时序差分

TD方法结合了**蒙特卡洛方法**和**动态规划（DP）方法**的优点，像蒙特卡洛方法一样，**直接从原始经验中学习**，不需要环境的模型。像DP方法一样，使用**自举（bootstrapping）**，即基于其他已学习的价值估计来更新当前估计，而无需等待最终结果。

> **Note**
>
> 复习：动态规划（DP）如何实现自举
>
> 在强化学习中，**自举**指的是在更新一个估计值时，**使用了其他已经存在的、同样是估计值的量**作为计算的一部分。DP通过**迭代应用贝尔曼期望方程**来求解策略 π 的价值函数 v_π。该过程从一个任意的初始价值函数 $V_0(s)$ 开始（例如所有状态设为0），然后反复更新，逐步逼近真实的价值函数 $v_\pi(s)$。
>
> **贝尔曼期望方程**
>
> 一个状态 s 的真实价值由其后续的即时奖励和所有可能后继状态的未来价值共同决定：
>
> $$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]$$
>
> 在计算 $v_\pi(s)$ 时，需要知道所有后继状态 s' 的真实价值 $v_\pi(s')$，但这些值正是我们试图求解的，因此是未知的。在第 k 次迭代中，DP使用**上一次迭代得到的、对后继状态价值的估计值** $V_{k-1}(s')$ 来代替贝尔曼方程中的真实值 $v_\pi(s')$。更新规则为：
>
> $$V_k(s) \leftarrow \mathbb{E}_\pi[R_{t+1} + \gamma V_{k-1}(S_{t+1}) \mid S_t = s]$$
>
> DP在更新 $V_k(s)$ 时，并未使用最终的真实回报或真实价值，而是**依赖于自身在上一轮迭代中产生的估计值** V_{k-1}。新的估计 $V_k(s)$ 是“基于旧的估计 $V_{k-1}(s')$ 被拉起来的”，即利用已有的（不完美的）知识来改进当前的知识。这个过程不等待幕结束，而是利用模型进行“内部预测”，因此被称为“自举”。尽管每次更新都使用了不完美的估计 V_{k-1}，但只要迭代次数足够多，这个过程会**保证收敛**到真实的 v_π。随着迭代进行，估计值 V_k 会越来越接近真实值 v_π。

> **Tip**
>
> **强化学习中的自举实例：网格世界策略评估**
>
> **1. 环境与任务设定**
>
> - **网格结构**：3×3 网格，状态用坐标 $(行, 列)$ 表示，例如 $(1,1)$ 为左下角，$(3,3)$ 为右上角。
> - **特殊状态**：
> - **终止状态**：$(3,3)$，其价值固定为 $v_\pi(3,3) = 0$。
> - **陷阱状态**：$(2,2)$，一旦进入，立即获得奖励 -10，并被强制转移到 $(3,3)$ 终止。
> - **动作空间**：在每个非终止状态，智能体可执行“上、下、左、右”四个动作，动作确定性执行（无滑动）。
> - **奖励函数**：
> - 每次移动（每步）获得 -1 的奖励（代表时间成本）。
> - 进入陷阱 $(2,2)$：获得 -10。
> - 到达终点 $(3,3)$：无额外奖励（已终止）。
> - **折扣因子**：$\gamma = 0.9$。
> - **目标策略** π：一个固定策略，智能体总是朝着目标 $(3,3)$ 移动。例如：

- 在 $(1,1)$：选择“右” → $(1,2)$
- 在 $(1,2)$：选择“上” → $(2,2)$（不幸进入陷阱）
- 在 $(2,1)$：选择“上” → $(3,1)$
- 在 $(3,1)$：选择“右” → $(3,2)$
- 在 $(3,2)$：选择“右” → $(3,3)$

任务是使用**动态规划的策略评估**（Policy Evaluation）算法，计算该策略 π 下所有状态的价值函数 $v_\pi(s)$。

DP通过迭代更新价值函数：

$$V_k(s) \leftarrow \mathbb{E}_\pi \left[R_{t+1} + \gamma V_{k-1}(S_{t+1}) \mid S_t = s \right]$$

自举体现在更新 $V_k(s)$ 时，使用的是上一轮的估计值 $V_{k-1}(S_{t+1})$，而不是真实值 $v_\pi(S_{t+1})$ 或最终回报。这是一种“用估计更新估计”的过程，无需等待慕结束。

第0轮：初始化

设所有状态的初始价值为0：

$$V_0(s) = 0, \quad \forall s$$

第1轮迭代（k=1）

使用 V_0 更新所有状态。

- **状态 (3,3)**（终止状态）：

$$V_1(3,3) = 0 \quad (\text{固定})$$

- **状态 (2,2)**（陷阱）：
  - 无论动作，立即获得 -10 并进入 (3,3)

$$V_1(2,2) = -10 + \gamma \cdot V_0(3,3) = -10 + 0.9 \times 0 = -10$$

- **状态 (3,2)**：
  - 动作“右” → (3,3)，奖励 -1

$$V_1(3,2) = -1 + \gamma \cdot V_0(3,3) = -1 + 0.9 \times 0 = -1$$

- **状态 (2,3)**：
  - 动作“右” → (3,3)，奖励 -1

$$V_1(2,3) = -1 + \gamma \cdot V_0(3,3) = -1$$

- **其他状态**（如 (1,1), (1,2), (2,1) 等）：
  - 它们的后继状态在 V_0 中价值为0，因此 $V_1(s) = -1$（仅包含即时奖励）

> **本轮小结**：价值信息开始从终止状态“泄露”出来。虽然 V_0 全为0，但通过自举，我们已得到 $V_1(3,2) = -1$、$V_1(2,2) = -10$。

第2轮迭代（k=2）

使用 V_1 的值更新 V_2。

- $V_1(3,3)=0,\quad V_1(3,2)=-1,\quad V_1(2,3)=-1,\quad V_1(2,2)=-10$
- **状态 (3,2)**:

$$V_2(3,2)=-1+\gamma\cdot V_1(3,3)=-1+0.9\times 0=-1\quad (\text{不变})$$

- **状态 (2,1)**:
  - 动作“上”→(3,1)，假设 $V_1(3,1)=-1$

$$V_2(2,1)=-1+\gamma\cdot V_1(3,1)=-1+0.9\times(-1)=-1.9$$

> 使用 $V_1(3,1)=-1$ 自举

- **状态 (1,2)**:
  - 动作“上”→(2,2)，奖励 -1

$$V_2(1,2)=-1+\gamma\cdot V_1(2,2)=-1+0.9\times(-10)=-1-9=-10$$

> **关键自举**：我们使用了上一轮刚计算出的 $V_1(2,2)=-10$，**立即**将 (1,2) 的价值更新为 -10。这意味着系统“知道”进入 (2,2) 是灾难性的。

- **状态 (1,1)**:
  - 动作“右”→(1,2)，$V_1(1,2)$ 在上一轮为 -1

$$V_2(1,1)=-1+\gamma\cdot V_1(1,2)=-1+0.9\times(-1)=-1.9$$

> **本轮小结**：**陷阱的负价值通过自举开始向后传播。**(1,2) 的价值因“感知”到 (2,2) 的危险而急剧下降。

第3轮迭代（k=3）

使用 V_2 更新 V_3。

- **状态 (1,1)**:
  - 现在 $V_2(1,2)=-10$

$$V_3(1,1)=-1+\gamma\cdot V_2(1,2)=-1+0.9\times(-10)=-1-9=-10$$

> (1,1) 的价值也因自举而“崩溃”，反映出该策略的高风险。

- **状态 (3,1)**:
  - 动作“右”→(3,2)，$V_2(3,2)=-1$

$$V_3(3,1)=-1+\gamma\cdot V_2(3,2)=-1+0.9\times(-1)=-1.9$$

- **状态 (2,3)**:
  - 动作“上”→(3,3)，$V_2(3,3)=0$

$$V_3(2,3)=-1+\gamma\cdot V_2(3,3)=-1$$

> **本轮小结**：负面价值继续沿路径反向传播。(1,1) 的价值从 -1.9 变为 -10，反映出其后继路径的危险性。

---

4. 收敛趋势

随着迭代继续，价值函数会稳定下来。例如：

- $v_\pi(3,3) = 0$
- $v_\pi(3,2) = -1$
- $v_\pi(3,1) = -1 + 0.9 \times (-1) = -1.9$
- $v_\pi(2,1) = -1 + 0.9 \times (-1.9) = -2.71$
- $v_\pi(2,2) = -10$
- $v_\pi(1,2) = -1 + 0.9 \times (-10) = -10$
- $v_\pi(1,1) = -1 + 0.9 \times (-10) = -10$

迭代轮次	自举行为	说明
k=1	使用 $V_0(3,3) = 0$ 计算 $V_1(3,2)$	即使初始估计为0，也能开始传播价值信息
k=2	使用 $V_1(2,2) = -10$ 计算 $V_2(1,2)$	**关键一步**：将陷阱的严重性传递给前驱状态
k=3	使用 $V_2(1,2) = -10$ 计算 $V_3(1,1)$	负面影响进一步向起点传播

核心结论：

DP的自举不是一次性完成的，而是通过**迭代过程逐步展开。**每一轮更新都"站在上一轮的肩膀上"，利用已有的估计值作为新计算的**基础。**价值信息（无论是正向的还是负向的）像波纹一样，从已知点（如终止状态、陷阱）**向外扩散**，最终形成完整的价值函数。

> **Note**
>
> 蒙特卡洛方法（Monte Carlo Method）复习
>
> 蒙特卡洛（MC）方法是一种**基于完整经验序列**（即从开始到结束的完整交互过程）来解决强化学习中**策略评估**（预测）问题的方法。它的目标是估计一个给定策略 π 下，每个状态 s 的价值函数 $v_\pi(s)$。价值函数 $v_\pi(s)$ 的定义是：**从状态 s 开始，遵循策略 π 所能获得的预期回报**：
>
> $$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] \tag{6.3}$$
>
> 其中 G_t 是从时间 t 开始的**实际回报**（即从 S_t 到终止状态的总折扣奖励）。
>
> 由于期望 $\mathbb{E}_\pi[G_t \mid S_t = s]$ 未知，蒙特卡洛方法通过**采样**来估计它：
>
> 每当智能体在遵循策略 π 的过程中**访问了状态** s，就记录下从该状态开始直到序列结束所获得的**实际回报** G_t。然后，将所有访问 s 时获得的 G_t 值取**平均**，作为 $v_\pi(s)$ 的估计值 $V(s)$。**简单来说：蒙特卡洛就是"用多次经历的平均结果来估计期望"。**
>
> **常步长-α蒙特卡洛**
>
> $$V(S_t) \leftarrow V(S_t) + \alpha\Big[G_t - V(S_t)\Big] \tag{6.1}$$
>
> - $V(S_t)$：当前对状态 S_t 价值的估计。
> - G_t：从 S_t 开始，直到序列结束的实际回报（即"完整幕回报"）。

- α：步长参数，控制学习速率。

这个更新规则本质上是一个**随机近似**，逐步将 $V(S_t)$ 向实际观察到的 G_t 调整。

为什么蒙特卡洛方法必须依赖"完整的幕回报"?

1. 回报 G_t 的定义决定了它必须等到序列结束

回报 G_t 的定义是：

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$$

它是从时间 t 到**终止时间** T 的所有未来奖励的加权和。只有当序列真正结束（即到达终止状态）时，我们才能知道所有的 $R_{t+1}, R_{t+2}, \ldots, R_T$，从而计算出 G_t。在序列进行中（$t < T$），G_t 是**未知的**，因为它依赖于未来的、尚未发生的奖励。

2. 蒙特卡洛不使用模型，也不进行自举

与DP不同，蒙特卡洛**不需要环境模型**（即不知道状态转移概率和奖励函数）。与TD不同，蒙特卡洛**不使用自举**（不依赖对后继状态价值的估计来更新当前状态）。它唯一能依赖的信息就是**实际发生的经验**。因此，它无法像DP那样"预测"未来，也无法像TD那样用 $R_{t+1} + \gamma V(S_{t+1})$ 来构造一个近似目标。它只能等待，直到**完整的轨迹走完**，才能获得一个真实的 G_t。

3. 必须等到"幕"结束才能更新

蒙特卡洛方法必须等到剧集结束才能确定 $V(S_t)$ 的增量（只有那时 G_t 才已知）。这意味着所有更新都必须是**离线的**（offline）或**幕后的**（at the end of the episode）。在序列进行过程中，即使你已经获得了 R_{t+1}，也无法更新 $V(S_t)$，因为你不知道未来会发生什么。

Caution

困惑：DP似乎也是从最后一步"倒推"价值，那它不也是依赖完整的路径信息吗?

DP并不依赖任何"幕"或经验序列，它依赖的是环境的完整模型。 DP的"倒推"是基于模型的计算，而不是基于经验的回溯。

方法	依赖什么?	是否需要经历一个完整的幕?
蒙特卡洛	实际发生的**经验序列**（即"幕"）	必须经历并等待幕结束
DP	环境的**数学模型**（状态转移概率和奖励函数）	完全不需要经历任何幕

蒙特卡洛必须**先走完一条路径**，比如：

$$S_0 \to S_1 \to S_2 \to \cdots \to S_T$$

然后，从终点开始，**逆向计算每个** G_t：

- $G_T = 0$
- $G_{T-1} = R_T$
- $G_{T-2} = R_{T-1} + \gamma R_T$
- ...
- $G_0 = R_1 + \gamma R_2 + \cdots + \gamma^{T-1} R_T$
- 最后，用这些 G_t 去更新 $V(S_t)$。

> 这个过程**完全依赖于你实际走过的这条路径**。没有这条路径，就没有 G_t，就无法更新。
>
> DP的“倒推”不是在一条具体路径上进行的，而是在**所有可能的状态空间上进行的系统性迭代**。其**输入是模型，不是数据**。DP知道：从状态 s 执行动作 a，会以概率 $p(s', r \mid s, a)$ 转移到 s' 并获得奖励 r。这个信息是**预先给定的**，不需要通过交互获得。**更新规则是贝尔曼方程**：
>
> $$V(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s', r} p(s', r \mid s, a) \left[r + \gamma V(s') \right]$$
>
> 这个公式**直接计算了从 s 出发的期望回报**，而不需要模拟任何路径。**“倒推”体现在迭代顺序，而非路径依赖**：
>
> - 在第一次迭代中，所有 $V(s)$ 都是任意值（如0）。
> - 在第二次迭代中，终止状态的 $V(s)$ 被正确设置为0。
> - 在第三次迭代中，那些**一步就能到终止状态**的状态，其 $V(s)$ 被更新。
> - 在第四次迭代中，**两步能到终止状态**的状态被更新。
> - 以此类推，价值信息像波纹一样从终止状态向外扩散。
>
> DP不需要“走完一条幕”来获得信息。它通过模型**一次性考虑所有可能的后继状态**，并用它们的当前估计值来更新当前状态。

时序差分预测

TD 和蒙特卡洛方法都使用经验来解决**预测问题**。给定遵循策略 π 的一些经验，两种方法都会**更新对非终止状态 S_t 的 v_π 的估计 $V(S_t)$**。TD方法不需要等待序列结束，只需等到下一个时间步 $t+1$。用即时奖励 R_{t+1} 和下一状态的估计值 $V(S_{t+1})$ 构造目标。更新公式为：

$$V(S_t) \leftarrow V(S_t) + \alpha \Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \Big] \tag{6.2}$$

蒙特卡洛更新的目标是 G_t，而TD更新的目标是 $R_{t+1} + \gamma V(S_{t+1})$。这种方法称为 **TD(0)**或者单步 TD，是更一般化的 TD(λ) 和 n步TD 方法的特例。

表格型TD(0)算法流程

输入：要评估的策略 π
参数：步长参数 $\alpha \in (0, 1]$
初始化：对所有状态 $s \in S^+$，任意初始化 $V(s)$，但终止状态满足 $V = 0$
循环（对每个序列）：
    初始化初始状态 S
    循环（对每一步）：
        根据策略 π 选择动作 A
        执行动作，观察奖励 R 和下一状态 S'
        更新价值函数：$V(S) \leftarrow V(S) + \alpha \big[R + \gamma V(S') - V(S) \big]$
        更新状态：$S \leftarrow S'$
    直到 S 为终止状态

