Chapter7
Temporal-Difference Learning

南京邮电大学计算机学院、软件学院、网络空间安全学院

2025 年 3 月

Table of contents

Introduction

- 本章介绍 temporal-difference (TD) learning，这是强化学习中最著名的方法之一。
- Monte Carlo (MC) learning 是第一个无模型方法，TD learning 是第二个无模型方法。TD learning 与 MC learning 相比有一些优势。
- 我们将看到上一讲研究的随机逼近方法是如何发挥作用的。

Motivating example: stochastic algorithms

我们考虑一些随机问题并且展示如何使用RM 算法求解。
首先，回顾均值估计问题：基于 X 的独立同分布样本 $\{x\}$，计算

$$w = E[X]$$

- 通过定义 $g(w) = w - E[X]$，可以将问题转化为求根问题

$$g(w) = 0$$

- 由于我们只能获得 X 的样本 $\{x\}$，因此噪声观测为

$$\tilde{g}(w, \eta) = w - x = (w - E[X]) + (E[X] - x) \doteq g(w) + \eta$$

- 根据上一讲，求解 $g(w) = 0$ 的 RM 算法为

$$w_{k+1} = w_k - \alpha_k \tilde{g}(w_k, \eta_k) = w_k - \alpha_k (w_k - x_k)$$

Motivating example: stochastic algorithms

接下来，考虑一个稍微复杂一点的问题：基于 X 的独立同分布随机样本 $\{x\}$，估计函数 $v(X)$ 的均值，

$$w = E[v(X)]$$

- 为了解决这个问题，定义

$$g(w) = w - E[v(X)]$$

$$\tilde{g}(w, \eta) = w - v(x) = (w - E[v(X)]) + (E[v(X)] - v(x)) \doteq g(w) + \eta$$

- 于是问题变成了求根问题 $g(w) = 0$，对应的 RM 算法为

$$w_{k+1} = w_k - \alpha_k \tilde{g}(w_k, \eta_k) = w_k - \alpha_k [w_k - v(x_k)]$$

再考虑一个更复杂的问题：计算

$$w = E[R + \gamma v(X)]$$

其中 R, X 是随机变量，γ 是常数，$v(\cdot)$ 是函数。

- 假设我们可以获得 X 和 R 的样本 $\{x\}$ 和 $\{r\}$。定义

$$g(w) = w - E[R + \gamma v(X)]$$

$$\tilde{g}(w, \eta) = w - [r + \gamma v(x)] = (w - E[R + \gamma v(X)]) + (E[R + \gamma v(X)] - [r + \gamma v(x$$

- 于是问题变成了求根问题 $g(w) = 0$，对应的 RM 算法为

$$w_{k+1} = w_k - \alpha_k \tilde{g}(w_k, \eta_k) = w_k - \alpha_k [w_k - (r_k + \gamma v(x_k))]$$

这个算法看起来像后面将要介绍的 TD 算法。

Motivating example: stochastic algorithms

小结:

- 上述三个例子越来越复杂。
- 它们都可以通过 RM 算法解决。
- 我们将看到 TD 算法有类似的表达式。

Table of contents

问题陈述：

- 给定策略 π，目标是估计在 π 下的状态值 $\{v_\pi(s)\}_{s\in\mathcal{S}}$
- 经验样本：由 π 生成的 $(s_0, r_1, s_1, \dots, s_t, r_{t+1}, s_{t+1}, \dots)$ 或 $\{(s_t, r_{t+1}, s_{t+1})\}_t$

TD learning 算法公式：

$$v_{t+1}(s_t) = v_t(s_t) - \alpha_t(s_t)\left[v_t(s_t) - [r_{t+1} + \gamma v_t(s_{t+1})]\right] \tag{1}$$

$$v_{t+1}(s) = v_t(s), \quad \forall s \neq s_t \tag{2}$$

其中 $t = 0, 1, 2, \ldots$。
这里，$v_t(s_t)$ 是 $v_\pi(s_t)$ 的估计值；$\alpha_t(s_t)$ 是 s_t 在时间 t 的学习率。

