编程作业三——火焰杯试炼问题报告

2023011004　自 35　夏弘宇　xiahy23@mails.tsinghua.edu.cn

2025 年 5 月 20 日

1　强化学习建模

1.1　状态空间

- **定义**：所有可能的迷宫格子坐标，共 $5 \times 5 = 25$ 个状态

$$S = \{(x, y) \mid x, y \in \{0, 1, 2, 3, 4\}\} \tag{1}$$

1.2　动作集合

- **定义**：上下左右四个移动方向：

$$A = \{\text{上}, \text{下}, \text{左}, \text{右}\} \tag{2}$$

- **边界处理**：若移动导致出界，则保持原状态

1.3　状态转移概率

对于状态 $s = (x, y)$ 和动作 a，计算目标位置 $s'_0 = (x', y')$

- 若 s'_0 是陷阱或火焰杯，则转移到起点 $s' = (0, 0)$，概率为 1，游戏结束。
- 若 s'_0 不是陷阱或火焰杯，则转移到 $s' = s'_0$，概率为 1。

数学表示为：$P(s' \mid s, a) = 1$

1.4　Reward 函数

虽然作业模板中给了一些默认值，但我觉得-1,0,1 的 reward 差异性不够大，就设计了如下比较激进的 reward 函数：

$$R(s, a, s') = \begin{cases} +100 & \text{到达火焰杯} \\ -10 & \text{触发陷阱} \\ -5 & \text{撞墙} \\ -2 & \text{其他移动} \end{cases} \tag{3}$$

- 每移动一步施加 -2 惩罚，减少无效探索
- 撞墙惩罚 -5，减少步数浪费
- 陷阱惩罚 -10，避免重复触发
- 火焰杯奖励 $+100$，激励快速找到目标

2 Q-learning 算法实现

2.1 Q-table 定义与状态编码

- **状态空间建模**：
  - 状态索引公式：$\text{state} = y \times \text{maze\_width} + x$
  - 其中 (x, y) 为网格坐标（范围 0-4）
- **动作空间设计**：
  - 0: 向上移动
  - 1: 向下移动
  - 2: 向右移动
  - 3: 向左移动
- **初始化策略**：
  - Q-table 维度为 25×4 矩阵
  - 初始值设为 0

2.2 核心参数设置与使用

对于这个比较简单的迷宫问题，训练轮数应该不用很多，设置了 100 轮，相应地，学习率就要大一些，取了 $\alpha = 0.1$；由于这个问题的确定性很强，所以折现因子设置得大一点 $\gamma = 0.9$，看看效果；初始探索率 $\epsilon = 0.1$ 是一个比较经典的取值，由于知识不断积累，需要探索的内容会变少，所以需要进行探索率的衰减，这里使用比较经典的衰减：$\epsilon_{new} = \max(0.01, 0.995\epsilon)$。

Q table 更新公式：$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_{a \in A} Q(S_{t+1}, a) - Q(S_t, A_t)\right]$

$\epsilon - greedy$ 策略的公式：

$$\pi(a|s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A|} & \text{如果} a = \arg\max_{a \in A} Q(s, a) \\ \frac{\epsilon}{|A|} & \text{其他情况} \end{cases} \tag{4}$$

2.3 智能体训练流程

如下伪代码展示了训练过程（忽略了可视化的部分）

Algorithm 1 Q-learning 训练过程

输入： 迷宫环境 $\mathcal{M}$, 最大训练轮数 $E = 100$, 最大步数 $T = 100$
输出： 最优 Q-tableQ^*

初始化 Q-table $Q(s, a) \leftarrow 0,\ \forall s \in \mathcal{S}, a \in \mathcal{A}$
设置初始探索率 $\epsilon \leftarrow 0.1$
for 回合数 $e = 1$ **to** E **do**
  重置环境获得初始状态 s_0
  初始化累计奖励 $G \leftarrow 0$
  for 时间步 $t = 0$ **to** $T - 1$ **do**
    动作选择:

$$a_t \leftarrow \begin{cases} \text{随机选择}, & \text{概率}\epsilon \\ \arg\max_a Q(s_t, a), & \text{概率}1 - \epsilon \end{cases}$$

    执行动作: $(s_{t+1}, r_t, \text{done}) \leftarrow \text{EnvStep}(a_t)$
    Q 值更新:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\big[r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\big]$$