自举特性

TD 方法使用当前估计 $V(S_{t+1})$ 来更新 $V(S_t)$，因此属于**自举方法**，与 DP 类似。

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] \tag{6.3}$$

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \tag{6.4}$$

蒙特卡洛方法使用样本回报 G_t 来近似期望（基于公式6.3），不使用自举。而**DP方法**使用完整模型计算期望（基于公式6.4），但用当前估计 $V(S_{t+1})$ 替代真实值 $v_\pi(S_{t+1})$，属于自举。**TD方法**既使用单个样本转换（采样），又使用当前估计值（自举），是两者的结合。

> **Note**
>
> DP 用于**策略评估，目标是计算给定策略 π 的状态值函数 $v_\pi(s)$。该值函数表示从状态 s 开始，遵循策略 π 所能获得的长期期望回报。**DP **假设环境模型 $p(s', r \mid s, a)$ 已知且完整**。这意味着期望值 $\mathbb{E}_\pi[\cdot]$ 可被**精确计算**，无需采样估计。
>
> > **“DP 的目标之所以是一个'估计值' 则不是因为期望值的原因,其会假设由环境模型完整地提供期望值”**
>
> DP 中的"估计值"属性**并非源于对期望值 $\mathbb{E}_\pi[\cdot]$ 的估计**，因为环境模型已知，期望值可**精确计算**。在 Bellman 更新公式中，$\sum_{s',r} p(s', r \mid s, a)\,[r + \gamma V_k(s')]$ 是**精确的期望计算**。由于 $p(s', r \mid s, a)$ 已知，无需用样本“猜测”期望。例如，在已知规则的棋盘游戏中，DP 可直接计算所有可能的下一状态和奖励的加权平均（权重为转移概率），该计算是确定性的、无噪声的。**因此，期望值部分不是“估计”的而是精确的。**
>
> > **“真正的原因是因为真实的 $v_\pi(S_{t+1})$ 是未知的,因此要使用当前的估计值 $V(S_{t+1})$ 来替代。”**
>
> DP 的目标（即计算出的 $V(s)$）是“估计值”，**根本原因在于值函数 v_π 本身未知**。我们不知真实的 $v_\pi(s)$，故在迭代中必须用当前估计 $V_k(s')$ 替代真实的 $v_\pi(s')$。Bellman 方程要求：$v_\pi(s)$ 依赖于**未来的状态值** $v_\pi(S_{t+1})$。但问题在于：**真实的 $v_\pi(S_{t+1})$ 是未知的。**我们正在计算整个 v_π，因此它无法预先知晓。因此，在 DP 迭代更新中我们用**当前的估计值** $V_k(s')$ 代替**真实的** $v_\pi(s')$。例如，更新 $V_{k+1}(s)$ 时，公式中的 $V_k(s')$ 是上一轮的估计，而非真实值。这导致：初始时 $V_0(s)$ 是任意猜测（如全零），误差很大。每次迭代，$V_k(s)$ 逐步逼近真实值 $v_\pi(s)$，但**在收敛前，它始终是估计值**。仅当迭代无限进行（理论上收敛），$V_k(s)$ 才等于 $v_\pi(s)$。
>
> 这样做的原因是**值函数的递归依赖性**，$v_\pi(s)$ 依赖于 $v_\pi(s')$，而 $v_\pi(s')$ 又依赖于 $v_\pi(s'')$，形成循环。无法直接求解，只能通过迭代“猜测”未来值。同时也因为**DP 的自举（bootstrapping）本质**，真实更新应为：$v_\pi(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(S_{t+1})\right]$，但 DP 实际执行 $V_{k+1}(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma V_k(S_{t+1})\right]$，用 $V_k(S_{t+1})$（估计值）替代了 $v_\pi(S_{t+1})$（真实值）。

样本更新和期望更新

TD 和蒙特卡洛的更新称为**样本更新**，基于一个样本后继状态（或状态-动作对），使用实际观察到的奖励和后继状态价值来计算备份值。DP 使用**期望更新**，基于所有可能后继状态及其概率分布进行加权平均。因此，TD 和蒙特卡洛属于基于经验的方法，而 DP 需要完整模型。

TD误差

TD 更新中括号内的项表示估计误差：

$$\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \tag{6.5}$$

δ_t 是对 $V(S_t)$ 的误差估计。它依赖于 R_{t+1} 和 S_{t+1}，因此直到 $t+1$ 时刻才可计算。即：δ_t 在时间 $t+1$ 可用。

我们希望证明：在价值函数 V 不随时间变化的前提下，从状态 S_t 开始的蒙特卡洛误差 $G_t - V(S_t)$ 可以表示为从时间 t 到 $T-1$ 所有TD误差 δ_k 的折扣加权和，即：

$$G_t - V(S_t) = \sum_{k=t}^{T-1} \gamma^{k-t} \delta_k$$

$$G_t = R_{t+1} + \gamma G_{t+1}$$

代入蒙特卡洛误差 $G_t - V(S_t)$ 中，得到：

$$G_t - V(S_t) = (R_{t+1} + \gamma G_{t+1}) - V(S_t)$$

TD误差 δ_t 的定义是：

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

为了引入 δ_t，在表达式中添加并减去 $\gamma V(S_{t+1})$，从而将 $R_{t+1} + \gamma G_{t+1} - V(S_t)$ 重新组织为两个部分：

- 一部分是 $R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$，这正是 δ_t；
- 另一部分是 $\gamma G_{t+1} - \gamma V(S_{t+1}) = \gamma(G_{t+1} - V(S_{t+1}))$。

因此，整个误差可以写成：

$$G_t - V(S_t) = \delta_t + \gamma(G_{t+1} - V(S_{t+1}))$$

同理有：

$$G_{t+1} - V(S_{t+1}) = \delta_{t+1} + \gamma(G_{t+2} - V(S_{t+2}))$$

将这个结果代回前一步的表达式中：

$$G_t - V(S_t) = \delta_t + \gamma\left(\delta_{t+1} + \gamma(G_{t+2} - V(S_{t+2}))\right)$$

展开后得到：

$$G_t - V(S_t) = \delta_t + \gamma\delta_{t+1} + \gamma^2(G_{t+2} - V(S_{t+2}))$$

上述过程可以不断重复。在每一步，都将剩余的误差项 $G_{t+k} - V(S_{t+k})$ 分解为当前的TD误差 δ_{t+k} 和下一个时间点的误差 $\gamma(G_{t+k+1} - V(S_{t+k+1}))$。

经过 $T-t$ 次这样的展开，会到达序列的最后一个非终止状态 S_{T-1}。此时：

$$G_{T-1} - V(S_{T-1}) = \delta_{T-1} + \gamma(G_T - V(S_T))$$

由于 S_T 是终止状态，根据定义，其价值 $V(S_T) = 0$，且在终止后不再有奖励，因此 $G_T = 0$。于是：

$$G_T - V(S_T) = 0 - 0 = 0$$

这意味着最后一项 $\gamma^{T-t}(G_T - V(S_T)) = 0$，它不会对总和产生任何贡献。

将所有中间步骤中的TD误差项收集起来，我们得到：

$$G_t - V(S_t) = \delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2} + \cdots + \gamma^{T-t-1}\delta_{T-1}$$

这正是从 $k = t$ 到 $k = T-1$ 的求和形式：

$$G_t - V(S_t) = \sum_{k=t}^{T-1} \gamma^{k-t}\delta_k \tag{6.6}$$

练习

> 分析：当值函数 V 在幕（episode）中发生变化时，式 (6.6)只能近似成立的原因，并推导出需要在 TD 误差之和上添加的额外项，使得等式右侧仍等于左侧的蒙特卡洛误差。

首先，明确相关公式：

- TD 误差公式 (6.5)：$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$
- TD 更新公式 (6.2)：$V(S_t) \leftarrow V(S_t) + \alpha\delta_t$
- 式 (6.6)：$\sum_{t=0}^{T-1} \gamma^t \delta_t = G_0 - V(S_0)$

其中 $G_0 = \sum_{k=0}^{T-1} \gamma^k R_{k+1}$ 是从初始状态 S_0 开始的蒙特卡洛目标（总回报）。

当 V 在幕中保持不变时，式 (6.6) 精确成立。但当 V 在幕中发生变化时（例如，使用函数近似或状态被重复访问），我们需要重新推导。

定义 V_t 为在时间 t 时用于计算 TD 误差 δ_t 和执行 TD 更新的**值函数数组**。

TD 误差定义为：

$$\delta_t = R_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t)$$

需要计算加权 TD 误差之和：

$$\sum_{t=0}^{T-1} \gamma^t \delta_t = \sum_{t=0}^{T-1} \gamma^t \left(R_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t)\right)$$

展开此式：

$$\sum_{t=0}^{T-1} \gamma^t \delta_t = \sum_{t=0}^{T-1} \gamma^t R_{t+1} + \sum_{t=0}^{T-1} \gamma^{t+1} V_t(S_{t+1}) - \sum_{t=0}^{T-1} \gamma^t V_t(S_t)$$

注意到 $\sum_{t=0}^{T-1} \gamma^t R_{t+1} = G_0$，因此：

$$\sum_{t=0}^{T-1} \gamma^t \delta_t = G_0 + \sum_{t=0}^{T-1} \gamma^{t+1} V_t(S_{t+1}) - \sum_{t=0}^{T-1} \gamma^t V_t(S_t)$$

现在，分析求和项：

$$\sum_{t=0}^{T-1} \gamma^t V_t(S_t) = \gamma^0 V_0(S_0) + \gamma^1 V_1(S_1) + \gamma^2 V_2(S_2) + \cdots + \gamma^{T-1} V_{T-1}(S_{T-1})$$

$$\sum_{t=0}^{T-1} \gamma^{t+1} V_t(S_{t+1}) = \gamma^1 V_0(S_1) + \gamma^2 V_1(S_2) + \cdots + \gamma^T V_{T-1}(S_T)$$

两式相减有：

$$\sum_{t=0}^{T-1}\gamma^{t+1}V_t(S_{t+1})-\sum_{t=0}^{T-1}\gamma^t V_t(S_t)=-\gamma^0 V_0(S_0)+\sum_{t=1}^{T-1}\gamma^t(V_{t-1}(S_t)-V_t(S_t))+\gamma^T V_{T-1}(S_T)$$

代入原式：

$$\sum_{t=0}^{T-1}\gamma^t\delta_t=G_0-V_0(S_0)+\sum_{t=1}^{T-1}\gamma^t(V_{t-1}(S_t)-V_t(S_t))+\gamma^T V_{T-1}(S_T)$$

假设终止状态 S_T 的值为 0（即 $V_{T-1}(S_T)=0$），则 $\gamma^T V_{T-1}(S_T)=0$，因此：

$$\sum_{t=0}^{T-1}\gamma^t\delta_t=G_0-V_0(S_0)+\sum_{t=1}^{T-1}\gamma^t(V_{t-1}(S_t)-V_t(S_t))$$

由上式可知，当 V 在幕中变化时，式 (6.6) 两侧的差异为：

$$\sum_{t=0}^{T-1}\gamma^t\delta_t-(G_0-V_0(S_0))=\sum_{t=1}^{T-1}\gamma^t(V_{t-1}(S_t)-V_t(S_t))$$

为了让等式右侧仍等于左侧的蒙特卡洛误差 $G_0-V_0(S_0)$，我们需要在 TD 误差之和 $\sum_{t=0}^{T-1}\gamma^t\delta_t$ 上添加额外项：

$$-\sum_{t=1}^{T-1}\gamma^t(V_{t-1}(S_t)-V_t(S_t))=\sum_{t=1}^{T-1}\gamma^t(V_t(S_t)-V_{t-1}(S_t))$$

实例 通勤

这个例子通过一个日常生活的场景——下班回家的通勤预测——来直观地说明**蒙特卡洛方法**和**时序差分（TD）方法**在策略评估中的不同行为。

- 每次从办公室回家，你都会根据当前情况（时间、天气、交通等）**预测回家所需的总时间**。
- 这是一个**策略评估问题**：你遵循一个固定的"策略"（比如走固定路线），目标是学习每个状态（如"离开办公室"、"驶出高速"等）下的**预期剩余时间**，即该状态的价值 $V(s)$。
- 状态包括：时间、地点、天气、交通状况等影响因素。
- 奖励定义为**每段旅程的实际耗时**（以分钟为单位）。
- 不进行折扣，即 $\gamma=1$。因此，从某个状态出发的**回报** G_t 就是从该状态到家的实际剩余时间。
- 每个状态的**真实价值** $v_\pi(s)$ 是在该状态下回家的**期望剩余时间**，而 $V(s)$ 是你当前的估计。

以下是某次具体通勤的完整记录：

状态	已用时间（分钟）	预计剩余时间（分钟）	预计总时间（分钟）
星期五6点离开办公室	0	30	30
到达汽车，下雨	5	35	40
驶出高速公路	20	15	35
在路上堵在卡车后面	30	10	40
开到居住的街道	40	3	43
到家	43	—	43

- **总耗时**：43分钟。

- 在“驶出高速公路”时，你预计只需15分钟到家，但实际用了23分钟（从20分钟到43分钟）。
- 在“进入家街”时，你预计还需3分钟，实际也用了3分钟，预测准确。

蒙特卡洛（MC）方法必须等到**整个旅程结束**，才能知道从每个状态出发的**实际回报** G_t，然后用这个回报来更新对该状态的预测。

- **更新规则**（常步长-α蒙特卡洛）：

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[G_t - V(S_t)\Big] \tag{6.1}$$

  其中 G_t 是从 S_t 到家的实际剩余时间。

- **以“驶出高速公路”状态为例**：
  - 当时你估计剩余时间 $V(S_t) = 15$ 分钟。
  - 实际从该点到家用了 $43 - 20 = 23$ 分钟，即 $G_t = 23$。
  - 误差为 $G_t - V(S_t) = 23 - 15 = 8$ 分钟。
  - 若步长 $\alpha = 0.5$，则更新为：

$$V(\text{驶出高速}) \leftarrow 15 + 0.5 \times 8 = 19$$

    即预测值向上调整4分钟。

- **关键限制**：
  - 所有这些更新**只能在到家之后进行。**
  - 在你还在路上时，即使你意识到预测偏差（如被卡车挡住），也无法更新之前的估计。
  - 更新是**离线的**（offline），必须等待“幕”结束。

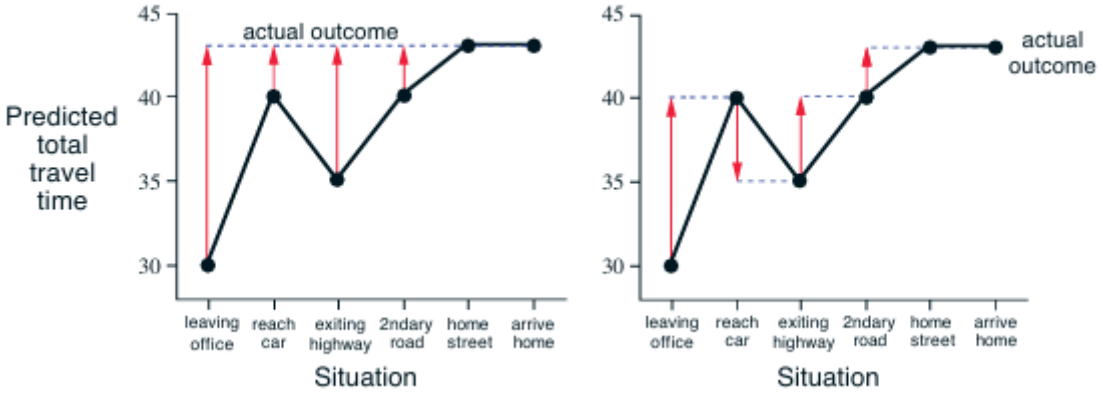


图6.1（左）说明横轴表示状态序列，纵轴表示“预计总时间”。红色箭头表示蒙特卡洛方法建议的更新方向（当 $\alpha = 1$ 时，箭头直接指向实际总耗时43分钟）。每个箭头的长度就是 $G_t - V(S_t)$，即预测误差。