- 在时间 t，只有访问过的状态 s_t 的值被更新，而未访问过的状态 $s \neq s_t$ 的值保持不变。
- 当上下文明确时，将省略更新式 (2)。

TD learning of state values - Algorithm properties

TD 算法可以标注为

$$\underbrace{v_{t+1}(s_t)}_{\text{new estimate}} = \underbrace{v_t(s_t)}_{\text{current estimate}} - \alpha_t(s_t)\Big[\overbrace{v_t(s_t) - \underbrace{[r_{t+1} + \gamma v_t(s_{t+1})]}_{\text{TD target } \bar{v}_t}}^{\text{TD error } \delta_t}\Big], \tag{3}$$

其中，

$$\bar{v}_t \doteq r_{t+1} + \gamma v_t(s_{t+1})$$

被称为 TD 目标。

$$\delta_t \doteq v_t(s_t) - [r_{t+1} + \gamma v_t(s_{t+1})] = v_t(s_t) - \bar{v}_t$$

被称为 TD 误差。

新估计 $v_{t+1}(s_t)$ 是当前估计 $v_t(s_t)$ 和 TD 误差的组合。

TD learning of state values - Algorithm properties

首先，为什么 $\bar{v}_t$ 被称为 TD 目标?
这是因为算法使 $v(s_t)$ 逐渐逼近 $\bar{v}_t$。
证明如下，

$$v_{t+1}(s_t) = v_t(s_t) - \alpha_t(s_t)[v_t(s_t) - \bar{v}_t]$$

$$\Rightarrow v_{t+1}(s_t) - \bar{v}_t = v_t(s_t) - \bar{v}_t - \alpha_t(s_t)[v_t(s_t) - \bar{v}_t]$$

$$\Rightarrow v_{t+1}(s_t) - \bar{v}_t = [1 - \alpha_t(s_t)][v_t(s_t) - \bar{v}_t]$$

$$\Rightarrow |v_{t+1}(s_t) - \bar{v}_t| = |1 - \alpha_t(s_t)||v_t(s_t) - \bar{v}_t|$$

由于 $\alpha_t(s_t)$ 是一个小的正数，我们有

$$0 < 1 - \alpha_t(s_t) < 1$$

因此，

$$|v_{t+1}(s_t) - \bar{v}_t| \leq |v_t(s_t) - \bar{v}_t|$$

这意味着 $v(s_t)$ 逐步逼近 $\bar{v}_t$!

其次，TD 误差的解释是什么?

$$\delta_t = v_t(s_t) - [r_{t+1} + \gamma v_t(s_{t+1})]$$

- 反映了两个时间步之间的差异。
- 反映了 v_t 和 v_π 之间的差异。证明如下，记

  $$\delta_{\pi,t} \doteq v_\pi(s_t) - [r_{t+1} + \gamma v_\pi(s_{t+1})]$$

  注意到

  $$E[\delta_{\pi,t}|S_t = s_t] = v_\pi(s_t) - E[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s_t] = 0$$

  - 如果 $v_t = v_\pi$，那么 δ_t 应该为零（在期望意义上）。
  - 因此，如果 δ_t 不为零，那么 v_t 不等于 v_π。
- TD 误差可以被解释为即从经验 (s_t, r_{t+1}, s_{t+1}) 中获得的信息。

TD learning of state values - Algorithm properties

其他性质:

- 公式 (3) 中的 TD 算法只估计给定策略的值函数。
  - 不估计动作值函数。
  - 不寻找最优策略。
- 这个算法稍后将被扩展为估计动作值，然后寻找最优策略。
- 公式 (3) 中的 TD 算法是理解更复杂 TD 算法的基础。

问题：该 TD 算法在数学上做了什么?

回答: 用于求解给定策略 π 的 Bellman 方程，在无模型的情况下。

- 第二章介绍了求解 Bellman 方程的基于模型的算法：封闭形式解 + 迭代算法。

首先，Bellman 方程的新表达式。
策略 π 下的状态 s 的值函数定义为

$$v_\pi(s) = E[R + \gamma G | S = s], \quad s \in \mathcal{S} \tag{4}$$

其中 G 是未来奖励的折扣总和。由于

$$E[G|S = s] = \sum_a \pi(a|s) \sum_{s'} p(s'|s, a) v_\pi(s') = E[v_\pi(S')|S = s]$$

