

人工智能实验报告 第十三周

姓名:丁晓琪 学号:22336057

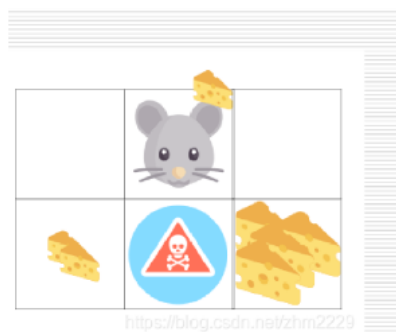
一.实验题目

You will implement Sarsa and Q-learning for the Maze environment from **OpenAI Gym**. We have provided custom versions of this environment. In the scenario, a red rectangle (agent) are initialized at the maze made up of 4×4 grids and can only observe its location. At each time step, agent can move to one of four neighboring grids. The reward is +1 if agent is located at the yellow grid, -1 if agent reaches the black grid, otherwise 0.

二. 实验内容

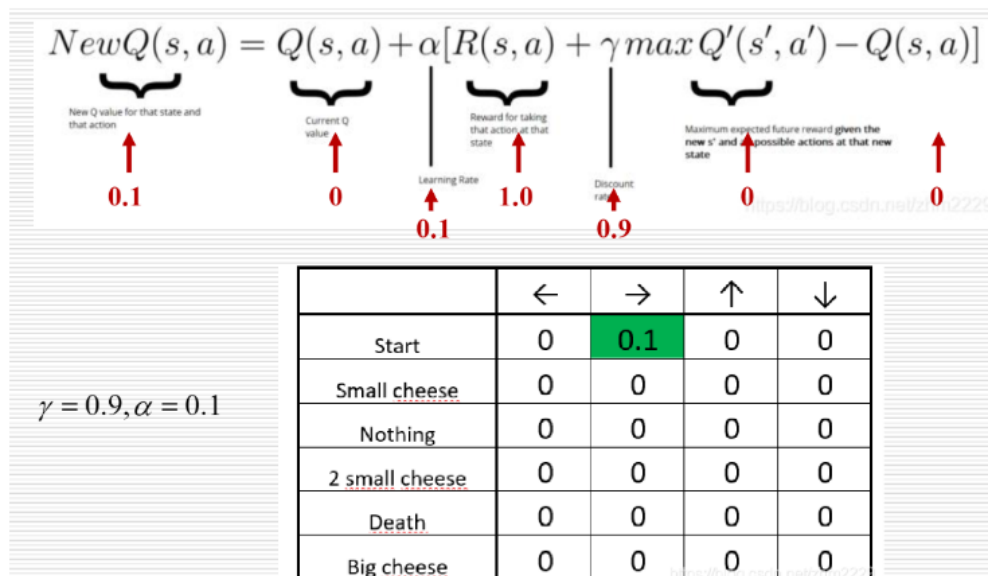
1.算法原理

- 背景:
 - s :当前状态, a :当前状态选择动作, s' :下一个状态, a' :下一个状态选择的动作
 - $Q(s, a)$: Q值函数, 给定 s 状态选择 a 动作得到的预期回报。实验中 $Q(s, a)$ 用二维表格数组存储, 行代表状态, 列代表采取动作



	←	→	↑	↓
Start	0	0	0	0
Small cheese	0	0	0	0
Nothing	0	0	0	0
2 small cheese	0	0	0	0
Death	0	0	0	0
Big cheese	0	0	0	0

- $TD\ target$: 时间差分目标, $TD = reward(s, a) + \gamma Q(s', a')$ (当前动作 a 在 s 下获得的真实奖励+衰减因子*下一个状态和选择动作下的期望值)。比 $Q(s, a)$ 可信, 用于迭代更新 $Q(s, a)$
- Q_learning:
 - 已知: 当前状态 s .
 - 训练中动作选择: $\epsilon - greedy$ 策略, 规定一个 ϵ 值 (选择贪心策略的概率), 当随机数概率大于 ϵ 时选择 Q 值最大的动作, 否则随机选择动作
 - (1) 选择动作能得到最大的期望回报
 - (2) 能够探索环境状态, 从而学习整合不同动作选择下的期望回报
 - $Q(s, a)$ 的更新: 根据学习率 α 和当前 $Q(s, a)$ 与贪心策略下 $TD\ target$ 的差距更新:
$$Q(s, a) = Q(s, a) + \alpha [reward(s, a) + \gamma \max_a Q(s', a') - Q(s, a)]$$



