20337270_钟海财_实验10

中山大学计算机学院

本科生实验报告

(2021学年春季学期)

课程名称：Artificial Intelligence

教学班级	**20级软工+网安**	**专业（方向）**	**软件工程**
学号	**20337270**	**姓名**	**钟海财**

一、 实验题目

实验任务

 在给定迷宫环境中实现Q-learning和Sarsa算法。

二、 实验内容

1. 算法原理

1.1 时间差分方法

时间差分方法是一种估计值函数的方法，相较于蒙特卡洛使用完整序列进行更新，时间差分使用当前回报和下一时刻的价值进行估计，它直接从环境中采样观测数据进行迭代更新，时间差分方法学习的基本形式为：

$$V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$$

因上式只采样单步，所以利用上式进行更新的方法称为单步时间差分方法（one-step TD，TD(0)），其实时间差分不仅可以采样一步还可采样多步，得到n步时间差分算法的更新公式：

$$V(s) \leftarrow V(s) + \alpha[r + \gamma r' + \gamma^2 r^2 + \ldots + \gamma^n V(s^n, a^n) - V(s, a)]$$

其需要的观测数据形式为：

$$(s, a, r, s', a', r', \ldots\ldots, s^n, a^n)$$

1.2 Q-learning和Sara

Q−learning与Sarsa都是基于Qtable的算法， Q-learning属于离线学习策略，Sarsa属于在线学习策略。

1.3 Q-learning和Sara的主要区别

Q−learning与Sarsa的唯一区别在于Qtable的更新方式。

Q−learning更新Q值的方式：

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big].$$

Sarsa更新Q值的方式：

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big].$$

2. 伪代码

Q-learning伪代码：

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ϵ-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伪代码：

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-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

3. 关键代码展示（带注释）

Q-learning和Sarsa类里相同的函数

```
def __init__(self, actions, learning_rate=0.01, reward_decay=0.9, e_greedy=0.9):
    self.actions = actions  # a list
    self.lr = learning_rate
    self.gamma = reward_decay
    self.epsilon = e_greedy

    ''' build q table'''
    ############################
    # YOUR IMPLEMENTATION HERE #
    # 构建Q表
    self.q_table = pd.DataFrame(columns=self.actions, dtype=np.float64)
    ############################

def choose_action(self, observation):
    ''' choose action from q table '''
    ############################
    # YOUR IMPLEMENTATION HERE #
    self.check_state_exist(observation)
    # 动作选择
    if np.random.uniform() < self.epsilon:
        # 选择最佳动作
        state_action = self.q_table.loc[observation, :]
        # 有些动作可能有相同的值，在这些动作中随机选择
        action = np.random.choice(state_action[state_action ==
np.max(state_action)].index)
    else:
        # 随机选取一个动作
        action = np.random.choice(self.actions)
    return action
    ############################

def check_state_exist(self, state):
    ''' check state '''
    ############################
    # YOUR IMPLEMENTATION HERE #
    if state not in self.q_table.index:
        # 将新状态附加到Q表
        self.q_table = self.q_table.append(
            pd.Series(
                [0] * len(self.actions),
                index=self.q_table.columns,
                name=state,
            )
        )
    ############################
```

Q-learning的learn函数

```
def learn(self, s, a, r, s_):
    ''' update q table '''
    ############################
    # YOUR IMPLEMENTATION HERE #
    self.check_state_exist(s_)
    q_predict = self.q_table.loc[s, a]
```

```
        if s_ != 'terminal':
            # Q-learning和Sarsa的唯一不同处：计算Q值的方法
            q_target = r + self.gamma * self.q_table.loc[s_, :].max()
        # next state is not terminal
        else:
            q_target = r  # next state is terminal
        # 更新Q值
        self.q_table.loc[s, a] += self.lr * (q_target - q_predict)
        ############################
```

Sarsa的learn函数

```
    def learn(self, s, a, r, s_):
        ''' update q table '''
        ############################
        # YOUR IMPLEMENTATION HERE #
        self.check_state_exist(s_)
        q_predict = self.q_table.loc[s, a]
        if s_ != 'terminal':
            # Q-learning和Sarsa的唯一不同处：计算Q值的方法
            a_ = self.choose_action(s_)
            q_target = r + self.gamma * self.q_table.loc[s_, a_]
        # next state is not terminal
        else:
            q_target = r  # next state is terminal
        # 更新Q值
        self.q_table.loc[s, a] += self.lr * (q_target - q_predict)
        ############################
```