其中 S' 是下一个状态，因此我们可以将 (4) 重写为

$$v_\pi(s) = E[R + \gamma v_\pi(S')|S = s], \quad s \in \mathcal{S} \tag{5}$$

方程 (5) 是 Bellman 方程的另一种表达式，有时被称为Bellman 期望方程，是设计和分析 TD 算法的重要工具。

其次，使用 RM 算法求解 (5) 中的 Bellman 方程。
特别地，通过定义

$$g(v(s)) = v(s) - E[R + \gamma v_\pi(S')|s]$$

我们可以将 (5) 重写为

$$g(v(s)) = 0$$

由于我们只能获得 R 和 S' 的样本 r 和 s'，我们得到的噪声观测是

$$\begin{aligned}\tilde{g}(v(s)) &= v(s) - \left[r + \gamma v_\pi(s')\right] \\ &= \underbrace{\left(v(s) - \mathbb{E}\left[R + \gamma v_\pi(S')|s\right]\right)}_{g(v(s))} + \underbrace{\left(\mathbb{E}\left[R + \gamma v_\pi(S')|s\right] - \left[r + \gamma v_\pi(s')\right]\right)}_{\eta}\end{aligned}$$

因此，求解 $g(v(s))=0$ 的 RM 算法为

$$\begin{aligned} v_{k+1}(s) &= v_k(s)-\alpha_k\tilde{g}(v_k(s)) \\ &= v_k(s)-\alpha_k\left(v_k(s)-\left[r_k+\gamma v_\pi(s'_k)\right]\right), \quad k=1,2,3,\ldots \end{aligned} \tag{6}$$

其中 $v_k(s)$ 是 $v_\pi(s)$ 在第 k 步的估计值；r_k, s'_k 是在第 k 步获得的 R, S' 的样本。

RM 算法 (6) 与 TD 算法看起来非常相似。然而，存在两个差异。

- 差异 1：RM 算法需要 $\{(s, r_k, s'_k)\}$ 对于 $k=1,2,3,\ldots$
  - 修改：将 $\{(s, r_k, s'_k)\}$ 改为 $\{(s_t, r_{t+1}, s_{t+1})\}$，以便算法可以利用一个 Episode 中连续的样本。
- 差异 2：RM 算法需要 $v_\pi(s'_k)$。
  - 修改：用估计值 $v_t(s_{t+1})$ 替换 $v_\pi(s'_k)$。

经过上述修改后，RM 算法就变成了 TD 算法。

TD learning of state values - Algorithm convergence

定理 (TD Learning 的收敛性)

通过 *TD* 算法 *(1)*，对于所有 $s \in \mathcal{S}$，$v_t(s)$ 随 $t \to \infty$ 以概率 *1* 收敛到 $v_\pi(s)$，条件是 $\sum_t \alpha_t(s) = \infty$ 和 $\sum_t \alpha_t^2(s) < \infty$ 对所有 $s \in \mathcal{S}$ 成立。

- 这个定理表明，对于给定的策略 π，TD 算法可以找到状态值。
- $\sum_t \alpha_t(s) = \infty$ 和 $\sum_t \alpha_t^2(s) < \infty$ 必须对所有 $s \in \mathcal{S}$ 成立。
  - 对于条件 $\sum_t \alpha_t(s) = \infty$：在时间步 t，
    - ★ 如果 $s = s_t$，则 $\alpha_t(s) > 0$；
    - ★ 如果 $s \neq s_t$，则 $\alpha_t(s) = 0$。

    因此，$\sum_t \alpha_t(s) = \infty$ 要求每个状态必须被访问无限次（或足够多次）。
  - 对于条件 $\sum_t \alpha_t^2(s) < \infty$：在实践中，学习率 α 通常被选为一个小的常数。在这种情况下，条件 $\sum_t \alpha_t^2(s) < \infty$ 不再有效。当 α 为常数时，仍然可以证明算法在期望意义上收敛。

TD learning of state values - Algorithm properties

虽然时序差分（TD）学习和蒙特卡罗（MC）学习都是无模型的，但 TD 学习与 MC 学习相比有哪些优缺点?

表: TD 学习与 MC 学习的比较

TD/Sarsa 学习	**MC 学习**
Online：TD 学习是在线的。它可以在接收到奖励后立即更新状态/动作值。	**Offline**：MC 学习是离线的。它必须等待一个完整集被完全收集。
Continuing tasks：由于 TD 学习是在线的，它可以处理连续任务和 episodic 任务。	**Episodic tasks**：由于 MC 学习是离线的，它只能处理有终止状态的 episodic 任务。

TD learning of state values - Algorithm properties

虽然时序差分（TD）学习和蒙特卡罗（MC）学习都是无模型的学习方法，但相较于蒙特卡罗学习，时序差分学习的优缺点分别是什么呢?

