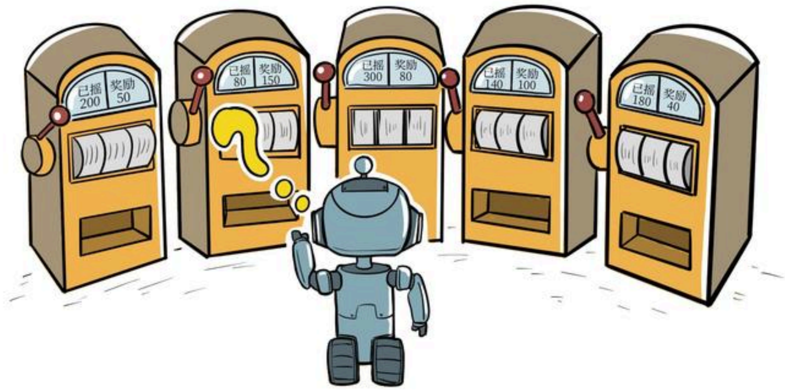
多臂老虎机

强化学习使用对所采取动作的**评价**进行训练，而不是通过给出正确动作为指导。这导致其需要**主动探索、显式搜索良**好行为。

- **评价性反馈**：表明所采取动作的好坏，但不说明是否为最优或最差动作。
- **指导性反馈**：直接给出正确动作，与实际采取的动作无关。

强化学习使用**评价性反馈**（evaluative feedback），而非指导性反馈。**评价性反馈完全依赖于所采取的动作(不同的动作会有不用的反馈)，而指导性反馈则与所采取的动作无关(不同的动作会有相同的反馈)。**这部分中只研究强化学习的**评价性特征**，在一个**简化场景**中进行，该场景为**非关联性**问题，不涉及在多种情境下学习如何行动。



> **Note**
>
> 想象你面前有多个老虎机（比如10台），每台机器拉下拉杆后会以某种未知概率给你不同金额的奖励。你的目标是在有限次数的拉杆操作中，**最大化总奖励。**
>
> 但问题是：你不知道每台机器的真实收益分布 —— 有些机器可能回报高，有些可能回报低。

k臂老虎机问题可以定义为面临 k 个不同动作（或选项）的重复选择；每次选择一个动作后，获得一个数值奖励；奖励来自**依赖于所选动作的平稳概率分布**，你的目标是在有限时间步（如1000步）内，**最大化累计奖励的期望值**。当前的“老虎机问题”特指此**简单非关联性情形**，不涉及状态变化。

动作价值

每个动作 a 有一个真实价值 $q_*(a)$，定义为**选择该动作时的期望奖励**：

$$q_*(a) \doteq \mathbb{E}[R_t \mid A_t = a]$$

- A_t：时间步 t 选择的动作
- R_t：时间步 t 获得的奖励

当前动作价值的真实价值未知，需进行**估计**。用 $Q_t(a)$ 表示在时间步 t 时对动作 a 的**价值估计**。当前的目标是使 $Q_t(a)$ 尽可能接近 $q_*(a)$。

贪婪动作、探索与价值

当前时间步估计价值最高的动作被称为**贪婪动作**。由此我们可以引出利用和探索的定义。**利用**是选择贪婪动作，以最大化当前步的期望奖励。**探索**是选择非贪婪动作，以改进其价值估计。探索短期收益低，但可能发现更优动作，长期收益更高。探索与利用之间存在**冲突**，何时探索或利用，取决于估计值的精度、不确定性大小、剩余时间步数。

动作价值方法

动作价值方法指用于**估计动作价值**并**基于估计进行动作选择**的一类方法。**动作的价值的真实值是选择这个动作时的期望收益。**

估计价值

样本平均法通过计算实际收益的**平均值**来估计动作的价值：

$$Q_t(a) \doteq \frac{\text{在 } t \text{ 前选择 } a \text{ 所获奖励之和}}{\text{在 } t \text{ 前选择 } a \text{ 的次数}} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}}$$

- 其中 $\mathbb{1}_{predicate}$ 表示一个随机变量，当 *predicate* 为真时取值为1，否则为0。
- 若分母为0，定义 $Q_t(a)$ 为默认值（如0）。
- 根据大数定律，当选择次数趋于无穷时，$Q_t(a) \to q_*(a)$。

贪婪动作选择

贪婪动作选择指的是选择当前**估计价值最高**的动作：

$$A_t \doteq \underset{a}{\operatorname{argmax}}\ Q_t(a)$$

若有多个最大值，任意选择（如随机）。其始终**利用**当前知识，最大化即时奖励。存在**不探索**，可能错过更优动作的缺点。

对贪心策略的优化

一种优化方法是大多数时间选择贪婪动作；以小概率 ε，**随机均匀选择所有动作**（包括贪婪动作），与估计值无关，从而实现**探索与利用的平衡**。这个方法的优点是，随着步数趋于无穷，每个动作都会被无限次采样，从而确保所有 $Q_t(a)$ 收敛到 $q_*(a)$。这意味着选择最优动作的概率将收敛到大于 $1-\varepsilon$ 的值，即接近确定。然而，这些只是渐近性保证，对于方法在实际中的有效性说明有限。

练习2.1

> 在 ε-贪婪动作选择中，在有两个动作及 ε = 0.5 的情况下，**贪婪动作被选择的概率是多少？**

在 ε-贪婪策略中：

- 以概率 **1 - ε**，选择当前**估计价值最高的动作（即贪婪动作）** → **利用**
- 以概率 **ε**，**从所有 k 个动作中均匀随机选择一个动作** → **探索**

注意：在“探索”阶段，**即使选中了当前的贪婪动作，那也是随机选中的，不是因为它是贪婪动作。**

- 动作数量：k = 2
- ε = 0.5

贪婪动作可以在两种情况下被选中：

1. **利用阶段（概率 = 1 - ε = 0.5）** → 一定选择贪婪动作
   → 贡献概率：**0.5 × 1 = 0.5**
2. **探索阶段（概率 = ε = 0.5）** → 随机均匀选择两个动作中的一个
   → 贪婪动作被随机选中的概率 = 1/2
   → 贡献概率：**0.5 × (1/2) = 0.2

$$P(\text{选择贪婪动作}) = 0.5 + 0.25 = \boxed{0.75}$$

> 在 ε = 0.5 且有两个动作的情况下，贪婪动作被选择的概率是 **0.75**。

实验一

为评估贪婪方法与 ε-贪婪方法的性能，将使用 **样本平均法** 估计动作价值，并在 **2000 个随机生成的 10 臂老虎机问题** 上，每个问题运行 **1000 步**，最后报告：

1. **平均奖励随时间步的变化曲线**
2. **最优动作选择比例随时间步的变化曲线**

其中

- 每个动作的真实价值 $q_*(a)$ 从均值为0、方差为1的正态分布中随机生成；
- 每次选择动作后，实际奖励 R_t 从均值为 $q_*(A_t)$、方差为1的正态分布中采样。

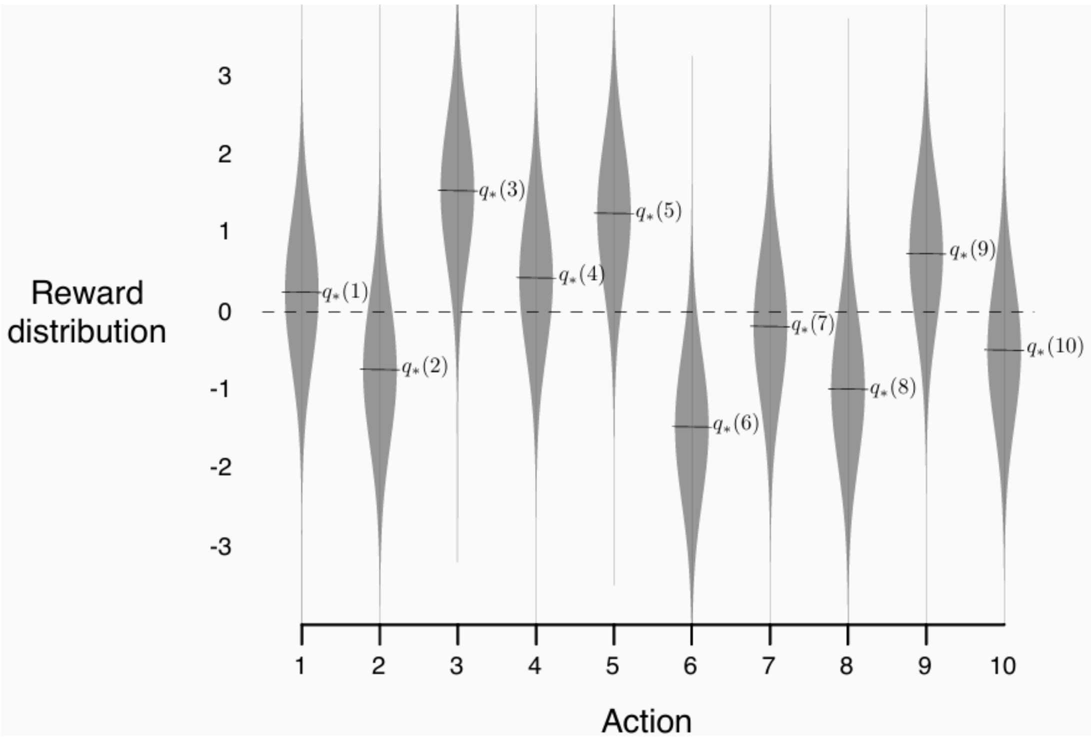


图2.1： 十个动作中每个动作的真实值 $q_*(a)$ 是根据均值为0、方差为1的正态分布选取的，随后实际奖励根据均值为 $q_*(a)$、方差为1的正态分布选取，如灰色分布所示。

```
import numpy as np
```

```
import matplotlib.pyplot as plt

k = 10                  # 臂的数量
n_steps = 1000          # 每个任务的时间步数
n_problems = 2000       # 任务数量
epsilons = [0.0, 0.01, 0.1]  # 不同的 epsilon 值

# rewards、optimal_actions 是典，键是eps，值是形状为 (2000, 1000)的NumPy数组。
rewards = {eps: np.zeros((n_problems, n_steps)) for eps in epsilons}
optimal_actions = {eps: np.zeros((n_problems, n_steps)) for eps in epsilons}

# 主循环 进行两千次实验
for problem_idx in range(n_problems):
    # 生成当前问题的真实动作价值 q*(a) ~ N(0, 1)
    true_q = np.random.randn(k)
    best_action = np.argmax(true_q)  # 最优动作

    # 对每个 epsilon 策略进行实验
    for eps in epsilons:
        # 初始化估计值 Q(a) 和选择次数 N(a)
        Q = np.zeros(k)
        N = np.zeros(k)

        for t in range(n_steps):
            # ϵ-贪婪动作选择
            if np.random.rand() < eps:
                action = np.random.randint(k)  # 探索：随机选动作
            else:
                action = np.argmax(Q)          # 利用：选当前最优

            # 记录是否选择了最优动作
            optimal_actions[eps][problem_idx, t] = 1 if action == best_action
else 0

            # 生成奖励 R ~ N(q*(a), 1)
            reward = np.random.randn() + true_q[action]
            rewards[eps][problem_idx, t] = reward

            # 增量更新 Q(a)
            N[action] += 1
            Q[action] += (reward - Q[action]) / N[action]

# 计算平均结果
avg_rewards = {eps: np.mean(rewards[eps], axis=0) for eps in epsilons}
avg_optimal = {eps: np.mean(optimal_actions[eps], axis=0) for eps in epsilons}

# 绘图
plt.figure(figsize=(12, 8))

# 上图：平均奖励
plt.subplot(2, 1, 1)
for eps in epsilons:
    label = f"ϵ = {eps}" if eps > 0 else "Greedy (ϵ = 0)"
    plt.plot(avg_rewards[eps], label=label)
plt.title("10-Armed Testbed: Average Reward over Time")
plt.xlabel("Time Step")
```

```python
plt.ylabel("Average Reward")
plt.legend()
plt.grid(True)

# 下图：最优动作选择比例
plt.subplot(2, 1, 2)
for eps in epsilons:
    label = f"ε = {eps}" if eps > 0 else "Greedy (ε = 0)"
    plt.plot(avg_optimal[eps], label=label)
plt.title("10-Armed Testbed: % Optimal Action over Time")
plt.xlabel("Time Step")
plt.ylabel("% Optimal Action")
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.savefig("10_arm_testbed_results.png", dpi=300)
plt.show()
```

