作业 4：TD_Sarsa_Q-learning_有风网格世界实验报告

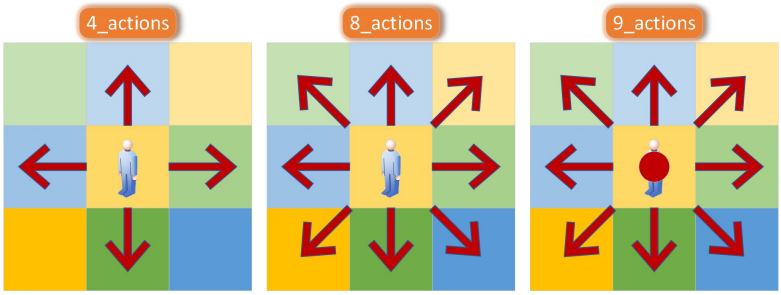
人工智能 91 卢佳源 2191121196

一、 实验目的：

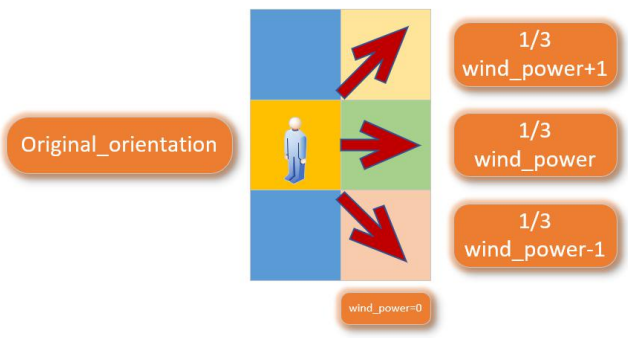
a) 理解时序差分学习的原理，以及同轨策略下的时序差分控制 Sarsa 和离轨策略下的时序差分控制 Q-learning；
b) 代码实现 TD 学习的 Sarsa 和 Q-learning，解决有风网格世界问题。

二、 问题重述：有风网格世界问题

a) 定义一个网格世界，指定起始状态和目标（终止）状态，网格世界的每一列都存在不同风力的向上的侧风，风会使 Agent 在完成该时刻的动作后按照 Agent 所处的列定义的风力向上移动相应风力的格子数（在保证不出边界的情况下）；
b) 每一幕：
   i. 任务：要求 Agent 从规定的起始状态位置按照一定的动作序列，同时遵循风力影响的规则，到达规定的终止状态位置。
c) 一幕中的每个时刻（每一步）：
   i. 动作：三种动作集合，分别包括 4，8 和 9 种动作：
      1. 4 动作集合：向上一格、向下一格、向左一格、向右一格；
      2. 8 动作集合：向上一格、向下一格、向左一格、向右一格、向左上一格、向左下一格、向右上一格、向右下一格；
      3. 9 动作集合：向上一格、向下一格、向左一格、向右一格、向左上一格、向左下一格、向右上一格、向右下一格、保持原位置不动；
      4. 上述三种动作集合的可视化简图如下：



      5. 加入随机性强度的风的可视化见图如下：



ii. 每一步的动作选择采用ε-greedy 算法，其中ε=0.1；
iii. 状态：Agent 当前所处的位置；
iv. 动作价值更新时选用的步长α=0.5；
v. 每一步的即时奖励：-1
vi. 折扣因子：1

d) 约束：
i. 后继状态由当前状态、当前选择的动作以及当前位置对应的风力决定；
ii. 当上述后继状态计算结果超出了规定的网格世界的边界，则水平或垂直移动到当前行或列的最近的边界处作为下一状态位置，具体描述如下：
1. 若 i 中计算的后继状态位置的行或列小于 0，则后继状态位置相应的行或列置为 0；
2. 若 i 中计算的后继状态位置的行或列大于等于行数或列数，则后继状态位置相应的行或列置为（行数-1）或（列数-1）；

三、 实验环境：

a) IDE：VSCode，Python-3.9.7；
b) 编程语言：Python；
c) 文件路径：C:\Users\jiayuan lu\OneDrive - MSRA\桌面\大三下\RL\作业 4 TD_Sarsa_QL_6_9\TD_Sarsa.ipynb 和 TD_QL.ipynb；