表: TD 学习与 MC 学习的比较（续）

TD/Sarsa 学习	MC 学习
Bootstrapping：TD 学习运用自举法，因为值的更新依赖于该值之前的估计值。因此，它需要初始猜测值。	**Non - bootstrapping**：MC 学习不使用自举法，因为它能够在无需任何初始猜测的情况下直接估计状态/动作值。
Low estimation variance：TD 学习的估计方差比 MC 学习低，因为涉及的随机变量更少。例如，Sarsa 算法仅需要 R_{t+1}（$t+1$ 时刻的奖励）、S_{t+1}（$t+1$ 时刻的状态）和 A_{t+1}（$t+1$ 时刻的动作 Sarsa）。	**High estimation variance**：为了估计 $q_\pi(s_t, a_t)$(在策略 π 下，状态 s_t 执行动作 a_t 的值函数)，我们需要 $R_{t+1}+\gamma R_{t+2}+\gamma^2 R_{t+3}+\ldots$(后续一系列折扣奖励之和) 的样本。假设每个 Episode 的长度为 L，那么就有 $\|A\|^L$ 种可能的 Episode。

Table of contents

TD learning of action values - Sarsa

- 上一节介绍的 TD 算法只能估计状态值。
- 接下来，我们介绍 Sarsa，这是一种可以直接估计动作值的算法。
- 我们还将看到如何使用 Sarsa 来寻找最优策略。

首先，我们的目标是估计给定策略 π 的动作值函数。
假设我们有一些经验 $\{(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})\}_t$。我们可以使用以下 Sarsa 算法来估计动作值：

$$
\begin{aligned}
q_{t+1}(s_t, a_t) &= q_t(s_t, a_t) - \alpha_t(s_t, a_t)\big(q_t(s_t, a_t) - (r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1}))\big) \\
q_{t+1}(s, a) &= q_t(s, a), \quad \forall (s, a) \neq (s_t, a_t)
\end{aligned}
$$

其中 $t = 0, 1, 2, \ldots$。

- $q_t(s_t, a_t)$ 是 $q_\pi(s_t, a_t)$ 的估计值；
- $\alpha_t(s_t, a_t)$ 是取决于 s_t, a_t 的学习率。

- 为什么这个算法被称为 Sarsa？这是因为算法的每一步都涉及 $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$。Sarsa 是 state-action-reward-state-action 的缩写。
- Sarsa 算法与前面的 TD 学习算法有什么关系？我们可以通过将 TD 算法中的状态值估计 $v(s)$ 替换为动作值估计 $q(s, a)$ 来获得 Sarsa。因此，Sarsa 是 TD 算法的动作值版本。
- Sarsa 算法在数学上做了什么？Sarsa 的表达式表明，它是一个随机逼近算法，用于求解以下方程：

$$q_\pi(s, a) = E[R + \gamma q_\pi(S', A')|s, a], \quad \forall s, a$$

这是用动作值函数表示的 Bellman 方程的另一种表达式。

Sarsa - Algorithm

定理 (Sarsa learning 的收敛性)

通过 *Sarsa* 算法，对于所有 (s, a)， $q_t(s, a)$ 以概率 *1* 收敛到动作值 $q_\pi(s, a)$，当 $t \to \infty$，条件式 $\sum_t \alpha_t(s, a) = \infty$ 和 $\sum_t \alpha_t^2(s, a) < \infty$ 对所有 (s, a) 成立。