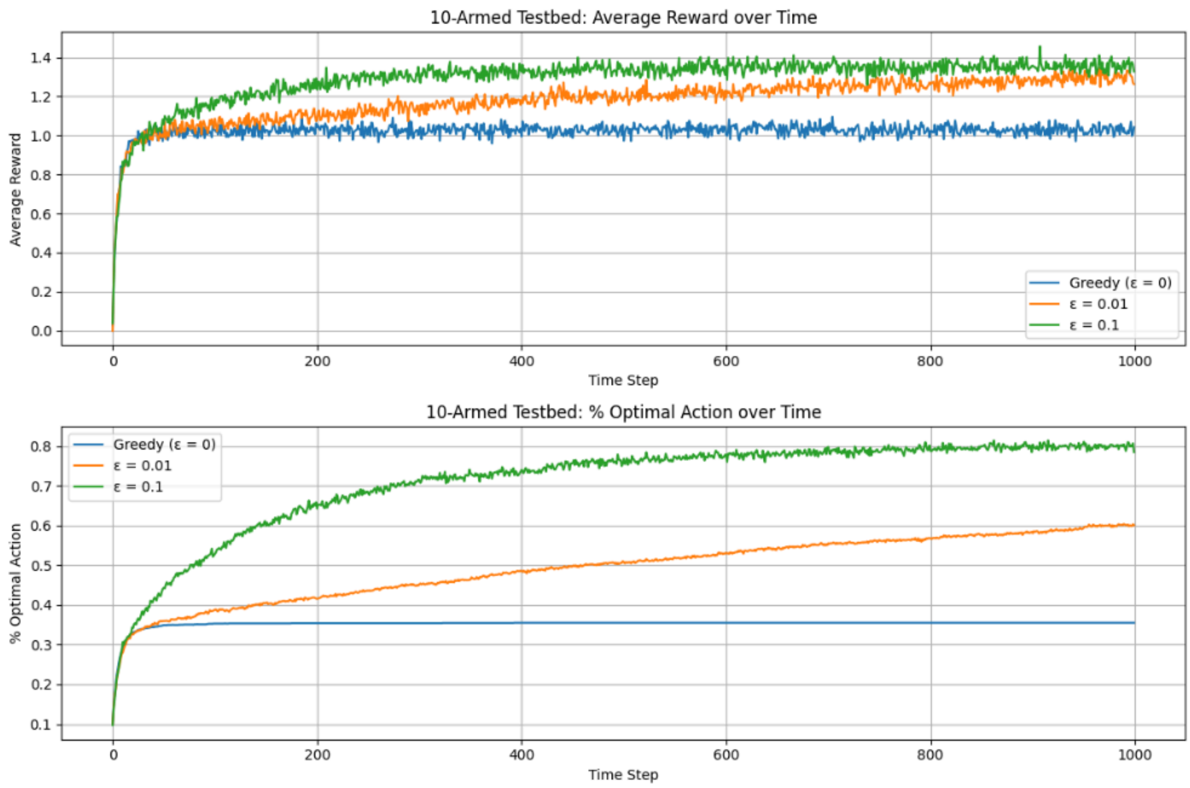


图2.2：比较了贪婪方法与两种ε-贪婪方法（$\varepsilon = 0.01$ 和 $\varepsilon = 0.1$）在10臂测试平台上的表现。所有方法均使用样本平均法来估计动作价值。上图显示了期望奖励随经验增长的情况。贪婪方法在最开始略微提升更快，但随后趋于平稳，且稳定在较低水平。其每步平均奖励仅为约1，而该测试平台上的最优可能值约为1.55。贪婪方法在长期表现较差，因为它常常陷入执行次优动作的情况。

主要结果：

- **贪婪方法**：
  - 初期提升较快，但很快陷入局部最优；
  - 在约三分之二的任务中未能识别最优动作；
  - 长期平均每步奖励约为1（远低于最优值约1.55）；

    - 因缺乏探索，常被次优动作“锁定”。
- ε**-贪婪方法**：
    - 通过持续探索，能更可靠地发现最优动作；
    - $\varepsilon = 0.1$：探索频繁，更快找到最优动作，但最多仅以91%的概率选择最优动作；
    - $\varepsilon = 0.01$：探索较少，初期进展慢，但长期性能优于 $\varepsilon = 0.1$；
    - 可通过随时间降低 ε 来兼顾探索与利用。

关键结论：

- ε-贪婪方法在长期表现上显著优于纯贪婪方法；
- 探索的重要性取决于任务特性：
    - 奖励噪声越大（方差高），越需要探索；
    - 即使在确定性环境中，若任务**非平稳**（动作价值随时间变化），探索仍必不可少；
- 在强化学习中，由于策略演化导致决策任务动态变化，**探索与利用的平衡是必要且普遍的要求。**

练习 2.2

> **题目：**
> 考虑一个 k = 4 的多臂赌博机问题，记作动作 1、2、3、4。
> 算法使用 **ε-贪婪动作选择**，基于采样平均的动作价值估计，初始估计 $Q_1(a) = 0, \forall a$。
> 假设动作及收益的最初顺序是：
> $A_1 = 1, R_1 = -1$
> $A_2 = 2, R_2 = 1$
> $A_3 = 2, R_3 = -2$
> $A_4 = 2, R_4 = 2$
> $A_5 = 3, R_5 = 0$
>
> 问：在哪些时刻中，**肯定发生了随机探索（即 ε 情形）**？在哪些时刻中，**可能发生**？

- 初始时，所有 $Q_1(a) = 0$，所以第 1 步时所有动作价值相等 → **任意动作都是“贪婪动作”。**
- ε-贪婪策略：以概率 $1 - \varepsilon$ 选择当前估计最优动作（若有多个，可任选其一）；以概率 ε 随机选动作。
- 关键：**只有当所选动作不是当前估计最优动作时，才“肯定”是探索。**
  如果所选动作是当前最优动作，则可能是利用（大概率），也可能是探索时“碰巧”选中了最优动作（小概率）。

我们用采样平均法更新 $Q_t(a)$：

时间步 t=1：

- 选 $A_1 = 1$，得 $R_1 = -1$
- 所有 Q 初始为 0 → 任意动作都是贪婪动作 → 选动作 1 **可以是利用（任意选一个最优），也可以是探索**
- **结论：可能发生了探索（不能肯定）**

更新后：

- $Q_2(1) = -1$，其余仍为 0

时间步 t=2：

- 当前估计：
  - Q(1) = -1
  - Q(2)=Q(3)=Q(4)=0 → 最优动作是 {2,3,4}（任选其一）
- 实际选 $A_2 = 2$ → 是当前最优动作之一
- **结论：可能是利用（选了最优），也可能是探索时碰巧选中 → 不能肯定发生了探索**

更新后：

- $Q_3(2) = 1$（第一次选动作 2，直接赋值）
- Q(1) = -1，Q(3)=Q(4)=0

时间步 t=3：

- 当前估计：
  - Q(1) = -1
  - Q(2) = 1 ← 最优！
  - Q(3)=Q(4)=0
- 实际选 $A_3 = 2$ → 是当前唯一最优动作
- 所以：**如果是利用，一定选 2；如果是探索，有 1/4 概率选 2**
- **结论：不能肯定发生了探索（可能是利用，也可能是探索碰巧）**

更新后（动作 2 第二次被选）：

- $Q_4(2) = \frac{1+(-2)}{2} = -0.5$

时间步 t=4：

- 当前估计：
  - Q(1) = -1
  - Q(2) = -0.5
  - Q(3)=Q(4)=0 → 最优动作是 {3,4}
- 实际选 $A_4 = 2$ → **不是最优动作**
- **关键点：只有探索阶段才会选择非贪婪动作**
- **结论：肯定发生了探索（ε情形）**

更新后：

- 动作 2 第三次被选：$Q_5(2) = \frac{1+(-2)+2}{3} = \frac{1}{3} \approx 0.333$

时间步 t=5：

- 当前估计：
  - Q(1) = -1
  - Q(2) ≈ 0.333
  - Q(3)=Q(4)=0 → 最优动作是 {2}（唯一最大）
- 实际选 $A_5 = 3$ → **不是最优动作**
- **结论：肯定发生了探索**

> **肯定发生了探索（ε情形）的时刻：**
> **t = 4 和 t = 5**

> **可能发生了探索的时刻：**
> **t = 1, 2, 3**

增量实现

在强化学习中，动作价值方法通过估计每个动作的期望奖励来指导决策。最直接的方式是将动作价值 $Q(a)$ 估计为该动作所获得奖励的**样本平均值**。但若每次更新都重新计算所有历史奖励的平均值，会导致：**内存消耗随时间增长**和**每步计算量逐渐增加**因此引入**增量更新**。

设某动作被选择 $n-1$ 次后，其价值估计为：

$$Q_n = \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}$$

当第 n 次选择该动作并获得奖励 R_n 后，新的估计应为：

$$Q_{n+1} = \frac{1}{n}\sum_{i=1}^{n} R_i$$

将其改写为仅依赖旧估计 Q_n 和新奖励 R_n 的形式：

$$\begin{aligned}
Q_{n+1} &= \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\left(R_n + (n-1)\cdot\frac{1}{n-1}\sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}(R_n + (n-1)Q_n) \\
&= \frac{1}{n}(R_n + nQ_n - Q_n) \\
&= Q_n + \frac{1}{n}(R_n - Q_n)
\end{aligned}$$

$$\boxed{Q_{n+1} = Q_n + \frac{1}{n}[R_n - Q_n]} \tag{2.3}$$

即使 $n=1$，公式也成立（此时 $Q_2 = R_1$，无论初始 Q_1 是什么）。

通用更新形式

公式 (2.3) 属于一个更广泛的学习模式：

$$\boxed{\text{新估计值} \leftarrow \text{旧估计值} + \alpha \times [\text{目标} - \text{旧估计值}]} \tag{2.4}$$

其中：

- $[\text{目标} - \text{旧估计值}]$：称为**误差项**（error），表示当前估计与新信息之间的差距

- α：**步长参数（step-size）**，控制更新幅度。通常记作 α 或 $\alpha_t(a)$，表示在时间 t 对动作 a 使用的步长

在样本平均法中，$\alpha = \frac{1}{n}$，即随着观测次数增加，每次更新的影响逐渐减小。

多臂老虎机伪代码

> 使用**增量样本平均法** + ε**-贪婪策略**

```
初始化：
  对每个动作 a ∈ {1, 2, ..., k}：
    Q(a) ← 0           // 动作价值估计
    N(a) ← 0           // 动作 a 被选择的次数

重复执行（每一步）：
  以概率 1-ε：
      A ← argmax_{a} Q(a)     // 选择估计价值最高的动作（打破平局随机）
  以概率 ε：
      A ← 随机选择一个动作   // 从所有动作中均匀随机选择
  R ← bandit(A)              // 执行动作 A，获得奖励
  N(A) ← N(A) + 1            // 更新动作 A 的选择次数
  Q(A) ← Q(A) + (1 / N(A)) × [R − Q(A)]    // 增量更新价值估计
```

非平稳问题

指数近期加权平均

在实际强化学习任务中，大多数问题是**非平稳的**，因此需要能适应变化的估值方法。

类型	**定义**	**特点**
平稳问题（Stationary）	动作的真实价值 $q_*(a)$ 不随时间变化	可使用**样本平均法**（sample average）有效估计
非平稳问题（Nonstationary）	动作的真实价值 $q_*(a)$ 随时间变化	样本平均法效果差，需更关注**近期奖励**

> **Note**
>
> 在样本平均法中，$\alpha = \frac{1}{n}$，即随着观测次数增加，每次更新的影响逐渐减小。
>
> $$\boxed{Q_{n+1} = Q_n + \frac{1}{n}[R_n - Q_n]} \tag{2.3}$$

为了更好地应对非平稳环境，给予近期奖励比远期奖励更高的权重，引入**固定步长参数** $\alpha \in (0,1]$ 的更新方式：

$$Q_{n+1} = Q_n + \alpha\left[R_n - Q_n\right] \tag{2.5}$$

这相当于对历史奖励进行**加权平均**，但更重视最近的奖励，就是**指数近期加权平均**

$$
\begin{aligned}
Q_{n+1} &= Q_n + \alpha\Big[R_n - Q_n\Big] \\
&= \alpha R_n + (1-\alpha)Q_n \\
&= \alpha R_n + (1-\alpha)[\alpha R_{n-1} + (1-\alpha)Q_{n-1}] \\
&= \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2 Q_{n-1} \\
&= \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2\alpha R_{n-2} + \cdots \\
&\quad + (1-\alpha)^{n-1}\alpha R_1 + (1-\alpha)^n Q_1 \\
&= (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i.
\end{aligned} \tag{2.6}
$$

