西安电子科技大学

专业基础实践 课程实践报告

实践名称 基于 Q-Table 的强化学习路径规划实验

人工智能 学院 _____ 班

姓名_____ 学号_____

同作者 无

实践日期 2025 年 ____月—____月

成　绩 （教师组打分）

实践报告内容基本要求及参考格式

一、实践题目功能描述；

二、问题分析及算法或方案阐述（包括理论基础简要介绍）；

三、实践结果与分析（给出结果图或表，并讨论分析）；

四、程序源代码（添加注释）；

五、实践过程或算法总结（取得的结果或效果，下一步工作展望）；

六、心得体会；（如解决了什么问题，还存在什么问题，哪些方面得到了提高或受到了启发；在设计该算法或解决方案时，遇到的最大困难是什么？是如何解决的？）

七、小组每个人的详细分工及自我评价（说明每人承担的具体任务和完成情况）

姓名	学号	小组分工与贡献	自我评价（优秀、良好，中等、及格、不及格）

实践报告内容

一、实践题目功能描述；

1.实验题目

基于 Q-Table 的强化学习路径规划实验

2.功能描述

本实验通过 Q-learning 算法训练个体（Agent）在给定的网格地图中寻找从起点（S）到终点（R）的最优路径，同时避开陷阱（T）和墙壁（W）等障碍物。实验内容包括：

(1)Q-Table 构建：通过强化学习动态更新状态-动作价值表。

(2)路径规划：个体基于 Q-Table 选择最优动作序列。

(3)可视化展示：实时显示训练过程、Q-Table 更新及最终路径。

二、问题分析及算法或方案阐述（包括理论基础简要介绍）；

1. 问题分析

为了简化问题建模，我们采用一个 7×7 的网格地图作为环境表示。该地图中包含以下关键元素：起点（标记为"S"）、终点（标记为"R"）、障碍物（包括陷阱"T"和墙壁"W"）。这种离散化的网格表示方法既能保持问题的本质特征，又便于算法实现和结果分析。

核心挑战：

个体需要在全局环境信息未知的情况下，通过试错学习逐步优化决策策略。这一过程的关键在于平衡探索与利用：

探索：尝试新动作以发现潜在更优路径，避免陷入局部最优；

利用：基于当前知识选择已知最优动作，确保策略的稳定性。

2. Q-learning 算法流程

（1）Q-Table 初始化

状态+动作空间：7×7 网格共 49 个状态 +4 个离散动作（右、下、左、上）

初始 Q 值统一设为 0

（2）动作选择策略（ε-greedy）

探索（Exploration）：以概率 ε 随机选择动作，以发现潜在更优策略

利用（Exploitation）：以概率 1-ε 选择当前 Q 值最高的动作，最大化即时收益

（3）Q-Table 更新机制

执行动作后，环境返回奖励 r 和新状态 s'