TD方法不需要等待旅程结束。它在**每一步之后立即进行更新**，利用当前观察到的奖励和对下一状态的预测来构造一个目标。

- **更新规则**（TD(0)）：

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big] \tag{6.2}$$

  由于 $\gamma = 1$，目标为 $R_{t+1} + V(S_{t+1})$，即“已用时间 + 下一状态的预计剩余时间”。

- **以“离开办公室”状态为例**：
  - 初始估计：$V(\text{离开办公室}) = 30$ 分钟。

- 到达汽车时，已用5分钟（$R_{t+1} = 5$），新估计剩余35分钟（$V(S_{t+1}) = 35$）。
- TD目标为 $5 + 35 = 40$ 分钟。
- 误差为 $40 - 30 = 10$ 分钟。
- 若 $\alpha = 1$，则立即更新：

$$V(\text{离开办公室}) \leftarrow 30 + 1 \times 10 = 40$$

即初始预测从30分钟上调至40分钟。

- **TD更新是在线的**（online）：
  - 在你到达汽车的那一刻，就已经开始学习并调整对“离开办公室”状态的预测。
  - 无需等到回家。

图6.1（右）说明红色箭头表示TD方法建议的更新（当 $\alpha = 1$ 时）。每个箭头的方向是**从当前预测指向“下一步预测”**。例如，从“离开办公室”的30分钟指向“到达汽车”后的40分钟。这种变化反映了**预测的时序差异**（temporal difference），即相邻状态预测之间的不一致。

练习2

> 思考一个场景，其中TD更新在平均意义上优于蒙特卡洛更新。

当环境发生**局部变化**时，TD方法可以立即利用**已知的后续部分知识**进行学习，而蒙特卡洛方法必须等待整个幕结束才能更新。这种"知识迁移"能力使TD在环境部分改变时显著更高效。

初始情况

- **旧路线经验**（已学习充分）：
  - 旧办公楼 → 高速入口：5分钟
  - 高速入口 → 家：25分钟
  - 总时间：30分钟（稳定估计）
- **新情况**：
  - 搬到新办公楼，新路线：
    - 新办公楼 → 高速入口：15分钟（未知）
    - 高速入口 → 家：仍为25分钟（已知，不变）
    - 总时间：40分钟（真实值）

第一次尝试回家

状态	消耗时间	估计剩余时间	估计总时间	实际后续时间
新办公楼出发	-	30	30	-
到达高速入口	15	25	40	25（已知）
到家	40	0	40	-

蒙特卡洛方法

- **更新时机**：必须等到到家（40分钟）后才能更新
- **第一次尝试后的更新**：
  - 从新办公楼出发：$V(\text{新办公楼}) \leftarrow 30 + \alpha(40 - 30)$

    - 仍需要完整经历一次出行才能进行任何更新
- **问题**：
    - 无法利用"高速入口→家"的已知知识
    - 即使到达高速入口时已知后续需要25分钟，也无法立即调整对新路段的估计
    - 学习效率低，需要多次完整出行才能准确估计

TD方法

- **更新时机**：到达高速入口时即可更新
- **关键更新点**（到达高速入口时）：
    - 当前TD误差：$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t) = 15 + 1 \times 25 - 30 = 10$
    - 更新：$V(\text{新办公楼}) \leftarrow 30 + \alpha \times 10$
- **优势**：
    - 到达高速入口时（出行中途）就能利用已知的"高速入口→家"部分知识
    - 立即对新路段进行合理估计，无需等待到家
    - 下一次出行开始时，估计已显著改进

为什么TD更好

假设 $\alpha = 0.5$：

尝试次数	蒙特卡洛方法估计	TD方法估计	说明
初始	30	30	两者相同
第1次中途(高速入口)	仍为30	35	TD已开始更新
第1次结束	35	35	两者更新到相同值
第2次中途(高速入口)	仍为35	37.5	TD继续提前更新
第2次结束	37.5	37.5	两者更新到相同值

- **关键区别**：TD方法在每次出行的**中途**就已经开始学习和调整，而蒙特卡洛必须等到结束
- **收敛速度**：TD方法实际上比蒙特卡洛方法多进行了一次有效更新（每次出行有两次更新机会：中途和结束，而非仅结束）
- **知识利用**：TD充分利用了环境的马尔可夫性质和已知的稳定部分

在环境发生**局部变化**的场景中（如搬到新办公楼但保持部分路线不变），TD方法通过**立即利用已知的后续部分知识**进行学习，显著优于必须等待完整幕结束的蒙特卡洛方法。这种优势源于TD方法对马尔可夫性质的充分利用和"自举"(bootstrapping)特性，使其在现实世界中频繁遇到的部分环境变化情况下学习效率更高。

TD预测的优势

TD方法通过**使用现有估计来更新其他估计**，实现“从一个猜测学习另一个猜测”，这一过程称为**自举**。相对于TP而言，DP方法依赖于**完整的环境模型**，即需要知道状态转移概率 $p(s', r \mid s, a)$ 和奖励函数。TD方法**不需要环境模型**，可以直接从实际经验（状态、奖励、后继状态的序列）中学习。这使得TD方法适用于**模型未知或难以建模的现实任务**。**蒙特卡洛方法**则必须收集从 S_t 到终止状态的完整轨迹，才能计算

G_t 并更新 $V(S_t)$。**TD方法**在 $t+1$ 时刻，利用 R_{t+1} 和 $V(S_{t+1})$ 构造目标 $R_{t+1}+\gamma V(S_{t+1})$，立即更新 $V(S_t)$。

这种"即时更新"的特性在以下场景中至关重要：

- **长序列任务**：某些任务的序列非常长（如复杂的棋类游戏），将所有学习延迟到结束会导致学习速度极慢。
- **持续性任务（Continuing Tasks）**：有些任务没有明确的起点和终点（如机器人持续巡逻），不存在“序列”的概念。蒙特卡洛方法无法直接应用，而TD方法天然适用。
- **探索性动作的影响**：在某些蒙特卡洛方法中，如果序列中包含了探索性动作（非策略动作），整个序列可能被忽略或折扣，这会显著减慢学习。而TD方法**从每一个转换（transition）中学习**，无论后续动作如何，因此对探索的敏感性更低。

尽管TD方法基于“猜测”进行更新，但它**是可靠的**，并且能收敛到正确答案。对于任何固定策略 π，**TD(0)** 已被证明会收敛到真实价值函数 v_π。若使用**常数步长参数** α，则在**均值意义下收敛**。若步长参数 α_t 满足随机逼近的条件（即 $\sum \alpha_t = \infty, \sum \alpha_t^2 < \infty$），则以**概率1收敛**。

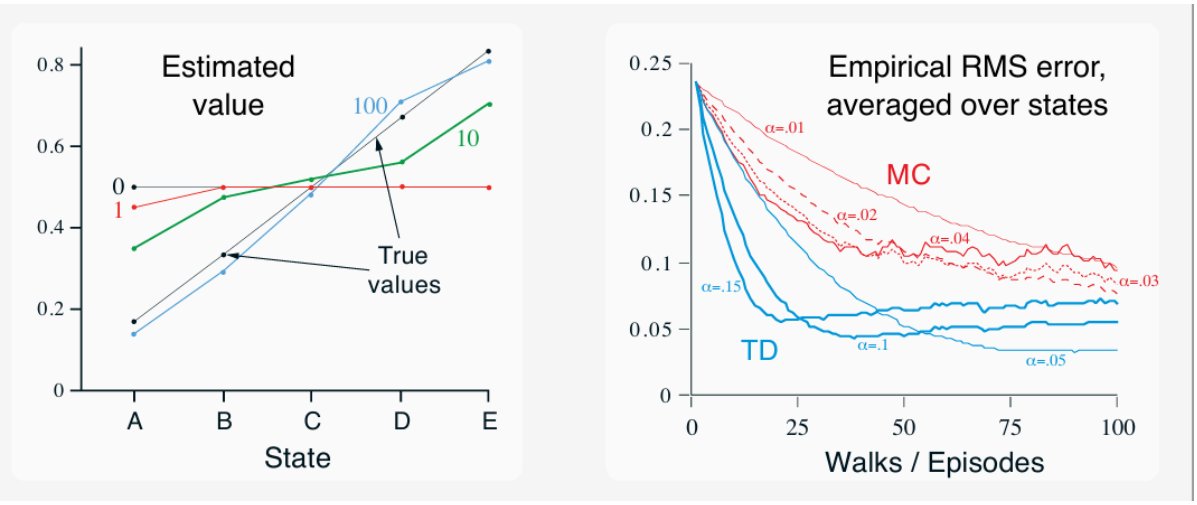
当TD和蒙特卡洛方法都能收敛于正确的预测是，按当前并没有有效的方法证明哪一个方法更快。

实例 随机游走

- **马尔可夫奖励过程（MRP）**：一个没有动作的MDP，用于专注于预测问题。
- **状态空间**：五个非终止状态 A, B, C, D, E，以及两个终止状态（左端和右端）。
- **动态**：
  - 所有序列从中心状态 C 开始。
  - 每一步，以 0.5 的概率向左或向右移动一个状态。
  - 到达最左端或最右端时，序列终止。
- **奖励**：
  - 仅在**从非终止状态进入右端终止状态时**获得奖励 +1。
  - 所有其他转移的奖励为 0。
- **无折扣**：$\gamma = 1$。
- **价值函数的含义**：每个状态的真实价值 $v(s)$ 等于从该状态出发，最终在右侧终止的**概率**。

由于任务对称且无折扣，真实价值如下：

- $v(A) = 1/6$
- $v(B) = 2/6 = 1/3$
- $v(C) = 3/6 = 0.5$
- $v(D) = 4/6 = 2/3$
- $v(E) = 5/6$



实验比较了 **TD(0)** 和 **常步长-α蒙特卡洛** 两种方法。

- **左侧图表**：显示了TD(0)在单次运行中，经过不同数量的序列后学习到的价值。
  - 初始化：所有状态 $V(s) = 0.5$。
  - 使用常数步长 $\alpha = 0.1$。
  - 学习曲线会围绕真实值波动，因为α是常数，不会完全收敛。
- **右侧图表**：显示了两种方法的**学习曲线**。
  - **性能度量**：均方根误差（RMS），计算方式为：

$$\text{RMS} = \sqrt{\frac{1}{5} \sum_{s \in \{A,B,C,D,E\}} (v(s) - V(s))^2}$$

    然后对100次独立运行取平均。
  - **结果**：在所有测试的 α 值下，**TD方法的RMS误差均低于蒙特卡洛方法**，表明TD在此任务上学习得更快、更稳定。

练习

> 在随机游走例子的左图中，第一幕只导致 $V(A)$ 发生了变化。由此你觉得第一幕发生了什么？为什么只有这一个状态的估计值改变了？它的值改变了多少？

第一幕的路径是：**C → B → A → 左端终止**

在 TD(0) 算法中，状态值更新基于 TD 误差：
$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$

更新规则为：
$V(S_t) \leftarrow V(S_t) + \alpha \delta_t$

在这个特定场景中（初始值 $V(s) = 0.5$ 对所有状态，$\gamma = 1$，$\alpha = 0.1$）：

1. **C → B 转移**：
   - $R_{t+1} = 0$（非终止转移）
   - $V(C) = 0.5$，$V(B) = 0.5$
   - $\delta = 0 + 1 \times 0.5 - 0.5 = 0$
   - 更新：$V(C) \leftarrow 0.5 + 0.1 \times 0 = 0.5$（**无变化**）
2. **B → A 转移**：
   - $R_{t+1} = 0$

- $V(B) = 0.5$，$V(A) = 0.5$
- $\delta = 0 + 1 \times 0.5 - 0.5 = 0$
- 更新：$V(B) \leftarrow 0.5 + 0.1 \times 0 = 0.5$（**无变化**）

3. **A → 左端终止转移**：
   - $R_{t+1} = 0$（进入左端终止无奖励）
   - $V(A) = 0.5$，$V(\text{终止}) = 0$
   - $\delta = 0 + 1 \times 0 - 0.5 = -0.5$
   - 更新：$V(A) \leftarrow 0.5 + 0.1 \times (-0.5) = 0.45$（**发生变化**）

关键原因：

- 前两个转移的 TD 误差为 0，因为初始估计值对称（均为 0.5）
- 只有最后一个转移（A → 左端终止）的 TD 误差非零（-0.5）
- **只有产生非零 TD 误差的转移才会导致状态值更新**
- 这体现了 TD 方法的特性：它只更新那些预测与后续预测不一致的状态

改变量是多少?

$V(A)$ 的值从 0.5 更新为:
$V(A) \leftarrow 0.5 + 0.1 \times (-0.5) = 0.5 - 0.05 = 0.45$

因此，$V(A)$ **减少了 0.05。**

> 在随机游走例子的右图中,具体结果与步长参数α的值是相关的。如果α从一个更宽广的值域中取值,会影响哪种算法更好的结论吗?是否存在另一个固定的 α , 使 得两种算法都能表现出比图中更好的性能?为什么?

1. α 取值范围扩大是否影响"TD 更好"的结论?

不会根本改变结论，但会影响优势程度：

- **在马尔可夫任务中**，TD 方法理论上应始终优于或等于蒙特卡洛方法
- **极小 α 值**（如 $\alpha = 0.01$）：
  - 两者都学习缓慢，但 TD 仍略优
  - 优势可能不太明显，因为学习速度都较慢
- **极大 α 值**（如 $\alpha = 0.5$）：
  - TD 可能因振荡而性能下降
  - 蒙特卡洛受高方差影响更大，性能下降更严重
  - TD 优势可能缩小，但通常仍保持
- **极端 α 值**（如 $\alpha = 1.0$）：
  - TD 可能不稳定，但蒙特卡洛方差问题更严重
  - 在随机游走这种简单任务中，TD 通常仍表现更好

> **理论依据**：在马尔可夫环境中，TD 方法利用了马尔可夫性质，其更新具有更低的方差，这是本质优势，不依赖于特定 α 值。

2. 是否存在更好的固定 α 值?

存在，但对两种算法的最优 α 不同：

- **对 TD 方法**：
  - 最优 α 通常在 0.05-0.15 之间（取决于任务）
  - 在随机游走任务中，$\alpha \approx 0.1$ 可能接近最优
  - 理由：平衡学习速度和稳定性
- **对蒙特卡洛方法**：
  - 最优 α 通常更小（如 0.01-0.05）
  - 理由：蒙特卡洛回报方差高，需要更小的步长来平滑更新
- **不存在单一 α 使两者都达到最优**：
  - TD 和蒙特卡洛有不同的**偏差-方差权衡**
  - TD 有偏差但方差低，适合稍大的 α
  - 蒙特卡洛无偏但方差高，需要更小的 α
  - 这种根本差异意味着它们对 α 的敏感度不同

3. 为什么?

- **TD 方法特性**：
  - 利用马尔可夫性质，通过"自举"(bootstrapping)减少方差
  - 适合稍大的 α，因为更新本身已经相对稳定
  - 在随机游走这种强马尔可夫任务中，优势尤为明显
- **蒙特卡洛方法特性**：
  - 无偏但高方差（因为依赖完整序列）
  - 需要更小的 α 来平滑高方差的回报估计
  - 在非马尔可夫任务中可能有优势，但在本任务中处于劣势
- **根本原因**：
  - 两种算法有不同的理论基础和统计特性
  - TD 更适合马尔可夫环境，这是随机游走任务的本质特性
  - 不存在"万能"的 α 值能同时优化两种不同机制的算法

总结：在随机游走任务中，TD 方法在大多数合理的 α 值下都优于蒙特卡洛。虽然可以找到各自算法的最优 α，但不存在一个 α 值能让两种算法同时达到理论最优性能，因为它们的更新机制和统计特性本质不同。

> 在随机游走例子的右图中, TD 方法的均方根误差先下降后上升, 尤其在 α 较大时更明显。这是由什么导致的? 你认为这种情况总是会发生,还是与近似价值函数如何 初始化相关?