- R_i：第 i 次获得的奖励
- 权重为：$w_i = \alpha(1-\alpha)^{n-i}$
- 初始估计 Q_1 的权重为：$(1-\alpha)^n$

权重的性质

- 所有权重之和为 1：

$$
(1-\alpha)^n + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} = 1
$$

> ⓘ **Note**
>
> 总权重：
>
> $$\text{Total Weight} = (1-\alpha)^n + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i}$$
>
> 先处理求和部分：
>
> 令 $k = n - i$，当 $i = 1$ 时，$k = n - 1$；当 $i = n$ 时，$k = 0$
>
> 所以：
>
> $$\sum_{i=1}^{n} (1-\alpha)^{n-i} = \sum_{k=0}^{n-1} (1-\alpha)^k$$
>
> 回忆等比数列求和公式：
>
> $$\sum_{k=0}^{m} r^k = \frac{1-r^{m+1}}{1-r}, \quad \text{for } r \neq 1$$
>
> 这里 $r = 1 - \alpha \in [0, 1)$（因为 $\alpha \in (0, 1]$），所以可以使用公式：
>
> $$\sum_{k=0}^{n-1} (1-\alpha)^k = \frac{1-(1-\alpha)^n}{1-(1-\alpha)} = \frac{1-(1-\alpha)^n}{\alpha}$$
>
> 现在乘上前面的 α：
>
> $$\sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} = \alpha \cdot \sum_{k=0}^{n-1} (1-\alpha)^k = \alpha \cdot \frac{1-(1-\alpha)^n}{\alpha} = 1 - (1-\alpha)^n$$
>
> 代入原式：
>
> $$(1-\alpha)^n + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} = (1-\alpha)^n + [1 - (1-\alpha)^n] = 1$$

> 得证

- 权重随时间呈**指数衰减**
- 越早的奖励（$i \ll n$），权重越小
- 近期奖励影响更大 → 更适合**动态变化环境**

变步长参数和收敛

令 $\alpha_n(a)$ 表示在第 n 次选择动作 a 后处理所获奖励时使用的步长参数。

要保证估计值以概率 1 **收敛到真实值**，需满足：

$$\sum_{n=1}^{\infty} \alpha_n(a) = \infty \quad \text{且} \quad \sum_{n=1}^{\infty} \alpha_n^2(a) < \infty \tag{2.7}$$

条件	含义
$\sum \alpha_n = \infty$	保证步长足够大，步长总和发散 → 能克服初始偏差或噪声
$\sum \alpha_n^2 < \infty$	平方和收敛 → 步长最终足够小，防止震荡

练习 2.4

当步长参数 α_n **不是常数**时，估计值 Q_n 是对之前所有收益 R_i 的加权平均。要求给出每个历史收益 R_i 在 Q_n 中的权重公式，并推广标准形式（即常数步长）的情况。

已知：

$$Q_{k+1} = Q_k + \alpha_k(R_k - Q_k)$$

$$Q_1 = Q_1 \quad (\text{初始估计})$$

$$Q_2 = Q_1 + \alpha_1(R_1 - Q_1) = (1 - \alpha_1)Q_1 + \alpha_1 R_1$$

$$Q_3 = Q_2 + \alpha_2(R_2 - Q_2) = (1 - \alpha_2)Q_2 + \alpha_2 R_2 = (1 - \alpha_2)\left[(1 - \alpha_1)Q_1 + \alpha_1 R_1\right] + \alpha_2 R_2$$

$$Q_3 = (1 - \alpha_1)(1 - \alpha_2)Q_1 + \alpha_1(1 - \alpha_2)R_1 + \alpha_2 R_2$$

可以看出：

- R_1 的系数是：$\alpha_1(1 - \alpha_2)$
- R_2 的系数是：α_2

使用数学归纳法证明以下通项公式成立：

$$Q_n = \left(\prod_{j=1}^{n-1}(1 - \alpha_j)\right) Q_1 + \sum_{i=1}^{n-1}\left(\alpha_i \prod_{j=i+1}^{n-1}(1 - \alpha_j)\right) R_i$$

n = 2时有

$$Q_2 = (1 - \alpha_1)Q_1 + \alpha_1 R_1$$

代入公式右边：

$$\prod_{j=1}^{1}(1-\alpha_j)=1-\alpha_1,\quad \sum_{i=1}^{1}\alpha_i\prod_{j=i+1}^{1}(1-\alpha_j)=\alpha_1\cdot 1=\alpha_1$$

所以：

$$Q_2=(1-\alpha_1)Q_1+\alpha_1 R_1\text{成立。}$$

假设对于某个 $k\geq 2$，有：

$$Q_k=\left(\prod_{j=1}^{k-1}(1-\alpha_j)\right)Q_1+\sum_{i=1}^{k-1}\left(\alpha_i\prod_{j=i+1}^{k-1}(1-\alpha_j)\right)R_i$$

由递推关系：

$$Q_{k+1}=Q_k+\alpha_k(R_k-Q_k)=(1-\alpha_k)Q_k+\alpha_k R_k$$

将归纳假设代入：

$$Q_{k+1}=(1-\alpha_k)\left[\left(\prod_{j=1}^{k-1}(1-\alpha_j)\right)Q_1+\sum_{i=1}^{k-1}\left(\alpha_i\prod_{j=i+1}^{k-1}(1-\alpha_j)\right)R_i\right]+\alpha_k R_k$$

分别处理两个部分：

$$(1-\alpha_k)\prod_{j=1}^{k-1}(1-\alpha_j)=\prod_{j=1}^{k}(1-\alpha_j)$$

$$(1-\alpha_k)\sum_{i=1}^{k-1}\left(\alpha_i\prod_{j=i+1}^{k-1}(1-\alpha_j)\right)R_i=\sum_{i=1}^{k-1}\left(\alpha_i\prod_{j=i+1}^{k}(1-\alpha_j)\right)R_i$$

再加上 $\alpha_k R_k$ 这一项：

$$Q_{k+1}=\left(\prod_{j=1}^{k}(1-\alpha_j)\right)Q_1+\sum_{i=1}^{k}\left(\alpha_i\prod_{j=i+1}^{k}(1-\alpha_j)\right)R_i$$

因此，对于任意非恒定步长序列 $\{\alpha_n\}$，收益 R_i 在 Q_n 中的权重为：

$$w_i^{(n)}=\alpha_i\cdot\prod_{j=i+1}^{n-1}(1-\alpha_j),\quad i=1,2,\ldots,n-1$$

特别地，当所有 $\alpha_j=\alpha$（常数），则：

$$w_i^{(n)}=\alpha(1-\alpha)^{n-1-i}$$

这正是公式 (2.6) 的形式。

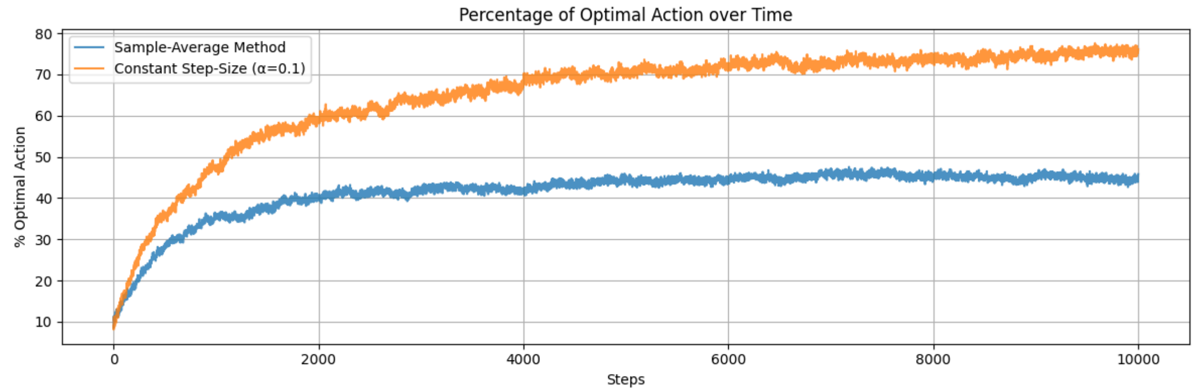
练习 2.5

设计并且实施一项实验来证实**采用采样平均方法去解决非平稳问题的困难**。使用一个 10 臂测试平台的修改版本，其中所有的 $q_*(a)$ 初始时相等，然后进行随机游走（例如，在每一步所有 $q_*(a)$ 都加上一个均值为 0、标准差为 0.01 的正态分布增量）。为其中一个方法使用采样平均（增量式计算）的动作-价值估计，为另一个方法使用常数步长参数（$\alpha=0.1$）的动作-价值估计。绘制如图 2.2 所示的分析曲线（平

均奖励与最优动作选择百分比随时间步的变化）。采用 $\varepsilon = 0.1$ 的 ε-贪婪策略，并运行足够长的时间（如 10,000 步）。

```
import numpy as np
import matplotlib.pyplot as plt

# 参数设置
k = 10                    # 臂数
epsilon = 0.1             # ε-greedy 探索率
alpha = 0.1               # 常数步长
steps = 10000             # 每次实验运行步数
runs = 2000               # 独立实验次数
q_star_std = 0.01         # 真值随机游走的标准差

# 初始化记录数组
rewards_sample_avg = np.zeros(steps)
optimal_actions_sample_avg = np.zeros(steps)

rewards_const_alpha = np.zeros(steps)
optimal_actions_const_alpha = np.zeros(steps)

# 开始实验
for run in range(runs):
    if run % 100 == 0:
        print(f"Run {run}/{runs}")

    # 初始化真值
    q_star = np.zeros(k)

    # 方法1：采样平均
    Q1 = np.zeros(k)
    N1 = np.zeros(k, dtype=int)

    # 方法2：常数步长
    Q2 = np.zeros(k)

    for t in range(steps):
        # 真值随机游走
        q_star += np.random.normal(0, q_star_std, k)

        # 找当前最优动作
        optimal_action = np.argmax(q_star)

        # 采样平均
        if np.random.rand() < epsilon:
            action1 = np.random.randint(k)
        else:
            action1 = np.argmax(Q1)

        # 获取奖励
        reward1 = np.random.normal(q_star[action1], 1)
        # 更新计数和估计
        N1[action1] += 1
        Q1[action1] += (1 / N1[action1]) * (reward1 - Q1[action1])
```

```python
        # 记录
        rewards_sample_avg[t] += reward1
        if action1 == optimal_action:
            optimal_actions_sample_avg[t] += 1

        # 常数步长 α=0.1
        if np.random.rand() < epsilon:
            action2 = np.random.randint(k)
        else:
            action2 = np.argmax(Q2)

        # 获取奖励
        reward2 = np.random.normal(q_star[action2], 1)
        # 更新估计
        Q2[action2] += alpha * (reward2 - Q2[action2])

        # 记录
        rewards_const_alpha[t] += reward2
        if action2 == optimal_action:
            optimal_actions_const_alpha[t] += 1

# 计算平均值
rewards_sample_avg /= runs
optimal_actions_sample_avg /= runs
optimal_actions_sample_avg *= 100  # 转换为百分比

rewards_const_alpha /= runs
optimal_actions_const_alpha /= runs
optimal_actions_const_alpha *= 100

# 绘图
fig, ax = plt.subplots(2, 1, figsize=(12, 8))

# 平均奖励
ax[0].plot(rewards_sample_avg, label='Sample-Average Method', alpha=0.8)
ax[0].plot(rewards_const_alpha, label='Constant Step-Size (α=0.1)', alpha=0.8)
ax[0].set_xlabel('Steps')
ax[0].set_ylabel('Average Reward')
ax[0].legend()
ax[0].set_title('Average Reward over Time')
ax[0].grid(True)

# 最优动作百分比
ax[1].plot(optimal_actions_sample_avg, label='Sample-Average Method', alpha=0.8)
ax[1].plot(optimal_actions_const_alpha, label='Constant Step-Size (α=0.1)',
alpha=0.8)
ax[1].set_xlabel('Steps')
ax[1].set_ylabel('% Optimal Action')
ax[1].legend()
ax[1].set_title('Percentage of Optimal Action over Time')
ax[1].grid(True)

plt.tight_layout()
plt.show()
```