四、 实验原理和思路：

a) 首先初始化动作集合、策略数组、动作价值函数（初始化为 0）、ε、α、风力大小、幕数、每一步的即时奖励等参数和变量；
b) 按照上述约束设计计算下一状态的函数；
c) 按照ε-greedy 算法原理设计选择当前要执行的动作的函数；
d) 对每一幕循环（Sarsa）：
i. 初始化一幕中的初始状态，其中起始状态已由题目设定；
ii. 用ε-greedy 算法在当前状态 S 下选择当前要执行的动作 A：
1. 对该幕中的每一步循环：
a) 在 S 下执行动作 A，得到即时奖励 R 和下一状态位置 S‘；
b) 使用ε-greedy 算法在 S‘状态下选择下一时刻要执行的动作 A‘；
c) 利用如下公式进行动作价值更新：

$$\mathrm{Q}(S,\mathrm{A}) \leftarrow \mathrm{Q}(S,\mathrm{A}) + \alpha[R + \Upsilon\mathrm{Q}(S',\mathrm{A}') - \mathrm{Q}(S,\mathrm{A})]$$

d) 更新状态和动作：

$$S \leftarrow S'$$
$$\mathrm{A} \leftarrow \mathrm{A}'$$

e) 直到当前状态到达题目设定的终止状态位置为止，结束当前幕，继续进入下一幕进行计算；

e) 对每一幕循环（Q-learning）：
i. 初始化一幕中的初始状态，其中起始状态已由题目设定；

1. 对该幕中的每一步循环：
   a) 使用ε-greedy 算法在当前状态 S 下选择当前时刻要执行的动作 A；
   b) 在 S 下执行动作 A，得到即时奖励 R 和下一状态位置 S'；
   c) 利用如下公式进行动作价值更新：

$$Q(S,A) \leftarrow Q(S,A) + \alpha[R + \Upsilon \max_a Q(S',a) - Q(S,A)]$$

   d) 更新状态：

$$S \leftarrow S'$$

   e) 直到当前状态到达题目设定的终止状态位置为止，结束当前幕，继续进入下一幕进行计算；

f) 绘制三种动作集合对应的逐幕累积起来的 Agent 所走的总步数和 Agent 完成任务的总幕数（即 Agent 成功到达目标状态的总次数）的曲线图，比较不同动作集合到达最优策略的性能；

五、实验代码：只粘贴 Sarsa 和 Q-learning 的核心部分代码，完整代码见附件。

a) Sarsa：

```
step_8=[]
EP_8=[]
for i in range(episode):
    state={}
    t=0
    R=0
    start_row=3
    start_col=0
    state[t]=[]
    state[t].append(start_row)
    state[t].append(start_col)
    curr_state=state[t]
    c=[]
    for j in range (0,len(action)):
        c.append(j)
    pr=[]
    for k in range(len(action)):
        pr.append(1/len(action))
    if np.random.random()<epsilon:
        A=action[(np.random.choice(c, p=pr))]
    else:
        A=epsilon_greedy_8(curr_state)
    while True:
        next_s=next_state(curr_state,A)
        R+=reward_each_step
        if np.random.random()<epsilon:
            A1=action[(np.random.choice(c, p=pr))]
```

```
        else:
            A1=epsilon_greedy_8(next_s)
        Q[curr_state[0],curr_state[1],action.index(A)]+=alpha*(R+gamma*Q
[next_s[0],next_s[1],action.index(A1)]-Q[curr_state[0],curr_state[1],ac
tion.index(A)])
        curr_state=next_s
        A=A1
        t+=1
        state[t]=curr_state
        if curr_state[0]==3 and curr_state[1]==7:
            break
```

