人工智能基础 第三次大作业

来昆 自 72 2017011607

一、 题目要求

Gym 是 OpenAI 发布的用于研究和比较学习算法的工具包

必做：

1. 使用强化学习算法，解决 MountainCar-v0.
2. 使用强化学习算法，解决 MountainCarContinous-v0.

选做：

1. 使用多种强化学习算法解决上述问题。

二、 原理分析

1. 题目分析

   GitHub 中两个游戏的基本信息如下：

   ① MountainCar-v0

   目标：登顶山坡（位于 0.5 高度的位置(position)）

   环境解析：

   Observation：

   Type: Box(2)

Num	Observation	Min	Max
0	position	-1.2	0.6
1	velocity	-0.07	0.07

   Actions：

   Type: Discrete(3)

Num	Action
0	push left
1	no push
2	push right

   Reward：

   每步-1，直到达到 0.5position 的位置。

   Starting State：

   Position：-0.6~-0.4 的随机范围；

Velocity：0

终止条件：

登顶或进行 200 步操作。

② MountainCarContinuous-v0

目标：

环境解析：

Observation：

Type: Box(2)

Num	Observation	Min	Max
0	Car Position	-1.2	0.6
1	Car Velocity	-0.07	0.07

Action

负值向左，正值向右。

Type: Box(1)

Num	Action
0	Push car to the left (negative value) or to the right (positive value)

Reward：

到达山顶：100 减去所有 Action 的平方和

Starting State：

Position：Random-0.6~-0.4

Velocity：0

终止条件：

Position:0.5

2. 强化学习算法分析

上课老师提到 Q-Learning 是现在实现的主流方案，因此必做任务中使用了 Q-Learning 算法，课件中的截图如下：

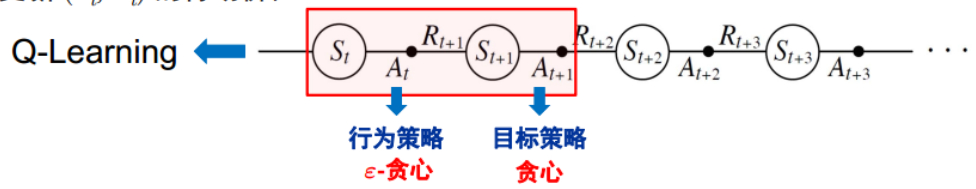
Q-Learning 属于 Off-Policy 的策略，行为策略和目标策略使用不同的算法，行为策略选择使用ε-贪心算法，目标策略使用贪心算法，即：谨慎行动，大胆探索。

Q-Learning

- 行动价值递推 $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_{a \in \mathbf{A}} Q(S_{t+1}, a) - Q(S_t, A_t))$
- 行为策略选择 ε-贪心算法 $\pi(a \mid s) = \begin{cases} 1 - \varepsilon + \varepsilon / m & \text{if } a = \arg\max_{a \in \mathbf{A}} Q(s, a) \\ \varepsilon / m & \text{otherwise} \end{cases}$
- 目标策略选择 贪心算法 $\pi'(S_{t+1}) = \arg\max_{a \in \mathbf{A}} Q(S_{t+1}, a)$

$$Q(S_{t+1}, A_{t+1}) = Q(S_{t+1}, \arg\max_{a \in \mathbf{A}} Q(S_{t+1}, a)) = \max_{a \in \mathbf{A}} Q(S_{t+1}, a)$$

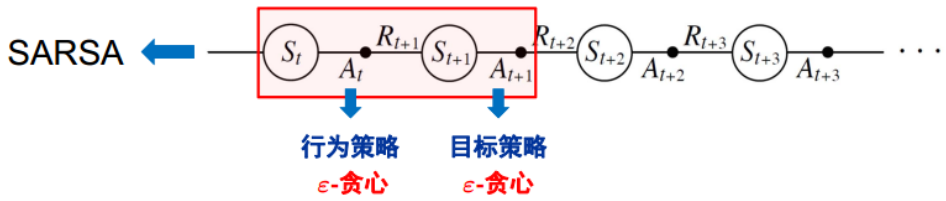
- 根据行为策略从 S_t 产生 A_t, 获得回报 R_{t+1} 和下一状态 S_{t+1}
- 根据目标策略从 S_{t+1} 产生 A_{t+1}, 计算 (S_{t+1}, A_{t+1}) 的行动价值
- 更新 (S_t, A_t) 的行动价值



选做中使用了 SARSA 和期望 SARSA，SARSA 算法与 Q-Learning 算法的不同就在于目标策略的选择， SARSA 属于 on-policy 的策略，目标策略和行为策略相同，都使用ε-贪心算法，表现在具体地推过程中为行动价值递推公式的不同；期望 SARSA 的目标策略设为行动的期望。

