电子科技大学

计算机科学与工程学院

标 准 实 验 报 告

（实验）课程名称 人工智能

电子科技大学教务处制表

电 子 科 技 大 学

电 子 科 技 大 学

实 验 报 告

学生姓名：李天　　学 号：2020080904021

指导教师：段立新 张彦如 顾实

实验地点： 主楼 A2-413-1　　实验时间：2022.12.3

一、实验室名称：计算机学院实验中心

二、实验项目名称：MDP 实验

三、实验学时：5 学时

四、实验原理：

（1）迷宫游戏说明

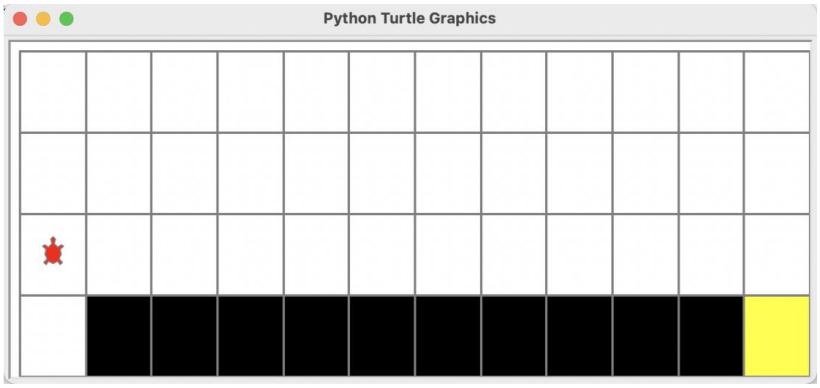

图 1 迷宫游戏示例

如上图所示，本实验所研究的迷宫由一个二维表格构成。其中白色区域是可行走区域，黑色区域是陷阱区域，黄色区域是终点。规则如下：

1） 智能体从左下角出发，到达黄色区域即游戏成功。

2） 智能体每次可选择上、下、左、右四种移动动作，每次动作得到-1 奖励。

3） 智能体不能移动出网络，如果下一步的动作命令会让智能体移动出边界，那么这一步将不会执行，即智能体原地不动，得到-1 奖励。

4） 智能体移动到黑色区域，得到-100 奖励。

5） 智能体移动到黄色区域，该回合结束。

由图可知，最优的路线需要 13 步，因此最后智能体获得的奖励指在-13 左右为最佳结果。

（2）Q-Learning 算法

Q-Learning 是一种记录行为值（Q value）的方法，每种行为在一定的状态都会有一个值 Q(s,a)，就是说行为 a 在 s 状态的值是 Q（s,a）。对于迷宫游戏，s 就是当前 agent 所在的地点了。每一步，智能体可以选择四种动作，所以动作 a 有四种可能性。

在本实验里，已经提供了强化学习基本的训练接口，只需要实现 Q 表格的强化学习方法即可。算法框架如下：

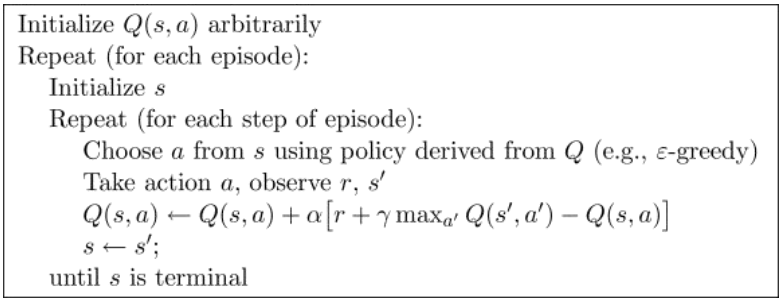

图 2 强化学习算法框架

可以看到，算法的核心部分是更新 Q 表格。训练目标是在评价阶段，智能体的平均奖励达到-13。做出可视化结果如下图。

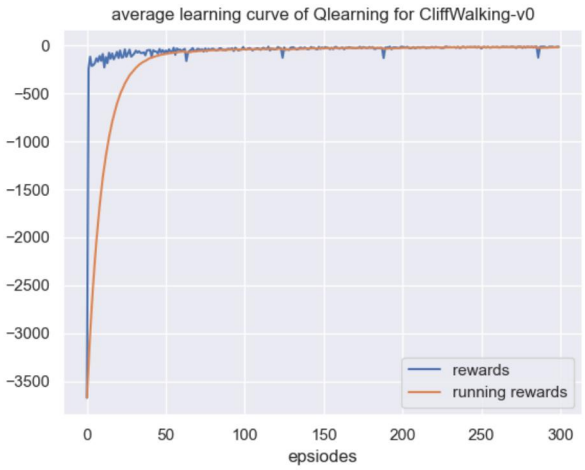

图 3 Q-Learning 训练奖励曲线

五、实验目的：

实验使用 Q 表格方法解决迷宫寻路问题，理解强化学习算法原理。

六、实验内容：

阅读代码，补全代码中缺失的部分，完成迷宫实验。

七、实验器材（设备、元器件）：

PC 微机一台

八、实验步骤：

1.补全./agent.py 中 choose_action()方法

choose_action()方法用于训练过程中针对一个 state 选取 action，因此按照 Q-Learning 算法来看，应该选取该 state 对应的四个 Q(s,a)中最大的 Q 所采取的 action，即如下公式所示：

$$\mathbf{action} = \underset{\mathbf{a} \in \mathbf{a_i}}{\mathbf{argmax}}\,(\mathbf{Q(state, a)})$$