b) Q-learning:

```
step_8=[]
EP_8=[]
for i in range(episode):
    state={}
    t=0
    R=0
    start_row=3
    start_col=0
    state[t]=[]
    state[t].append(start_row)
    state[t].append(start_col)
    curr_state=state[t]
    c=[]
    for j in range (0,len(action)):
        c.append(j)
    pr=[]
    for k in range(len(action)):
        pr.append(1/len(action))
    while True:
        if np.random.random()<epsilon:
            A=action[(np.random.choice(c, p=pr))]
        else:
            A=epsilon_greedy_8(curr_state)
        next_s=next_state(curr_state,A)
        R+=reward_each_step
        q_max=[]
        for m in range(len(action)):
            q_max.append(Q[next_s[0],next_s[1],m])
        Q[curr_state[0],curr_state[1],action.index(A)]+=alpha*(R+gamma*m
ax(q_max)-Q[curr_state[0],curr_state[1],action.index(A)])
```

```
        curr_state=next_s
        t+=1
        state[t]=curr_state
        if curr_state[0]==3 and curr_state[1]==7:
            break
```

c) 随机性强度的风的实现：

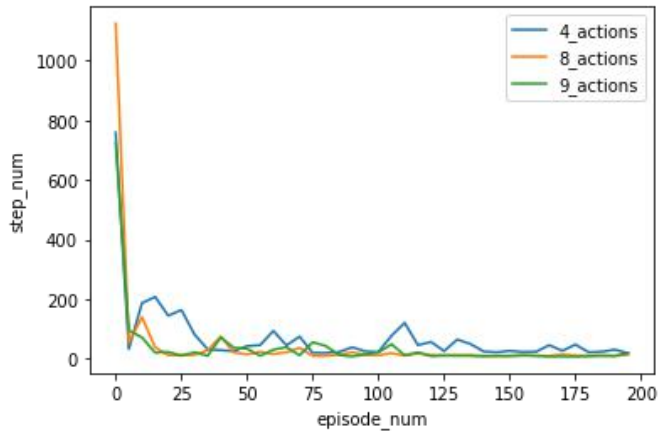
```
wind_power=[0,0,0,1,1,1,2,2,1,0]
rand=[-1,0,1]
prob = np.array([1/3,1/3,1/3])
index = np.random.choice(rand, p = prob)
next_s_row=s[0]-act[1]-(wind_power[position_col]+rand[index])
```