初始值

在强化学习中，所有动作价值估计方法都依赖于**初始估计值** $Q_1(a)$。这些初始值会影响学习过程，甚至引入**偏差**（bias）。

方法	是否存在偏差	偏差是否消失
样本平均法	有（初期）	一旦每个动作至少选一次，偏差消失
固定步长法	有	偏差**永久存在**，但随时间指数衰减

对于**样本平均法**有，

$$Q_{n+1} = Q_n + \frac{1}{n}(R_n - Q_n)$$

假设你从 Q_1 开始，然后：

- $Q_2 = Q_1 + \frac{1}{1}(R_1 - Q_1) = R_1$
- $Q_3 = Q_2 + \frac{1}{2}(R_2 - Q_2) = \frac{R_1+R_2}{2}$
- $Q_4 = Q_3 + \frac{1}{3}(R_3 - Q_3) = \frac{R_1+R_2+R_3}{3}$

当一个动作**至少被选择一次**，它的估计值 $Q(a)$ 就会从 $Q_1(a)$更新为基于真实奖励的平均值。 如果某个动作**从未被选过**，它的估计值就**永远停留在初始值** $Q_1(a)$，也就是偏差**永远不会消失。只要每个动作至少被选择一次，之后的估计就完全基于实际收到的奖励，不再受初始值影响。**

对于**固定步长法**，

$$Q_{n+1} = Q_n + \alpha(R_n - Q_n)$$

我们之前推导过它的闭式解（公式 2.6）：

$$Q_{n+1} = (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i \tag{2.6}$$

固定步长法对初始值有永久性偏差，只是随时间指数衰减。

优缺点

初始值的**缺点**在于**增加可调参数**，用户必须设定 $Q_1(a)$，即使设为 0 也是一种选择，不恰当的初始值可能误导早期决策。优点是若对奖励水平有先验知识，合理初始化可加速收敛，更重要的是：**可用于鼓励探索。**

乐观初始值

乐观初始值的核心思想将所有动作的初始价值估计设为一个**远高于真实期望奖励的值**（例如 $Q_1(a) = +5$），从而**激励算法去尝试所有动作。乐观初始值通过"高估所有动作的价值"，让智能体在尝试每个动作后都感到"失望"，从而驱使它不断尝试其他还没充分尝试的动作——即使它始终执行的是"贪婪"策略。**

> ⓘ **Note**
>
> 假设只使用贪婪策略，总是选择当前估计价值最高的动作，一旦某个动作获得一次正奖励，就可能一直被选，导致**缺乏探索**。但使用乐观初始值，假设将把所有动作的初始估计设为：
>
> $$Q_1(a) = +5 \quad \text{（远高于任何可能的真实奖励）}$$
>
> 假设仍然使用**贪婪策略**（$\varepsilon = 0$），即每次都选当前 $Q(a)$ 最高的动作。初始时，所有动作价值相同，系统会**随机选择一个动作**（比如 a_3）。执行 a_3，获得真实奖励 R，对a_3进行更新后，由于初始值远高于真是奖励，所以$Q(a_3)$会下降。此时则会选择其余的动作继续执行(因为初始化都是5)。

- 真实奖励 R_t 通常低于初始估计（"失望"）
- 算法不断更新 $Q(a)$ 向下调整
- 由于所有动作初始值相同，任何未充分探索的动作仍具有较高估计值
- 即使使用**贪婪策略**（$\varepsilon = 0$），也会持续探索，直到价值收敛

实验二

验证 **乐观初始值** 如何在贪心策略（ε=0）下自动驱动探索，从而在平稳环境中获得比标准 ε-greedy 方法更优的长期性能。

在标准 ε-greedy 方法中，我们通过随机选择动作（概率 ε）来探索。**乐观初始值方法**则将所有动作的初始估计值 $Q_1(a)$ 设置为一个**远高于真实期望值**的值（如 +5），而真实值 $q_*(a) \sim \mathcal{N}(0,1)$。由于初始估计"过于乐观"，无论选择哪个动作，实际奖励都会"令人失望"，从而促使算法尝试其他动作，**即使使用纯贪心策略（ε=0）也能自动探索。**该方法在**平稳环境**中效果显著，但在**非平稳环境**中无效（因为探索动力是一次性的）。

- **10 臂赌博机问题（k=10）**
- 真实动作价值：$q_*(a) \sim \mathcal{N}(0,1)$，**固定不变**（平稳环境）
- 奖励分布：$R_t \sim \mathcal{N}(q_*(A_t), 1)$
- 比较两种方法：
  1. **乐观初始值 + 贪心策略**：$Q_1(a) = +5, \varepsilon = 0$
  2. **零初始值 + ε-greedy 策略**：$Q_1(a) = 0, \varepsilon = 0.1$
- 使用 **常数步长法** 更新动作价值：

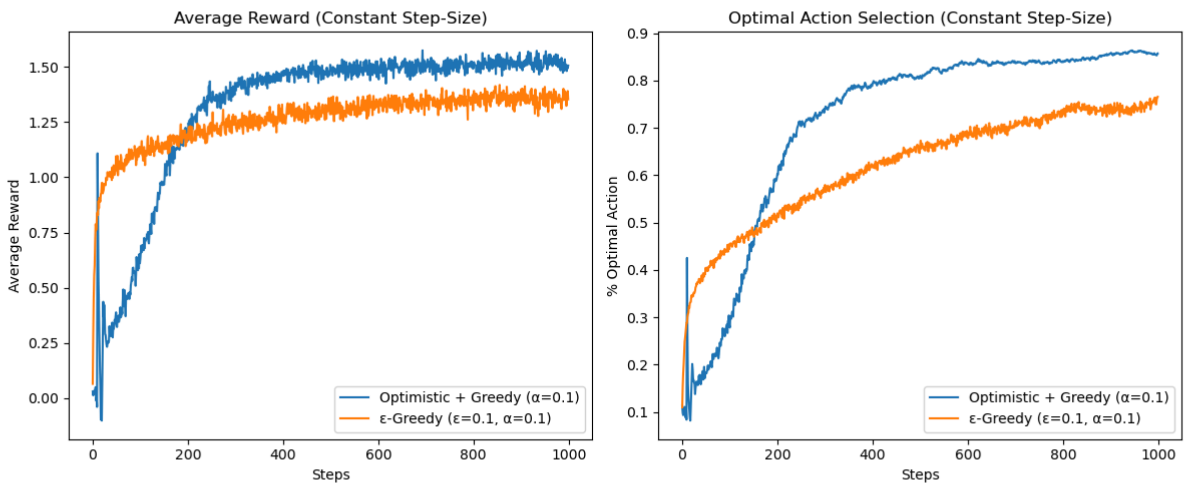
$$Q_{n+1} = Q_n + \alpha(R_n - Q_n)$$

- 运行步数：1000步
- 独立重复实验次数：2000次（取平均以减少噪声）
- 输出曲线：
  - 平均奖励随时间变化
  - 最优动作选择百分比随时间变化

```
import numpy as np
import matplotlib.pyplot as plt

# 多臂赌博机类
class Bandit:
    def __init__(self, k=10, mean=0, std=1):
        self.k = k
        self.arm_means = np.random.normal(mean, std, k)  # 每个臂的真实收益均值
        self.optimal_action = np.argmax(self.arm_means)

    def pull(self, a):
        # 拉动第 a 个臂，返回收益
        return np.random.normal(self.arm_means[a], 1)

# 乐观初始值 + 贪心策略（常数步长）
def optimistic_greedy(bandit, steps=1000, initial_value=5.0, alpha=0.1):
    Q = np.full(bandit.k, initial_value)  # 乐观初始值
    rewards = []
    optimal_action_selected = []

    for t in range(steps):
        a = np.argmax(Q)  # 贪心动作选择
        reward = bandit.pull(a)
        # 常数步长更新
        Q[a] += alpha * (reward - Q[a])

        rewards.append(reward)
        optimal_action_selected.append(int(a == bandit.optimal_action))

    return rewards, optimal_action_selected

# ε-贪心策略（常数步长）
def epsilon_greedy(bandit, steps=1000, epsilon=0.1, alpha=0.1):
    Q = np.zeros(bandit.k)  # 零初始值
    rewards = []
    optimal_action_selected = []

    for t in range(steps):
        if np.random.rand() < epsilon:
            a = np.random.choice(bandit.k)  # 随机探索
        else:
            a = np.argmax(Q)  # 贪心动作选择

        reward = bandit.pull(a)
        # 常数步长更新
        Q[a] += alpha * (reward - Q[a])
```

```python
        rewards.append(reward)
        optimal_action_selected.append(int(a == bandit.optimal_action))

    return rewards, optimal_action_selected

# 主实验函数
def run_experiment(runs=2000, steps=1000):
    rewards_opt = np.zeros(steps)
    opt_action_opt = np.zeros(steps)
    rewards_eps = np.zeros(steps)
    opt_action_eps = np.zeros(steps)

    print("运行实验...")
    for i in range(runs):
        if i % 500 == 0:
            print(f"运行 {i}/{runs}")
        bandit = Bandit()

        r1, o1 = optimistic_greedy(bandit, steps, initial_value=5.0, alpha=0.1)
        r2, o2 = epsilon_greedy(bandit, steps, epsilon=0.1, alpha=0.1)

        rewards_opt += r1
        opt_action_opt += o1
        rewards_eps += r2
        opt_action_eps += o2

    rewards_opt /= runs
    opt_action_opt /= runs
    rewards_eps /= runs
    opt_action_eps /= runs

    # 绘图
    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
    plt.plot(rewards_opt, label='乐观初始值 + 贪心')
    plt.plot(rewards_eps, label='ε-贪心 (ε=0.1)')
    plt.xlabel('Steps')
    plt.ylabel('Average Reward')
    plt.legend()
    plt.title('Average Reward over Time')

    plt.subplot(1, 2, 2)
    plt.plot(opt_action_opt, label='乐观初始值 + 贪心')
    plt.plot(opt_action_eps, label='ε-贪心 (ε=0.1)')
    plt.xlabel('Steps')
    plt.ylabel('% Optimal Action')
    plt.legend()
    plt.title('Optimal Action Selection over Time')

    plt.tight_layout()
    plt.show()

# 运行实验
if __name__ == "__main__":
```

```python
    np.random.seed(42)  # 为了可重现性
    run_experiment(runs=2000, steps=1000)
```



方法	初始值	探索方式	表现特点
贪婪 + 乐观初始化	$Q_1(a) = +5$	内生探索（失望驱动）	初期差（探索多），后期好（探索充分）
ε-greedy	$Q_1(a) = 0$	外部随机探索（$\varepsilon = 0.1$）	初期较好，后期持续随机扰动

结论：在**平稳问题**中，乐观初始化能在不依赖外部噪声的情况下实现高效探索，长期性能更优。

乐观初始值不适用于**非平稳问题**，一旦价值估计收敛，探索停止；若环境变化，无法重新激发探索。

练习2.6

上图显示，使用**乐观初始值**（如将所有动作的初始价值设为 +5）并配合**贪心策略**（ε=0）的方法，在学习曲线的早期（大约第 10 步左右）出现了**一个明显的峰值**（performance peak），即平均回报突然升高，随后略有下降。为什么乐观初始化方法在曲线的早期会出现振荡和峰值呢? 是什么使得这种方法在特定的早期步骤中表现得特别好或更糟?