Update函数

```
def update():
    for episode in range(episodes):
        # initial observation
        observation = env.reset()
        step = 0
        while True:
            # 记录步数
            step += 1
            # fresh env
            '''Renders policy once on environment. Watch your agent play!'''
            env.render()
            # RL根据观察选择动作
            action = RL.choose_action(str(observation))
            # RL采取行动并获得下一次观察和奖励
            observation_, reward, done = env.step(action)
            # 更新Q值
            RL.learn(str(observation), action, reward, str(observation_))
            # swap observation
            observation = observation_
            # break while loop when end of this episode
            if done:
                # 如果找到了目标
                if reward == 1:
                    print("episode = ", episode, ",  steps = ", step, "\n")
                    steps.append(step)
```

```
                # 否则进入陷阱
                else:
                    steps.append(steps[-1])
                break
    # end of game
    print('game over')
    env.destroy()
```

选择Q-learning/Sarsa方法，并将结果画图

```
if __name__ == "__main__":
    env = Maze()

    '''
    build RL Class
    RL = QLearning(actions=list(range(env.n_actions)))
    RL = Sarsa(actions=list(range(env.n_actions)))
    '''
    ############################
    # YOUR IMPLEMENTATION HERE #
    choice = input("请选择使用Q-learning/Sarsa: 1/2\n")
    episodes = 100
    steps = [100]
    if choice == "1":
        print("使用Q-learning")
        RL = QLearning(actions=list(range(env.n_actions)))
    else:
        print("使用Sarsa")
        RL = Sarsa(actions=list(range(env.n_actions)))
    ############################
    env.after(100, update)
    env.mainloop()

    # 输出最终的q表
    print(RL.q_table)
    # 画图
    plt.plot(np.linspace(0, episodes, len(steps)), steps)
    plt.xlabel("迭代次数episode", fontsize=18)
    plt.ylabel("步数", rotation=0, fontsize=18)
    plt.title('达到目标的步数和训练Epoch数', fontsize=18)
    plt.show()
```