基于 Bellman 方程 更新 Q 值：

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

其中，α 为学习率，γ 为折扣因子

（4）终止条件

成功到达终点（R）→ 终止并记录成功

落入陷阱（T）→ 终止并记录失败

该框架使个体在未知环境中逐步优化策略，同时平衡探索与利用。

3. 关键参数

参数	值	说明
学习率 α	0.1	控制 Q 值更新幅度
折扣因子 γ	0.9	衡量未来奖励的重要性
探索率 ε	0.1→0.01	随训练衰减，平衡探索与利用

三、实践结果与分析（给出结果图或表，并讨论分析）；

1. 训练过程

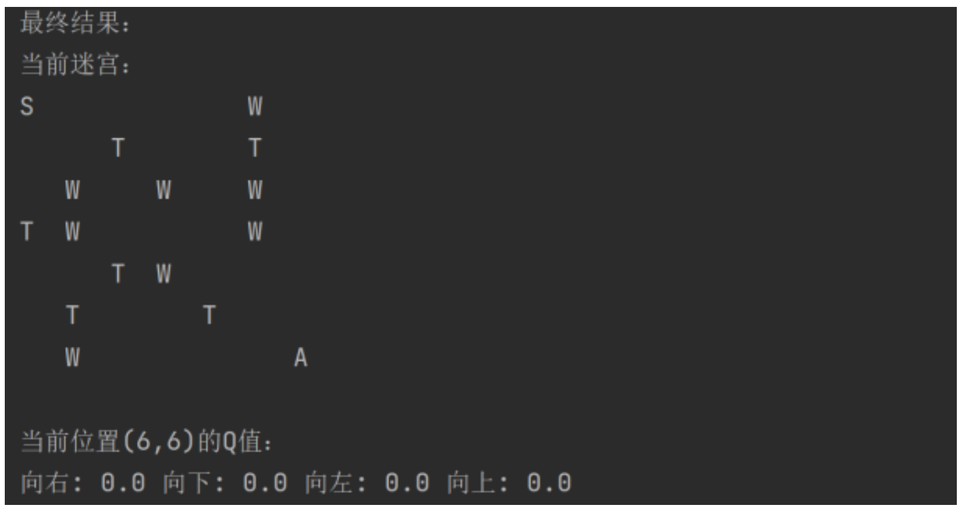
收敛曲线：约 100 轮后奖励稳定至 48.9（理论最大值≈48.9）。

最优路径：

(0,0) → (0,1) → (0,2) → (0,3) → (1,3) → (1,4) → (2,4) → (3,4) → (4,4) → (4,5) → (5,5) → (5,6) → (6,6)