- 这个定理表明，Sarsa 可以找到给定的策略 π 动作值。

Sarsa - Implementation

强化学习的最终目标是找到最优策略。为此，我们可以将 Sarsa 与策略改进步骤结合起来。这种组合算法也被称为 Sarsa。

算法步骤：通过 Sarsa 进行策略搜索

对每个 Episode：

在初始状态 s_0 处，根据初始策略 $\pi_0(s_0)$ 生成动作 a_0。当状态 s_t $(t=0,1,2,\ldots)$ 不是目标状态时：

1. 收集经验样本 $(r_{t+1}, s_{t+1}, a_{t+1})$：通过与环境交互生成奖励 r_{t+1} 和下一状态 s_{t+1}，再根据当前策略 $\pi_t(s_{t+1})$ 生成动作 a_{t+1}。
2. 更新动作值函数 $q(s_t, a_t)$：
$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)\left[q_t(s_t, a_t) - (r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1}))\right]$$
3. 更新策略 $\pi_{t+1}(a|s_t)$：
$$\pi_{t+1}(a|s_t) = \begin{cases} 1 - \epsilon\left(\frac{|A(s_t)|-1}{|A(s_t)|}\right) & \text{如果} a = \arg\max_a q_{t+1}(s_t, a) \\ \frac{\epsilon}{|A(s_t)|} & \text{其他情况} \end{cases}$$
4. 转移状态和动作：$s_t \leftarrow s_{t+1}$，$a_t \leftarrow a_{t+1}$

Sarsa - Implementation

关于这个算法的一些说明：

- 在更新 $q(s_t, a_t)$ 后立即更新 s_t 的策略。这是基于广义策略迭代的思想。
- 策略是 ϵ-greedy 的，而不是贪婪的，以很好地平衡利用和探索。

任务描述：

- 任务是从一个特定的起始状态找到一条通往目标状态的 good path。
  - 这个任务与前面的所有任务不同，前面的任务需要为每个状态找到最优策略！
  - 每个集从左上角状态开始，以目标状态结束。
  - 在未来，要注意任务是什么。
- $r_{\text{target}} = 0, r_{\text{forbidden}} = r_{\text{boundary}} = -10$，且 $r_{\text{other}} = -1$。学习率为 $\alpha = 0.1$，ϵ 的值为 0.1。

结果：

- 左侧图表显示了 Sarsa 找到的最终策略。
- 并非所有状态都有最优策略。
- 右侧图表显示了每个集的总奖励和长度。
- 每个 Episode 的 Total rewards 这一指标将被频繁使用。

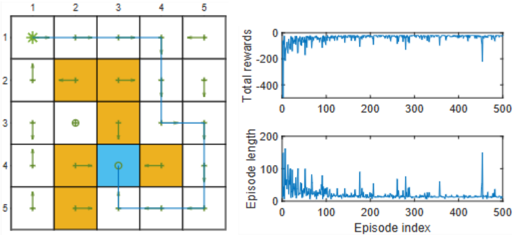


Table of contents

TD learning of action values: n-step Sarsa

n-step Sarsa 可以统一 Sarsa 和 Monte Carlo 学习
动作值函数的定义为

$$q_\pi(s,a) = E[G_t|S_t = s, A_t = a]$$

折扣回报 G_t 可以写成不同的形式：

$$\begin{aligned}
\text{Sarsa} \leftarrow G_t^{(1)} &= R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \\
G_t^{(2)} &= R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi(S_{t+2}, A_{t+2}) \\
&\vdots \\
\text{n-step Sarsa} \leftarrow G_t^{(n)} &= R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^n q_\pi(S_{t+n}, A_{t+n}) \\
&\vdots \\
\text{MC} \leftarrow G_t^{(\infty)} &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots
\end{aligned}$$

应该注意的是 $G_t = G_t^{(1)} = G_t^{(2)} = \cdots = G_t^{(n)} = G_t^{(\infty)}$，其中上标仅表示 G_t 的不同分解结构。

TD learning of action values: n-step Sarsa

- Sarsa 的目标是求解

$$q_\pi(s,a) = E[G_t^{(1)}|s,a] = E[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1})|s,a]$$

- MC 学习的目标是求解

$$q_\pi(s,a) = E[G_t^{(\infty)}|s,a] = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots|s,a]$$

- 中间算法 n-step Sarsa 的目标是求解

$$q_\pi(s,a) = E[G_t^{(n)}|s,a] = E[R_{t+1} + \gamma R_{t+2} + \dots + \gamma^n q_\pi(S_{t+n}, A_{t+n})|s,a]$$

- n-step Sarsa 的算法为

$$\begin{aligned} q_{t+1}(s_t,a_t) = {} & q_t(s_t,a_t) - \\ & \alpha_t(s_t,a_t)[q_t(s_t,a_t) - [r_{t+1} + \gamma r_{t+2} + \dots + \gamma^n q_t(s_{t+n}, a_{t+n})]] \end{aligned}$$