伪代码:

```

Initialize  $Q(s, a)$  arbitrarily
Repeat (for each episode):
  Initialize  $S$ 
  Repeat (for each step of episode):
    Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
    Take action  $A$ , observe  $R, S'$ 
     $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$ 
     $S \leftarrow S'$ 
  until  $S$  is terminal
  
```

• Sarsa:

- 已知: 当前状态 s .
- 训练中动作选择: $\epsilon - greedy$ 策略, 规定一个 ϵ 值 (选择贪心策略的概率), 当随机数概率大于 ϵ 时选择 Q 值最大的动作, 否则随机选择动作

(1) 选择动作能得到最大的期望回报

(2) 能够探索环境状态, 从而学习整合不同动作选择下的期望回报

- $Q(s, a)$ 更新: 与 Q -learning 不同, $TDtarget$ 中动作 a' 的选择不是完全的贪心策略 (选择让 $Q(s', a')$ 最大的 a'), 而是和训练中动作选择一样的 $\epsilon - greedy$ 策略

$$Q(s, a) = Q(s, a) + \alpha [reward(s, a) + \gamma Q(s', a') - Q(s, a)]$$

伪代码:

```

Initialize  $Q(s, a)$  arbitrarily
Repeat (for each episode):
  Initialize  $S$ 
  Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
  Repeat (for each step of episode):
    Take action  $A$ , observe  $R, S'$ 
    Choose  $A'$  from  $S'$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
     $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$ 
     $S \leftarrow S'; A \leftarrow A'$ 
  until  $S$  is terminal
  
```

2.关键代码展示

- 游戏背景：

4x4 的网格，黑色代表障碍 reward=-1,黄色代表终点 reward=1

```
1         # 获得当前所在网格索引：
2         s = env.canvas.coords(env.rect)
3         state[0]=int((s[1]-5)//40)
4         state[1]=int((s[0]-5)//40)
5         # 选择动作和获得下一个状态和当前奖励：
6         observation_, reward, done = env.step(action)
```

- Q_learning:

- 相关参数设置和Q(s,a)表格建立（全部初始化为0）

```
1         self.actions = actions # a list
2         self.lr = learning_rate
3         self.gamma = reward_decay
4         self.epsilon = e_greedy #随机选择动作的概率
5         #表格存储16个状态s在4个a的状态下的Q(s,a)，用二维数组存储，行为0-15
        (i+j)，列为0-3对应a
6         #全部初始化为0
7         self.q_table=np.zeros((16,len(self.actions)))
8
```

- 动作选择： $\epsilon - greedy$ 策略

注意：选择动作后要判断下一个状态是否越界

在贪心策略时，如果最大Q值对应的动作有多个要随机选择，否则可能出现前期Q值都初始化为0，每次开始探索选择总是选择一个特定方向的动作，探索别的动作的概率低，学习缓慢。还可能出现在两个状态间来回运动切换，陷入循环。

```
1
2         def choose_action(self, observation):
3             i=observation[0] #行：
4             j=observation[1] #列：
5             s=i*4+j #在q_table中的位置
6             state_=[i,j]
7             # 2.生成随机数概率与epsilon比较 epsilon-greedy策略
8             action=0
9             p=np.random.rand()
10            # 【1】在0-epsilon范围内，随机选择 #注意选择动作检查下一个状态是否存
        在
11            if(p<=self.epsilon):
12                action = np.random.randint(0, len(self.actions))
13                self.get_s_(state_,action)
14                while(self.check_state_exist(state_)==0): #越界重新生成
15                    state_=[i,j] #复原状态
16                    action = np.random.randint(0, len(self.actions))
17                    self.get_s_(state_,action)
18                return action
19            # 【2】.选择Q值最大的动作
20            else:
```

```

21         #坚持越界行为, 越界重新生成
22         q_s_a=np.copy(self.q_table[s]) #深拷贝不要破坏q_table
23         max_arr=[] #存放最大值对应的索引
24         max=np.amax(q_s_a)
25         for index in range(0,len(q_s_a)):
26             if(q_s_a[index]==max):
27                 max_arr.append(index)
28         ran= np.random.randint(0, len(max_arr)) #在拥有最大值的动作中随机选一个
29         action=max_arr[ran]
30
31         self.get_s_(state_,action)
32         while(self.check_state_exist(state_)==0): #越界重新生成
33             state_=[i,j]
34             q_s_a[action]=-10000 #失去竞选资格
35             max_arr=[]
36             max=np.amax(q_s_a)
37             for index in range(0,len(q_s_a)):
38                 if(q_s_a[index]==max):
39                     max_arr.append(index)
40             ran= np.random.randint(0, len(max_arr))
41             action=max_arr[ran]
42             self.get_s_(state_,action)
43         return action
44