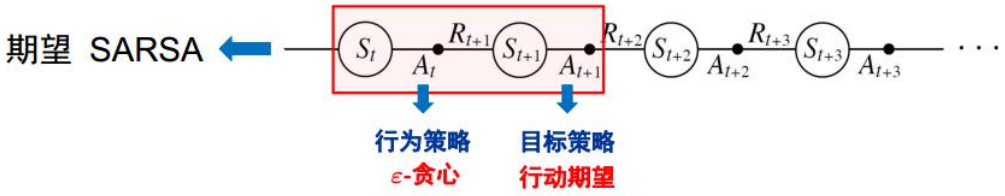
SARSA

- 行动价值递推 $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$
- 行为策略选择 ε-贪心算法 $\pi(a \mid s) = \begin{cases} 1 - \varepsilon + \varepsilon / m & \text{if } a = \arg\max_{a \in \mathbf{A}} Q(s, a) \\ \varepsilon / m & \text{otherwise} \end{cases}$
- 目标策略选择 ε-贪心算法
- 根据行为策略从 S_t 产生 A_t, 获得回报 R_{t+1} 和下一状态 S_{t+1}
- 根据目标策略从 S_{t+1} 产生 A_{t+1}, 计算 (S_{t+1}, A_{t+1}) 的行动价值
- 更新 (S_t, A_t) 的行动价值



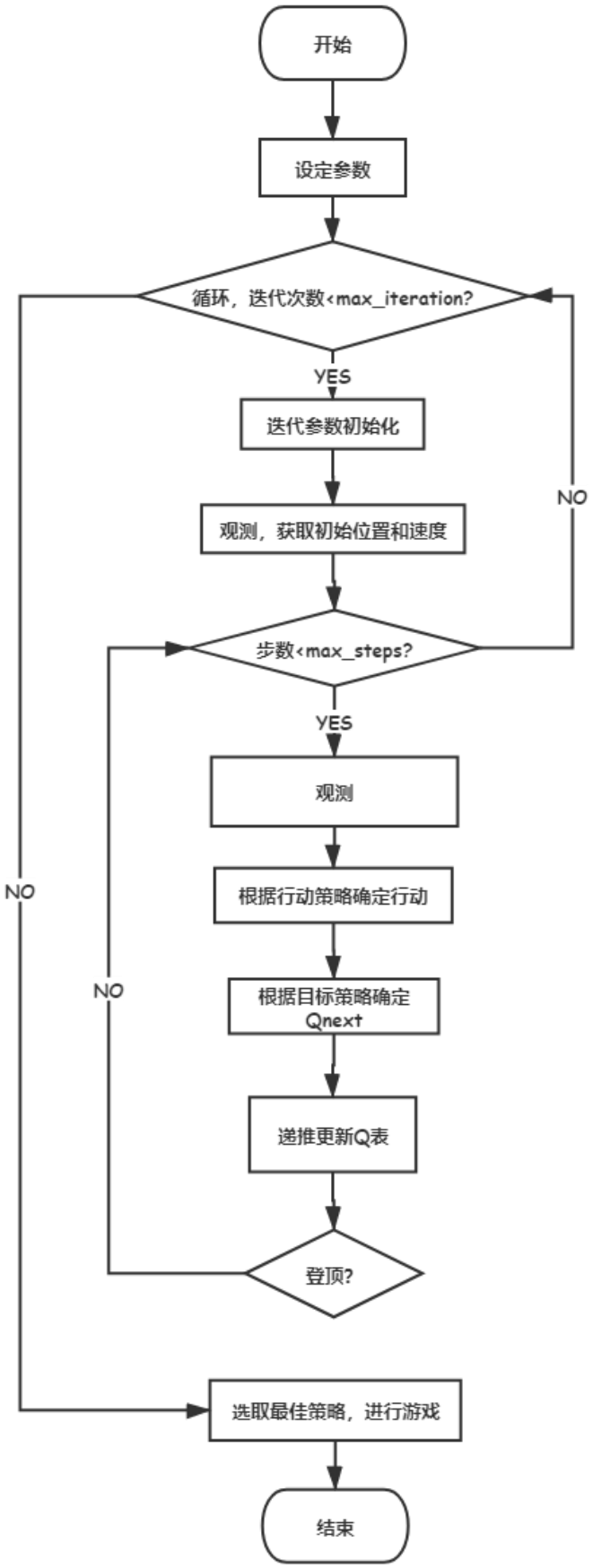
期望 SARSA　　Expected SARSA

- 行动价值递推　$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \sum_{a \in \mathbf{A}} \pi(a \mid S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t))$
- 行为策略选择　ε-贪心算法　$\pi(a \mid s) = \begin{cases} 1 - \varepsilon + \varepsilon / m & \text{if } a = \arg\max_{a \in \mathbf{A}} Q(s, a) \\ \varepsilon / m & \text{otherwise} \end{cases}$
- 目标策略选择　行动的期望
- 根据行为策略从 S_t 产生 A_t，获得回报 R_{t+1} 和下一状态 S_{t+1}
- 根据目标策略从 S_{t+1} 产生 A_{t+1}，计算 (S_{t+1}, A_{t+1}) 的行动价值
- 更新 (S_t, A_t) 的行动价值