  - 当 $n=1$ 时，n-step Sarsa 变为（一步）Sarsa 算法。
  - 当 $n=\infty$ 时，n-step Sarsa 变为 MC 学习算法。

TD learning of action values: n-step Sarsa

- 数据：n-step Sarsa 需要 $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, \ldots, r_{t+n}, s_{t+n}, a_{t+n})$。
- 由于在时间 t 时尚未收集到 $(r_{t+n}, s_{t+n}, a_{t+n})$，因此我们无法在时间步 t 实施 n-step Sarsa。我们需要等到时间 $t+n$ 才能更新 (s_t, a_t) 的 q 值：

$$q_{t+n}(s_t, a_t) = q_{t+n-1}(s_t, a_t) -$$
$$\alpha_{t+n-1}(s_t, a_t)[q_{t+n-1}(s_t, a_t) - [r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^n q_{t+n-1}(s_{t+n}, a_{t+n}$$

- 由于 n-step Sarsa 包含 Sarsa 和 MC 学习作为两个极端情况，其性能是 Sarsa 和 MC 学习的混合：
  - 如果 n 较大，其性能接近 MC 学习，因此具有较大的方差但偏差较小。
  - 如果 n 较小，其性能接近 Sarsa，因此由于初始猜测而具有相对较大的偏差和相对较低的方差。
- 最后，n-step Sarsa 也是用于策略评估的。它可以与策略改进步骤结合起来寻找最优策略。

Table of contents

TD learning of optimal action values: Q-learning

- 接下来，我们介绍 Q-learning，这是最广泛使用的强化学习算法之一。
- Sarsa 可以估计给定策略的动作值。它必须与策略改进步骤结合起来才能找到最优策略。
- Q-learning 可以直接估计最优动作值，因此可以直接找到最优策略。

Q-learning - Algorithm

Q-learning 算法为

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)[q_t(s_t, a_t) - [r_{t+1} + \gamma \max_{a \in \mathcal{A}} q_t(s_{t+1}, a)]]$$

$$q_{t+1}(s, a) = q_t(s, a), \quad \forall (s, a) \neq (s_t, a_t)$$

Q-learning 与 Sarsa 非常相似。它们在 TD target 方面有所不同：

- Q-learning 的 TD target 是 $r_{t+1} + \gamma \max_{a \in \mathcal{A}} q_t(s_{t+1}, a)$
- Sarsa 的 TD target 是 $r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1})$

Q-learning 在数学上做了什么？它旨在求解

$$q(s, a) = E[R_{t+1} + \gamma \max_a q(S_{t+1}, a) | S_t = s, A_t = a], \quad \forall s, a$$

这是用动作值表示的 Bellman 最优性方程。

- 因此，Q-learning 可以直接估计最优动作值，而不是给定策略的动作值。

Off-policy vs On-policy

在进一步研究 Q-learning 之前，我们首先介绍两个重要概念：on-policy 学习和 off-policy 学习。
在 TD 学习任务中存在两种策略：

- 行为策略用于生成经验样本。
- 目标策略不断更新，以接近最优策略。

On-policy 与 off-policy：

- 当行为策略与目标策略相同时，这种学习被称为 on-policy。
- 当它们不同时，学习被称为 off-policy。

Off-policy vs On-policy

Off-policy 学习的优点：

- 它可以根据任何其他策略生成的经验样本来搜索最优策略。
- 例如：行为策略是探索性的，因此我们可以生成足够多次访问每个状态-动作对的集。

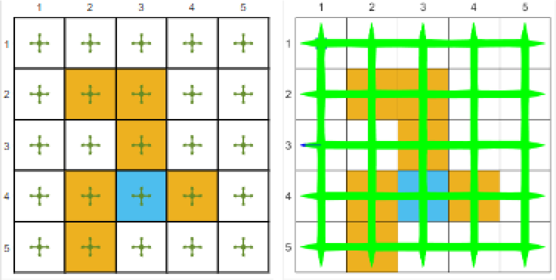


图: (a) 探索性行为策略 (b) 生成的集

Off-policy vs On-policy

- Sarsa 旨在通过求解如下贝尔曼方程，来评估给定策略 π，

$$q_\pi(s,a) = E[R + \gamma q_\pi(S', A') | s, a], \quad \forall s, a$$

其中 $R \sim p(R|s,a)$，$S' \sim p(S'|s,a)$，$A' \sim \pi(A'|S')$。

- MC 旨在通过求解

$$q_\pi(s,a) = E[R_{t+1} + \gamma R_{t+2} + \cdots | S_t = s, A_t = a], \quad \forall s, a$$