1. 为什么误差先降后升?

这是由**常数步长 α** 导致的**稳态振荡**现象：

- **初始阶段**（误差下降）：
  - 初始估计远离真实值
  - TD 更新使估计快速接近真实值
  - 均方根误差迅速减小
- **后期阶段**（误差上升）：

- 估计值接近真实值后
- 由于**常数步长 α 不衰减**
- 每次更新仍有一定幅度
- 导致在真实值附近持续振荡
- 这种振荡的平均幅度可能大于接近过程中的误差，导致均方根误差上升

2. α 较大时为什么更明显?

- **α 直接控制更新幅度**:
  - α 越大，每次更新的步长越大
  - 导致更大的振荡幅度
  - 稳态误差（振荡幅度）与 α 正相关
- **数学解释**:
  - 对于常数步长 α，TD(0) 的稳态均方误差近似为:

$$\text{MSE} \approx \frac{\alpha}{2-\alpha} \cdot \text{方差项}$$

  - 当 α 接近 1 时，MSE 会显著增大
  - 这就是为什么 $\alpha = 0.15$ 的曲线比 $\alpha = 0.05$ 的曲线更早上升

3. 这种情况是否总是发生?

在以下条件下总是发生:

- **使用常数步长 α**（不随时间衰减）
- **任务有随机性**（本任务中转移是随机的）
- **进行足够多次更新**

> 如果 α 随时间衰减（如 $\alpha_t = 1/t$），则会收敛到真实值，误差最终会降至 0。但本实验使用**常数步长**，因此必然出现稳态振荡。

4. 是否与初始化相关?

与初始化有一定关系，但不是根本原因:

- **初始化影响初期行为**:
  - 如果初始值恰好等于真实值，误差会立即上升（因为随机波动）
  - 如果初始值远离真实值，会先有下降阶段
- **长期行为主要由 α 决定**:
  - 无论初始化如何，常数步长 TD 最终都会达到相似的稳态振荡
  - 稳态误差大小主要取决于 α 和任务特性，而非初始化
- **具体到本任务**:
  - 所有状态初始化为 $V(s) = 0.5$
  - 这恰好是状态 C 的真实值，但其他状态有误差
  - 初始误差分布导致明显的先降后升曲线
  - 但即使初始化不同，只要 α 不衰减，长期仍会出现振荡

5. 数学解释

考虑一个简单状态的价值更新：

$$V_{t+1}(s) = V_t(s) + \alpha\delta_t$$

其中 $\delta_t = R_{t+1} + \gamma V_t(s') - V_t(s)$

当 $V_t(s)$ 接近真实值 $v(s)$ 时，δ_t 接近 0，但仍有随机波动。常数步长 α 会使这些波动持续影响估计值，导致：

$$\lim_{t\to\infty} \mathbb{E}[(V_t(s) - v(s))^2] = \text{非零常数}$$

这个非零极限就是稳态误差，它随 α 增大而增大。

TD 方法的均方根误差先降后升是**常数步长 α 导致的稳态振荡**的结果，这在使用常数步长的随机近似算法中是普遍现象。α 越大，振荡越剧烈，误差上升越明显。虽然初始化会影响曲线的具体形状，但只要使用常数步长且任务有随机性，这种现象就会发生。

在随机游走任务中，状态 A~E 的真实价值分别为 $\frac{1}{6}$, $\frac{2}{6}$, $\frac{3}{6}$, $\frac{4}{6}$ 和 $\frac{5}{6}$。以下是计算这些值的至少两种方法：

方法一：解 Bellman 方程组

步骤：

1. 为每个状态写出 Bellman 方程：

$$\begin{aligned}
v(A) &= 0.5 \times 0 + 0.5 \times v(B) \\
v(B) &= 0.5 \times v(A) + 0.5 \times v(C) \\
v(C) &= 0.5 \times v(B) + 0.5 \times v(D) \\
v(D) &= 0.5 \times v(C) + 0.5 \times v(E) \\
v(E) &= 0.5 \times v(D) + 0.5 \times 1
\end{aligned}$$

2. 将方程整理为线性系统：

$$\begin{aligned}
v(A) - 0.5v(B) &= 0 \\
-0.5v(A) + v(B) - 0.5v(C) &= 0 \\
-0.5v(B) + v(C) - 0.5v(D) &= 0 \\
-0.5v(C) + v(D) - 0.5v(E) &= 0 \\
-0.5v(D) + v(E) &= 0.5
\end{aligned}$$

3. 解这个线性方程组（可使用矩阵求逆或高斯消元法）：

$$\begin{pmatrix}
1 & -0.5 & 0 & 0 & 0 \\
-0.5 & 1 & -0.5 & 0 & 0 \\
0 & -0.5 & 1 & -0.5 & 0 \\
0 & 0 & -0.5 & 1 & -0.5 \\
0 & 0 & 0 & -0.5 & 1
\end{pmatrix}
\begin{pmatrix}
v(A) \\ v(B) \\ v(C) \\ v(D) \\ v(E)
\end{pmatrix}
=
\begin{pmatrix}
0 \\ 0 \\ 0 \\ 0 \\ 0.5
\end{pmatrix}$$

4. 解得：

$$v(A) = \frac{1}{6},\ v(B) = \frac{2}{6},\ v(C) = \frac{3}{6},\ v(D) = \frac{4}{6},\ v(E) = \frac{5}{6}$$

优点：精确、系统化，适用于任意 MRP
缺点：当状态空间大时，计算复杂

方法二：概率分析法

思路：价值函数 $v(s)$ 表示从状态 s 出发，最终从右侧终止的概率。

步骤：

1. 观察到状态空间有 7 个"位置"：左终止 + A + B + C + D + E + 右终止
2. 从状态 A 出发，需要向右移动 5 步才能到右终止，向左移动 0 步到左终止
3. 由于左右移动概率相等，到达右终止的概率 = (到右终止的步数) / (总步数)

$$v(A) = \frac{1}{1+5} = \frac{1}{6}$$

（解释：从 A 到右终止有 5 步，到左终止有 1 步，总"权重"为 6）

4. 一般化：
   - 状态 A（第 1 个非终止状态）：$v(A) = \frac{1}{6}$
   - 状态 B（第 2 个非终止状态）：$v(B) = \frac{2}{6}$
   - ...
   - 状态 E（第 5 个非终止状态）：$v(E) = \frac{5}{6}$
5. 公式化：

$$v(S_i) = \frac{i}{6}, \quad i = 1, 2, 3, 4, 5 \quad (\text{对应 } A, B, C, D, E)$$

优点：直观、计算简单，无需解方程
缺点：依赖任务的对称性和简单结构

方法三：动态规划迭代

步骤：

1. 初始化：$v_0(s) = 0$ 对所有 s
2. 迭代应用 Bellman 期望备份：

$$v_{k+1}(s) = \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s]$$

3. 具体计算：

$$\begin{aligned}
v_{k+1}(A) &= 0.5 \times 0 + 0.5 \times v_k(B) \\
v_{k+1}(B) &= 0.5 \times v_k(A) + 0.5 \times v_k(C) \\
v_{k+1}(C) &= 0.5 \times v_k(B) + 0.5 \times v_k(D) \\
v_{k+1}(D) &= 0.5 \times v_k(C) + 0.5 \times v_k(E) \\
v_{k+1}(E) &= 0.5 \times v_k(D) + 0.5 \times 1
\end{aligned}$$

4. 重复迭代直到收敛：
   - $k = 0$: $[0, 0, 0, 0, 0]$
   - $k = 1$: $[0, 0, 0, 0, 0.5]$
   - $k = 2$: $[0, 0, 0, 0.25, 0.5]$
   - $k = 3$: $[0, 0, 0.125, 0.25, 0.625]$
   - ...
   - 收敛到：$[\frac{1}{6}, \frac{2}{6}, \frac{3}{6}, \frac{4}{6}, \frac{5}{6}]$

优点：通用性强，适用于任意 MRP
缺点：需要多次迭代，可能较慢

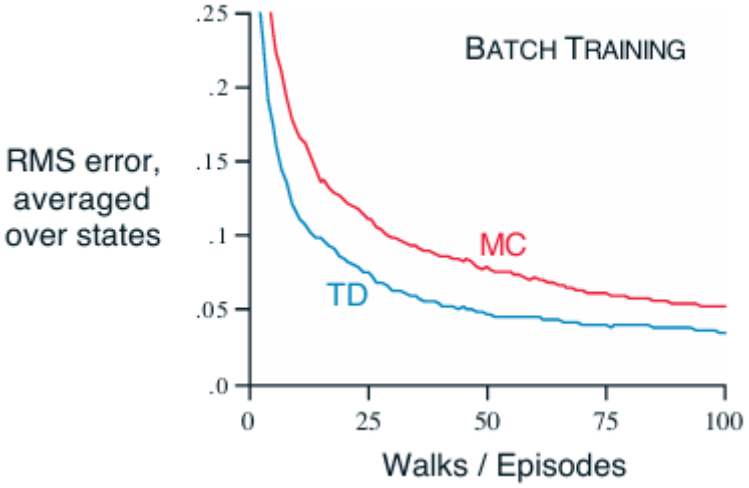
TD(0)的最优性

批量更新

在实际应用中，经验数据往往是有限的（如10个序列或100个时间步）。一种常见的做法是采用**批量更新**（batch updates），将所有可用经验视为一个**批次**（batch）。算法反复遍历这个批次，计算每次访问状态时的增量。**价值函数仅在处理完整个批次后才更新一次**，更新量为所有增量的总和。重复此过程，直到价值函数收敛。正常TD(0)或蒙特卡洛方法在每一步或每个序列结束后立即更新。批量更新则是在**完整数据集上进行多次迭代**，直到收敛。

在批量更新模式下，只要步长参数 α 足够小，**TD(0)** 会**确定性地收敛**到一个唯一的解，且该解与 α 无关。同样条件下，**常步长-α蒙特卡洛方法**也会收敛，但收敛到**不同的解**。

实例 批量TD与批量MD



实验目标为比较在**批量更新模式**下，**TD(0)** 和 **常步长-α蒙特卡洛** 方法在**随机游走任务**上的学习性能。选择价值函数估计与真实值 v_π 之间的**均方根误差（RMS）**作为评价指标，表示随经验量增长的变化趋势。

实验采用批量更新的方式进行训练。所谓"批量更新"，是指每完成一幕（episode），将之前所有幕的数据汇总成一个批次，并基于整个数据集对价值函数进行一次全局更新。具体而言，在每次迭代中，算法遍历所有历史经验，重新计算每个状态的价值更新量，并在该批次结束后统一调整价值参数。这一过程持续进行，直到价值函数趋于稳定。

实验采用一种**增量式批量学习**的方式，模拟智能体在不断获得新经验的同时，反复利用所有历史数据进行学习。每完成一幕（episode），将之前所有幕的数据汇总成一个批次，并基于整个数据集对价值函数进行一次全局更新。

> 首先是**生成一个新剧集**，从状态 C 开始每一步以 0.5 概率向左或向右移动，直到到达最左端或最右端终止；记录完整的状态-奖励序列（如：C, 0, B, 0, A, 0）。然后**构建当前批次**，将**迄今为止观察到的所有剧集**（包括刚生成的这一个）合并成一个**训练数据集**（即"批次"）。例如，在第10次迭代时，批次包含前10个剧集。

对于蒙特卡洛方法，使用首次访问型（first-visit）MC 估计，结合固定的步长 α 进行价值更新。其基本规则为：当某一状态 s 在一幕中被访问后，记录从该时刻起至终幕的实际回报 G_t，然后按照公式

$$V(s) \leftarrow V(s) + \alpha\left[G_t - V(s)\right]$$

进行更新。在批量模式下，这意味着每一幕结束后，收集所有状态及其对应的完整回报序列，形成训练样本集合。随后，算法在整个批次上反复应用上述更新规则，直至收敛。理论上，随着足够多幕的积累，批量 MC 方法所得到的价值估计 $V(s)$ 将趋近于所有经过状态 s 后获得的回报的算术平均值，即

$$V(s) = \frac{1}{N(s)} \sum_{i=1}^{N(s)} G^{(i)}(s)$$

其中 $N(s)$ 是状态 s 在批次中出现的次数，$G^{(i)}(s)$ 是第 i 次访问 s 时对应的实际回报。这种估计具有**最小化训练集中均方回报误差**的性质，因此在经验风险最小化的意义下是最优的。

相比之下，TD(0) 方法采用自举式（bootstrapping）更新机制。其单步更新规则为：

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right]$$

在批量模式下，每一幕完成后，系统会汇总所有时间步的经验 (S_t, R_{t+1}, S_{t+1})，并在每轮迭代中基于整个数据集重新计算 TD 误差并对价值函数进行批量调整。值得注意的是，批量 TD 并不直接最小化状态-回报对之间的均方误差，而是试图满足贝尔曼一致性条件，即期望下一状态价值加上即时奖励等于当前状态价值。

实验共运行 100 次独立重复，每次重复包括若干幕的交互过程。在每一次重复中，分别记录 TD(0) 和 MC 方法在各个训练阶段的价值函数估计结果，并计算它们相对于真实价值函数 $v_\pi(s)$ 的均方根误差（RMSE）：

$$\mathrm{RMSE} = \sqrt{\frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \left[V(s) - v_\pi(s) \right]^2}$$

其中 $\mathcal{S} = \{A, B, C, D, E\}$ 为非终止状态集合。最终的学习曲线由 100 次独立运行的 RMSE 值取平均得到。

实验结果显示，**尽管批量蒙特卡洛方法能够最小化训练数据上的实际回报误差，但其在测试阶段（即逼近真实价值函数）的表现却不如批量 TD 方法。**如图 6.2 所示，随着训练幕数的增加，TD(0) 的 RMSE 下降速度更快，并最终达到更低的稳态误差水平。这表明批量 **TD 方法在泛化能力方面更具优势。**

这一现象看似矛盾：既然 MC 方法最小化了训练集上的均方回报误差，为何其整体表现反而更差？关键在于目标的不同。**蒙特卡洛方法优化的是与观测回报之间的拟合误差，这是一种纯粹的数据驱动估计，适用于离策略评估或当模型未知且无需泛化的情况。**

实例二

考虑一个离散状态马尔可夫收益过程（MRP）：

- **状态空间**：$\mathcal{S} = \{A, B\}$，其中 B 为终止状态（terminal state）。
- **转移动态**：
  - 从状态 A，以概率1转移到 B，即时奖励 $R = 0$。
  - 从状态 B，立即终止，终止奖励 R 服从伯努利分布。
- **折扣因子**：$\gamma = 1$。
- **目标**：估计状态价值函数 $V(s) = \mathbb{E}[\text{return} \mid S_0 = s]$。

实验使用8幕序列数据，每幕以终止状态结束。数据整理如下：

幕编号	序列描述	转移奖励	终止奖励	总回报
1	$A \xrightarrow{R=0} B \rightarrow$ 终止	0	0	0
2	直接从 B 终止	-	1	1
3	直接从 B 终止	-	1	1
4	直接从 B 终止	-	1	1
5	直接从 B 终止	-	1	1
6	直接从 B 终止	-	1	1
7	直接从 B 终止	-	1	1
8	直接从 B 终止	-	0	0

关键统计：

- **状态 B 的终止次数**：8次（包括幕1的终止）。
  - **终止奖励为1**的次数：6次（幕2-7）。
  - **终止奖励为0**的次数：2次（幕1和幕8）。
- **状态 A 的出现次数**：仅1次（幕1），总回报为0。
- **状态转移观察**：
  - $A \rightarrow B$ 转移：仅1次，奖励恒为0。
  - B 无出转移（终止状态）。