```

- Q(s,a)的更新学习: a'选择是贪心策略。

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

```

1     def learn(self, s, a, r, s_):
2         ''' update q table ''' #s: 当前状态坐标, a: 动作, r: a动作后的实际奖励
           s_:a动作后的坐标
3         state=s[0]*4+s[1]
4         Q_s_a=self.q_table[state][a]
5         state_new=s_[0]*4+s_[1]
6         Q_s_a_newmax=np.amax(self.q_table[state_new])
7         #更新q_table
8         self.q_table[state][a]=Q_s_a+self.lr*
(r+self.gamma*Q_s_a_newmax-Q_s_a)

```

- 补充: 如何规避学习时, 总是只选择某方向的动作和 在某些状态间切换循环而不向外探索。
 ϵ 下降: 在训练初期将 ϵ 设置为较高值, 让主体有较大的概率随机向外探索学习更新 $Q(s,a)$ 。
 在训练过程中逐渐下降 ϵ ,使得在 $Q(s,a)$ 收敛时, 主体能够与较大的概率选择最优的动作, 学习走到终点的路径

```

1         if episode%10==0:
2             RL.epsilon-=0.02
3         if episode>90:
4             RL.epsilon=0.05

```

- 总: 让主体进行100次游戏, 在迭代中更新Q值和学习 reward 最大的路径

```

1         for episode in range(100):
2             # initial observation

```

```

3      observation = env.reset()
4      if episode%10==0:
5          RL.epsilon-=0.02
6      if episode<10:
7          RL.epsilon=0.05
8      while True:
9          env.render() #更新上一次的动作
10         #选择动作
11         state=[0,0]
12         s = env.canvas.coords(env.rect)
13         state[0]=int((s[1]-5)//40)
14         state[1]=int((s[0]-5)//40)
15         action = RL.choose_action(state)
16         #执行动作
17         observation_, reward, done = env.step(action)
18         #学习，更新表格
19         #####
20         state_=[0,0]
21         s = env.canvas.coords(env.rect)
22         state_[0]=int((s[1]-5)//40)
23         state_[1]=int((s[0]-5)//40 )
24         RL.learn(state, action, reward, state_)
25
26         observation = observation_
27
28         # break while loop when end of this episode
29         if done:
30             break

```

- Sara:
 - 相关参数设置：同Q_learning
 - 动作选择：同Q_learning
 - Q值更新：a'采用 $\epsilon - greedy$ 策略：随机数大于 ϵ 时选择s'下最大的Q(s',a')值，否则随机选择s'下的Q(s',a')值

```

1      def learn(self, s, a, r, s_):
2          ''' update q table ''' #s: 当前状态的像素坐标, a: 动作, r: a动作后
          的实际奖励 s_:a动作后的像素坐标
3          # YOUR IMPLEMENTATION HERE #
4          state=s[0]*4+s[1]
5          Q_s_a=self.q_table[state][a]
6          state_new=s_[0]*4+s_[1]
7          #也需要用epsilon来选择S_时的q
8          p=np.random.rand()
9          if p>self.epsilon:
10             Q_s_a_newmax=np.amax(self.q_table[state_new])
11         else:
12             action = np.random.randint(0, len(self.actions))
13             Q_s_a_newmax=self.q_table[state_new][action]
14         #更新q_table
15         #改一下每个空格子的奖励,
16
17         self.q_table[state][a]=Q_s_a+self.lr*
          (r+self.gamma*Q_s_a_newmax-Q_s_a)