在乐观初始值（$Q_0(a) = +5$）与完全贪心策略（$\varepsilon = 0$）的组合下，智能体并非“被迫”探索，而是被**高估的未尝试动作所吸引**，在前若干步中**系统性地轮流尝试所有动作**——因为任何未被选择的动作都保持 $Q(a) = +5$，而一旦被选，其价值估计会因实际回报远低于 +5 而逐步下降（使用固定步长 $\alpha = 0.1$）。因此，贪心策略自然倾向于优先选择尚未尝试的动作。

这一机制导致：

- 在前 10 步左右，绝大多数动作都被至少尝试一次；
- 由于奖励具有随机性（$\mathcal{N}(q_*(a), 1)$），某些真实价值一般但早期获得较高采样回报的动作，其 $Q(a)$ 下降较慢，甚至短暂上升；
- 当探索接近完成时（约第 10 步），这些“看起来好”的动作可能成为当前估计最优者，被贪心策略集中选择；
- 导致第 10 ~ 12 步的平均回报突然升高，形成“峰值”。

随后：

- 所有动作的 $Q(a)$ 继续向真实值收敛（缓慢下降）；

- 真正最优的动作逐渐显现，但部分任务中可能因早期噪声锁定在次优动作上；
- 平均回报略低于峰值阶段，曲线轻微回落并趋于稳定。

> **因此，这一"神秘峰值"本质上是以下两个因素共同作用的结果：**
>
> 1. **由乐观初始化驱动的系统性探索**（systematic exploration）；
> 2. **早期高方差奖励导致的暂时性价值高估**（transient overestimation due to reward noise）；
>
> 它是一个**过渡性现象**：出现在"探索结束、利用开始"的短暂窗口期，反映的是**群体平均行为中的统计性假象**，而非长期性能提升。

练习2.7

需证明：在步长设定为 $\beta_n = \alpha/\bar{o}_n$，其中 $\bar{o}_n$ 由递推关系

$$\bar{o}_n = \bar{o}_{n-1} + \alpha(1 - \bar{o}_{n-1}), \quad \bar{o}_0 = 0$$

定义时，动作价值估计 Q_n 是一个对初始值 Q_0 无偏的**指数近因加权平均。**

首先，求解 $\bar{o}_n$ 的显式表达式。该递推关系为线性一阶差分方程：

$$\bar{o}_n = (1-\alpha)\bar{o}_{n-1} + \alpha, \quad \bar{o}_0 = 0.$$

其通解为：

$$\bar{o}_n = 1 - (1-\alpha)^n.$$

验证：

- 当 $n = 0$，$\bar{o}_0 = 1 - (1-\alpha)^0 = 1 - 1 = 0$，成立。
- 假设对 $n = k$ 成立，则

  $$\bar{o}_{k+1} = (1-\alpha)\bar{o}_k + \alpha = (1-\alpha)(1-(1-\alpha)^k) + \alpha = 1 - (1-\alpha)^{k+1},$$

  成立。
  故由归纳法，$\bar{o}_n = 1 - (1-\alpha)^n$ 对所有 $n \geq 0$ 成立。

因此，步长为：

$$\beta_n = \frac{\alpha}{\bar{o}_n} = \frac{\alpha}{1-(1-\alpha)^n}.$$

更新规则为：

$$Q_n = Q_{n-1} + \beta_n(R_n - Q_{n-1}) = (1-\beta_n)Q_{n-1} + \beta_n R_n.$$

用数学归纳法证明：对任意 $n \geq 1$，Q_n 可表示为历史收益 $R_1, R_2, \ldots, R_n$ 的加权平均：

$$Q_n = \sum_{i=1}^{n} w_{n,i} R_i,$$

且满足 $\sum_{i=1}^{n} w_{n,i} = 1$，即权重归一化，从而 Q_n 不依赖初始值 Q_0，是**无偏估计。**

> ⓘ **Note**
>
> 在统计学中，“无偏估计”通常指：**估计量的期望等于真实参数**，即 $\mathbb{E}[\hat{\theta}] = \theta$。
>
> 但在本题语境中，“对初始值无偏”（bias-free with respect to initial value）并不是指估计动作真实价值 $q_*(a)$ 是无偏的，而是指：
>
> > - Q_n **的取值不依赖于人为设定的初始值** Q_0，即估计结果不会因为 Q_0 的选择而系统性地偏高或偏低。
>
> 这被称为 **“初始值偏差的消除”**（removal of initialization bias），而不是传统统计意义上的“无偏估计真实值”。

基础步骤（$n=1$）：

$$\bar{o}_1 = 1-(1-\alpha)^1 = \alpha, \quad \beta_1 = \frac{\alpha}{\alpha} = 1,$$

$$Q_1 = Q_0 + 1\cdot(R_1 - Q_0) = R_1.$$

故 $w_{1,1}=1$，$\sum w_{1,i}=1$，且 Q_1 与 Q_0 无关。

归纳步骤：
假设对 $n-1$，有

$$Q_{n-1} = \sum_{i=1}^{n-1} w_{n-1,i} R_i, \quad \sum_{i=1}^{n-1} w_{n-1,i} = 1.$$

则

$$Q_n = (1-\beta_n)Q_{n-1} + \beta_n R_n = (1-\beta_n)\sum_{i=1}^{n-1} w_{n-1,i} R_i + \beta_n R_n.$$

定义权重：

- 对 $i = 1,2,\ldots,n-1$：$w_{n,i} = (1-\beta_n)w_{n-1,i}$，
- 对 $i=n$：$w_{n,n} = \beta_n$。

则

$$\sum_{i=1}^{n} w_{n,i} = (1-\beta_n)\sum_{i=1}^{n-1} w_{n-1,i} + \beta_n = (1-\beta_n)\cdot 1 + \beta_n = 1.$$

故对任意 n，权重和恒为 1，且 Q_n 完全由 R_1 至 R_n 线性组合构成，不包含 Q_0，因此对初始值无偏。

进一步，说明其为“指数近因加权平均”。
由于 $\bar{o}_n = 1-(1-\alpha)^n$，当 n 增大时，$\bar{o}_n \to 1$，故 $\beta_n \to \alpha$，更新渐近等价于恒定步长 α，具有指数遗忘特性 —— 近期奖励权重更高，历史奖励权重呈指数衰减。

同时，由于每一步都进行归一化（权重和为1），避免了恒定步长方法中因 $(1-\alpha)^n Q_0$ 项导致的初始值偏差。

综上，采用步长 $\beta_n = \alpha/\bar{o}_n$，其中 $\bar{o}_n = 1-(1-\alpha)^n$，可使 Q_n 成为对初始值无偏的指数近因加权平均。

置信上界动作选择

在多臂赌博机问题中，我们对每个动作的真实价值 $q_*(a)$ 并不完全了解，只能通过有限的奖励样本进行估计（如 $Q_t(a)$）。这种**估计存在不确定性。探索的本质就是应对不确定性。**贪婪策略只利用当前最优估计，容易陷入局部最优。ε-贪婪策略进行随机探索，但**缺乏智能性**：它不区分"接近最优但不确定"的动作和"明显较差"的动作。理想的探索策略应**优先尝试那些"可能比当前最优更好的"动作**，尤其是那些**估计不准但潜力大**的动作。

通过按照以下规则选择动作来实现：

在时间步 t，选择动作：

$$A_t \doteq \underset{a}{\operatorname{argmax}} \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right] \tag{2.10}$$

符号说明：

符号	含义
$Q_t(a)$	动作 a 在时刻 t 的价值估计
$N_t(a)$	到时间 t 为止，动作 a 被选择的次数
$\ln t$	时间步 t 的自然对数（$\log_e t$）
$c > 0$	探索系数（控制探索强度）

如果某个动作 a **尚未被选择过**（即 $N_t(a) = 0$），则该动作被视为一个可最大化项（直接参与 argmax 运算），以确保所有动作**至少被尝试一次。**

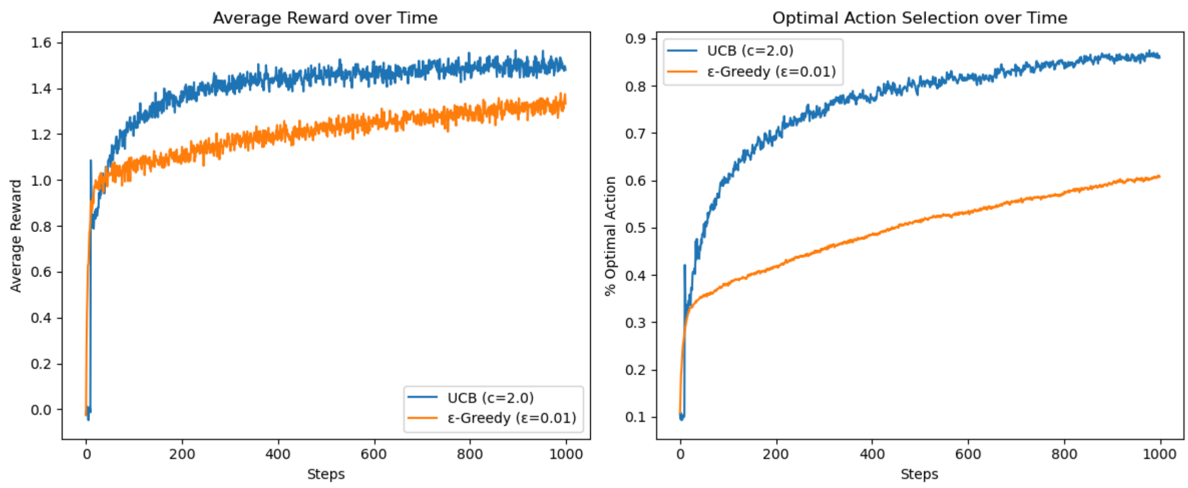
探索项

探索项反映了对动作 a 价值估计的**不确定性或方差。**被最大化的目标值可以看作是对动作 a 真实价值的一个**上界估计**，其中 c 决定了置信水平。

$$\sqrt{\frac{\ln t}{N_t(a)}}$$

情况	影响
$N_t(a)$ 小（尝试少）	分母小 → 探索项大 → 更可能被选（鼓励探索）
$N_t(a)$ 大（尝试多）	分母大 → 探索项小 → 了解充分 → 逐渐退场
t 增加（时间推进）	分子 $\ln t$ 缓慢增长 → 长期未被选的动作"不确定性"缓慢上升 → 重新获得被选择的机会

实验三



环境参数

- 赌博机臂数量：10臂
- 每个臂的真实收益：从均值为0、标准差为1的正态分布中采样
- 每次拉动臂的收益：从该臂的正态分布中采样（标准差=1）

算法参数

- **UCB算法**：参数$c = 2.0$
- **ε-贪心算法**：探索概率$\varepsilon = 0.01$
- **步长方法**：样本平均更新（无偏估计）
- 实验步数：1000步
- 实验重复次数：2000次

评估指标

1. 平均累积奖励
2. 最优动作选择比例

实验步骤

1. 初始化10臂赌博机环境
2. 对于每个算法分别运行：
   - UCB算法：前10步确保每个动作至少被选择一次，后续按UCB公式选择
   - ε-贪心算法：按ε-贪心策略选择动作
3. 使用**样本平均方法**更新动作价值估计：$Q_{n+1} = Q_n + \frac{1}{n}(R_n - Q_n)$
4. 记录每步的奖励和最优动作选择情况
5. 重复2000次实验，计算平均性能指标
6. 绘制性能对比图

性能对比

- **初期表现**：UCB算法在前10步需要探索所有动作，表现可能较差
- **中期表现**：UCB算法通过智能探索快速收敛到最优动作
- **长期表现**：UCB算法由于其平衡的探索利用机制，通常优于低ε值的ε-贪心算法

理论分析

- UCB算法能够自适应地平衡探索与利用
- ε-贪心算法($\varepsilon = 0.01$)探索频率较低，可能错过更好的动作
- 样本平均更新确保无偏估计，随着样本增加估计逐渐准确

局限性

UCB在实际强化学习中应用受限，其难以处理非平稳问题；因为必须为每个动作维护 $N_t(a)$ 和 $Q_t(a)$在状态空间巨大或使用函数逼近时，计算和存储成本过高。

> Tip
>
> # 点估计
>
> **点估计**是用一个具体的数值"点"来估计某个未知的总体参数。
>
> 在现实生活中，我们常常想知道一些“总体特征”，比如：
>
> - 全国成年人的平均身高
> - 某种疫苗的有效率
> - 一台机器生产零件的平均长度
>
> 但你不可能测量所有人、所有人接种后的结果、或每一个零件。
>
> 所以，我们只能：
>
> 1. 抽取一部分样本（sample）
> 2. 计算样本中的统计量（如平均值）
> 3. 用这个统计量去**估计**总体的真实参数
>
> 这个过程就是**点估计**。
>
> ## 例子
>
> 例子1：估计平均身高
>
> - **总体参数**（未知）：全国成年人的平均身高 μ
> - **我们做了什么**：随机调查了 100 个人，算出他们的平均身高是 172 cm
> - **点估计**：用 172 cm 来估计 μ
>
> 即：
>
> $$\hat{\mu} = 172 \text{ cm}$$
>
> 例子2：估计疫苗有效率
>
> - 真实有效率 p（未知）
> - 在试验中，1000 名接种者中有 950 人未感染 → 样本有效率 $\frac{950}{1000} = 0.95$
> - **点估计**：$\hat{p} = 0.95$

常见的点估计方法

总体参数	常用点估计量	公式
总体均值 μ	样本均值	$\bar{X} = \frac{1}{n}\sum X_i$
总体方差 σ^2	样本方差	$s^2 = \frac{1}{n-1}\sum(X_i - \bar{X})^2$
总体比例 p	样本比例	$\hat{p} = \frac{\text{成功次数}}{n}$

这些估计量被称为**估计量**（estimator），而代入数据后算出的具体数值叫**估计值**（estimate）。

点估计的局限性

虽然点估计给出了一个“最佳猜测”，但它有**重大缺陷**：它**不告诉我们“有多准”**

比如：

- 你用 10 个人的数据估计平均身高是 172 cm
- 我用 10000 个人的数据也得到 172 cm

两个点估计相同，但显然你的估计更不可靠。**点估计忽略了不确定性。**

为了弥补点估计的不足，统计学家引入了**区间估计**：

- 不只说“我认为是 172”
- 而是说：“我有 95% 的信心，真实平均身高在 168 到 176 cm 之间”