实验设置

- **批量学习模式**：使用全部8幕数据一次性计算估计值（非在线更新）。
- **价值函数更新**：
  - MC：直接计算样本平均回报。
  - TD(0)：求解贝尔曼方程 $V = \mathcal{R} + \gamma \mathcal{P} V$，其中 $\mathcal{P}$ 和 $\mathcal{R}$ 为最大似然估计的转移矩阵和奖励向量。
- **评估指标**：
  - 训练集MSE：衡量对已知数据的拟合程度。
  - 泛化能力：基于马尔可夫假设，预测未来数据的误差。

状态 B 的价值估计 $V(B)$

- **蒙特卡洛方法**：
  $V_{\text{MC}}(B) = \frac{\text{所有以 } B \text{ 开始的幕的回报和}}{\text{幕数}} = \frac{6\times1+2\times0}{8} = \frac{6}{8} = \frac{3}{4} = 0.75$
- **TD(0)方法**：
  B 为终止状态，无后续状态，故：
  $V_{\text{TD}}(B) = \mathbb{E}[R_{\text{terminate}}] = \frac{6\times1+2\times0}{8} = 0.75$
- **结论**：两种方法对 $V(B)$ 的估计一致，均为最优解（符合数据最大似然估计）。

状态 A 的价值估计 $V(A)$

蒙特卡洛方法结果

- 状态 A 仅出现1次（幕1），总回报为0。
  $V_{\text{MC}}(A) = \frac{0}{1} = 0$
- **训练集MSE**：
  $\text{MSE}_{\text{MC}} = (0 - V_{\text{MC}}(A))^2 = 0$
  在训练数据上误差为零，完美拟合。

TD(0)方法结果

- 基于马尔可夫转移动态：
  - 从 A 到 B 的转移概率 $P(B \mid A) = 1$（观测1次，转移1次）。
  - 即时奖励 $R(A \to B) = 0$（观测1次，恒为0）。
  - $V(B) = 0.75$（如前所述）。
- 代入贝尔曼方程（$\gamma = 1$）：
  $V_{\text{TD}}(A) = R(A \to B) + \gamma \cdot V_{\text{TD}}(B) = 0 + 1 \times 0.75 = \frac{3}{4} = 0.75$
- **训练集MSE**：
  $\text{MSE}_{\text{TD}} = (0 - V_{\text{TD}}(A))^2 = (0 - 0.75)^2 = 0.5625$
  在训练数据上存在误差，但符合马尔可夫结构。

估计结果对比

方法	$V(A)$	$V(B)$	**训练集MSE**	**是否利用转移结构**
蒙特卡洛	0	0.75	0	否
TD(0)	0.75	0.75	0.5625	是

分析与讨论

- **蒙特卡洛估计（$V(A) = 0$）**：
  - 优点：最小化训练集MSE（误差为0），在已知数据上表现最优。
  - 缺点：忽略马尔可夫结构，仅依赖单一观测。若过程为马尔可夫，此估计过拟合训练数据。
- **TD(0)估计（$V(A) = 0.75$）**：
  - 优点：利用 $A \to B$ 的转移动态和 $V(B)$ 的估计，符合贝尔曼方程。
  - 缺点：训练集MSE较高（0.5625），但泛化能力更强。

批量蒙特卡洛方法总是找出最小化训练集上均方误差的估计,而批量 TD(0) 总是找出完全符合马尔可夫过程模型的最大似然估计参数。通常,一个参数的最大似然估计是使得生成训练数据的概率最大的参数值。

模型参数的最大似然估计

基于多幕观察序列，模型参数可通过最大似然估计（MLE）直观获得：

转移概率估计：从状态 i 转移到状态 j 的概率估计值 $\hat{P}_{ij}$ 为观察数据中从 i 转移到 j 的次数占从 i 出发所有转移次数的比例：

$$\hat{P}_{ij} = \frac{N(i \to j)}{N(i)}$$

其中 $N(i \to j)$ 表示从 i 转移到 j 的观测次数，$N(i)$ 表示从 i 出发的总转移次数。

期望收益估计：状态 i 的期望即时收益估计值 $\hat{R}_i$ 为从 i 出发的所有转移中观察到的收益的平均值：

$$\hat{R}_i = \frac{1}{N(i)} \sum_{\text{transitions from } i} r$$

其中 r 为每次转移的即时收益（可能依赖于下一个状态）。

若模型正确（即真实过程为马尔可夫），此估计可精确计算价值函数 $V(i)$。

确定性等价估计

马尔可夫过程本质上是随机的，而非确定性的。真实的转移概率 P_{ij} 是概率性的（例如，从状态 i 转移到 j 的概率可能为 0.7，转移到 k 的概率为 0.3）。期望收益 R_i 也包含随机性（例如，即时收益 r 可能服从某种分布）。

“确定性等价”特指模型参数的估计方法，而非马尔可夫过程的性质。当我们通过最大似然估计（MLE）从数据中得到 $\hat{P}_{ij}$ 和 $\hat{R}_i$ 后，**假设这些估计值是确定的（即 $\hat{P}_{ij} = P_{ij}$, $\hat{R}_i = R_i$）**，忽略估计的不确定性（如方差或置信区间）。基于这个“确定的”模型，计算价值函数 $V(i)$（例如求解 $V = \hat{R} + \gamma \hat{P} V$）。这种做法被称为 **确定性等价估计**，因为：

> *它等价于将估计参数视为真实参数（确定性），而非近似值。*

基于估计模型，价值函数满足 Bellman 方程：

$$V(i) = \hat{R}_i + \gamma \sum_j \hat{P}_{ij} V(j)$$

其中 γ 为折扣因子。

批量TD(0)收敛到此估计，而TD方法的核心优势在于以$O(n)$内存和迭代更新高效逼近该解，避免了显式计算确定性等价模型所需的$O(n^3)$时间复杂度，从而在大规模状态空间中成为唯一可行的近似方法。

与MD的比较

在**批量学习场景**，TD(0) 比蒙特卡洛方法更快收敛，因为它直接计算确定性等价估计（例如，在随机游走任务中，TD(0) 在批量学习下表现优势）。蒙特卡洛方法需完整幕序列更新，而 TD(0) 利用自举（bootstrapping）更高效利用数据。在**非批量（在线）学习场景**，非批量 TD(0) 不能达到确定性等价估计或最小平方误差估计，但其更新方向大致朝向这些解：

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right]$$

因此，它通常比常数步长 α 的蒙特卡洛方法更快，但目前对在线 TD 和蒙特卡洛方法的效率比较尚无明确理论结论。

但**确定性等价估计存在计算瓶颈**，在状态数 $n = |S|$时，需存储 $n \times n$ 转移概率矩阵，内存复杂度为 $O(n^2)$；解线性系统 $V = R + \gamma P V$（例如通过矩阵求逆），时间复杂度为 $O(n^3)$，直接计算在大型状态空间中不可行（n 大时 n^2 或 n^3 开销过大）。**TD 方法**仅需存储价值函数 V，内存复杂度为 $O(n)$；通过迭代更新（在训练集上反复计算），以较低开销逼近确定性等价解。对于状态空间巨大的任务（如 n 极大），TD 方法是唯一可行的近似确定性等价解的方法，其效率优势显著。

练习

> 设计一个离轨策略版本的TD(0)算法，用于估计任意目标策略π的状态价值函数$V^{\pi}(s)$，其中数据由行为策略b生成。该算法使用重要度采样比$\rho_{t,t}$来调整由不同策略产生的样本分布差异。

初始化

对所有状态 $s \in \mathcal{S}$，任意初始化价值函数 $V(s)$

选择步长参数 $\alpha \in (0, 1]$

选择折扣因子 $\gamma \in [0, 1]$

对于每个幕：

初始化起始状态 S

重复以下步骤直到达到终止状态：

从行为策略 b 选择动作：$A \sim b(\cdot|S)$

执行动作 A，观察奖励 R 和下一状态 S'

计算重要度采样比：$\rho_{t,t} = \frac{\pi(A|S)}{b(A|S)}$

更新价值函数：$V(S) \leftarrow V(S) + \alpha \cdot \rho_{t,t} \cdot \left[R + \gamma \cdot V(S') - V(S)\right]$

将当前状态更新为下一状态：$S \leftarrow S'$

1. **重要度采样比定义**：$\rho_{t,t}$ 是式(5.3)中当$h = t$时的特殊情况，仅考虑当前时刻t的策略差异
2. **策略覆盖条件**：为确保算法有效，行为策略b必须满足"覆盖"目标策略π的条件：
   $\forall s \in \mathcal{S}, \forall a \in \mathcal{A}(s), \quad \pi(a|s) > 0 \implies b(a|s) > 0$
   即如果目标策略可能选择某个动作，则行为策略也必须有一定概率选择该动作
3. **特殊情况处理**：
   - 当 $b(A|S) = 0$ 时（行为策略不可能选择该动作），应跳过此样本（不进行更新）
   - 当 $\pi(A|S) = 0$ 且 $b(A|S) > 0$ 时，$\rho_{t,t} = 0$，表示该样本对目标策略无信息，不更新价值函数
4. **算法特性**：
   - 该算法是**单步离轨策略预测算法**
   - 仅使用一步重要度采样比 $\rho_{t,t}$ 而非完整轨迹的重要性采样
   - 与同轨策略TD(0)相比，增加了重要度采样比作为更新权重

该算法基于以下原理：

- TD误差：$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$
- 在离轨策略学习中，需要使用重要度采样调整期望：
  $\mathbb{E}_b[\rho_{t,t} \cdot \delta_t] = \mathbb{E}_\pi[\delta_t]$
- 因此，使用 $\rho_{t,t} \cdot \delta_t$ 作为TD误差的无偏估计

Sarsa:同轨策略时序差分控制

Sarsa 是一种**同策略**的时序差分控制算法，用于解决强化学习中的**控制问题**（即寻找最优策略）。它遵循**广义策略迭代**（GPI）模式，但使用TD方法替代蒙特卡洛方法进行策略评估作为同策略方法，Sarsa在**评估当前行为策略**的同时**改进该策略。**

与状态价值函数不同，Sarsa学习**动作价值函数** $q_\pi(s, a)$，即在策略 π 下从状态 s 执行动作 a 后的期望回报。Sarsa关注**状态-动作对**之间的转移，而非单纯的状态转移。

一个序列由状态和状态-动作对交替组成：

$$\cdots \begin{pmatrix} S_t \\ A_t \end{pmatrix} \xrightarrow{R_{t+1}} \begin{pmatrix} S_{t+1} \\ A_{t+1} \end{pmatrix} \xrightarrow{R_{t+2}} \begin{pmatrix} S_{t+2} \\ A_{t+2} \end{pmatrix} \xrightarrow{R_{t+3}} \cdots$$

Sarsa使用以下TD更新规则学习动作价值函数：

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big] \tag{6.7}$$

该更新在每次从非终止状态 S_t 转移后进行。如果 S_{t+1} 是终止状态，则定义 $Q(S_{t+1}, A_{t+1}) = 0$。更新基于**五元组** $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$，这也是算法名称"Sarsa"的由来。TD误差定义为：

$$\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$$

Sarsa（同策略TD控制）用于估计 $Q \approx q_*$

算法参数：步长 $\alpha \in (0, 1]$，较小的 $\varepsilon > 0$
初始化：对所有 $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$ 任意初始化 $Q(s, a)$，但要求 $Q(\text{终止状态}, \cdot) = 0$
循环，对每个episode：
　初始化状态 S
　根据由 Q 导出的策略（例如 ε-贪心策略）从 S 中选择动作 A
　执行动作 A，观察奖励 R 和下一状态 S'
　根据由 Q 导出的策略（例如 ε-贪心策略）从 S' 中选择动作 A'
　更新动作价值$Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
　更新状态和动作：$S \leftarrow S'$；$A \leftarrow A'$
直到 S 为终止状态

Sarsa的**收敛性**取决于策略 π 对 Q 函数的依赖方式。只要满足**所有状态-动作对被无限次访问**，**策略最终收敛到贪婪策略**（例如，通过令 $\varepsilon = 1/t$ 的 ε-贪心策略），Sarsa就能以概率1收敛到最优策略和最优动作价值函数。常见的做法是使用 ε-贪心策略或 ε-软策略作为行为策略。

练习

> 证明"状态-动作"二元组版本的 TD 误差 $\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$ 成立。这里假设价值函数在不同时刻不发生改变（即 Q 是固定的）。

要证明"状态-动作"二元组版本的 TD 误差公式成立，我们需要验证当 Q 为真实动作值函数 Q_π 时，TD 误差的条件期望为零。

设 π 为一个固定策略，Q_π 为策略 π 的真实动作值函数，满足贝尔曼方程：

$$Q_\pi(s, a) = \mathbb{E}_\pi\left[R_{t+1} + \gamma Q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\right]$$

对所有状态-动作对 (s, a) 成立，其中 $A_{t+1} \sim \pi(\cdot \mid S_{t+1})$。

考虑 TD 误差：

$$\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$$

在题目假设下（价值函数 Q 固定不变），当 $Q = Q_\pi$ 时，计算条件期望 $\mathbb{E}_\pi[\delta_t \mid S_t, A_t]$：

$$\mathbb{E}_\pi[\delta_t \mid S_t, A_t] = \mathbb{E}_\pi\left[R_{t+1} + \gamma Q_\pi(S_{t+1}, A_{t+1}) - Q_\pi(S_t, A_t) \mid S_t, A_t\right]$$

由于 $Q_\pi(S_t, A_t)$ 在给定 S_t 和 A_t 时为常数，可分离期望：

$$\mathbb{E}_\pi[\delta_t \mid S_t, A_t] = \mathbb{E}_\pi\left[R_{t+1} + \gamma Q_\pi(S_{t+1}, A_{t+1}) \mid S_t, A_t\right] - Q_\pi(S_t, A_t)$$

由贝尔曼方程，有：

$$\mathbb{E}_\pi\left[R_{t+1} + \gamma Q_\pi(S_{t+1}, A_{t+1}) \mid S_t, A_t\right] = Q_\pi(S_t, A_t)$$

代入上式得：

$$\mathbb{E}_\pi[\delta_t \mid S_t, A_t] = Q_\pi(S_t, A_t) - Q_\pi(S_t, A_t) = 0$$

因此，当 $Q = Q_\pi$ 时，$\mathbb{E}_\pi[\delta_t \mid S_t, A_t] = 0$，表明"状态-动作"二元组版本的 TD 误差在真实动作值函数下条件期望为零，与公式 (6.6)（状态值函数版本的 TD 误差）的性质一致。这证明了"状态-动作"二元组版本的 TD 误差公式 $\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$ 成立。

实例 有风的网格世界

- **网格结构**：
  - 标准尺寸为 **7 行 × 10 列**（行索引 0–6，列索引 0–9），行 0 为顶部边界，行 6 为底部边界。
  - **起始状态**：$(3, 0)$
  - **目标状态**：$(3, 7)$
  - **边界处理**：若移动导致位置越界（如行 < 0 或行 > 6），则停留在边界内最近的有效位置（例如，从行 0 向上移动仍停留在行 0）。
- **风力机制**（核心特性）：
  - **风力分布**（每列风力大小标注在列下方，单位为向上推动的格数）：

列索引	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
风力	0	0	0	0	1	1	2	2	2	0

  - **状态转移规则**：
    1. 智能体执行动作（上、下、左、右）后，先移动到相邻格子。
    2. **随后风力生效**：位置被向上推动 `风力值` 格（新行号 = $\max(0, \text{移动后行号} - \text{风力})$）。
    - *示例 1*：在目标右侧格子 $(3, 8)$ 执行“向左”动作：
      - 先移动到 $(3, 7)$（目标列），
      - 列 7 风力为 2，向上推动 2 格 → 新位置为 $(\max(0, 3 - 2), 7) = (1, 7)$。
    - *示例 2*：在列 6 的 $(4, 6)$ 执行“向右”动作：
      - 先移动到 $(4, 7)$，
      - 列 7 风力为 2，向上推动 2 格 → 新位置为 $(\max(0, 4 - 2), 7) = (2, 7)$。
  - **关键影响**：风力使状态转移**非确定性**（相同动作在不同列结果不同），且**破坏对称性**（例如，“向右”动作可能因风力导致向上偏移）。
- **任务规范**：
  - **分幕式任务**（episodic task）：每幕从起始状态开始，到达目标状态后终止。
  - **奖励设计**：
    - 每步（包括终止步）收益恒为 $R_t = -1$（无折扣，即 $\gamma = 1$）。
    - 目标：**最小化每幕步数**（因每步 -1，总收益 = $-$步数，最大化收益等价于最小化步数）。
  - **最优路径分析**：