```

```
18 print("now:",s," q_s_a:",self.q_table[state])
```

- 补充：为了规避训练时在某个地方陷入死循环，在选择动作时为可能陷入循环的动作加以惩罚

```
1 self.last_action=-3 #记录上一次选择动作
2 def choose_action(self, observation):
3     # 1.解锁状态（当前位置）
4     #行：左上角x坐标/一个方格像素值 #修改放在主函数解
5     i=observation[0]
6     #列：左上角y坐标/一个方格像素值
7     j=observation[1]
8     #在q_table中的位置
9     s=i*4+j
10    state_[i,j]
11    # YOUR IMPLEMENTATION HERE #
12    # 2.生成随机数概率与epsilon比较
13    action=0
14    p=np.random.rand()
15    # 3.在0-0.1范围内，随机选择 #这里选择动作要看动作后的状态是否存在
16    if(p<=self.epsilon):
17        action = np.random.randint(0, len(self.actions))
18        self.get_s_(state_,action)
19        while(self.check_state_exist(state_)==0): #越界重新生成
20            state_[i,j]#复原状态
21            action = np.random.randint(0, len(self.actions))
22            self.get_s_(state_,action)
23        self.last_action=action
24        return action
25    # 4.选择Q值最大的动作
26    else:
27        #坚持越界行为，越界重新生成
28        q_s_a=np.copy(self.q_table[s]) #深拷贝不要破坏q_table
29        #重复惩罚：要是和上一次动作相反，也就是会使当前状态回到上一个状态时加个惩罚
30
31        if(self.last_action!=-3):
32            if(self.last_action<=1 ):
33                q_s_a[1-self.last_action]-=0.005
34            else:
35                q_s_a[5-self.last_action]-=0.005
36
37        max_arr=[]
38        max=np.amax(q_s_a)
39        for index in range(0,len(q_s_a)):
40            if(q_s_a[index]==max):
41                max_arr.append(index)
42        ran= np.random.randint(0, len(max_arr))
43        action=max_arr[ran]
44
45        self.get_s_(state_,action)
46        while(self.check_state_exist(state_)==0): #越界重新生成
47            state_[i,j]
48            q_s_a[action]=-10000 #失去竞选资格
49            max_arr=[]
50            max=np.amax(q_s_a)
51            for index in range(0,len(q_s_a)):
52                if(q_s_a[index]==max):
```

```

52         max_arr.append(index)
53         ran= np.random.randint(0, len(max_arr))
54         action=max_arr[ran]
55         self.get_s_(state_,action)
56         self.last_action=action
57         return action

```

三.实验结果及分析

- Q_learning: 在最后10次迭代中，路线收敛到了一条，并且能够到达终点坐标（2，2）（由于最后一个状态的时候，一次游戏已经结束，所以没有打印出来）

```

91 time:
now: 0 0
now: 0 1
now: 0 2
now: 0 3
now: 1 3
now: 2 3
92 time:
now: 0 0
now: 0 1
now: 0 2
now: 0 3
now: 1 3
now: 2 3
93 time:
now: 0 0
now: 0 1
now: 0 2
now: 0 3
now: 1 3
now: 2 3

```

```

94 time:
now: 0 0
now: 0 1
now: 0 2
now: 0 3
now: 1 3
now: 2 3
95 time:
now: 0 0
now: 0 1
now: 0 2
now: 0 3
now: 1 3
now: 2 3
96 time:
now: 0 0
now: 0 1
now: 0 2
now: 0 3
now: 1 3
now: 2 3

```

```
97 time:
now: 0 0
now: 0 1
now: 0 2
now: 0 3
now: 1 3
now: 2 3
98 time:
now: 0 0
now: 0 1
now: 0 2
now: 0 3
now: 1 3
now: 2 3
99 time:
now: 0 0
now: 0 1
now: 0 2
now: 0 3
now: 1 3
now: 2 3
game over
```

- Sara:在最后10次迭代中，路线收敛到了一条，并且能够到达终点坐标（2，2）（由于最后一个状态的时候，一次游戏已经结束，所以没有打印出来）

```
91 time:
now: 0 0
now: 1 0
now: 2 0
now: 3 0
now: 3 1
now: 3 2
92 time:
now: 0 0
now: 1 0
now: 2 0
now: 3 0
now: 3 1
now: 3 2
93 time:
now: 0 0
now: 1 0
now: 2 0
now: 3 0
now: 3 1
now: 3 2
```



```
94 time:
now: 0 0
now: 1 0
now: 2 0
now: 3 0
now: 3 1
now: 3 2
95 time:
now: 0 0
now: 1 0
now: 2 0
now: 3 0
now: 3 1
now: 3 2
96 time:
now: 0 0
now: 1 0
now: 2 0
now: 3 0
now: 3 1
now: 3 2
```

```
97 time:
now: 0 0
now: 1 0
now: 2 0
now: 3 0
now: 3 1
now: 3 2
98 time:
now: 0 0
now: 1 0
now: 2 0
now: 3 0
now: 3 1
now: 3 2
99 time:
now: 0 0
now: 1 0
now: 2 0
now: 3 0
now: 3 1
now: 3 2
game over
```

四.参考内容

课程ppt