这个区间就是**置信区间**（Confidence Interval），它的上限就是**置信上界**（Upper Confidence Bound），下限是**置信下界。**

💡 **Tip**

置信上界

在统计学中，我们常常需要根据**有限的样本数据**来估计某个未知的总体参数。

例如：

- 估计某产品的平均寿命
- 估计某种药物的有效率
- 估计某地区居民的平均收入

我们用样本均值 $\bar{X}$ 作为总体均值 μ 的点估计：

$$\hat{\mu} = \bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$$

但问题是：**这个估计有多可信？**

为了量化这种不确定性，统计学家引入了**置信区间**（Confidence Interval）。

> ### 定义：
>
> 对于一个未知参数 μ，一个 95% 的置信区间是一个区间 $[L, U]$，使得：
>
> $$\mathbb{P}(L \leq \mu \leq U) = 0.95$$
>
> 即：如果我们重复抽样多次，构造出很多这样的区间，大约有 95% 的区间会包含真实的 μ。
>
> ### 标准正态情形下的置信区间：
>
> 当样本独立同分布且总体近似正态时，均值的置信区间为：
>
> $$\bar{X} \pm z_{\alpha/2} \cdot \frac{s}{\sqrt{n}}$$
>
> 其中：
>
> - $\bar{X}$：样本均值
> - s：样本标准差
> - n：样本量
> - $z_{\alpha/2}$：标准正态分布的分位数（如 95% 置信水平下为 1.96）
>
> 这个区间的**上界**就是：
>
> $$\text{UCB} = \bar{X} + z_{\alpha/2} \cdot \frac{s}{\sqrt{n}}$$
>
> 置信上界表示：**在给定置信水平下，真实参数可能达到的最大值。**
>
> 换句话说：
>
> - 我们不能确定 μ 到底是多少
> - 但我们可以说："以 95% 的信心，真实均值不会超过这个上界"
>
> 这在决策中非常有用，尤其是在**保守估计**或**风险控制**场景中。

练习2.8

> 为什么上图中使用 $c = 2$ 的 UCB 算法在第 11 步左右出现一个明显的奖励尖峰，随后性能略有下降。而 $c = 1$ 的曲线则更平滑，无明显尖峰？

UCB 算法在第 11 步出现奖励尖峰，是因为在前 10 步中，未被尝试的动作具有无限大的置信上界，导致算法系统性地遍历所有动作。到第 11 步时，所有动作都已被尝试，UCB 进入自由选择阶段。此时 $\ln t$ 相对较大，而 $N_t(a) = 1$ 对多数动作成立，不确定性项 $c\sqrt{\frac{\ln t}{N_t(a)}}$ 显著放大估计差异。当 $c = 2$ 时，算法强烈倾向于选择前阶段表现较好的动作，这些动作往往真实价值较高，导致第 11 步平均回报显著上升，形成"尖峰"。随后，由于被选动作的计数增加，不确定性下降，算法重新探索其他动作，平均回报回归稳定水平，曲线轻微回落。当 $c = 1$ 时，探索强度较弱，该效应不明显，因此无显著尖峰。

梯度赌博机算法

传统方法（如 ε-贪婪、UCB）通过**估计动作的真实价值** $Q(a)$ 来指导动作选择。另一种思路是**不直接估计价值，而是学习每个动作的"偏好值"** $H_t(a)$**，再通过偏好决定选择概率。**

- 偏好值越高 → 被选中的概率越大

- 偏好值本身**没有语义含义**（不是奖励估计）
- 只有**相对大小**重要（例如所有 $H_t(a)$ 加上常数不影响结果）

动作选择基于**Softmax 函数**（也称 Gibbs 或 Boltzmann 分布）：

$$\Pr\{A_t = a\} = \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} = \pi_t(a) \tag{2.11}$$

符号说明：

- $H_t(a)$：动作 a 在时刻 t 的偏好值
- $\pi_t(a)$：动作 a 在时刻 t 被选中的概率
- k：动作总数

特性：

- 所有动作初始偏好相等（如 $H_1(a) = 0$）→ 初始时各动作等概率被选
- 若某个动作的 $H_t(a)$ 显著大于其他，则其被选概率接近 1
- Softmax 是**连续、可微**的，便于使用梯度方法优化

练习2.9

在强化学习中，**softmax 动作选择策略**根据动作的偏好值 $H_t(a)$ 按如下概率选择动作：

$$\Pr\{A_t = a\} = \frac{e^{H_t(a)}}{\sum_{a'} e^{H_t(a')}}$$

而 **logistic 函数**（又称 sigmoid 函数）定义为：

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

> **问题**：证明在只有两个动作的情况下，softmax 分布退化为 logistic 函数形式，二者等价。

考虑仅有两个动作的情形：a_1 和 a_2，其对应的偏好值分别为 $H_t(a_1)$ 和 $H_t(a_2)$。

根据 softmax 分布，选择每个动作的概率为：

$$\Pr\{A_t = a_1\} = \frac{e^{H_t(a_1)}}{e^{H_t(a_1)} + e^{H_t(a_2)}} \quad , \quad \Pr\{A_t = a_2\} = \frac{e^{H_t(a_2)}}{e^{H_t(a_1)} + e^{H_t(a_2)}}$$

我们的目标是将 $\Pr\{A_t = a_1\}$ 化为标准 logistic 函数形式。

将分子分母同除以 $e^{H_t(a_2)}$：

$$\Pr\{A_t = a_1\} = \frac{e^{H_t(a_1)}/e^{H_t(a_2)}}{(e^{H_t(a_1)} + e^{H_t(a_2)})/e^{H_t(a_2)}} = \frac{e^{H_t(a_1)-H_t(a_2)}}{e^{H_t(a_1)-H_t(a_2)} + 1}$$

令：

$$z = H_t(a_1) - H_t(a_2)$$

则：

$$\Pr\{A_t = a_1\} = \frac{e^z}{e^z + 1}$$

进一步变形：

$$\frac{e^z}{e^z+1} = \frac{1}{1+e^{-z}} = \sigma(z)$$

$$\Pr\{A_t = a_1\} = \sigma\left(H_t(a_1) - H_t(a_2)\right)$$

这正是以 **偏好差值为输入** 的标准 logistic 函数。

对于动作 a_2：

$$\Pr\{A_t = a_2\} = \frac{e^{H_t(a_2)}}{e^{H_t(a_1)} + e^{H_t(a_2)}} = \frac{1}{1+e^{H_t(a_1)-H_t(a_2)}} = \frac{1}{1+e^z} = \sigma(-z)$$

而：

$$\sigma(-z) = 1 - \sigma(z)$$

因此：

$$\Pr\{A_t = a_2\} = 1 - \Pr\{A_t = a_1\}$$

且：

$$\Pr\{A_t = a_1\} + \Pr\{A_t = a_2\} = \sigma(z) + (1 - \sigma(z)) = 1$$

满足概率分布的归一化条件。

> 在仅有两个动作的情况下，softmax 动作选择策略退化为 logistic 函数：
>
> $$\Pr\{A_t = a_1\} = \sigma\left(H_t(a_1) - H_t(a_2)\right), \quad \Pr\{A_t = a_2\} = \sigma\left(H_t(a_2) - H_t(a_1)\right)$$
>
> 其中 $\sigma(z) = \dfrac{1}{1+e^{-z}}$ 是标准的 logistic（sigmoid）函数。

因此，**二动作下的 softmax 分布与 logistic 回归中的选择机制完全等价。**

在每一步 t：

1. 根据 $\pi_t(a)$ 选择动作 A_t
2. 观察奖励 R_t
3. 更新所有动作的偏好值：

> **Note**
>
> - 被选动作 A_t 的更新项是 $+(1-\pi_t(A_t))$
> - 其他动作 a 的更新项是 $-\pi_t(a)$
> - 所有动作的更新量加起来为：
>
> $$(1-\pi_t(A_t)) - \sum_{a \neq A_t} \pi_t(a) = (1-\pi_t(A_t)) - (1-\pi_t(A_t)) = 0$$
>
> → 保持整体平衡

$$H_{t+1}(A_t) \doteq H_t(A_t) + \alpha(R_t - \bar{R}_t)(1 - \pi_t(A_t)) \qquad \text{（所选动作）}$$

$$H_{t+1}(a) \doteq H_t(a) - \alpha(R_t - \bar{R}_t)\pi_t(a), \quad \forall a \neq A_t \tag{2.12}$$

参数解释：

- $\alpha > 0$：步长参数（学习率）
- $\bar{R}_t$：到时间 t 为止的**平均奖励**（可用增量方式计算）
- $R_t - \bar{R}_t$：当前奖励相对于历史平均水平的"优势"

更新逻辑：

情况	含义	更新方向
$R_t > \bar{R}_t$	当前表现优于平均水平	增加所选动作的偏好
$R_t < \bar{R}_t$	当前表现差于平均水平	降低所选动作的偏好

设计理念

- 如果你做了一件事得到了**高于平均的回报**，说明这件事"值得多做" → 提高它的偏好
- 反之，如果回报低于平均，"可能不是好选择" → 降低偏好
- 所有未被选择的动作偏好**同步下调**，以保持总概率为 1

$\bar{R}_t$ 被称为**基准项**（baseline），它的引入至关重要。其可以**消除奖励绝对水平的影响；减少更新方差，使学习更稳定。**

实验四

1. 问题设置

- **臂的数量**：10个
- **真实期望收益分布**：$q_*(a) \sim \mathcal{N}(4, 1)$（均值为+4，标准差为1）
- **奖励分布**：$R \sim \mathcal{N}(q_*(a), 1)$（每个动作的奖励服从正态分布，方差为1）
- **实验规模**：2000个独立任务
- **每任务步数**：1000步

2. 算法配置（四条对比曲线）

配置1：有基准项，小步长

- **收益基准项**：有（使用运行平均计算 $\bar{R}_t$）
- **步长参数**：$\alpha = 0.1$
- **标记**："with baseline, α=0.1"

配置2：有基准项，大步长

- **收益基准项**：有（使用运行平均计算 $\bar{R}_t$）
- **步长参数**：$\alpha = 0.4$
- **标记**："with baseline, α=0.4"

配置3：无基准项，小步长

- **收益基准项**：无（$\bar{R}_t = 0$，即更新中使用 R_t 而非 $R_t - \bar{R}_t$）
- **步长参数**：$\alpha = 0.1$
- **标记**："without baseline, α=0.1"