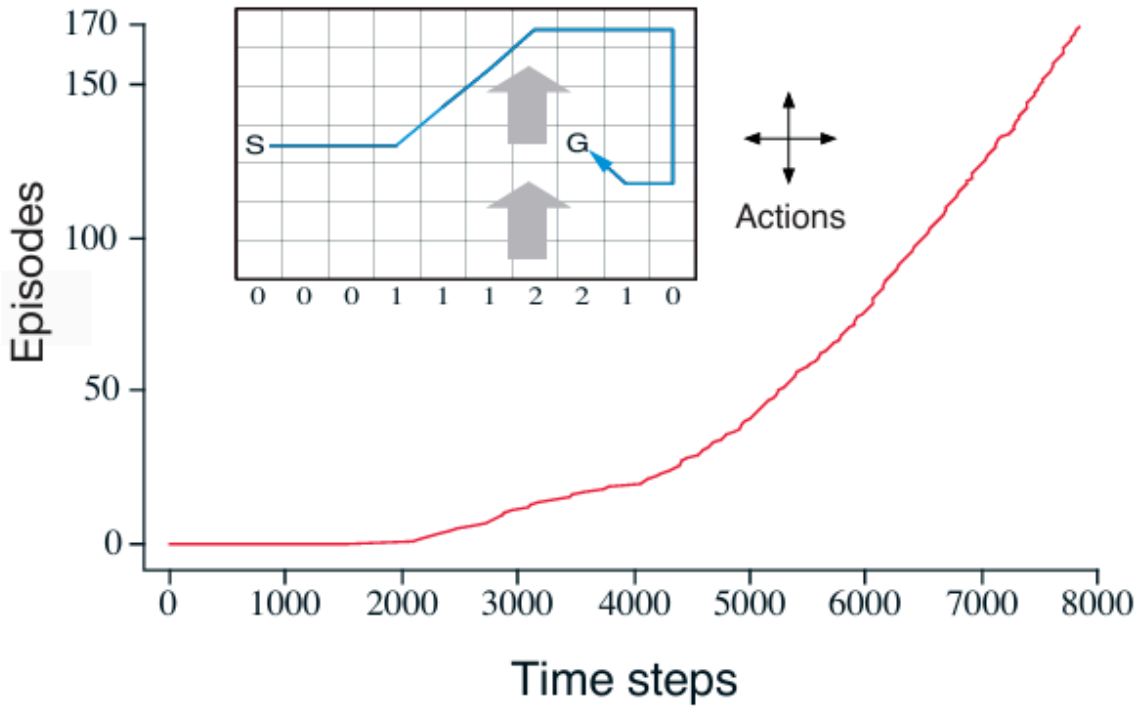
- 理论最小步数：**15 步**（需巧妙利用风力抵消部分移动）。
- *典型最优路径示例*：
  1. $(3,0) \xrightarrow{\text{下}} (4,0)$
  2. $(4,0) \xrightarrow{\text{右}} (4,1) \xrightarrow{\text{右}} (4,2) \xrightarrow{\text{右}} (4,3)$
  3. $(4,3) \xrightarrow{\text{右}} (4,4) \xrightarrow{\text{风力}} (3,4)$
  4. $(3,4) \xrightarrow{\text{右}} (3,5) \xrightarrow{\text{风力}} (2,5)$
  5. $(2,5) \xrightarrow{\text{右}} (2,6) \xrightarrow{\text{风力}} (0,6)$
  6. $(0,6) \xrightarrow{\text{下}} (1,6) \xrightarrow{\text{下}} (2,6)$
  7. $(2,6) \xrightarrow{\text{右}} (2,7) \xrightarrow{\text{风力}} (0,7)$
  8. $(0,7) \xrightarrow{\text{下}} (1,7) \xrightarrow{\text{下}} (2,7) \xrightarrow{\text{下}} (3,7)$
- **为什么 15 步?** 通过提前向下移动（如步骤 1），利用风力向上推回目标行（行 3），避免额外调整步数。

Sarsa 算法实验细节

- **算法选择**：ε-贪心 **Sarsa**。
- **超参数设置**：
  - $\varepsilon = 0.1$：90% 时间选择贪心动作，10% 时间随机探索（确保覆盖所有状态-动作对）。
  - $\alpha = 0.5$：学习率（适中值，平衡收敛速度与稳定性）。
  - **初始化**：所有 $Q(s,a) = 0$（对称初始化，无先验偏好）。
- **Sarsa 更新规则**：

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \right]$$

  其中 A_{t+1} 由当前 ε-贪心策略生成（即 Q 更新依赖于下一步的实际动作）。
- **关键实现特性**：
  - **一步更新（on-the-fly）**：每步转移后立即更新 Q 值，无需等待幕结束。
  - **探索策略**：ε-贪心在每步动态调整（例如，在风力强区域增加探索以发现绕行路径）。



学习曲线解读（横轴：累积总步数；纵轴：成功到达目标的幕数）：

"时间步/幕数" 是衡量算法性能的核心指标，特指完成单个任务（一幕）所需的平均动作执行次数。

阶段	行为特征	平均步数	原因分析
初始阶段（0–2000 步）	策略高度随机，常陷入风力陷阱（如在列 6–8 反复上下）	>30 步	Q 值初始化为 0，\varepsilon-贪心导致大量无效探索
学习中期（2000–8000 步）	曲线斜率增大（成功幕数增速加快）	18–20 步	Q 值逐渐收敛，智能体学会利用风力（如提前向下移动抵消向上风）
稳定阶段（>8000 步）	贪心策略已收敛至最优，但 \varepsilon-贪心维持探索	**17 步**	最优路径 15 步，但 10% 探索导致额外 2 步（如随机选择"向上"动作需后续补偿）

- **关键现象解析**：
  - **为什么平均 17 步?**
    - $\varepsilon = 0.1$ 时，每幕约有 1–2 次探索动作（例如在 $(2, 6)$ 随机选择"向上"而非"向右"），导致路径延长。
    - *具体案例*：在列 6 的 $(2, 6)$，最优动作为"向右"，但探索选择"向上"：
      - 移动到 $(1, 6)$，风力 2 推至 $(0, 6)$，
      - 需额外 2 步向下调整（$(0, 6) \rightarrow (1, 6) \rightarrow (2, 6)$）才能继续。
  - **收敛证据**：
    - 8000 步后，**贪心策略**（$\varepsilon = 0$）始终以 15 步到达目标（图中轨迹示例：从 $(3, 0)$ 沿最优路径移动）。

    - 但 ε-贪心策略下，**平均步数稳定在 17 步**（15 步 × 0.9 + 17 步 × 0.1 ≈ 15.2，实际因探索位置影响略高）。

蒙特卡洛方法在此问题上不适用， 因为非终止策略（non-terminating policies）导致**幕永不结束。**

- *示例*：若策略在列 4–8 总是选择"向上"，则：
    - 从 $(3,5)$ 执行"向上" → $(2,5)$，风力 1 推至 $(1,5)$，
    - 再次"向上" → $(0,5)$，风力 1 推至 $(0,5)$（边界），
    - 永久循环于 $(0,5)$，**幕无限持续。**

蒙特卡洛需完整幕数据更新 Q 值 → 无法获取回报 G_t → 算法停滞。 无法检测策略失效（因无更新信号），可能无限等待。

Sarsa 的核心优势：

- **在线更新机制**：每步转移后立即更新 Q 值，**无需幕终止。**
    - *示例*：在 $(0,5)$ 执行"向上"后，Sarsa 立即更新 $Q((0,5),\text{上})$，因 $R_{t+1}=-1$ 且 Q 值降低，后续 ε-贪心会减少选择此动作。
- **自动规避非终止策略**：
    - 若策略导致循环，Q 值在循环步骤中迅速降低（因持续 -1 收益），促使策略切换。
    - 实验中，Sarsa 在 8000 步内收敛，证明其**对非终止问题的鲁棒性。**
- **Sarsa 的实践价值**：
    - 在**存在干扰**（如风力）的环境中，Sarsa 通过**在线学习**和**探索-利用平衡**，高效收敛至近优策略。
    - ε-贪心的**持续探索**虽牺牲最优性（17 步 vs 15 步），但确保策略**鲁棒性**（适应动态变化）。
- **理论启示**：
    - **TD 方法优势**：一步更新（TD(0)）天然适合**非终止任务**，而蒙特卡洛依赖幕完整性的局限性在此凸显。
    - **探索必要性**：在风力干扰下，纯贪心策略可能陷入局部最优（如忽略向下移动），$\varepsilon > 0$ 是收敛关键。

Q学习：离轨策略下的时序差分控制

Q学习是强化学习早期的重要突破，作为离轨策略下的时序差分控制算法，由 Watkins (1989) 提出。其**更新规则**为：

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1},a) - Q(S_t,A_t)\Big]. \tag{6.8}$$

在这个公式中：

- S_t：当前状态
- A_t：**本回合**（当前时刻 t）选择并执行的动作
- R_{t+1}：执行 A_t 后获得的即时奖励
- S_{t+1}：执行 A_t 后到达的**下一个状态**
- a **表示在状态** S_{t+1} **下所有可能的动作集合中的任意一个动作**
- $\max_a Q(S_{t+1},a)$：**假设**在状态 S_{t+1} 下采取了最优动作

> [!NOTE]
>
> 同轨策略会直接受生成策略（行为策略）的影响。这是因为同轨策略方法中：**行为策略和目标策略是相同的**（或高度相关）；算法学习的目标就是当前用于与环境交互的策略。更新规则中使用的下一动作 A_{t+1} 是**实际由行为策略选择的**，因此，学习过程会考虑行为策略的所有特性，包括探索机制（如 ε-贪心中的随机性）。
>
> Sarsa的更新规则为：
>
> $$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$
>
> 更新时使用 $Q(S_{t+1}, A_{t+1})$，其中 A_{t+1} 是**实际执行的动作**，是由当前行为策略（通常是基于当前 Q 的 ε-贪心策略）在 S_{t+1} 选择的。其**行为策略与目标策略相同**，Sarsa学习的正是这个用于生成数据的 ε-贪心策略。
>
> Q学习的更新规则为：
>
> $$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$
>
> 更新时使用 $\max_a Q(S_{t+1}, a)$，即**假设在 S_{t+1} 采取了最优动作**。这个最大值操作对应的是**贪心策略**，与实际执行的动作 A_{t+1} 无关。实现了**行为策略与目标策略分离**，行为策略可以是 ε-贪心等任意策略（用于生成数据），目标策略总是贪心策略（$\pi(s) = \arg\max_a Q(s, a)$）。

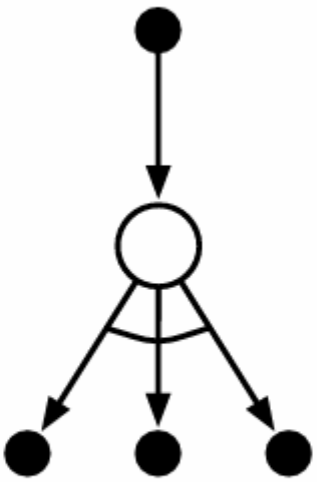
待学习的动作价值函数 Q 的**学习目标**是**直接逼近最优动作价值函数 q_*，与行为策略（生成决策序列的策略）无关。** Sarsa的学习目标则依赖于待学习的动作价值函数本身（需知道下一时刻动作 A_{t+1}），因此与行为策略相关；而 Q 学习使用 $\max_a Q(S_{t+1}, a)$，完全解耦了目标策略（最优策略）与行为策略。其**优势**是简化算法分析，早期即给出收敛性证明。

Q 学习的目标策略是贪心策略（$\pi(s) = \arg\max_a Q(s, a)$），但行为策略（如 ε-贪心）可任意生成数据。 行为策略仅影响哪些“状态-动作”二元组被访问和更新，**不改变学习目标。** 假设所有“状态-动作”二元组被**持续更新**（即所有状态-动作对无限次访问）；步长参数 α 满足**随机近似条件**（如 $\sum \alpha = \infty, \sum \alpha^2 < \infty$），$Q$ 以概率 1 收敛到最优动作价值函数 q_*。

Q 学习（离轨策略下的时序差分控制）算法，用于预测 $\pi \approx \pi_*$
算法参数：步长 $\alpha \in (0, 1]$，很小的 $\varepsilon > 0$
初始化：对所有 $s \in S^+, a \in A(s)$，任意初始化 $Q(s, a)$；其中 $Q(\text{终止状态}, \cdot) = 0$
对每幕：
　初始化 S
　对幕中的每一步循环：
　　使用从 Q 得到的策略（例如 ε-贪心），在 S 处选择 A
　　执行 A，观察到 R, S'
　　$Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$
　　$S \leftarrow S'$
　直到 S 是终止状态

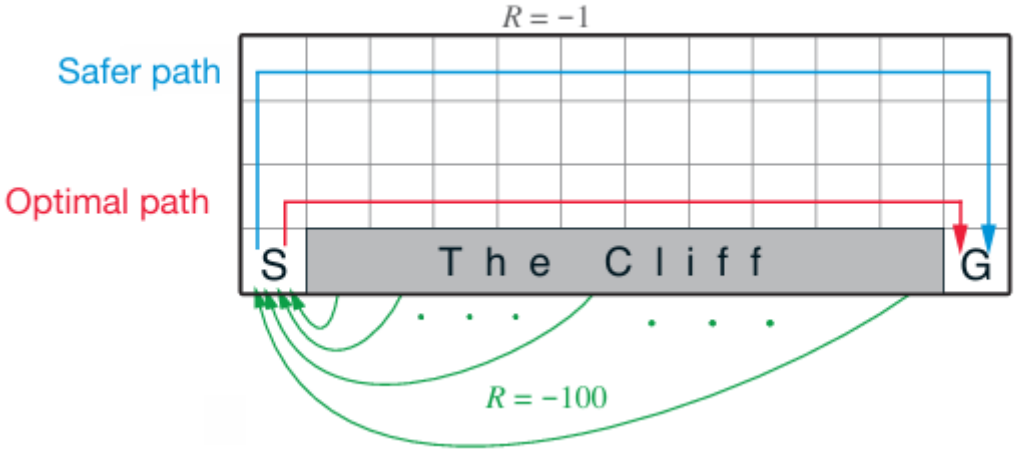
Q学习的回溯图

顶部节点（更新根节点）是代表动作的小实心圆点（因更新针对“状态-动作”二元组）。底部节点是下一状态 S_{t+1} 的所有可能动作节点（因 $\max_a Q(S_{t+1}, a)$ 涉及所有动作）。**一条弧线跨过所有“下一步的动作节点”，表示取最大值操作。**