三、 实现及流程

不同游戏的整体流程基本相同，分为训练和观测结果两部分，流程如下:


开始
设定参数
循环，迭代次数<max_iteration?
YES
NO
迭代参数初始化
观测，获取初始位置和速度
步数<max_steps?
YES
NO
观测
根据行动策略确定行动
根据目标策略确定Qnext
递推更新Q表
登顶?
NO
选取最佳策略，进行游戏
结束

具体实现

1. 参数初始化及衰减

初始化：状态数，迭代上线，初始学习率，学习率下限，初始ε，γ，α，建立 Q 表

参数衰减：随着迭代次数上升，ε逐渐减小，α逐渐减小，初始时大胆探索，迅速学习，后期逐渐谨慎，收敛到最优策略。

```
#   参数初始化
num_states = 40
max_iteration = 5000
initial_learning_rate = 1.0
min_learning_rate = 0.005
max_steps = 200
epsilon_initial = 1.0
gamma = 0.99
num_actions = 20
Q_Table = np.zeros((num_states, num_states, num_actions))
```

```
epsilon = epsilon_initial / (np.sqrt(episode))
```

```
alpha = max(min_learning_rate, initial_learning_rate * (0.85 ** (episode // 250)))#
```

2. 状态离散化

速度和位置是离散的，通过离散化将 Observation 观测到的结果分配到有限的离散状态中，便于学习和策略迭代。

```
def Observation_2_State(observation, environment, num_states):

    env_low = environment.observation_space.low
        #   [-1.2,-0.07]
    env_high = environment.observation_space.high
        #   [0.6,0.07]

    #   根据需要的状态数目确定的最小分片单位
    env_piece = (env_high - env_low) / num_states

    position = int((observation[0] - env_low[0]) / env_piece[0])
    velocity = int((observation[1] - env_low[1]) / env_piece[1])
    return position, velocity
```

在 MountainCarContinuous 中，

3. Q 表递推

不同的算法使用不同的策略迭代。具体实现如下：

Q-Learning

```
Q_Table[p][v][action_seq] = Q_Table[p][v][action_seq] + alpha * (
    reward + gamma * np.max(Q_Table[p_][v_]) - Q_Table[p][v][action_seq])
```

SARSA

```
p_, v_ = Observation_2_State(observation, env, num_states)
if np.random.uniform(0,1) < epsilon:
    #  random choice
    Q_next = np.random.choice((Q_Table[p_][v_]))
else:
    Q_next = np.max(Q_Table[p_][v_])

Q_Table[p][v][action_seq] = Q_Table[p][v][action_seq] + alpha * (
    reward + gamma * Q_next - Q_Table[p][v][action_seq])
```

期望 SARSA

```
for m in range(num_actions):
    if Q_Table[p_][v_][m]==np.max(Q_Table[p_][v_]):
        Q_next += (1 - epsilon + (epsilon / num_actions)) * Q_Table[p_][v_][m]
    else:
        Q_next += (epsilon / num_actions) * Q_Table[p_][v_][m]

Q_Table[p][v][action_seq] = Q_Table[p][v][action_seq] + alpha * (
    reward + gamma * Q_next - Q_Table[p][v][action_seq])
```

四、 实验结果

爬山视频附在附件中，根据不同的策略、算法、游戏调整了不同的参数（ε，α，γ衰减速率，迭代步数等），都完成了游戏。不同的算法下最终的效果不同，收敛的步数不同，最终实现爬山的策略也不同。其中 Q-Learning 的最终策略并不如其他两种，但收敛较快，迭代速度也较快，可以通过改善参数提升效果，SARSA 和 E-SARSA 的迭代速度较慢，但最终实现的视频效果看起来更快更流畅，这也体现了不同算法的不同特点和各自的优缺点。

五、 总结与反思

1. 遇到的问题

   由于之前没有使用过 gym 平台和 openAI 的相关内容，而 gym 平台上给的信息也十分有限，刚开始感到无从下手，后来阅读了官网提供的 github 文件，对照着上课的课件和参考资料，找了几份代码阅读，才理解了 gym 环境中的相关概念和包的使用方法。

   在实现 MountainCarContinuous 时，Action 和 Reward 都十分模糊，而且非常容易为了防止 Reward 减少而静止在谷底，经过尝试，我离散化了 Action，并在每一个 step 后 Reward 减去固定值，这个值的大小也会影响最终的收敛效果，经过不断调节，得到了较好的结果。

2. 总结与反思

   本次大作业是这学期的最后一次大作业，通过 gym 平台的小游戏学习和探索，对上课学到的强化学习的相关概念有了更深刻的理解，也有了更好的掌握，虽然

期末考试结果十分不理想，但这个学期还是学到了很多的东西，也受到了老师和助教的熏陶和影响，受益匪浅。感谢助教和老师一学期的帮助指导和耐心答疑，辛苦了！