中国科学技术大学
University of Science and Technology of China

《人工智能数学原理与算法》
第 7 章：强化学习

7.4 Q 学习算法（Q-Learning）

吉建民
jianmin@ustc.edu.cn

目录

目录

什么是 Q 学习?

Q 学习算法（Q-learning）：是一种基于价值的强化学习算法，用于解决**具有明确奖励的马尔可夫决策问题**。其基本思想是通过**学习一个 Q 函数**来选择最优的行为。

- 对所有的状态-动作对下的最优 Q 值 q*(s,a) 进行估计，对于任意状态s，令 q*(s，a) 最大的 a 即为最优动作

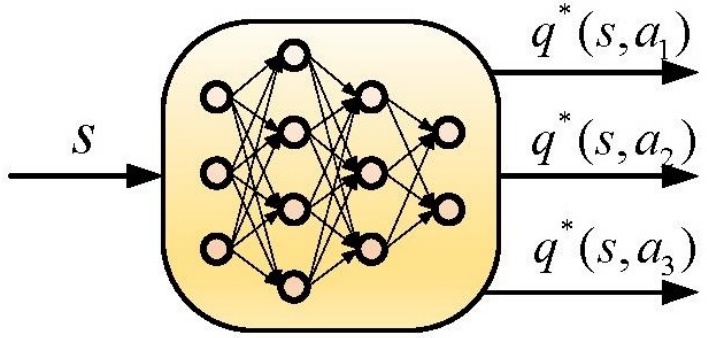
◆ **Q-learning**

离散状态、离散动作

	a_1	a_2	a_3
s_1	$q^*(s_1,a_1)$	$q^*(s_1,a_2)$	$q^*(s_1,a_3)$
s_2	$q^*(s_2,a_1)$	$q^*(s_2,a_2)$	$q^*(s_2,a_3)$
s_3	$q^*(s_3,a_1)$	$q^*(s_3,a_2)$	$q^*(s_3,a_3)$

◆ **Deep Q-learning**

连续状态、离散动作



最优动作：

$a^*(s) = \max_a q^*(s,a)$

最优Q值：

$q^*(s,a) = \sum_{s`,r} p(s`,r|s,a)[r + \gamma\, max_a\, q^*(s`,a`)]$

算法特点

- 依据贝尔曼方程对最优 Q 值进行迭代更新；
- 动作空间必须离散；
- 当动作维度增大时，神经网络规模指数增加。

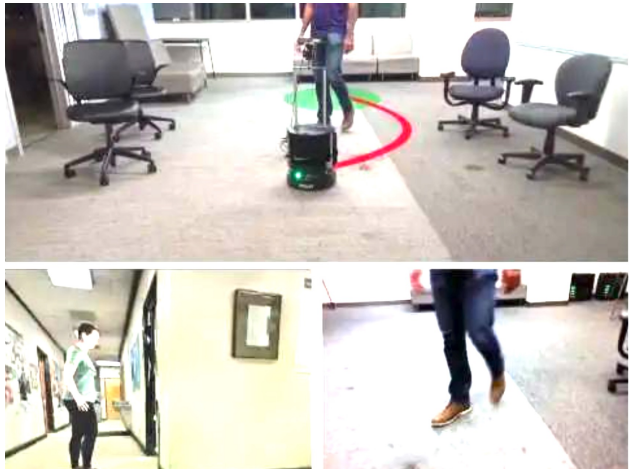
什么是 Q 函数?

- **Q 函数**：也称动作价值函数（Action-Value Function），是一种特殊的价值函数。
  - Q 函数用来评估每个状态和每个行为的预期奖励（即 Q 值）。Q 值越大，说明在该状态下采取相应行动越有利。
  - Q-学习算法通过迭代地更新 Q 值来逼近最优 Q 函数。
  - Q 函数的表格形式如下所示：

	第 1 种动作	第 2 种动作	第 3 种动作	第 4 种动作
第 1 种状态	380	-95	20	173
第 2 种状态	-7	64	-195	210
第 3 种状态	152	72	413	-80