    路径记录: $\mathcal{P} \leftarrow \mathcal{P} \cup \{(s_t, a_t)\}$
    奖励累计: $G \leftarrow G + r_t$
    if done = True **then**
      退出循环
    end if
  end for
  探索率衰减: $\epsilon \leftarrow \max(0.01, 0.995\epsilon)$
  奖励记录: $\mathcal{R} \leftarrow \mathcal{R} \cup \{G\}$
end for

2.4 核心代码解释

本项目的核心代码由三部分组成：Q-learning 智能体、训练框架和迷宫环境。以下是关键代码的解释：

2.4.1 Q-learning 智能体（agent.py）

- **Q 表初始化**

```
self.q_table = np.zeros((self.n_states, self.n_actions))
```

创建状态-动作价值矩阵，维度为（迷宫格子总数 × 4 个动作）。每个元素表示在特定状态下采取某动作的预期收益。

- **状态编码**

```
x = int((coords[0] - 50) / 100)
y = int((coords[1] - 50) / 100)
return y * self.maze_width + x
```

将像素坐标 (如 [150,250]) 转换为网格坐标（y=2, x=1），再编码为状态索引（2×5 +1=11）。这是连接 GUI 环境与算法的关键转换。

- **动作选择**

```
if random.uniform(0, 1) < self.epsilon:
    return random.randint(0, 3)
max_actions = np.where(...)
return np.random.choice(max_actions)
```

采用 -greedy 策略：以概率 ϵ 随机探索，以 $1-\epsilon$ 概率选择当前最优动作。随机选择 Q 值并列情况确保探索多样性。

- **Q 值更新**（learn 方法）

```
q_target = r + self.gamma * np.max(self.q_table[next_state])
self.q_table[state, a] += self.alpha * (q_target - ...)
```

实现 Q-learning 更新规则：新 Q 值 = 旧值 + 学习率 ×(即时奖励 + 折扣因子 × 下状态最大 Q 值 - 旧值)

2.4.2 训练框架

- **训练循环**（train 函数）

```
for episode in range(n_episodes):
    observation = env.reset()
    for step in range(max_steps):
        action = agent.choose_action(observation)
        observation_, reward, done = env.step(action)
        agent.learn(observation, action, reward, observation_)
```

包含完整的" 选择动作 → 执行动作 → 获得反馈 → 更新 Q 表" 循环。每个 episode 最多执行 100 步，动态调整探索率。

- **探索率衰减**

```
agent.epsilon = max(0.01, agent.epsilon * 0.995)
```

随着训练进行逐步降低探索率，平衡探索与利用。设置下限 0.01 防止完全停止探索。

- **路径可视化**

```
visualize_path(episode + 1, path, success)
```

调用可视化函数保存路径图像，如图 1 所示。处理终端状态时推断最终位置（到达目标或最近陷阱）。

2.4.3 迷宫环境

- **状态转移逻辑**（step 方法）

```
if s_ == self.canvas.coords(self.goal):
    reward = 100
```

```
elif s_ in trap_coords:
    reward = -10
elif hit_wall:
    reward = -5
else:
    reward = -2
```

定义强化学习的关键反馈机制：到达目标 +100，触碰陷阱-10，撞墙-5，正常移动-2。负奖励加速收敛。

- **动作空间限制**

```
if action == 0 and s[1] > UNIT: # 上移边界检测
    base_action[1] -= UNIT
```

实现网格世界的运动约束，防止移出迷宫边界。每个动作对应 100 像素的位移。

3 训练结果分析

3.1 训练过程指标

表 1: 训练阶段关键指标

训练回合	单回合奖励	探索率 (ϵ)	最近 10 回合成功率
10	-38.0	0.096	10%
20	52.0	0.091	60%
30	78.0	0.086	70%
40	90.0	0.082	80%
50	90.0	0.078	100%
60	90.0	0.074	90%
100	90.0	0.061	100%

收敛性分析：

- **reward 提升**：平均单回合奖励从-38（第 10 轮）提升至 +90（第 40 轮后）
- **稳定期**：60 回合后保持接近 100% 成功率，表明策略收敛