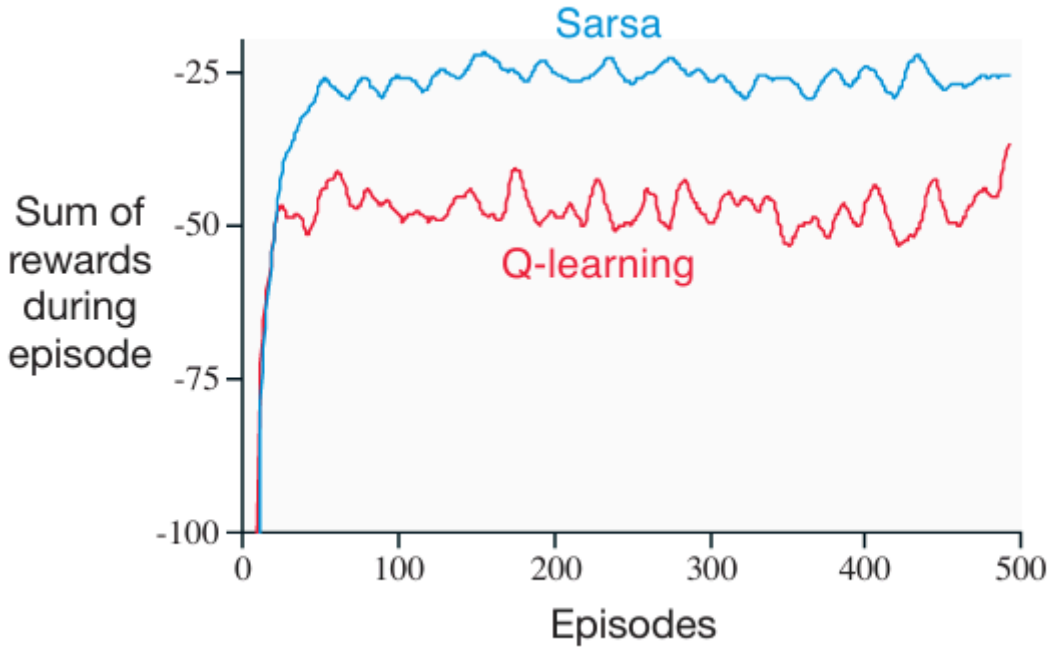


Q-learning

实例



标准无折扣分幕式网格世界，S是起点，G是重点，可以上下左右移动；除悬崖外转移收益为 -1了，掉入悬崖收益为 -100 并立即重置到起点。 使用ε-贪心（$\varepsilon = 0.1$）策略。



算法	学习到的策略	在线性能	原因分析
Q 学习	沿悬崖边行走的**最优策略**（最短路径）	**较差**：因 ε-贪心偶尔掉入悬崖	离轨策略：目标策略与行为策略解耦，忽略探索风险
Sarsa	通过网格上半部分的**安全迂回策略**	**较好**：避免悬崖，更稳定	同轨策略：学习目标依赖行为策略，考虑 ε-贪心的随机性

Q 学习学到最优策略价值，但其行为策略可能带来风险。**若 ε 逐步减小，两者均收敛到最优策略。**

> [!NOTE]
>
> **Q学习更新规则**
>
> $$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$
>
> **Sarsa更新规则**
>
> $$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$
>
> **关键区别**：Q学习使用 $\max_a Q(S_{t+1}, a)$，而Sarsa使用 $Q(S_{t+1}, A_{t+1})$，其中 A_{t+1} 是**实际选择的下一个动作**。
>
> **为什么Q学习不考虑探索行为的风险**
>
> Q学习中，**目标策略是贪心策略**（$\pi(s) = \arg\max_a Q(s, a)$）。**行为策略可以是任意策略**（如ε-贪心），仅用于收集数据。更新时**完全忽略行为策略的特性**，只关注"如果采取最优动作"的价值。
>
> 在危险环境中（例如靠近悬崖的状态），Q-learning 的更新机制隐含地假设："**只要下一个状态 S_{t+1} 存在一个安全的最优动作，那么从当前状态 S_t 执行动作 A_t 就是可接受的。**" 然而，这一假设忽略了一个关键事实：即使 S_{t+1} 中存在最优的安全动作，使用 ε-贪心策略的智能体仍有 ε 的概率在 S_{t+1} 中选择一个危险动作，这可能导致灾难性后果（如掉下悬崖）。 因此，Q-learning 在追求长期回报最大化的同时，**未能充分考虑策略执行过程中的风险暴露**，尤其是在高风险转移路径上的短期脆弱性。

> **在该问题中，最优策略确实应该选择最短路径（沿悬崖边），而安全路径（远离悬崖）会增加行走时间，因此不是最优的。**但Q学习在学习过程中可能**错误地高估最短路径的价值**，导致它**无法可靠地学到真正的最优策略。**
>
> **为什么最短路径（沿悬崖边）是最优策略?**
>
> 在经典的悬崖行走问题中，环境是**确定性的**（动作执行无随机性），且奖励设计如下：
>
> - **每一步的奖励**：`-1`（鼓励智能体尽快到达目标）。
> - **掉下悬崖的奖励**：`-100`（灾难性惩罚）。
> - **到达目标的奖励**：`0`（或 `+10`）。
>
> 关键逻辑：**最优策略 = 最大化累积折扣奖励**
>
> 最短路径步数少则积累奖励大，在确定性环境中，只要智能体不犯错，它永远不会掉下悬崖。安全路径步数则累积奖励 小，累积了更多负奖励，因此价值更低。在**确定性环境**中，**最短路径（悬崖边）是真正的最优策略**，因为它最小化步数。
>
> **为什么Q学习可能学不到这个最优策略?**
>
> 尽管最短路径是最优的，但Q学习在**训练过程中**可能失败，这是因为**"忽略探索风险"**,问题出在**学习机制**，而非最优策略本身。
>
> **目标策略**收敛后，它应该选择最短路径（悬崖边），因为这是理论最优解。**行为策略（ε-贪心）**在训练中强制探索，以 `ε=0.1` 概率随机选择动作（包括"掉下悬崖"的动作）。**Q学习的致命缺陷**则在于更新规则 $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$ 假设在 S总是选择安全动作，即忽略行为策略的探索风险。但在实际交互中，当智能体处于悬崖边状态S 时，行为策略有概率选择危险动作，导致掉下悬崖。Q学习却在更新时假装"在 s' 总是安全"，因此**高估了最短路径的价值。**

Q-learning 在理论上能收敛到最优策略，但在实践中常被诟病"危险""不稳定"或"不适合安全敏感场景"。其被诟病，并不是因为它不能找到数学意义上的最优策略，而是因为它把"最优"定义得太狭隘：只看期望回报，不顾风险、容错性和执行稳定性，这使得它在**安全关键领域**（如机器人控制、自动驾驶、医疗决策）中难以直接应用。

> [!NOTE]
>
> 为什么Sarsa会考虑探索行为的风险
>
> 1. **目标策略与行为策略一致**
>
> Sarsa中，**目标策略就是行为策略**（如ε-贪心），更新时使用的 A_{t+1} 是**实际由行为策略选择的动作。**因此，Sarsa学习的是"**在当前探索水平下**"的策略价值
>
> 2. **使用实际执行的动作：真实反映风险**
>
> $Q(S_{t+1}, A_{t+1})$ 中的 A_{t+1} 是由行为策略实际选择的，如果行为策略是ε-贪心，那么 A_{t+1} 有ε概率是随机选择的，这意味着Sarsa的更新**直接反映了探索行为带来的风险。**
>
> 3. **风险评估机制：预期回报的真实计算**
>
> Sarsa计算的是："**在当前行为策略下，从 (S_t, A_t) 开始的预期回报**"。这个预期回报**自然包含了**：选择次优动作的概率（ε），由于Q值不准确导致的错误选择，任何与行为策略相关的风险。

> ## 4. 悬崖行走例子的数学解释
>
> 继续悬崖边缘状态 s 的例子：
>
> **Sarsa的评估**：
>
> - $Q(s, a_2)$ 的更新基于 $Q(s', A_{t+1})$，其中 A_{t+1} 是实际选择的动作
> - 考虑到ε-贪心：
>   - 90%概率选择"沿悬崖边走"（假设这是当前认为的最优动作）
>   - 10%概率随机选择，可能选择"掉下悬崖"的动作
> - 因此，Sarsa计算：
> $R + \gamma[(0.9 \times Q(s', \text{安全动作})) + (0.1 \times Q(s', \text{危险动作}))]$
> - 由于"掉下悬崖"的回报是-100，这个值会显著降低 $Q(s, a_2)$
> - 结果：Sarsa认为 a_2 的价值低于 a_1，选择更安全的路径

练习

> 假设使用贪心的方式选择动作，那么此时 Q 学习和 Sarsa 是完全相同的算法吗? 它们会做出完全相同的动作选择，进行完全相同的权重更新吗?

在纯贪心策略（ε=0）下，Q-learning 和 Sarsa 的更新规则在数学上是等价的，但它们不一定做出完全相同的动作选择，也不一定进行完全相同的权重更新。

原因在于：**算法的更新规则相同 ≠ 学习过程和最终结果相同。**

先看两种算法的标准更新公式：

Q-learning (off-policy):

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

Sarsa (on-policy):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

当使用**纯贪心策略**（ε=0）时：

- 选择的动作 $a_{t+1} = \arg\max_a Q(s_{t+1}, a)$
- 因此 $Q(s_{t+1}, a_{t+1}) = \max_a Q(s_{t+1}, a)$

此时，Sarsa 的更新公式变为：
$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$

尽管更新规则在数学上等价，但以下因素导致它们**不一定**做出相同的动作选择或更新：

原因一：学习路径不同（历史依赖性）

即使更新规则相同，**初始Q值不同**会导致学习轨迹不同，早期的小差异会被放大，导致后续选择不同的状态-动作序列。一旦轨迹分叉，即使更新规则相同，它们也会基于不同的经验更新。

原因二：探索历史的影响（即使现在使用贪心）

如果算法在**学习过程中**曾使用过 ε > 0，Q-learning 和 Sarsa 会积累**不同的经验。**Q-learning 会认为"靠近悬崖但不掉下去"是可行的（因为它假设下一步总是最优）

- Sarsa 会因为"偶尔掉下悬崖"而给这些状态更低的估值

- **即使现在都使用贪心策略，它们已经学到了不同的Q函数**

原因三：悬崖漫步的特殊性

在经典的悬崖漫步环境中：

- Q-learning 会学到**紧贴悬崖的最短路径**（因为它只考虑最优动作）
- Sarsa 会学到**远离悬崖的更安全路径**（因为它考虑了探索时的失败）
- 即使在测试阶段都使用 ε=0：
  - Q-learning 仍会选择紧贴悬崖的路径
  - Sarsa 仍会选择绕远的安全路径
- **它们做出了完全不同的动作选择**

原因四：收敛到不同的局部最优

- 在复杂环境中，即使更新规则相同
- 不同的初始条件或学习轨迹可能导致收敛到**不同的局部最优解**
- Q-learning 可能收敛到高风险高回报的策略
- Sarsa 可能收敛到低风险低回报的策略

期望 Sarsa

期望 Sarsa 是一种时序差分 (TD) 控制算法，与 Q-learning 十分类似，但将 Q-learning 中**对下一个“状态-动作”二元组取最大值的操作替换为取期望值。**它遵循 Sarsa 的模式，给定下一个状态 S_{t+1}，**算法确定地向期望意义上的方向移动**（即基于策略 π 的期望值），因此称为“期望 Sarsa”。

核心更新规则为：

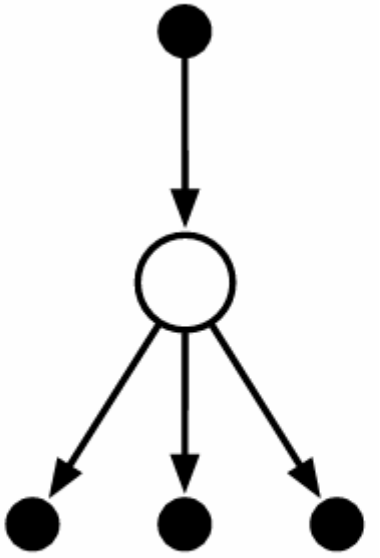
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma\mathbb{E}[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t)\Big]$$

等价于：

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma\sum_a \pi(a \mid S_{t+1})Q(S_{t+1}, a) - Q(S_t, A_t)\Big] \quad (6.9)$$

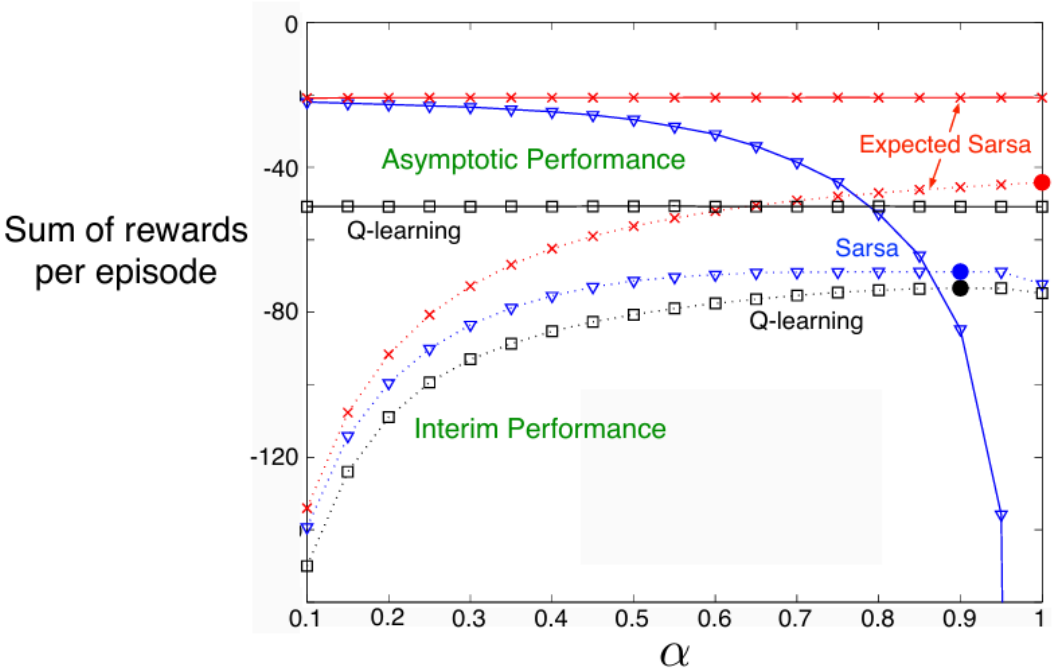
其中：

- α 是步长参数（学习率），
- γ 是折扣因子，
- $\pi(a \mid S_{t+1})$ 是策略 π 在状态 S_{t+1} 下选择动作 a 的概率，
- 期望 $\mathbb{E}[\cdot \mid S_{t+1}]$ 通过策略 π 对所有可能动作求和计算。



Expected Sarsa

期望 Sarsa 在计算上比 Sarsa 更加复杂。但作为回报, 它消除了因为随机选择 A_{t+1} 而 产生的方差。在相同数量的经验下,我们可能会预想它的表现能略好于 Sarsa, 期望 Sarsa 也确实表现更好。



该图表示在悬崖边行走这个任务中, TD 控制方法的启动期瞬态性能和长期稳态性能与步长参数 a 之间的关系函数。这里所有算法都用的是 ε = 0.1 的 ε-贪心策略。每次实验中, **稳态性能**是100 000 幕数据上的平均值, 而**瞬态性能**则是最初 100 幕数据上的平均值。图中启动期瞬态曲线和长期稳态曲线中的数据则分别是 50 000 次实验运行和 10 次实验运行的**平均值**。图中实心圆表示的是每种方法最佳的瞬态性能。

期望 Sarsa 保持了 Sarsa 优于 Q 学习的显著优势。此外,在步长参数 α 大量不同的取值下, 期望 Sarsa 都显著地优于 Sarsa。在悬崖边上行走这个例子 中,状态的转移都是确定的, 所有的随机性都来自于策略。在这种情况下, 期望 Sarsa 可 以放心地设定 α=1, 而不需要担心长期稳态性能的损失。与之相比, Sarsa 仅仅在 α 的 值较小时能够有良好的长期表现,而启动期瞬态表现就很糟糕了。不管是在这个例子还是 在其他例子中, 期望 Sarsa 相比 Sarsa 在实验中都表现出了明显的优势。