Q 学习应用

- Q 学习目前广泛用于游戏、机器人和自动驾驶等领域。



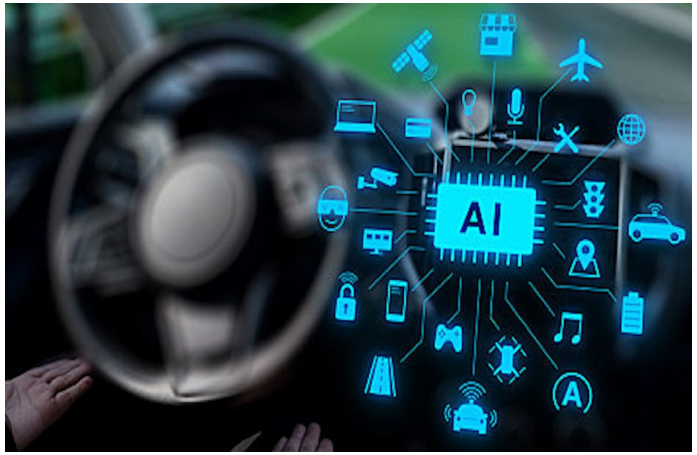
训练机器人路径规划和导航

优化资源分配例如网络流量控制

各种棋盘游戏和视频游戏的开发

在医疗领域用来辅助诊断

可以用于开发智能教学系统



帮助汽车在不同交通状况做决策

目录

Q 值更新公式

- Q 学习算法的核心在于**更新 Q 值**的公式，该公式基于**贝尔曼方程**，并用于迭代地更新**状态-动作对的价值估计**。Q值的更新公式如下：

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

其中：

- $Q(s_t, a_t)$ 是时间步 t的状态下采取动作 a_t 的 Q 值。
- α 是学习率，它决定了新信息对 Q 值更新的影响程度。
- r_{t+1} 是在采取动作。
- γ 是折扣因子，它反映了未来奖励相对于即时奖励的重要性。
- $max_a Q(s_{t+1}, a)$ 是在新状态 s_{t+1}下所有可能动作的最大 Q 值，代表了对未来奖励的最大预期。

- 这个公式结合了**即时奖励**和对**未来奖励的预期**，通过不断地更新 Q 值，智能体可以学习到在每个状态下应该采取哪个动作以**最大化长期累积奖励**。

贝尔曼方程与最优策略

- **贝尔曼方程**描述了一个状态的价值可以通过**即时奖励**和**未来价值**的综合来计算。对于 Q 学习来说，贝尔曼方程用来计算最优策略下的 Q 值，即：

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a) \max_{a' \in A} Q^*(s',a')$$

其中：

- $Q^*(s,a)$ 是在最优策略下状态 s 采取动作 a 的最优 Q 值。
- $R(s,a)$ 是在状态 s 下采取动作 a 获得的即时奖励。
- γ 是折扣因子。
- $P(s'|s,a)$ 是从状态 s 采取动作 a 转移到状态 s' 的概率。

- Q 学习的目标是找到是 Q 值最大化的策略，即：

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

这意味着在给定状态下，最优策略是选择能够使 Q 值最大的动作。通过不断迭代地更新 Q 值，Q学习算法能够收敛到最优 Q 值，从而学习到最优策略。

目录

共同点

基于贝尔曼最优方程

- 两者都通过迭代逼近 **最优价值函数**（V^* 或 Q^*），最终目标是找到最优策略 π^*。
- **值迭代**：

$$Q_{k+1}(s,a) = R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) \max_{a'} Q_k(s',a').$$

- Q–learning：

$$Q_{t+1}(s_t,a_t) = Q_t(s_t,a_t) + \alpha \left[r_{t+1} + \gamma \max_{a'} Q_t(s_{t+1},a') - Q_t(s_t,a_t) \right]$$

- 本质上，Q–learning 是贝尔曼方程的**随机采样版本**，而值迭代是**精确期望计算版本**。

Q-Value Iteration	Q-Learning
$Q(s,a) \leftarrow \mathbb{E}\left[R + \gamma \max_{a' \in \mathcal{A}} Q(S',a') \mid s,a\right]$	$Q(S,A) \overset{\alpha}{\leftarrow} R + \gamma \max_{a' \in \mathcal{A}} Q(S',a')$ $\quad x \overset{\alpha}{\leftarrow} y \equiv x \leftarrow x + \alpha(y-x)$