4. 创新点&优化（如果有）

无

三、 实验结果及分析

1. 实验结果展示示例（可图可表可文字，尽量可视化）

1.视频

见压缩包里的视频 lab10.mp4。

2.最终的q值表

Q-learing

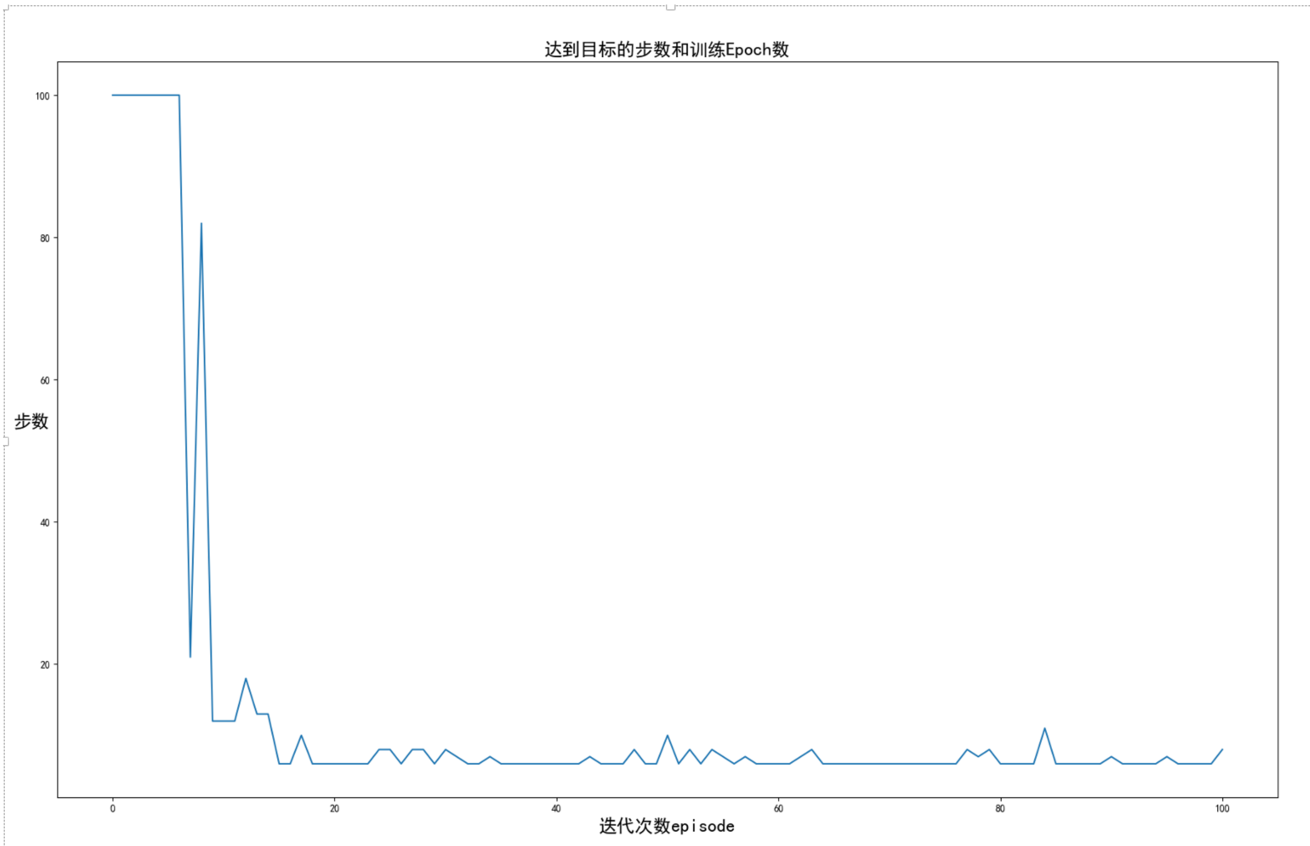
```
game over
                                  0  ...             3
[5.0, 5.0, 35.0, 35.0]         0.000001  ...  1.104538e-09
[45.0, 5.0, 75.0, 35.0]        0.000011  ...  1.303800e-06
[45.0, 45.0, 75.0, 75.0]       0.000002  ...  3.676208e-13
[5.0, 45.0, 35.0, 75.0]        0.000000  ...  0.000000e+00
terminal                       0.000000  ...  0.000000e+00
[85.0, 5.0, 115.0, 35.0]       0.000000  ...  8.533842e-06
[125.0, 5.0, 155.0, 35.0]      0.000009  ...  2.811113e-06
[125.0, 45.0, 155.0, 75.0]     0.000015  ... -5.851985e-02
[125.0, 85.0, 155.0, 115.0]    0.002168  ...  5.569520e-01
[5.0, 85.0, 35.0, 115.0]       0.000000  ...  0.000000e+00
[5.0, 125.0, 35.0, 155.0]      0.000000  ...  0.000000e+00
[45.0, 125.0, 75.0, 155.0]    -0.010000  ...  0.000000e+00
[85.0, 125.0, 115.0, 155.0]    0.019900  ...  0.000000e+00
[125.0, 125.0, 155.0, 155.0]   0.000000  ...  9.000000e-05

[14 rows x 4 columns]
```