在悬崖边上行走这个例子的实验结果中, 期望 Sarsa 被用作一种同轨策略的算法。 但在一般情况下, 它可以采用与目标策略 π 不同的策略来生成行为。在这种情况下期 望 Sarsa 就成了离轨策略的算法。举个例子, 假设目标策略 π 是一个贪心策略, 而行动 策略却更注重于试探, 那么此时期望 Sarsa 与 Q 学习完全相同。从这个角度来看, 期 望 Sarsa 推广了 Q 学习, 可以将 Q 学习视作期望 Sarsa 的一种特例, 同时期望 Sarsa 比 起 Sarsa 也稳定地提升了性能。除了增加少许的计算量之外,期望 Sarsa 应该完全优于这两种更知名的时序差分控制算法。

> [!IMPORTANT]
>
> **期望 Sarsa 的 "同轨/离轨" 属性取决于目标策略 π 是否与行为策略一致。**在悬崖实验中，π 被设为行为策略本身（同轨）；在目标策略为贪心的案例中，π 与行为策略分离（离轨）。这种灵活性是其优于 Sarsa 和 Q-learning 的核心原因之一。

最大化偏差与双学习

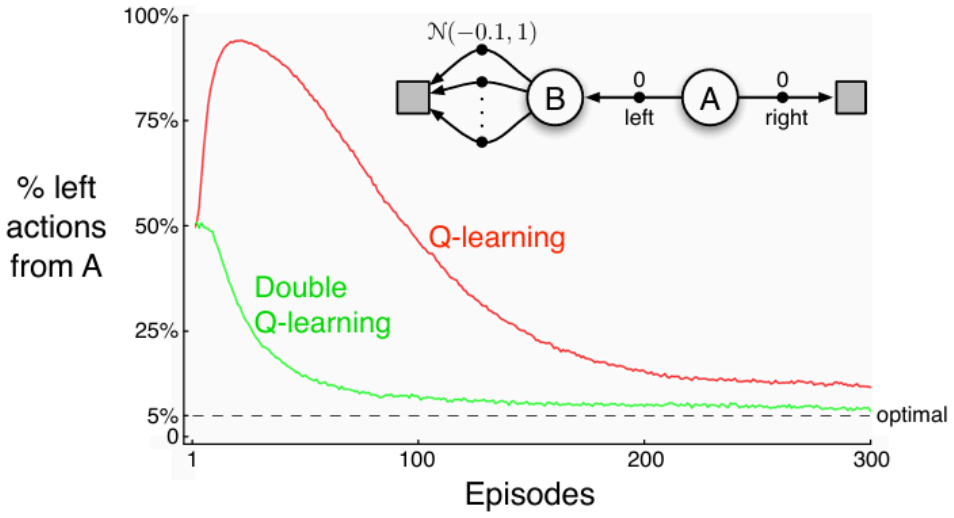
最大化偏差

在强化学习控制算法（如 Q 学习、Sarsa）中，目标策略的构建通常包含 **最大化操作**（如 $\max_a Q(s,a)$ 或 ϵ-贪心中的 $\max$）。这会导致对动作价值的最大值产生 **正偏差**（估计值高于真实值）。

假设状态 s 下所有动作的真实价值 $q(s,a)=0$（$\forall a$），但估计值 $Q(s,a)$ 因噪声存在波动（部分 >0，部分 <0）。其真实最大值：$\max_a q(s,a)=0$，但估计最大值：$\max_a Q(s,a)>0$（因噪声的非对称性，正偏差占主导）。最大化偏差会显著损害 TD 控制算法的性能，导致策略过度偏好某些动作（即使其真实价值较低）。

实例

- 状态：非终止状态 A 和 B，每幕从 A 开始。
- 动作：
  - A 处向右：立即终止，收益 $R=0$。
  - A 处向左：转移到 B，收益 $R=0$。
  - B 处有多个动作：每个动作立即终止，收益服从正态分布 $\mathcal{N}(-0.1,1.0)$。
- **真实最优策略**：在 A 应选择向右（因向左的期望回报为 $-0.1<0$）。
- **问题**：
  - 由于最大化偏差，B 处的估计价值 $\max_a Q(B,a)>-0.1$（真实期望为 -0.1）。
  - Q 学习算法（$\epsilon=0.1,\alpha=0.1,\gamma=1$）错误地偏好向左动作：
    - 初始阶段：选择向左的概率显著高于向右。
    - 收敛后：选择向左的概率仍比理论最优值高约 5%（ϵ-贪心下最低向左概率为 5%）。
- **关键结论**：最大化偏差导致算法学习次优策略（下图显示 Q 学习持续高估向左动作的价值）。



> 本图是Q 学习和双 Q 学习在一个简单的分幕式 MDP 中的对比。Q 学习在开始阶段学到 的行为是: 执行向左的概率会远比执行向右来得高, 并且向左的概率会一直显著地高于 5%。这个 5% 是使 用 ε = 0.1 的 ε-贪心策略选择动作而引起的最低的向左运动的概率。与之相比, 双 Q 学习实质上并没有受到最 大化偏差的影响。这些数据都是 10 000 次运行的平均值, 动作价值都被初始化为零。在用贪心策略选择动作 时, 如果有多个最大值, 那么会随机选择一个。

双学习

为了消除最大化偏差，提出了一种称为 **双学习（Double Learning）** 的方法。

核心思想：

> 将数据划分为两个独立的部分，分别训练两个独立的价值函数估计器。
> 一个用于**选择**最优动作，另一个用于**评估**该动作的价值。

这样可以打破“用同一数据同时做选择和评估”的循环，从而减少偏差。

假设我们有两个独立的数据集（或多幕序列），可以用来学习两个独立的动作价值估计：

- $Q_1(a)$：基于第一组样本对动作 a 的价值估计
- $Q_2(a)$：基于第二组样本对动作 a 的价值估计

步骤如下：

1. 使用 Q_1 来选择最优动作：

$$A^* = \arg\max_a Q_1(a)$$

2. 使用 Q_2 来评估该动作的价值：

$$\text{Estimate} = Q_2(A^*) = Q_2\left(\arg\max_a Q_1(a)\right)$$

3. 因为 Q_2 是独立于选择过程的，所以：

$$\mathbb{E}[Q_2(A^*)] = q(A^*)$$

   即该估计是**无偏的**。

4. 同样地，也可以反过来：

$$Q_1\left(\arg\max_a Q_2(a)\right)$$

得到另一个无偏估计。

> 通过交替使用两个估计器进行选择与评估，可以在不引入偏差的前提下更准确地估计最大价值。

> [!NOTE]
>
> 如何理解
>
> > 因为 Q_2 是独立于选择过程的，所以：
> >
> > $$\mathbb{E}[Q_2(A^*)] = q(A^*)$$
> >
> > 即该估计是**无偏的**

考虑一个**多臂赌博机（multi-armed bandit）**问题：

- 有若干动作（臂）$a \in \mathcal{A}$
- 每个动作 a 的真实价值为 $q(a) = \mathbb{E}[R|A = a]$
- 我们通过采样得到带有噪声的回报，从而对每个动作的价值进行估计
- 设我们有两个独立的数据集（或两组独立的样本轨迹），可以用来训练两个独立的价值估计函数：
  - $Q_1(a)$：基于第一组数据对 $q(a)$ 的估计
  - $Q_2(a)$：基于第二组数据对 $q(a)$ 的估计

假设这两个估计满足：

- $\mathbb{E}[Q_1(a)] = q(a)$，即无偏
- $\mathbb{E}[Q_2(a)] = q(a)$，即无偏
- 并且对于所有 a，$Q_1(a)$ 与 $Q_2(a)$ 是**相互独立的随机变量**

我们使用 Q_1 来选择最优动作：

$$A^* = \arg\max_a Q_1(a)$$

注意：这里的 A^* 是一个**随机变量**，因为它依赖于随机的估计值 $Q_1(a)$，它不是固定的。

我们的目标是估计真实价值中最大的那个：$\max_a q(a)$，但我们实际计算的是：

$$Q_2(A^*) = Q_2\left(\arg\max_a Q_1(a)\right)$$

> 用 Q_1 “选出”了一个看起来最好的动作 A^*，然后用另一个独立的估计器 Q_2 去评估它的**真实表现。**

核心分析：为什么 $\mathbb{E}[Q_2(A^*)] = q(A^*)$?

我们要理解这个等式的含义：

$$\mathbb{E}[Q_2(A^*)] = q(A^*)$$

更准确地说，我们应该说：

> 对任意固定的动作选择结果 $A^* = a$，在给定 $A^* = a$ 的条件下，
>
> $$\mathbb{E}[Q_2(A^*) \mid A^* = a] = q(a)$$

> 或者整体地看：
>
> $$\mathbb{E}[Q_2(A^*)] = \mathbb{E}[q(A^*)]$$
>
> 这才是正确的无偏性表述。
>
> **所以将命题改写为**
>
> > 如果 $Q_2(a)$ 是 $q(a)$ 的无偏估计，且 $Q_2(a)$ 与 A^*（由 Q_1 决定）**独立**，那么：
> >
> > $$\mathbb{E}[Q_2(A^*)] = \mathbb{E}[q(A^*)]$$
> >
> > 即：$Q_2(A^*)$ 是 $q(A^*)$ 的**无偏估计。**
>
> 按 A^* 的可能取值分解期望：
>
> $$\mathbb{E}[Q_2(A^*)] = \sum_{a \in \mathcal{A}} \mathbb{P}(A^* = a) \cdot \mathbb{E}[Q_2(A^*) \mid A^* = a]$$
>
> 由于当 $A^* = a$ 时，$Q_2(A^*) = Q_2(a)$，所以上式变为：
>
> $$\mathbb{E}[Q_2(A^*)] = \sum_{a} \mathbb{P}(A^* = a) \cdot \mathbb{E}[Q_2(a) \mid A^* = a]$$
>
> 关键假设：
>
> > $Q_2(a)$ 与 $Q_1(\cdot)$ 独立 → 所以 $Q_2(a)$ 与 $A^* = \arg\max_b Q_1(b)$ 也独立
>
> 这意味着：
>
> $$\mathbb{E}[Q_2(a) \mid A^* = a] = \mathbb{E}[Q_2(a)]$$
>
> 因为条件不影响分布。
>
> 又因为 $Q_2(a)$ 是无偏估计：
>
> $$\mathbb{E}[Q_2(a)] = q(a)$$
>
> 代入上式：
>
> $$\mathbb{E}[Q_2(A^*)] = \sum_{a} \mathbb{P}(A^* = a) \cdot q(a) = \mathbb{E}[q(A^*)]$$

双学习的优点在于**消除或显著降低最大化偏差**，但需要两倍内存存储两个价值函数，每步计算量不变。

双 Q 学习

将双学习的思想扩展到完整的 MDP 场景，得到 **Double Q-Learning（双Q学习）**。

- 维护两个动作价值函数：$Q_1(s,a)$ 和 $Q_2(s,a)$
- 所有时间步被随机分成两部分（如抛硬币决定）
- 每次只更新其中一个 Q 函数，另一个保持不变

更新规则

根据抛硬币结果决定更新哪个函数：

情况1：正面朝上 → 更新 Q_1

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q_2 \left(S_{t+1}, \arg\max_a Q_1(S_{t+1}, a) \right) - Q_1(S_t, A_t) \right]$$

- 动作选择由 Q_1 决定（$\arg\max_a Q_1$）

- 动作价值评估由 Q_2 提供
- 更新的是 Q_1

情况2：反面朝上 → 更新 Q_2

$$Q_2(S_t, A_t) \leftarrow Q_2(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q_1 \left(S_{t+1}, \arg\max_a Q_2(S_{t+1}, a) \right) - Q_2(S_t, A_t) \right]$$

- 动作选择由 Q_2 决定
- 动作价值评估由 Q_1 提供
- 更新的是 Q_2

注意：两个价值函数地位对等，轮流担任"选择者"和"评估者"。

行为策略

虽然有两个价值函数，但在实际决策时仍需一个统一的行为策略。

常见做法：

- 使用 $Q_1 + Q_2$ 或 $(Q_1 + Q_2)/2$ 的平均值
- 结合 ε-贪心策略选择动作

例如，在每一步中：

$$A = \begin{cases} \arg\max_a \frac{Q_1(s,a)+Q_2(s,a)}{2}, & \text{概率 } 1-\varepsilon \\ \text{随机动作}, & \text{概率 } \varepsilon \end{cases}$$

习题

> **题目**：使用 ε-贪心目标策略的双期望 Sarsa 的更新步骤是怎样的?

首先回顾**期望 Sarsa**的更新公式：

$$Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma \sum_a \pi(a|S') Q(S', a) - Q(S, A) \right]$$

其中 $\pi(a|S')$ 是目标策略下动作的概率分布（如 ε-贪心策略）。

现在将其扩展为**双期望 Sarsa（Double Expected Sarsa）**：

维护两个估计：Q_1 和 Q_2

更新规则（分情况）：

情况1：更新 Q_1

$$Q_1(S, A) \leftarrow Q_1(S, A) + \alpha \left[R + \gamma \sum_a \pi_2(a|S') Q_1(S', a) - Q_1(S, A) \right]$$

其中：

- $\pi_2(a|S')$ 是基于 $Q_2(S', a)$ 构造的 ε-贪心策略
- 即：选择动作分布依赖于 Q_2，但价值评估使用 Q_1

情况2：更新 Q_2

$$Q_2(S,A) \leftarrow Q_2(S,A) + \alpha \left[R + \gamma \sum_a \pi_1(a|S') Q_2(S',a) - Q_2(S,A) \right]$$

其中：

- $\pi_1(a|S')$ 是基于 $Q_1(S',a)$ 的 ε-贪心策略
- 动作选择依据 Q_1，价值评估使用 Q_2

游戏、后位状态与其他特殊例子

后位状态

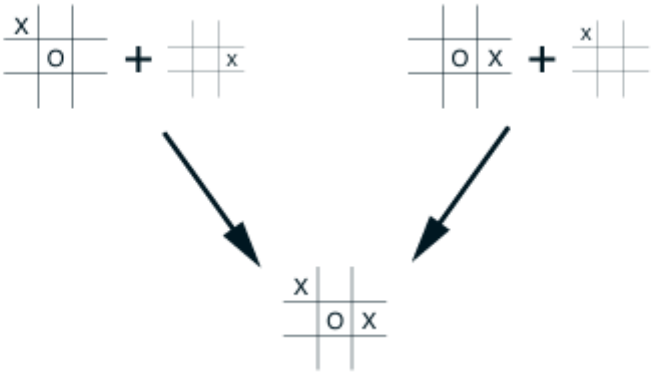
后位状态（Afterstate）指智能体执行某个动作后立即进入的状态，**后位状态价值函数**则用于评估在智能体执行动作后所达到状态的价值。

类型	评估对象	定义域	示例
状态价值函数 V(s)	当前状态s的价值	状态空间S	评估当前棋盘局面的价值
动作价值函数 Q(s,a)	在状态s选择动作a的价值	状态-动作对 (s,a)	评估"当前局面+下某步"的价值
后位状态价值函数 V(s')	执行动作后达到的状态 s'的价值	后位状态空间	评估"下完某步后的局面"的价值

当我们**部分了解环境动态**，但不知道完整环境动态时后位状态特别有用。例如在许多游戏中，我们知道自己的动作会导致什么局面（确定性部分），但不知道对手的反应或随机事件的结果（不确定性部分）。**后位状态利用了"动作→立即结果"这一确定性部分的先验知识，使学习更高效。**

实例 井字棋

传统的动作价值函数将当前局面和出棋动作映射到价值的估计值。$Q(\text{当前局面}, \text{下法}) \rightarrow \text{价值估计}$。但现实中存在大量不同的"(局面,下法)"对可能产生**相同的后位状态**（即相同的棋盘局面）。



如图所示：两个不同的"(局面,下法)"对产生了相同的后位状态。传统方法会分别学习这两个不同的(s,a)对，但实际上它们应该有**相同的价值**。

后位状态价值函数直接学习$V(\text{下完棋后的局面}) \rightarrow \text{价值估计}$，可以**自动共享经验**，无论通过何种方式达到相同的后位状态，其价值估计是相同的。其关心关于某一"(局面,下法)"对的学习会立即迁移到所有能产生相同后位状态的其他组合，减少需要学习的独立参数数量，提高样本效率，加速收敛。