迭代收敛性

- 在理想条件下（如无限次访问所有状态–动作对），两者均能收敛到最优解。

核心区别

特性	值迭代	Q-learning
模型需求	已知状态转移概率 $P(s'\|s,a)$ 和奖励函数 $R(s,a)$（如网格世界、规划问题）	环境模型未知，需通过试错交互学习（如游戏、机器人控制）
更新方式	需遍历所有状态同步更新	异步更新，仅更新访问的 (s,a)
收敛条件	理论保证收敛到 Q^*	需充分探索才能收敛到 Q^*
适用场景	小规模离散状态空间（如网格世界）	中小规模离散状态（需遍历所有 (s,a)）
计算效率	高（批量更新，适合小规模问题）	低（单样本更新，适合在线学习）
探索机制	无需探索（直接计算最优值）	需探索策略（如 ε-greedy）

目录

算法流程

Q 学习的算法流程如下：

- **初始化 Q 表**：创建一个 Q 表，通常初始化为 0。
- **选择动作**：在每个时间步骤中，智能体根据当前状态和 Q 表选择一个动作。这通常涉及到探索和利用的平衡，以确保在学习过程中不断地探索新的动作策略。
- **执行动作**：智能体执行所选择的动作，并观察环境的响应，包括获得的奖励信号和新的状态。
- **更新 Q 值**：根据观察到的奖励信号和新的状态，智能体更新 Q 值，这涉及到上面提到的 Q-学习更新规则即贝尔曼方程。
- **重复迭代**：智能体不断地执行上述步骤，与环境互动，学习和改进 Q 函数，直到达到停止条件。

算法流程

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'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

初始化 Q 表

- **初始化 Q 值**：Q-学习算法的第一步是初始化 Q 值。Q 值表示特定状态下采取特定动作的预期回报。在算法开始的时候，Q 值通常被初始化为零，以表示我们对环境的无知。这个 Q 表会随着智能体与环境的交互不断更新和改进。
  - Q 表初始化：Q 表是一个二维数组，其中行表示状态，列表示动作。每个单元格 $Q(s,a)$ 表示在状态 s 下采取动作 a 的预期回报。初始化时，$Q(s,a)=0$ 对于所有 s 和 a 。

选择动作与执行

- **选择动作与执行**：在每个时间步骤，智能体需要根据当前状态选择一个动作。Q-学习使用 *ε-greedy* 策略来平衡探索和利用。
  - ε-greedy 策略：以 *1-ε* 的概率选择当前 Q 值最高的动作，以 *ε* 的概率随机选择一个动作。这种策略允许智能体在大多数时间利用已知的最佳动作，同时保留一定的概率去探索新的动作，以发现可能更好的策略。
  - 动作执行：智能体执行选择的动作，并观察环境的响应，包括获得的奖励信号和新的状态。

$$\pi(a \mid s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a* = argmax_{a \in A} Q(s, a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

环境反馈与 Q 值更新

- **环境反馈与 Q 值更新**：智能体根据环境的反馈更新 Q 值，这是 Q-学习算法的核心步骤。
  - 奖励和新状态：智能体执行动作之后，会收到一个奖励r和一个新的状态 s′。
  - Q 值更新：使用以下公式更新 Q 值：

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

    - α 是学习率，控制新信息的影响程度。
    - γ 是折扣因子，控制未来奖励的当前价值。
    - $max_{a'}Q(s',a')$ 是在新状态 s' 下所有可能动作的最大 Q 值，表示对未来奖励的最大预期。
  - 迭代过程：智能体重复选择动作、执行动作和更新 Q 值的过程，直到达到某个终止条件，如达到最大迭代次数或 Q 值收敛。

目录

Q 学习的优缺点

优点:

- 无模型（Model-Free）
  - 无需环境动态信息：不需要知道状态转移概率或者奖励函数仅通过与环境交互的样本学习。
  - 适用性广：可以用于真实世界的问题（如机器人控制、游戏 AI），其中环境模型可能未知或复杂。
- 离线策略（Off-Policy）
  - 灵活探索：使用 ε-*greedy* 等策略探索环境，但通过更新公式学习最优策略（贪婪策略）。
  - 可复用历史数据：可以从过去的经验（如经验回放缓冲区）中学习，提高数据效率。

缺点:

- 维度灾难
  - Q 表的局限性：状态和动作空间需离散化，若状态高维或者连续，表格存储不可行。
- 对超参数敏感
  - 学习率 α、折扣因子 γ、探索率 ε 的选择直接影响性能。