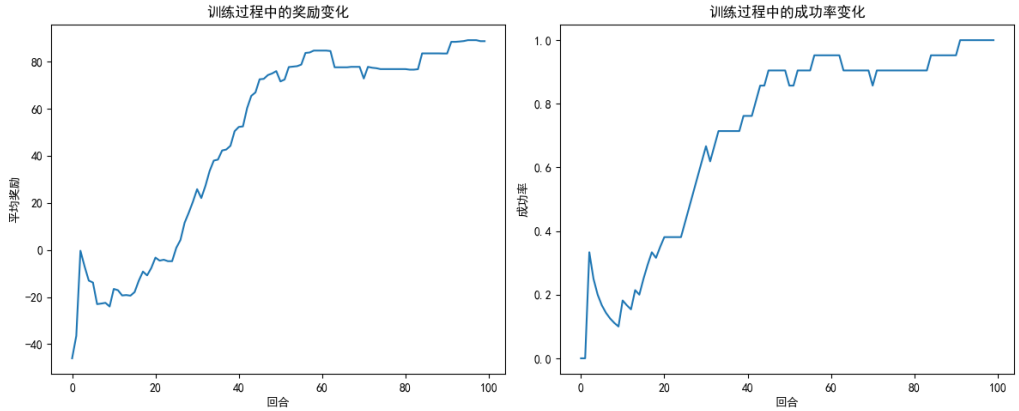

图 1: 训练指标变化曲线

为了方便分析，我将成功率绘制成曲线图。可以发现**成功率曲线**呈现三阶段增长：

- 探索期（1-30 回合）：成功率 $< 60\%$
- 过渡期（30-50 回合）：成功率快速提升
- 稳定期（50+ 回合）：几乎保持 100% 成功率

3.2 路径演化

3.3 Q-table 特征分析

完整学习后的 Q-table 如下（状态编码为 5×5 网格的行优先排列）：

表 2: 最终 Q-table 示例（按网格坐标索引）

坐标 (x, y)	上	下	右	左
(0,0)	-0.83	-4.08	40.60	-1.03
(1,0)	-3.76	-2.95	52.83	0.38
(2,0)	-2.40	-2.71	64.73	1.24
(3,0)	10.88	76.25	-0.74	2.99
(4,0)	-0.50	-0.54	-0.50	-0.42
(0,1)	-0.43	-3.32	-3.31	-3.40
(1,1)	4.97	-3.44	-3.44	-3.01
(2,1)	0.00	0.00	0.00	0.00
(3,1)	5.72	87.78	-0.20	-1.00
(4,1)	-0.22	-0.38	-0.50	-0.14
(0,2)	-2.81	-2.80	-2.71	-2.93
(1,2)	0.00	0.00	0.00	0.00
(2,2)	0.00	0.00	0.00	0.00
(3,2)	-0.08	5.68	0.28	99.98
(4,2)	-0.22	-0.40	0.00	14.02
(0,3)	-2.30	-2.23	-2.71	-2.42
(1,3)	0.00	0.00	0.00	0.00
(2,3)	0.00	-0.22	-0.14	-1.00
(3,3)	34.61	0.00	-0.32	0.00
(4,3)	-0.20	-0.20	-0.50	4.07
(0,4)	-1.78	-1.97	-1.73	-2.33
(1,4)	-1.90	-1.44	-1.03	-1.15
(2,4)	-0.54	-0.50	-0.54	-0.44
(3,4)	0.40	-0.50	-0.20	-0.20
(4,4)	-0.20	-0.50	-0.50	-0.20