配置4：无基准项，大步长

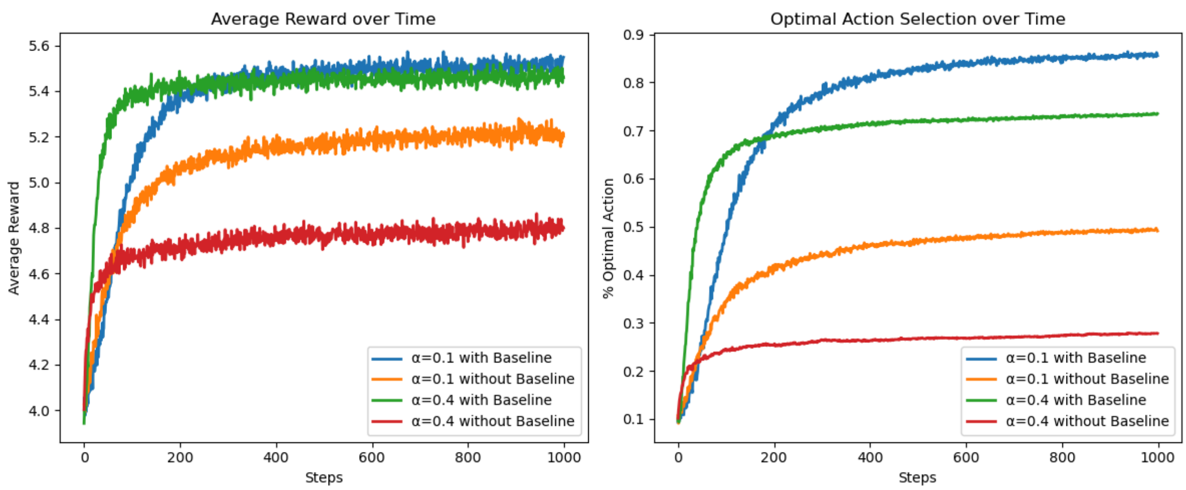
- **收益基准项**：无（$\bar{R}_t = 0$，即更新中使用 R_t 而非 $R_t - \bar{R}_t$）
- **步长参数**：$\alpha = 0.4$
- **标记**："without baseline, α=0.4"

```
import numpy as np
import matplotlib.pyplot as plt

# 多臂赌博机类（期望收益均值为+4）
class Bandit:
    def __init__(self, k=10, mean=4, std=1):
        self.k = k
        self.arm_means = np.random.normal(mean, std, k)  # 期望收益均值为+4
        self.optimal_action = np.argmax(self.arm_means)

    def pull(self, a):
        # 拉动第 a 个臂，返回收益
        return np.random.normal(self.arm_means[a], 1)

# 梯度赌博机算法（含基准项）
def gradient_bandit_with_baseline(bandit, steps=1000, alpha=0.1):
    H = np.zeros(bandit.k)  # 偏好函数初始值
    R_avg = 0.0  # 平均收益
    rewards = []
    optimal_action_selected = []

    for t in range(steps):
        # 计算softmax概率
        exp_H = np.exp(H)
        pi = exp_H / np.sum(exp_H)

        # 根据概率选择动作
        a = np.random.choice(bandit.k, p=pi)

        # 获得收益
        reward = bandit.pull(a)

        # 更新平均收益（样本平均）
        R_avg += (reward - R_avg) / (t + 1)

        # 更新偏好函数（含基准项）
        for i in range(bandit.k):
            if i == a:
                H[i] += alpha * (reward - R_avg) * (1 - pi[i])
            else:
                H[i] -= alpha * (reward - R_avg) * pi[i]

        rewards.append(reward)
        optimal_action_selected.append(int(a == bandit.optimal_action))

    return rewards, optimal_action_selected

# 梯度赌博机算法（不含基准项）
def gradient_bandit_without_baseline(bandit, steps=1000, alpha=0.1):
```

```python
    H = np.zeros(bandit.k)  # 偏好函数初始值
    rewards = []
    optimal_action_selected = []

    for t in range(steps):
        # 计算softmax概率
        exp_H = np.exp(H)
        pi = exp_H / np.sum(exp_H)

        # 根据概率选择动作
        a = np.random.choice(bandit.k, p=pi)

        # 获得收益
        reward = bandit.pull(a)

        # 更新偏好函数（不含基准项，基准设为0）
        for i in range(bandit.k):
            if i == a:
                H[i] += alpha * (reward - 0) * (1 - pi[i])
            else:
                H[i] -= alpha * (reward - 0) * pi[i]

        rewards.append(reward)
        optimal_action_selected.append(int(a == bandit.optimal_action))

    return rewards, optimal_action_selected

# 主实验函数
def run_experiment(runs=2000, steps=1000):
    # 初始化结果数组
    rewards_with_baseline_alpha01 = np.zeros(steps)
    rewards_without_baseline_alpha01 = np.zeros(steps)
    rewards_with_baseline_alpha04 = np.zeros(steps)
    rewards_without_baseline_alpha04 = np.zeros(steps)

    opt_action_with_baseline_alpha01 = np.zeros(steps)
    opt_action_without_baseline_alpha01 = np.zeros(steps)
    opt_action_with_baseline_alpha04 = np.zeros(steps)
    opt_action_without_baseline_alpha04 = np.zeros(steps)

    print("运行实验...")
    for i in range(runs):
        if i % 500 == 0:
            print(f"运行 {i}/{runs}")
        bandit = Bandit()

        # alpha = 0.1
        r1, o1 = gradient_bandit_with_baseline(bandit, steps, alpha=0.1)
        r2, o2 = gradient_bandit_without_baseline(bandit, steps, alpha=0.1)

        # alpha = 0.4
        r3, o3 = gradient_bandit_with_baseline(bandit, steps, alpha=0.4)
        r4, o4 = gradient_bandit_without_baseline(bandit, steps, alpha=0.4)

        # 累加结果
        rewards_with_baseline_alpha01 += np.array(r1)
```

```
        rewards_without_baseline_alpha01 += np.array(r2)
        rewards_with_baseline_alpha04 += np.array(r3)
        rewards_without_baseline_alpha04 += np.array(r4)

        opt_action_with_baseline_alpha01 += np.array(o1)
        opt_action_without_baseline_alpha01 += np.array(o2)
        opt_action_with_baseline_alpha04 += np.array(o3)
        opt_action_without_baseline_alpha04 += np.array(o4)

    # 计算平均值
    rewards_with_baseline_alpha01 /= runs
    rewards_without_baseline_alpha01 /= runs
    rewards_with_baseline_alpha04 /= runs
    rewards_without_baseline_alpha04 /= runs

    opt_action_with_baseline_alpha01 /= runs
    opt_action_without_baseline_alpha01 /= runs
    opt_action_with_baseline_alpha04 /= runs
    opt_action_without_baseline_alpha04 /= runs

    # 绘图
    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
    plt.plot(rewards_with_baseline_alpha01, label='α=0.1 with Baseline',
linewidth=2)
    plt.plot(rewards_without_baseline_alpha01, label='α=0.1 without Baseline',
linewidth=2)
    plt.plot(rewards_with_baseline_alpha04, label='α=0.4 with Baseline',
linewidth=2)
    plt.plot(rewards_without_baseline_alpha04, label='α=0.4 without Baseline',
linewidth=2)
    plt.xlabel('Steps')
    plt.ylabel('Average Reward')
    plt.legend()
    plt.title('Average Reward over Time')

    plt.subplot(1, 2, 2)
    plt.plot(opt_action_with_baseline_alpha01, label='α=0.1 with Baseline',
linewidth=2)
    plt.plot(opt_action_without_baseline_alpha01, label='α=0.1 without Baseline',
linewidth=2)
    plt.plot(opt_action_with_baseline_alpha04, label='α=0.4 with Baseline',
linewidth=2)
    plt.plot(opt_action_without_baseline_alpha04, label='α=0.4 without Baseline',
linewidth=2)
    plt.xlabel('Steps')
    plt.ylabel('% Optimal Action')
    plt.legend()
    plt.title('Optimal Action Selection over Time')

    plt.tight_layout()
    plt.show()

# 运行实验
if __name__ == "__main__":
```

```
    np.random.seed(42)  # 为了可重现性
    run_experiment(runs=2000, steps=1000)
```



在真实期望收益均值为+4的10臂赌博机环境中，实验结果显示：

- **有基准项的算法**（两条曲线）：迅速适应高收益环境，平均奖励快速上升至约+4.1-4.2，在200步内达到接近最优性能
- **无基准项的算法**（两条曲线）：性能显著下降，平均奖励仅达到约+3.4-3.6，学习过程缓慢且无法充分利用高收益环境

基准项 $\bar{R}_t$ 作为关键机制，发挥以下作用：

1. **相对评估机制**：
    - 算法关注 $R_t - \bar{R}_t$（当前收益与历史平均的偏差）
    - 而非绝对收益值 R_t
    - 使算法能够区分"表现好于平均水平"和"表现差于平均水平"的动作
2. **环境适应能力**：
    - 当所有收益水平整体提高至+4时
    - 基准项 $\bar{R}_t$ 自动上升至接近+4
    - 保持 $R_t - \bar{R}_t$ 在合理范围（约-1至+1）
    - 使算法不受绝对收益水平影响，仅关注动作间的相对表现
3. **防止偏好值膨胀**：
    - 无基准项时，所有 $R_t \approx 4$ 都远大于0
    - 导致所有动作偏好值 $H_t(a)$ 快速增长
    - softmax输出趋于均匀分布（$\pi_t(a) \approx 1/10$）
    - 无法有效区分真正优秀的动作
4. **理论最优性**：
    - 理论证明最优基准为 $\bar{R}_t = \mathbb{E}[R_t]$
    - 运行平均是对该理论最优基准的有效近似
    - 显著降低策略梯度估计的方差

基准项是梯度赌博机算法成功的关键：它使算法能够**适应不同收益水平的环境**，通过**相对评估**而非**绝对评估**来区分动作优劣，从而在所有收益整体偏高的环境中仍能有效学习最优策略。无基准项的算法会因偏好值膨胀而失去区分能力，导致性能显著下降。

随机梯度上升

梯度赌博机算法的本质是**最大化期望奖励**的一种**随机梯度上升**方法。

目标函数：

$$J(\mathbf{H}_t) = \mathbb{E}[R_t] = \sum_x \pi_t(x) q_*(x)$$

1. $J(\mathbf{H}_t)$：性能函数

- J 是一个**目标函数**（也叫目标、性能指标）
- 它衡量的是：在偏好值为 $\mathbf{H}_t$ 的情况下，智能体的"表现有多好"
- 我们的目标是：**让 J 越大越好** → 即最大化期望奖励

---

2. $\mathbb{E}[R_t]$：期望奖励（Expected Reward）

- R_t：在时间 t 实际获得的奖励（是一个随机变量，因为动作是随机选的）
- $\mathbb{E}[R_t]$：表示"如果我们用当前策略运行一次，平均能得多少奖励"
- 这是一个**期望值**，不是某一次的具体奖励

---

3. $\sum_x \pi_t(x) q_*(x)$：期望奖励的展开形式

这是期望奖励的**具体计算方式**，我们来拆解：

符号	含义
x	动作的索引（同 a，）
$\pi_t(x)$	在时间 t，选择动作 x 的概率（由 Softmax 决定）
$q_*(x)$	动作 x 的**真实期望奖励**（未知但固定）

所以：

$$\mathbb{E}[R_t] = \sum_x \pi_t(x) \cdot q_*(x)$$

我们要最大化 $J(\mathbf{H}_t) = \mathbb{E}[R_t]$，所以使用**梯度上升法**：

$$H_{t+1}(a) = H_t(a) + \alpha \frac{\partial J}{\partial H_t(a)}$$

也就是说：

- 计算 J 对每个 $H_t(a)$ 的偏导数
- 沿着梯度方向更新偏好值
- 最终使 J 增大

虽然我们不知道 $q_*(x)$，但可以通过实际奖励 R_t 来**估计梯度。**

$$\frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} = \frac{\partial}{\partial H_t(a)} \left[\sum_x \pi_t(x) q_*(x) \right] = \sum_x q_*(x) \frac{\partial \pi_t(x)}{\partial H_t(a)} \qquad (1)$$

- $q_*(x)$ 是常数（真实价值，不依赖于 H_t）
- 只有 $\pi_t(x)$ 依赖于 $H_t(a)$
- 所以可以把导数放进求和里，只对 $\pi_t(x)$ 求导

引入基准项 B_t

$$= \sum_x (q_*(x) - B_t) \frac{\partial \pi_t(x)}{\partial H_t(a)} \tag{2}$$

其中 B_t 是任意一个**不依赖于动作** x 的标量（比如 $\bar{R}_t$、0、1000 都行）。

答案并没有发生改变，因为：

$$\sum_x \frac{\partial \pi_t(x)}{\partial H_t(a)} = \frac{\partial}{\partial H_t(a)} \left(\sum_x \pi_t(x) \right) = \frac{\partial}{\partial H_t(a)}(1) = 0$$

所以答案并没有发生改变

因此：

$$\sum_x B_t \frac{\partial \pi_t(x)}{\partial H_t(a)} = B_t \cdot \sum_x \frac{\partial \pi_t(x)}{\partial H_t(a)} = B_t \cdot 0 = 0$$

所以：

$$\sum_x q_*(x) \frac{\partial \pi_t(x)}{\partial H_t(a)} = \sum_x (q_*(x) - B_t) \frac{\partial \pi_t(x)}{\partial H_t(a)}$$

将上式**改写成期望形式**有

乘以 $\frac{\pi_t(x)}{\pi_t(x)} = 1$：

$$= \sum_x \pi_t(x) \cdot (q_*(x) - B_t) \cdot \frac{1}{\pi_t(x)} \frac{\partial \pi_t(x)}{\partial H_t(a)} \tag{3}$$

回忆期望的定义：

$$\mathbb{E}[f(X)] = \sum_x p(x) f(x)$$

在这里：

- $p(x) = \pi_t(x)$：选择动作 x 的概率
- $f(x) = (q_*(x) - B_t) \cdot \frac{1}{\pi_t(x)} \frac{\partial \pi_t(x)}{\partial H_t(a)}$

所以整个表达式就是：

$$= \mathbb{E}\left[(q_*(A_t) - B_t) \cdot \frac{1}{\pi_t(A_t)} \frac{\partial \pi_t(A_t)}{\partial H_t(a)} \right] \tag{4}$$