Q 学习的扩展

- **Double Q-learning**：传统的 Q 学习的 max 操作会导致 Q 值高估的问题。改进后使用 2 个 Q 表交替更新来减少偏差。

$$
\begin{aligned}
Q_1(S_t, A_t) \leftarrow\ & Q_1(S_t, A_t) \\
& + \alpha\left(R_{t+1} + \gamma Q_2(S_{t+1}, \mathit{argmax}_a Q_1(S_{t+1}, a)) - Q_1(S_t, A_t)\right) \\
Q_2(S_t, A_t) \leftarrow\ & Q_2(S_t, A_t) \\
& + \alpha\left(R_{t+1} + \gamma Q_1(S_{t+1}, \mathit{argmax}_a Q_2(S_{t+1}, a)) - Q_2(S_t, A_t)\right)
\end{aligned}
$$

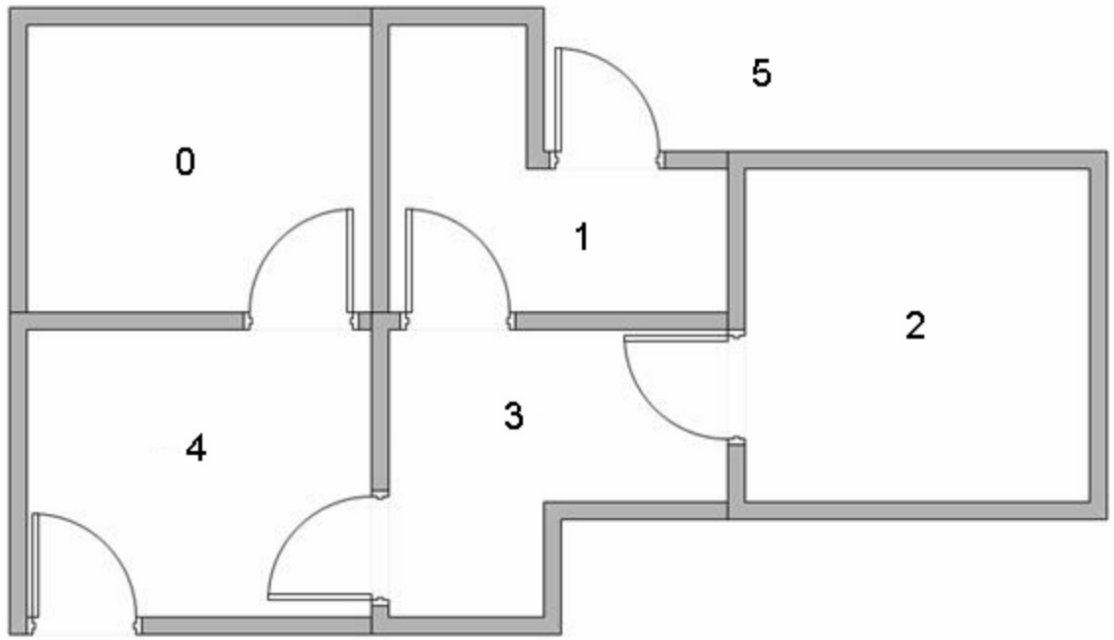
- **Multi-step Q-learning (n-step Bootstrapping)**：单步 Q–learning 更新方差大（依赖单次采样）。改进后使用多步奖励（类似 Monte Carlo 和 TD 的折中）。