Sarsa

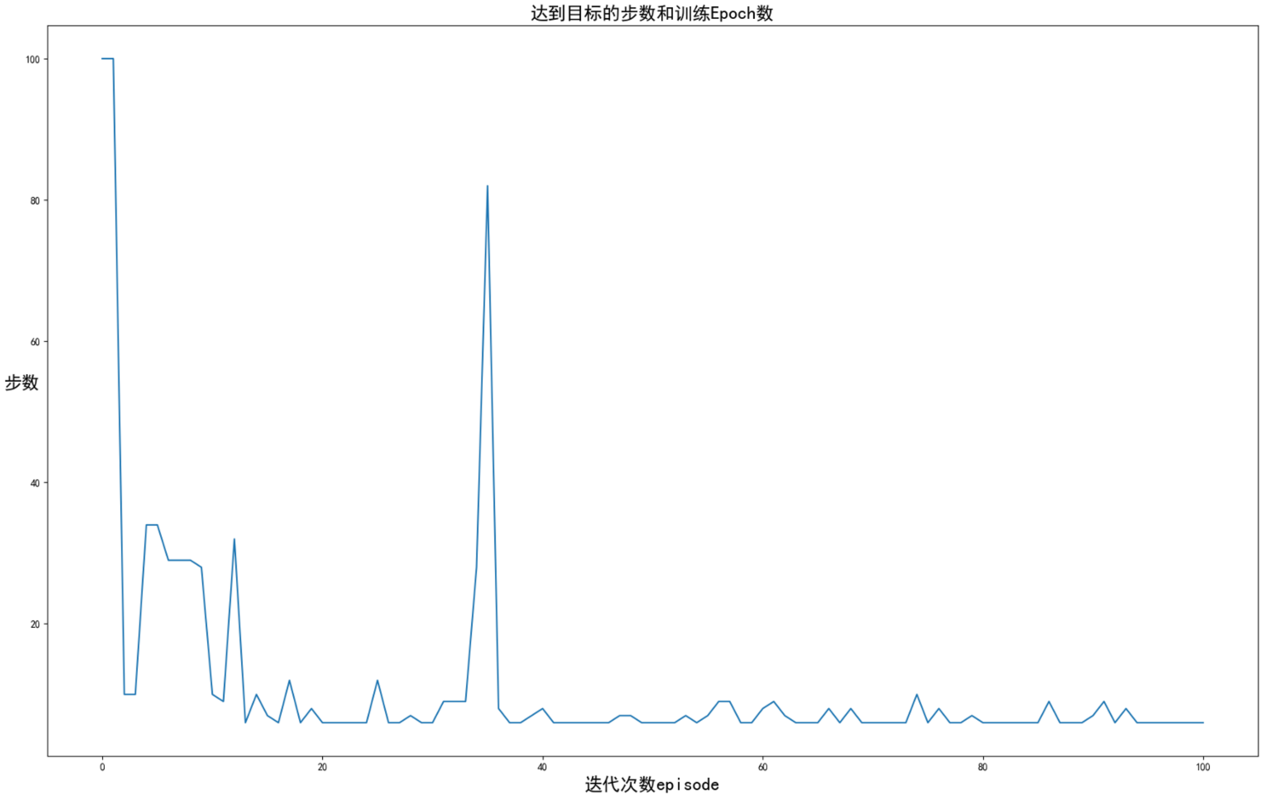
```
game over
                                           0  ...             3
[5.0, 5.0, 35.0, 35.0]        -5.918728e-09  ...  9.221336e-08
[5.0, 45.0, 35.0, 75.0]        1.068387e-07  ...  7.112503e-12
[5.0, 85.0, 35.0, 115.0]       2.489915e-10  ...  0.000000e+00
[45.0, 45.0, 75.0, 75.0]       8.331300e-12  ... -7.893312e-07
terminal                       0.000000e+00  ...  0.000000e+00
[45.0, 5.0, 75.0, 35.0]        2.769664e-06  ...  2.003608e-07
[85.0, 5.0, 115.0, 35.0]      -8.738422e-05  ...  2.171787e-07
[125.0, 5.0, 155.0, 35.0]      2.468705e-05  ... -8.026292e-05
[125.0, 45.0, 155.0, 75.0]     9.172566e-05  ... -1.000000e-02
[125.0, 85.0, 155.0, 115.0]    6.184183e-04  ...  5.101097e-01
[5.0, 125.0, 35.0, 155.0]      6.050845e-09  ...  3.091381e-07
[45.0, 125.0, 75.0, 155.0]     0.000000e+00  ...  9.372754e-06
[85.0, 125.0, 115.0, 155.0]    1.983694e-01  ...  0.000000e+00
[125.0, 125.0, 155.0, 155.0]   1.213406e-03  ...  0.000000e+00

[14 rows x 4 columns]
```

3.达到目标的步数和迭代次数的曲线图

Q-learing



Sarsa



2. 评测指标展示及分析（机器学习实验必须有此项，其它可分析运行时间等）

评测指标：

达到目标的步数

分析：

无论是Q-learning还是Sarsa，都能较快地(迭代次数20次以内)找到到达目标地最优路径（步数最少为6步）。

且在找到最优路径后，两种方法都能稳定在最优路径附近，但Sarsa方法有时候会突然走一条莫名的较差的路径，推测可能是在根据观察选择动作时受随机因子（e_greedy=0.9，90%的几率选择最优动作）的影响多次没有选择最优动作，但Q-learning方法不会出现这种情况。这是因为Q-learning计算q值时是直接选择最优动作（没有随机因子的干扰），而Sarsa计算q值时使用了choose_action()函数受到了随机因子e_greedy的影响。

四、 思考题

本次实验无思考题。

五、 参考资料

1. 实验文档：16_q-learning.pdf
2. 课本：《人工智能》(第三版) 清华大学出版社
3. 参考网址：

   Q-learning原理及其实现方法北木.的博客-*CSDN*博客q-learning

   Q-learning 算法更新 - 强化学习 Reinforcement Learning | 莫烦Python (yulizi123.github.io)

   Sarsa 算法更新 - 强化学习 Reinforcement Learning | 莫烦Python (yulizi123.github.io)