- A_t 是一个随机变量，表示“在时间 t 实际被选择的动作”
- 这个期望是在当前策略 π_t 下对所有可能的动作取平均

现在把“梯度”写成了“某个随机变量的期望”。

用 R_t 替代 $q_*(A_t)$

我们现在有：

$$\text{梯度} = \mathbb{E}\left[(q_*(A_t) - B_t) \cdot \frac{1}{\pi_t(A_t)} \frac{\partial \pi_t(A_t)}{\partial H_t(a)}\right]$$

问题来了：我们仍然不知道 $q_*(A_t)$，怎么办？

虽然我们不知道 $q_*(A_t)$，但我们知道：**在给定选择了动作 A_t 的条件下，奖励 R_t 的期望就是它的真值。**

$$\mathbb{E}[R_t \mid A_t] = q_*(A_t)$$

R_t 是 $q_*(A_t)$ 的一个**无偏估计**

我们可以用 R_t 来代替 $q_*(A_t)$：

$$\approx \mathbb{E}\left[(R_t - B_t) \cdot \frac{1}{\pi_t(A_t)} \frac{\partial \pi_t(A_t)}{\partial H_t(a)}\right] \tag{5}$$

这个近似在**期望意义上是准确的**（无偏），所以得到了一个**可以用实际数据计算的梯度估计**

> ⓘ **Note**
>
> $$\frac{\partial \pi_t(x)}{\partial H_t(a)} = \pi_t(x)(\mathbb{1}_{a=x} - \pi_t(a)) \tag{*}$$
>
> 这表示：**动作 x 的选择概率 $\pi_t(x)$ 对偏好值 $H_t(a)$ 的偏导数。**
>
> | 符号 | 含义 | 类型 |
> | --- | --- | --- |
> | $\pi_t(x)$ | 在时间 t，选择动作 x 的概率 | 数值（在 0~1 之间） |
> | $H_t(a)$ | 在时间 t，动作 a 的偏好值（数值越高越可能被选） | 实数 |
> | x | 当前关注的动作 | 动作索引 |
> | a | 正在对其求导的那个动作 | 动作索引 |
> | Z | 归一化常数（也叫配分函数）：$Z = \sum_y e^{H_t(y)}$ | 标量 |
> | $\mathbb{1}_{a=x}$ | 指示函数：如果 $a = x$ 就是 1，否则是 0 | 0 或 1 |
>
> Softmax 将偏好值 $H_t(x)$ 转换为概率：
>
> $$\pi_t(x) = \frac{e^{H_t(x)}}{\sum_{y=1}^k e^{H_t(y)}} = \frac{e^{H_t(x)}}{Z} \quad \text{其中 } Z = \sum_y e^{H_t(y)}$$
>
> 因为 $\pi_t(x) = \frac{\text{分子}}{\text{分母}} = \frac{e^{H_t(x)}}{Z}$，所以我们用**商法则**求导：
>
> > 商法则：

$$\frac{d}{dx}\left(\frac{u}{v}\right)=\frac{u'v-uv'}{v^2}$$

在这里：

- $u=e^{H_t(x)}$
- $v=Z=\sum_y e^{H_t(y)}$
- 自变量是 $H_t(a)$（我们想看看当 $H_t(a)$ 变化时，$\pi_t(x)$ 怎么变）

所以：

$$\frac{\partial\pi_t(x)}{\partial H_t(a)}=\frac{\frac{\partial e^{H_t(x)}}{\partial H_t(a)}\cdot Z-e^{H_t(x)}\cdot\frac{\partial Z}{\partial H_t(a)}}{Z^2} \tag{1}$$

1. $\frac{\partial e^{H_t(x)}}{\partial H_t(a)}$

- $e^{H_t(x)}$ 是动作 x 的指数偏好
- 它只依赖于 $H_t(x)$，不直接依赖于其他动作的偏好

所以：

- 如果 $a=x$：改变 $H_t(a)$ 就是在改变 $H_t(x)\rightarrow$ 导数是 $e^{H_t(x)}$
- 如果 $a\neq x$：改变 $H_t(a)$ 不影响 $e^{H_t(x)}\rightarrow$ 导数是 0

$$\frac{\partial e^{H_t(x)}}{\partial H_t(a)}=\mathbb{1}_{a=x}\cdot e^{H_t(x)}$$

其中 $\mathbb{1}_{a=x}$ 是**指示函数**：

$$\mathbb{1}_{a=x}=\begin{cases}1 & \text{if } a=x\\ 0 & \text{if } a\neq x\end{cases}$$

---

2. $\frac{\partial Z}{\partial H_t(a)}$

回忆：

$$Z=\sum_y e^{H_t(y)}$$

当我们对 $H_t(a)$ 求偏导时：

- 只有 $y=a$ 那一项会变化
- 其他项视为常数

所以：

$$\frac{\partial Z}{\partial H_t(a)}=\frac{\partial}{\partial H_t(a)}e^{H_t(a)}=e^{H_t(a)}$$

现在把上面两个结果代入：

$$\frac{\partial\pi_t(x)}{\partial H_t(a)}=\frac{(\mathbb{1}_{a=x}e^{H_t(x)})\cdot Z-e^{H_t(x)}\cdot e^{H_t(a)}}{Z^2}$$

拆开写：

$$= \frac{\mathbb{1}_{a=x} e^{H_t(x)} Z}{Z^2} - \frac{e^{H_t(x)} e^{H_t(a)}}{Z^2} = \mathbb{1}_{a=x} \frac{e^{H_t(x)}}{Z} - \frac{e^{H_t(x)}}{Z} \cdot \frac{e^{H_t(a)}}{Z}$$

注意到：

- $\frac{e^{H_t(x)}}{Z} = \pi_t(x)$
- $\frac{e^{H_t(a)}}{Z} = \pi_t(a)$

所以：

$$= \mathbb{1}_{a=x} \pi_t(x) - \pi_t(x)\pi_t(a) = \pi_t(x)(\mathbb{1}_{a=x} - \pi_t(a))$$

代入 Softmax 导数

我们之前证明了：

$$\frac{\partial \pi_t(x)}{\partial H_t(a)} = \pi_t(x)(\mathbb{1}_{a=x} - \pi_t(a))$$

代入上式：

$$\frac{1}{\pi_t(A_t)} \frac{\partial \pi_t(A_t)}{\partial H_t(a)} = \frac{1}{\pi_t(A_t)} \cdot \pi_t(A_t)(\mathbb{1}_{a=A_t} - \pi_t(a)) = \mathbb{1}_{a=A_t} - \pi_t(a)$$

所以梯度变为：

$$\frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} = \mathbb{E}\left[(R_t - B_t)(\mathbb{1}_{a=A_t} - \pi_t(a))\right] \tag{6}$$

既然梯度是某个量的期望，那么我们可以用**一次采样**来估计它。

在时间 t，我们观察到了：

- 实际选择的动作 A_t
- 实际获得的奖励 R_t
- 当前策略 $\pi_t(a)$

于是我们可以构造一个**梯度的无偏估计**：

$$\widehat{\nabla} = (R_t - B_t)(\mathbb{1}_{a=A_t} - \pi_t(a))$$

然后使用**随机梯度上升**更新：

$$H_{t+1}(a) = H_t(a) + \alpha(R_t - B_t)(\mathbb{1}_{a=A_t} - \pi_t(a)) \tag{7}$$

这正是我们原始算法 (2.12) 的统一形式！

- 当 $a = A_t$：$\mathbb{1}_{a=A_t} = 1 \to +\alpha(R_t - B_t)(1 - \pi_t(a))$
- 当 $a \neq A_t$：$\mathbb{1}_{a=A_t} = 0 \to -\alpha(R_t - B_t)\pi_t(a)$

总结

梯度赌博机算法的**期望更新方向**等于期望奖励关于动作偏好的**梯度方向**；**梯度赌博机算法不估计动作价值，而是学习动作偏好，并通过与平均奖励的比较进行随机梯度上升，从而最大化长期期望奖励。它是一种简单而理论坚实的策略优化方法，体现了“相对优势驱动学习”的思想。**

关联搜索与情境赌博机

此前讨论的都只是**非关联性任务**，即不需要将不同动作与不同情境相关联的任务。

特点	说明
单一情境	每次决策面对的是同一个赌博机（或环境）
动作选择独立于上下文	不需要根据“当前情况”调整策略
目标	找出一个全局最优动作，或在非平稳中追踪最优动作

但现实中更常见的是：需要“看情况做事”

比如：

- 同一个医生面对不同病人，要用不同治疗方法
- 同一个推荐系统面对不同用户，要推荐不同商品
- 同一辆自动驾驶汽车在不同路况下要采取不同操作

这就需要将**情境（context）**与**动作（action）**关联起来。

关联搜索

> **关联搜索**是指智能体通过试错学习，将**特定情境**与**最优动作**建立联系的过程。

其核心的机制在于学习一个**策略**，一种从情境到在该情境下最优动作的映射。策略的本质是：**条件反射式的映射**，而非单一的最优动作

假设你面对一个会变色的老虎机：

显示屏颜色	最优动作
红色	拉第1号杠杆
绿色	拉第2号杠杆
蓝色	拉第5号杠杆

虽然你不知道每种颜色对应的真实奖励分布，但你可以通过观察发现：

- 红色时拉1号臂总得高奖励
- 绿色时拉2号臂表现最好

于是你学会了一个策略：“红→1，绿→2，蓝→5”

情境赌博机

关联搜索是**试错学习**与**条件映射学习**的结合，既包含通过试错来**搜索**最佳动作的过程，也包含将这些动作与最适合它们的**情境**进行**关联**的过程。

名称部分	含义
搜索（Search）	仍需通过试错探索，找到每个情境下的好动作（像多臂赌博机）
关联（Association）	需要将动作与特定情境配对，建立"条件-响应"关系

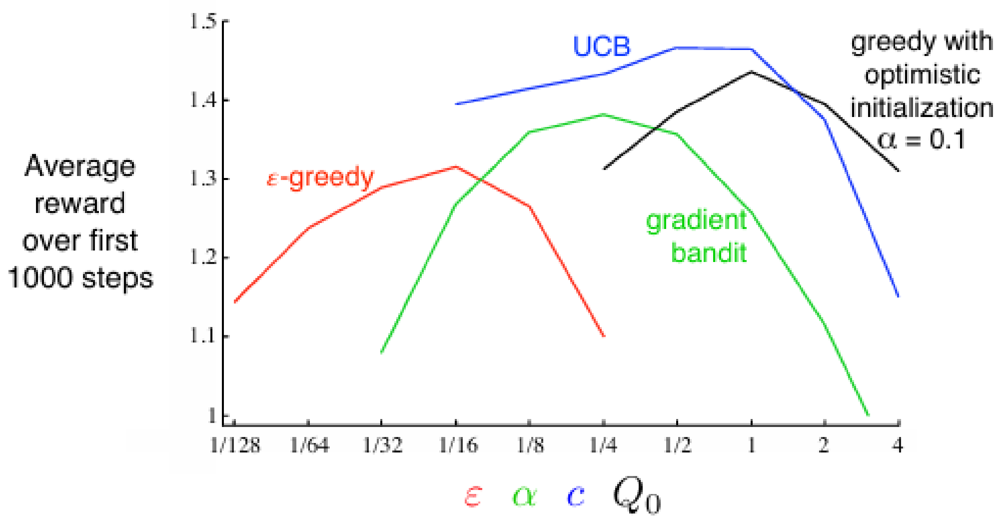
这类任务在当前文献中常被称为**情境赌博机**。

总结

方法	原理	探索机制	是否估计价值
ε-贪婪	大部分时间选最优动作，以概率 ε 随机探索	无差别随机探索	是（用 $Q(a)$）
UCB（置信上界）	选择 $Q(a)+c\sqrt{\frac{\ln t}{N(a)}}$ 最大的动作	偏向不确定性高的动作	是
梯度赌博机算法	学习动作偏好 $H(a)$，用 Softmax 输出选择概率	相对优势驱动更新（$R_t-\bar{R}_t$）	否（学的是偏好）
乐观初始值	将 $Q_1(a)$ 设得很高，使未尝试动作更具吸引力	初始高估 → "失望驱动"探索	是

所有方法都试图在**探索**（尝试未知）和**利用**（使用已知好动作）之间取得平衡。

使用**参数研究图**，用前1000步的平均奖励来总结整条学习曲线。



所有方法都呈现 **倒U型曲线**，参数太小时导致探索不足（过度利用）；参数太大时探索过多（浪费机会）；中间某个值能达成探索与利用平衡，达到性能最优。