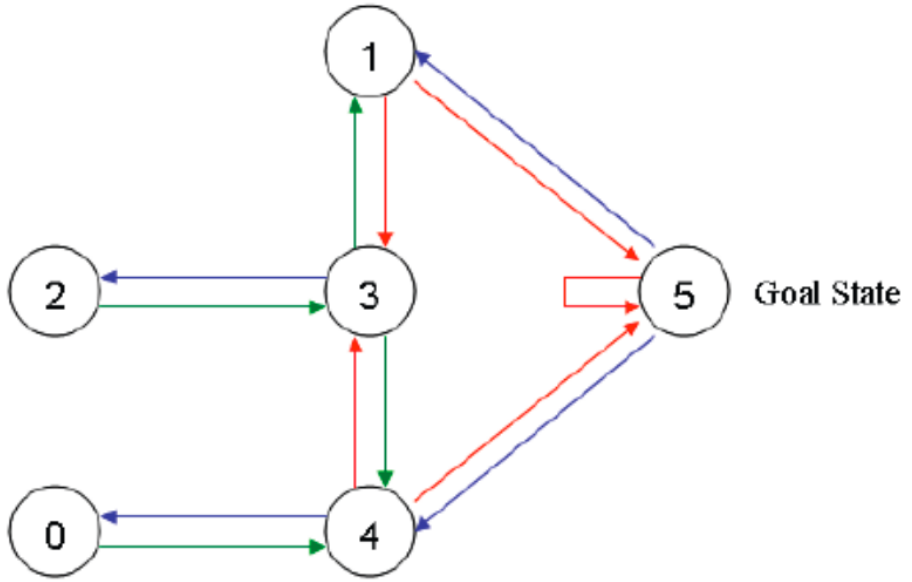
$$
G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \max_a Q_t(S_{t+n}, a)
$$

目录

房间问题

- 假设有一个这样的大房间，有门表示互相连通，将房间表示为点，连通关系表示为线，则上图可以建模为左图。





- 这就是房间对应的图。我们首先将 agent（机器人）处于任何一个位置，让他自己走动，直到走到 5 房间，表示成功。为了能够走出去，我们将每个节点之间设置一定的权重，能够直接到达 5 的边设置为 100，其他不能的设置为 0，这样网络的图为右图。

房间问题

- Q 学习中，最重要的就是“**状态**”和“**动作**”，状态表示处于图中的哪个节点，比如 2 节点，3 节点等等，而动作则表示从一个节点到另一个节点的操作。
- 首先我们生成一个奖赏矩阵：

$$R = \begin{array}{c} \\ \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} \multicolumn{6}{c}{\text{Action}} \\ 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

其中:
-1 表示不可以通过
0 表示可以通过
100 表示直接到达终点

总结就是：R 矩阵中非负的表示节点之间是可以相通的。

房间问题

- 同时，我们创建一个 Q 表，表示学习到的经验，首先初始化为 0 矩阵。
- 然后根据 Q 学习的转移方程更新 Q 表。
- 当 Q 表更新完以后，就可以根据 Q 表来选择路径。
- 例如，设置 ε 为 0.8，可以得到奖励函数矩阵和初始化 Q 表：

$$R=\begin{array}{c} \\ \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}\begin{array}{c} \begin{array}{cccccc} & & & \text{Action} & & \end{array} \\ \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

$$Q=\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}\begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

房间问题

- 随机选择一个状态，比如 1，查看状态 1 所对应的 R 表，也就是 1 可以到达 3 或 5，随机地，我们选择 5，根据转移方程：

$$\begin{aligned} Q(1,5) &= R(1,5) + 0.8 * \max\{Q(5,1), Q(5,4), Q(5,5)\} \\ &= 100 + 0.8 * \max\{0,0,0\} \\ &= 100. \end{aligned}$$

- 于是 Q 表为：

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

- 这样，到达目标，一次尝试结束。

房间问题

- 接下来再选择一个随机状态，比如 3，3 对应的下一个状态有 1，2，4 都是状态 3 对应的非负状态，随机地，我们选择 1，这样根据算法更新：

$$
\begin{aligned}
Q(3,1) &= R(3,1) + 0.8 * \max\{Q(1,3), Q(1,5)\} \\
&= 0 + 0.8 * \max\{0, 100\} \\
&= 80.
\end{aligned}
$$

- 于是 Q 表为：

$$
Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c}
\begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\
\left[\begin{array}{cccccc}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 100 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 80 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0
\end{array}\right]
\end{array}
$$

- 到达 1 状态以后，可以直接到达 5，这样一次训练也完成了。

课后作业

□ 思考题：

题目 1：

在 Q-learning 中，更新公式为：

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

请回答：

学习率 α 的作用是什么？若 $\alpha=1$ 或 $\alpha=0$，会发生什么？

题目 2：

假设 Q–learning 的探索策略为 ϵ-*greedy*（$\epsilon=0.1$），请回答：

训练初期和后期，ϵ 的值是否需要调整？为什么？如果完全去掉探索（$\epsilon=0$），可能会有什么问题？

谢谢！