来评估给定策略 π，其中样本由 π 生成。

- Sarsa 和 MC 都是 on-policy 的。
  - π 是行为策略，因为我们需要由 π 生成的经验样本来估计 π 的动作值。
  - π 也是目标策略，因为它不断更新，以接近最优策略。

Off-policy vs On-policy

Q-learning 是 off-policy 的。

- 首先，Q-learning 旨在求解 Bellman 最优性方程

$$q(s,a) = E[R_{t+1} + \gamma \max_a q(S_{t+1},a) | S_t = s, A_t = a], \quad \forall s, a$$

- 其次，算法为

$$q_{t+1}(s_t,a_t) = q_t(s_t,a_t) - \alpha_t(s_t,a_t)[q_t(s_t,a_t) - [r_{t+1} + \gamma \max_{a\in\mathcal{A}} q_t(s_{t+1},a)]]$$

  它需要 $(s_t, a_t, r_{t+1}, s_{t+1})$。

- 行为策略是用于在 s_t 中生成 a_t 的策略。目标策略是选择最大的 q 的 a 的策略，可以是任意策略。

Q-learning - Implementation (On-policy)

由于 Q-learning 是 off-policy 的，因此它可以以 off-policy 或 on-policy 的方式实现。

1. 对于每个 Episode：
   1. 如果 s_t $(t = 0, 1, 2, \ldots)$ 不是目标状态：
      1. 收集经验样本 (a_t, r_{t+1}, s_{t+1}) 给定 s_t：根据 $\pi_t(s_t)$ 生成 a_t；通过与环境交互生成 r_{t+1}, s_{t+1}
      2. 更新 q 值，对于 (s_t, a_t)：

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)[q_t(s_t, a_t) - (r_{t+1} + \gamma \max_a q_t(s_{t+1}, a))]$$

      3. 更新策略，对于 s_t：

$$\pi_{t+1}(a|s_t) = \begin{cases} 1 - \epsilon\left(\frac{1}{|A(s_t)|}\right) & \text{如果} a = \arg\max_a q_{t+1}(s_t, a) \\ \frac{\epsilon}{|A(s_t)|} & \text{其他情况} \end{cases}$$

Q-learning - Implementation (Off-policy)

目标：从由 π_b 生成的经验样本中学习最优目标策略 π_T，适用于所有状态。

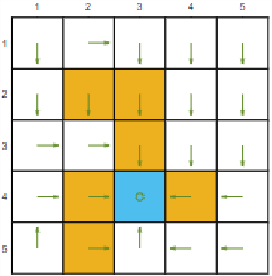
1. 对于每个由 π_b 生成的集 $\{s_0, a_0, r_1, s_1, a_1, r_2, \ldots\}$：
   1. 对于集中的每个步骤 $t = 0, 1, 2, \ldots$：
      1. 更新 q 值，对于 (s_t, a_t)：
         $$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)[q_t(s_t, a_t) - (r_{t+1} + \gamma \max_a q_t(s_{t+1}, a))]$$
      2. 更新目标策略，对于 s_t：
         $$\pi_{T,t+1}(a|s_t) = \begin{cases} 1 & \text{如果} a = \arg\max_a q_{t+1}(s_t, a) \\ 0 & \text{其他情况} \end{cases}$$

Q-learning - Examples

任务描述：

- 这些例子中的任务是为所有状态找到最优策略。
- 奖励设置为 $r_{\text{boundary}} = r_{\text{forbidden}} = -1$，且 $r_{\text{target}} = 1$。折扣率为 $\gamma = 0.9$。学习率为 $\alpha = 0.1$。

Ground-Truth：最优策略和相应的最优状态值。



	1	2	3	4	5
1	5.8	5.6	6.2	6.5	5.8
2	6.5	7.2	8.0	7.2	6.5
3	7.2	8.0	10.0	8.0	7.2
4	8.0	10.0	10.0	10.0	8.0
5	7.2	9.0	10.0	9.0	8.1