六、 实验结果：

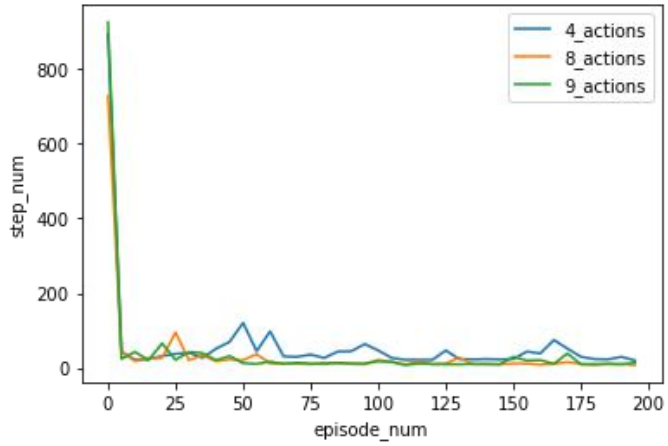
a) 实验结果：

i. 同轨策略下的时序差分控制（Sarsa）：

1. 确定强度的风：

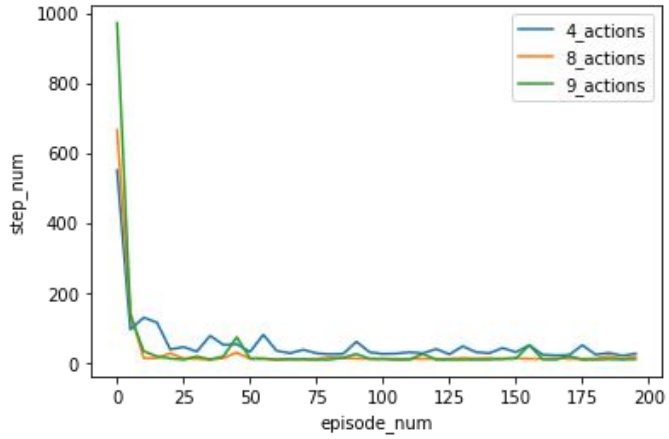


2. 加入随机强度的风：

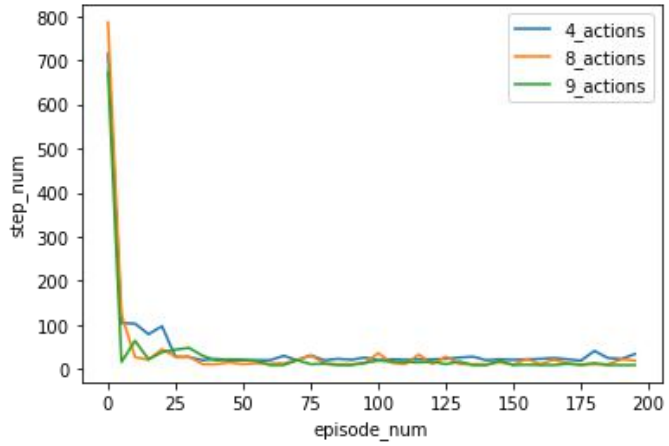


ii. 离轨策略下的时序差分控制（Q-learning）：

1. 确定强度的风：

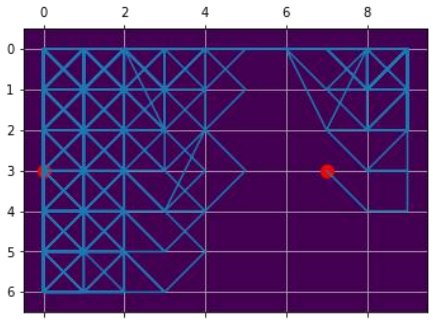


2. 加入随机强度的风：

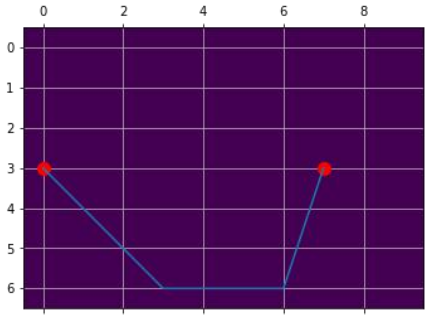


iii. 轨迹变化：

第一幕：



最后一幕：



b) 实验结果分析：

i. 由上图可知，随着时间（步数）的增加，Agent 能够更快地达到目标状态位置（所需幕数越少）；

ii. 对于包含不同动作数量的动作集合来说，动作集合中的动作数量越多，随着时间增加，Agent 能够更加迅速地达到目标状态（即上图中，总体上，蓝色曲线>橙色曲线>绿色曲线）

iii. 8 动作集合显示的性能要比较明显地优于 4 动作集合，而 8 动作集合与 9 动作集合显示出的性能较为相近；

iv. 随机性强度的风和确定性强度的风在三种动作集合上的性能表现均是 9 动作优于 8 动作，优于 4 动作；

v. Sarsa 和 Q-learning 所绘制的趋势图大致相同，两者都可以收敛到最优解上，但是运行速度上 Q-learning 要略快于 Sarsa（虽然两者都很快）；另外，Q-learning 曲线图在幕数较大的时候要比 Sarsa 的波动要小。上述两个区别的原因在“实验反思”中阐述。

七、 实验反思：

a) 当某个策略可能使 Agent 停留在同一个状态中，那么下一幕的任务就永远不会开始，此时 MC 方法就不适用了，因为 MC 必须等到一幕的末尾（结束）才能确定对价值函数的增量，而 TD 方法只需要等到下一时刻即可更新；

b) Sarsa（同轨策略）的学习目标中使用的是待学习的动作价值函数本身，由于它的计算需要知道下一时刻的动作，因此与生成数据的行动策略是相关的；Q-learning（离轨策略）计算待学习的动作价值函数采用了对最优动作价值函数的直接尽速作为学习目标，而与生成 Agent 决策序列轨迹的行动策略无关；

c) 上述实验结果中“Q-learning 曲线图在幕数较大的时候要比 Sarsa 的波动要小，且 Sarsa 收敛速度较慢”的原因，我认为是 Sarsa 的学习较为保守，在每幕的每一步都要进行ε-greedy 探索，而 Q-learning 则更倾向于利用经验的累积来学习到最优策略，因此 Sarsa 会收敛较慢一些，而且在幕数较大的时候仍然会进行ε-greedy 探索，导致时间达到较长的时候收敛曲线波动较大。