同时为了提升该算法的性能，我引入了**贪心策略（ε -greedy）**用于进一步优化 choose_action()方法。其表示在智能体做决策时，有一很小的正数 ε (<1)的概率**随机选择未知的一个动作**，剩下 1- ε 的概率选择已有动过中动作价值最大的

动作。这样做的好处是，如果我们每次都选择最好的动作，那会有很多动作没有被选择， ε -greedy 则可以解决这个问题。进一步思考会发现，所有的动作被选择的概率都满足 $\pi(a|s) \geq \varepsilon/|A|$，这就保证了每个（状态-动作）二元组都会有一定概率被访问到。

同时，为了使后期训练时尽可能选择最优路径，在这里我设置 ε **随着迭代次数而逐渐衰减（最开始为 0.95，最终收敛至 0.01）**，使得最后 Q-Learning 可以达到收敛。由此，可以补全 choose_action()方法，如图 4 所示。

```python
def choose_action(self, state):
    self.sample_count += 1
    # epsilon的更新
    self.epsilon = self.epsilon_end + (self.epsilon_start - self.epsilon_end) * math.exp(-1. * self.sample_count / self.epsilon_decay)
    if np.random.uniform(0, 1) > self.epsilon:
        # 选取对应Q最大的动作
        action = np.argmax(self.Q_table[int(state)])
    else:
        # 随机选取动作
        action = np.random.choice(self.action_dim)
    return action
```

图 4 choose_action 方法

2. 补全./agent.py 中 predict()方法

predict()方法用于评估模型时使用，即在模型训练好后，根据 Q 表格选取每个 state 下最优的 action。因此，实现该方法只需要取 Q 表格中对应当前 state 的最大值所采取的 action，predict()方法实现如图 5 所示。

```python
def predict(self, state):
    action = np.argmax(self.Q_table[int(state)])
    return action
```

图 5 predict 方法

3.补全./agent.py 中 update()方法

update()方法用于训练模型时更新 Q 表格。因此，实现该方法需要实现如下公式：

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

但需要注意，如果当前状态已经达到终点，那么其 Q 值更新时对应的未来 Q 值应该就等于当前的 reward。由此，update()方法实现如图 6 所示。

```python
def update(self, state, action, reward, next_state, done):
    # 计算Q估计
    Q_predict = self.Q_table[int(state)][action]
    # 计算Q现实
    if done:
        # 如果回合结束，则直接等于当前奖励
        Q_target = reward
    else:
        # 如果回合每结束，则按照
        Q_target = reward + self.gamma * np.max(self.Q_table[int(next_state)])
    # 根据Q估计和Q现实，差分地更新Q表格
    self.Q_table[int(state)][action] += self.lr * (Q_target - Q_predict)
```

图 6 update 方法

九、实验数据及结果分析：

（1）训练过程输出展示

训练过程中的输出为：

```
Episode:381/400: reward:-13.0
Episode:382/400: reward:-13.0
Episode:383/400: reward:-13.0
Episode:384/400: reward:-13.0
Episode:385/400: reward:-13.0
Episode:386/400: reward:-13.0
Episode:387/400: reward:-13.0
Episode:388/400: reward:-13.0
Episode:389/400: reward:-13.0
Episode:390/400: reward:-13.0
Episode:391/400: reward:-13.0
Episode:392/400: reward:-17.0
Episode:393/400: reward:-15.0
Episode:394/400: reward:-13.0
Episode:395/400: reward:-13.0
Episode:396/400: reward:-13.0
Episode:397/400: reward:-13.0
Episode:398/400: reward:-13.0
Episode:399/400: reward:-13.0
Episode:400/400: reward:-13.0
Complete training!
results saved!
```

本次实验我将 QlearningConfig 中的 train_eps 参数增加到了 400，可以看到（如图 7），到了训练后期，实验结果基本**趋于收敛**（有少数非-13 的结果是因为 **ε -greedy** 机制的存在）。