图: (a) 最优策略 (b) 最优状态值

Q-learning - Examples

行为策略和生成的经验（1 millon steps）：

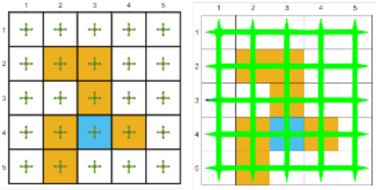


图: (a) 行为策略 (b) 生成的 episode

通过 off-policy Q-learning 找到的策略：

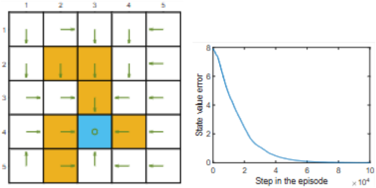


图: (a) 估计的策略 (b) 状态值误差

探索的重要性：1 millon steps 的 Episode

- 如果策略探索性不强，样本就不够好。

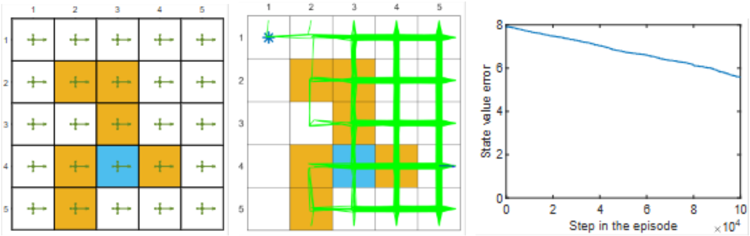


图: (a) 行为策略 $\epsilon = 0.5$ (b) 生成的 episode (c) Q-learning 结果

Q-learning - Examples

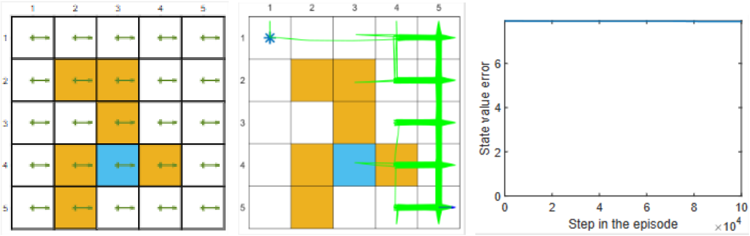


图: (a) 行为策略 $\epsilon = 0.1$ (b) 生成的 episode (c) Q-learning 结果

Table of contents

本章介绍的所有算法都可以用统一的表达式表示：

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)[q_t(s_t, a_t) - \bar{q}_t]$$

其中 $\bar{q}_t$ 是 TD 目标。不同的 TD 算法有不同的 $\bar{q}_t$。

算法	$\bar{q}_t$ 的表达式
Sarsa	$\bar{q}_t = r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1})$
n-step Sarsa	$\bar{q}_t = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^n q_t(s_{t+n}, a_{t+n})$
Q-learning	$\bar{q}_t = r_{t+1} + \gamma \max_a q_t(s_{t+1}, a)$
Monte Carlo	$\bar{q}_t = r_{t+1} + \gamma r_{t+2} + \cdots$

- 通过设置 $\alpha_t(s_t, a_t) = 1$，MC 方法也可以用这种统一的表达式表示。具体来说，表达式为 $q_{t+1}(s_t, a_t) = \bar{q}_t$。

A unified point of view

所有的 TD 算法都可以被视为求解 Bellman 方程或 Bellman 最优性方程的随机逼近算法：

算法	需要解决的方程
Sarsa	BE: $q_\pi(s,a) = E[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \| S_t = s, A_t = a]$
n-step Sarsa	BE: $q_\pi(s,a) = E[R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^n q_\pi(s_{t+n}, a_{t+n}) \| S_t = s, A_t$
Q-learning	BOE: $q(s,a) = E[R_{t+1} + \gamma \max_a q(S_{t+1}, a) \| S_t = s, A_t = a]$
Monte Carlo	BE: $q_\pi(s,a) = E[R_{t+1} + \gamma R_{t+2} + \cdots \| S_t = s, A_t = a]$

Table of contents

Summary

- 介绍了各种 TD 学习算法。
- 它们的表达式、数学解释、实现、关系、例子。
- 统一的观点。