = np.random.choice(self.actions)
81          else:
82              index = self.QTable.iloc[self.state, :].index
83              candidate_actions = self.QTable.take(np.random.permutation(len(index)),
    axis=1).iloc[self.state, :]
84              action = candidate_actions.idxmax()
85          return action
```

4、回报 R 值定义

终点的回报值为 1，其余每一个位置的回报值均设置为 0，每次预估观测时同时返回对应回报

```
        self.QTable = pd.DataFrame(
            np.zeros((length, len(actions))),
            columns=actions
        )
        self.rewords = [0] * (length - 1) + [1]
        np.random.seed(int(time.time()))
        self.state = 0
```

```
    def observe(self, action):
        """
        :param action: 采取的动作
        :return: 采取该动作后的下一个状态和回报
        """
        if action == "left":
            next_state = max(self.state - 1, 0)
        else:
            next_state = min(self.state + 1, self.length - 1)
        reward = self.rewords[next_state]
        return next_state, reward
```

5、Q 值更新过程

每次迭代的每一步都需要进行 Q 值学习

首先是根据当前状态和环境选择一个动作 action，

再从 Q 表中获取当前状态下采取 action 的预估价值 q_predict

预判采取动作 action 后的下一个状态 next_state 和获得的回报值 reward。

从 Q 表中获取 next_state 状态下的能获得的最大预估价值 q_real

再计算 reward + γ * q_real 作为在当前状态下采取 action 获得的实际价值

用实际价值-预估价值作为学习反馈，更新在当前状态下采取动作 action 的 q_predict

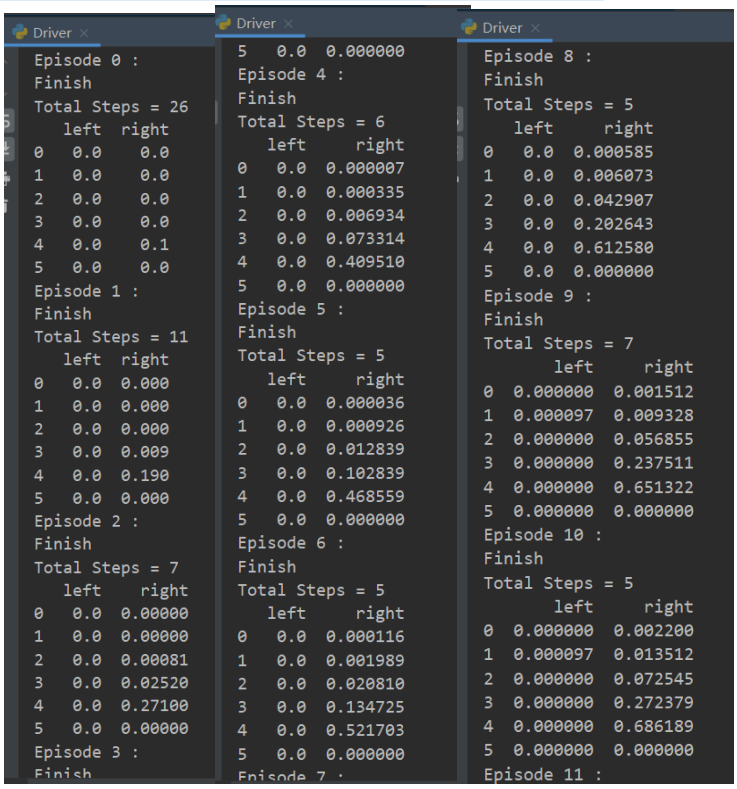
最后更新状态并进行下一步

Driver.py RL_game.py

```
    def learn(self, to_print=False):
        """
        一个回合的学习过程
        :param to_print: 是否打印过程信息
        :return: 该回合学习后的Q table
        """
        while not self.is_terminated():
            if to_print:
                self.print()
            self.stepCount += 1
            action = self.choose_action()
            q_predict = self.QTable.loc[self.state, action]
            next_state, reward = self.observe(action)
            q_real = self.QTable.iloc[next_state, :].max()
            self.QTable.loc[self.state, action] +=\
                self.alpha * (reward + self.gamma * q_real - q_predict)
            self.state = next_state
        if to_print:
            print("\rFinish")
            print("Total Steps = %s" % self.stepCount)
        return self.QTable
```

6、Q 表分析

右图是每一轮 Q 表的更新过程



从中可以看到，在第一轮结束时只有 right 的最后一个值有更新(第 5 是终点，不做动作预估处理)，这是因为初始化时全部为 0，随机移动后只在终点位置才会获得一个 reward=1，因此第一轮更新只会更新终点前的最后一步(4)的 q 值。

同理第二轮的时候只在倒数第二步(3)时采取了 right 动作才会获得预估回报，由于衰减系数为 0.9，故(3)更新为α*0.9*0.1，而 alpha=0.1，所以最后得到的是 0.009，而此时(4)因为已经有了一个 q 值，因此在学习时计算式是 q = q + 0.1*(1+0.9*0 – 0.1) = 0.19

以此类推，基本上在第 5 次学习之后基本就得到了只往右走的 Q table，此后除了随机选择到往左走外，都会选择往右走。可以看到每一轮后 Q 表都有更新，且其中采取 right 动作的预估值远高于采取 left 动作的预估值，left 中一些值为 0 是因为如果ε-贪婪策略没有随机到采取往左走就不会增加，而由于 90%的概率选取最高估值动作，所以一旦学习到 right，就很少选择往左走，也就很少更新 left 的值。这点从上图的 Episode9 可以看到，步数为 7，意味着在状态(1)随机往左走了 1 次，导致多了 2 步(还有一步往右)，此时更新了状态(1)处 left 的 q 值。

六、实验中出现的问题和相应的解决办法

1、 对 pandas 不熟悉，所以在编写的时候一直出 bug，然后去查函数的用法