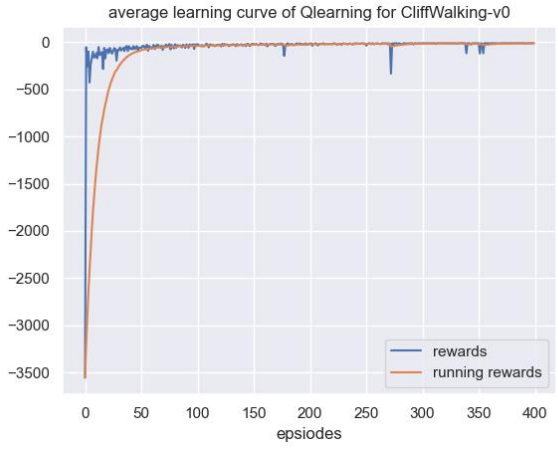

图 7 training_reward 曲线

（2）评估模型

评估模型过程输出为：

```
Start to eval !
Episode:1/30, reward:-13.0
Episode:2/30, reward:-13.0
Episode:3/30, reward:-13.0
Episode:4/30, reward:-13.0
Episode:5/30, reward:-13.0
Episode:6/30, reward:-13.0
Episode:7/30, reward:-13.0
Episode:8/30, reward:-13.0
Episode:9/30, reward:-13.0
Episode:10/30, reward:-13.0
Episode:11/30, reward:-13.0
Episode:12/30, reward:-13.0
Episode:13/30, reward:-13.0
Episode:14/30, reward:-13.0
Episode:15/30, reward:-13.0
Episode:16/30, reward:-13.0
Episode:17/30, reward:-13.0
Episode:18/30, reward:-13.0
Episode:19/30, reward:-13.0
Episode:20/30, reward:-13.0
Episode:21/30, reward:-13.0
Episode:22/30, reward:-13.0
Episode:23/30, reward:-13.0
Episode:24/30, reward:-13.0
Episode:25/30, reward:-13.0
Episode:26/30, reward:-13.0
Episode:27/30, reward:-13.0
Episode:28/30, reward:-13.0
Episode:29/30, reward:-13.0
Episode:30/30, reward:-13.0
Complete evaling!
results saved!

Process finished with exit code 0
```

可以发现，评估结果较为理想，总共 30 次 episode 都选择了最短的路径，**智能体的奖励每次都是-13**，成功达到了训练目标（如图 8）。

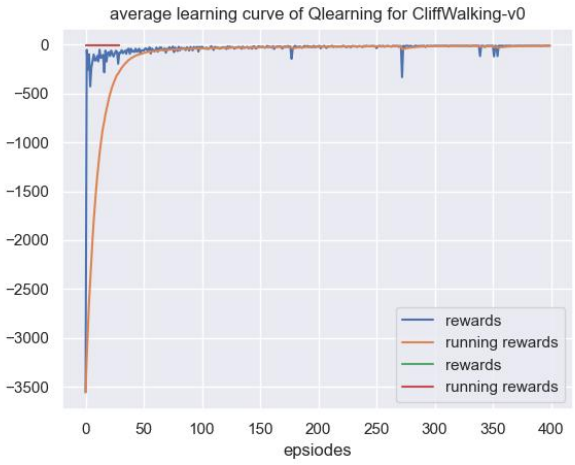

图 8 eval_reward 曲线

十、实验结论：

本次实验圆满完成，成功实现了所有实验要求，成功达到了训练目标。

十一、总结及心得体会：

本次实验我熟悉和掌握了 Q-table 算法，理解了算法原理，并利用该算法成功实现训练悬崖寻路问题，最终在评估过程中能达到预期的训练目标，使我对 Q-Learning 算法有了更深入的理解和思考，对该算法的应用有了一定的了解。

十二、对本实验过程及方法、手段的改进建议：

在实际实验过程中，对 gym 库的使用不太熟悉，花了很多的时间去了解这个库，以及训练的环境，我认为之后可以先简要的讲解一下整个项目。

十三、代码附件

实验三、补全 agent.py 文件

```
import numpy as np
import math

class QLearning(object):
    def __init__(self, state_dim, action_dim, cfg):
        self.action_dim = action_dim  # dimension of acgtion
        self.lr = cfg.lr  # learning rate
        self.gamma = cfg.gamma # 衰减系数
        self.epsilon = 0.1
        self.epsilon_start = 0.95
        self.epsilon_end = 0.01
        self.epsilon_decay = 300
        self.sample_count = 0
        self.Q_table = np.zeros((state_dim, action_dim))  # Q 表格

    def choose_action(self, state):
        self.sample_count += 1
        # epsilon 的更新
        self.epsilon = self.epsilon_end + (self.epsilon_start - self.epsilon_end) * math.exp(-1. * self.sample_count / self.epsilon_decay)
        if np.random.uniform(0, 1) > self.epsilon:
            # 选取对应 Q 最大的动作
            action = np.argmax(self.Q_table[int(state)])
```

```
            else:
                # 随机选取动作
                action = np.random.choice(self.action_dim)
            return action

        def predict(self, state):
            action = np.argmax(self.Q_table[int(state)])
            return action

        def update(self, state, action, reward, next_state, done):
            # 计算Q估计
            Q_predict = self.Q_table[int(state)][action]
            # 计算Q现实
            if done:
                # 如果回合结束，则直接等于当前奖励
                Q_target = reward
            else:
                # 如果回合每结束，则按照
                Q_target = reward + self.gamma * np.max(self.Q_table[int(next_state)
    ])
            # 根据Q估计和Q现实，差分地更新Q表格
            self.Q_table[int(state)][action] += self.lr * (Q_target - Q_predict)

        def save(self, path):
            np.save(path + "Q_table.npy", self.Q_table)

        def load(self, path):
            self.Q_table = np.load(path + "Q_table.npy")
```

报告评分：

指导教师签字：