由于 Q 值的表看着非常抽象，我把每一个单元格中最大值提取出来进行了热力图可视化：

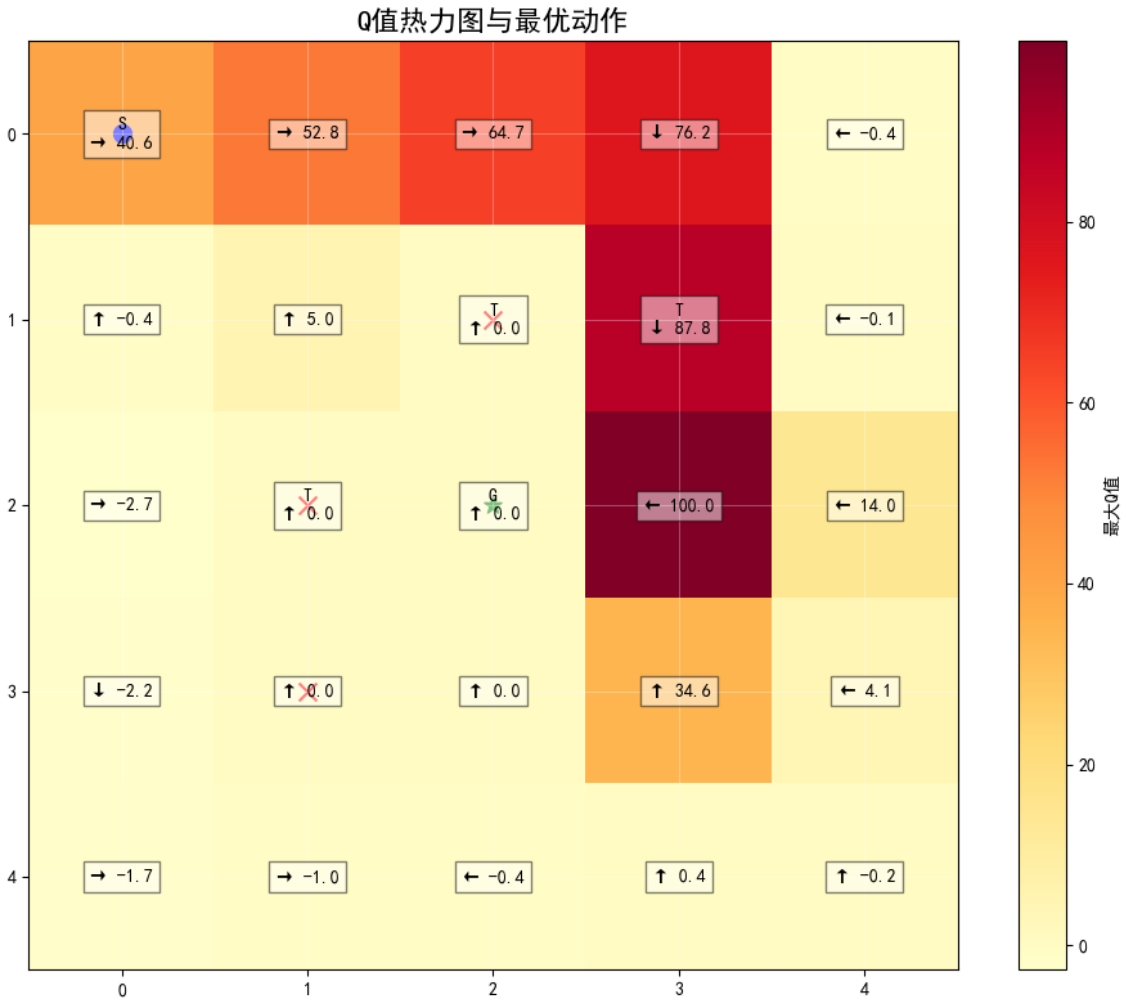

图 2: Q 值热力图

主要特征：

1. **目标导向**：火焰杯右侧格子 $(3,2)$ 向左的 Q 值达 99.94，形成明显梯度引导
2. **陷阱规避**：陷阱周围状态（如 $(0,2)$）所有动作 Q 值均 < 0，使得最大值竟然是指向陷阱的，这是不好的，可能是因为学习过程中探索到这一块的次数太少导致的。如果有需要，可以提升其探索率 ϵ
3. **最优路径**：很明显从起点开始有一条颜色很深的路径指向火焰杯，这条路径就是最优路径，引导得很到位

4 思考题：2-step Q-learning 算法设计

哈利提出的 2-step Q-learning 算法更新规则为：

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma R_{t+2} + \gamma^2 \max_{a \in A} Q(S_{t+2}, a) - Q(S_t, A_t) \right]$$

表 3: 公式符号定义

符号	含义
S_t	时刻 t 的状态
A_t	时刻 t 选择的动作
α	学习率 $(0 < \alpha \leq 1)$
γ	折扣因子 $(0 \leq \gamma < 1)$
R_{t+1}	执行 A_t 获得的即时奖励
R_{t+2}	执行 A_{t+1} 获得的次级奖励
$\max_{a \in A} Q(S_{t+2}, a)$	两步后的最大预期价值

Algorithm 2 2-step Q-learning 更新流程

在 t 时刻：执行 A_t，观测 (S_{t+1}, R_{t+1})
在 $t+1$ 时刻：执行 A_{t+1}，观测 (S_{t+2}, R_{t+2})
更新历史记录：$\mathcal{H} \leftarrow (S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, S_{t+2})$
计算 TD 目标：$target \leftarrow R_{t+1} + \gamma R_{t+2} + \gamma^2 \max_a Q(S_{t+2}, a)$
更新 Q 值：$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[target - Q(S_t, A_t)]$

这一设计，相比标准 Q-learning 的 1 步回溯，能识别跨越两个状态的因果关系，比较适合火焰杯试炼等需要预测多步之后情况的场景。