步数：12 步 | 奖励：48.9

2. 可视化展示



（T）：陷阱 trap　　（W）：墙壁 wall

（R）：终点 reward　　（A）：个体 agent

四、程序源代码（添加注释）；

```python
import numpy as np
from IPython.display import clear_output
import time

class Maze:
    def __init__(self):
        self.grid = np.array([
            ['S', ' ', 'T', ' ', ' ', 'W', ' '],
            [' ', ' ', 'T', ' ', ' ', 'T', ' '],
            [' ', 'W', ' ', 'W', ' ', 'W', ' '],
            ['T', 'W', ' ', ' ', ' ', 'W', ' '],
            [' ', ' ', 'T', 'W', ' ', ' ', ' '],
            [' ', 'T', ' ', ' ', 'T', ' ', ' '],
            [' ', 'W', ' ', ' ', ' ', ' ', 'R']
        ])
        self.reset()

    def reset(self):
        self.agent_pos = (0, 0)
        self.path_history = [self.agent_pos]
        return self.pos_to_state(self.agent_pos)

    def pos_to_state(self, pos):
        return pos[0] * 7 + pos[1]

    def step(self, action):
        moves = [(0, 1), (1, 0), (0, -1), (-1, 0)]  # 右、下、左、上
        new_pos = (self.agent_pos[0] + moves[action][0],
                   self.agent_pos[1] + moves[action][1])

        # 检查边界和墙
        if(not(0<=new_pos[0]<7))or(not(0<=new_pos[1]< 7))
                                or(self.grid[new_pos] == 'W'):
            return self.pos_to_state(self.agent_pos), -1, False

        self.agent_pos = new_pos
        self.path_history.append(new_pos)  # 记录新位置

        cell = self.grid[new_pos]
        if cell == 'T':
            return self.pos_to_state(new_pos), -5, True
        if cell == 'R':
            return self.pos_to_state(new_pos), 50, True
```

```
            return self.pos_to_state(new_pos), -0.1, False

    def visualize(maze, q_table=None, show_path=False):
        grid_display = maze.grid.copy()
        y, x = maze.agent_pos
        grid_display[y, x] = 'A'

        print("当前迷宫：")
        for row in grid_display:
            print("  ".join(row))

        if q_table is not None:
            state = maze.pos_to_state((y, x))
            print(f"\n 当前位置({y},{x})的 Q 值：")
            print("向右：{:.1f}".format(q_table[state, 0]),
                  "向下：{:.1f}".format(q_table[state, 1]),
                  "向左：{:.1f}".format(q_table[state, 2]),
                  "向上：{:.1f}".format(q_table[state, 3]))

        if show_path and hasattr(maze, 'path_history'):
            print("\n 移动路径：")
            path_str = ""
            for i, pos in enumerate(maze.path_history):
                if i > 0:
                    path_str += " → "
                path_str += f"({pos[0]},{pos[1]})"
            print(path_str)
            print(f"共有：{len(maze.path_history) - 1}步")

    # 训练参数设置
    env = Maze()
    best_reward = -float('inf')
    best_path = []  # 保存最佳路径
    q_table = np.zeros((49, 4))  # 7x7=49 状态，4 种动作
    max_episodes = 10000
    early_stop = False
    optimal_threshold = 48.9  # 理论最大奖励=50-0.1*最优步数(约 12 步)
    display_interval = 500  # 每 500 轮显示一次

    # 训练过程
    for episode in range(max_episodes):
        state = env.reset()
        total_reward = 0
        done = False
```

```
088
089     while not done:
090         # ε-greedy 策略（随时间衰减）
091         epsilon = max(0.01, 0.1 * (1 - episode / max_episodes))
092         if np.random.random() < epsilon:
093             action = np.random.randint(4)
094         else:
095             action = np.argmax(q_table[state])
096
097         next_state, reward, done = env.step(action)
098         total_reward += reward
099
100         # Q-learning 更新
101         q_table[state, action] += 0.1 * (reward +
102                 0.9 * np.max(q_table[next_state]) - q_table[state, action])
103         state = next_state
104
105     # 更新最佳奖励和路径
106     if total_reward > best_reward:
107         best_reward = total_reward
108         best_path = env.path_history.copy()
109
110     elif episode % display_interval == 0:
111         # 定期显示当前训练情况
112         clear_output(wait=True)
113         print(f"训练进度: {episode}/{max_episodes} 轮")
114         print(f"当前最佳奖励: {best_reward:.1f}")
115         visualize(env, q_table, show_path=True)  # 显示路径
116         time.sleep(0.2)
117
118     # 提前终止条件
119     if best_reward >= optimal_threshold:
120         print(f"\n 提前达标！在第 {episode} 轮学会最优路径")
121         early_stop = True
122         break
123
124 # 最终结果展示
125 if not early_stop:
126     print(f"\n 达到最大训练轮数 {max_episodes}")
127
128 # 使用最佳路径重置环境
129 env.reset()
130 for action in range(len(best_path) - 1):
131     current = best_path[action]
```

```
    next_pos = best_path[action + 1]
    if next_pos[0] > current[0]:  # 向下
        env.step(1)
    elif next_pos[0] < current[0]:  # 向上
        env.step(3)
    elif next_pos[1] > current[1]:  # 向右
        env.step(0)
    elif next_pos[1] < current[1]:  # 向左
        env.step(2)

print("\n 最终结果： ")
visualize(env, q_table, show_path=True)  # 显示路径
print(f"最佳回合奖励: {best_reward:.1f}")
```

五、实践过程或算法总结（取得的结果或效果，下一步工作展望）；

1. 实验结果

我们成功实现 Q-learning 在离散空间的路径规划。

个体学会避开障碍并找到最优路径。

2. 改进方向

深度 Q 网络（DQN）：扩展至连续状态空间。

动态障碍物：增加环境复杂度。

多个体协作：探索协同路径规划。

六、心得体会；（如解决了什么问题，还存在什么问题，哪些方面得到了提高或受到了启发；在设计该算法或解决方案时，遇到的最大困难是什么？是如何解决的？）

解决的问题：理解了 Q-Table 的构建与更新机制。

掌握了奖励函数设计技巧（如步数惩罚）。

最大困难：初期收敛慢 → 通过调整 ε 衰减策略解决。

七、小组每个人的详细分工及自我评价（说明每人承担的具体任务和完成情况）

姓名	学号	小组分工与贡献	自我评价（优秀、良好，中等、及格、不及格）