第7章 时序差分方法 (Temporal-Difference Methods)

TD 方法是增量形式 (incremental form)

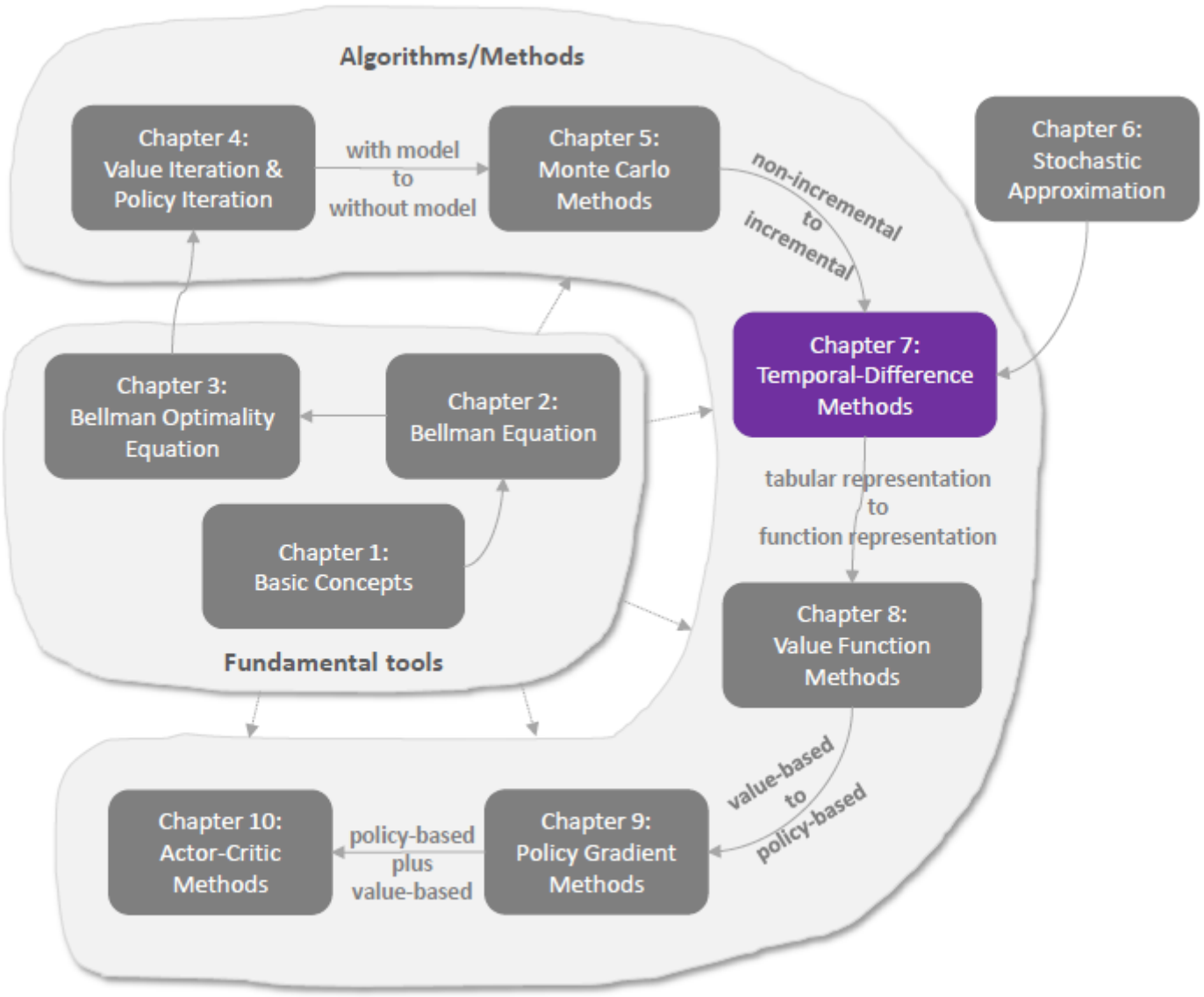


Figure 7.1: Where we are in this book.

图 7.1：我们在本书中的位置。

本章介绍用于强化学习的时序差分 (temporal-difference, TD) 方法。与蒙特卡洛 (MC) 学习类似，TD 学习也是无模型 (model-free) 的，但由于其**增量形式 (incremental form)**，它具有一些优势。有了第 6 章的准备，读者在看到 TD 学习算法时不会感到突兀。事实上，TD 学习算法可以被视为用于求解贝尔曼 (Bellman) 或贝尔曼最优方程的特殊随机算法。

由于本章介绍了相当多的 TD 算法，我们首先概述这些算法并澄清它们之间的关系。

- 7.1 节介绍了**最基本的 TD 算法，它可以估计 *状态 (state)* 值**。在学习其他 TD 算法之前，先理解这个基本算法非常重要。
- 7.2 节介绍了 **Sarsa 算法，它可以估计给定策略的 *动作价值 (action values)***。该算法可以与策略改进步骤相结合以**寻找最优策略**。通过将 7.1 节中 TD 算法的状态价值估计替换为动作价值估计，可以很容易地得到 Sarsa 算法。
- 7.3 节介绍了 n -步 Sarsa 算法，这是 Sarsa 算法的推广。我们将展示 Sarsa 和 MC 学习是 n -步 Sarsa 的两个特例。
- 7.4 节介绍了 Q-learning 算法，这是最经典的强化学习算法之一。虽然其他 TD 算法旨在求解给定策略的贝尔曼方程，但 Q-learning 旨在直接求解贝尔曼最优方程以获得最优策略。
- 7.5 节比较了本章介绍的 TD 算法，并提供了一个统一的视角。

7.1 状态价值的 TD 学习 (TD learning of state values)

TD 学习通常指的是一类广泛的强化学习算法。例如，本章介绍的所有算法都属于 TD 学习的范畴。然而，本节中的 **TD 学习特指一种用于估计状态价值的经典算法**。

7.1.1 算法描述 (Algorithm description)

给定一个策略 π，我们的目标是估计所有 $s \in \mathcal{S}$ 的 $v_\pi(s)$。

📍 注意：在式子里 s_t **的下标** t 和 v_{t+1} **的下标** $t+1$ 虽然都用同一个字母 t，但它们“指代的对象”不一样，一个是“状态序列的时间下标”，一个是“估计函数的迭代下标”

- s_t 的 t ：环境轨迹的时间步

$$(s_0, r_1, s_1, \ldots, s_t, r_{t+1}, s_{t+1}, \ldots)$$

这里的 s_t 表示：**第** t **个时间步（time step）智能体所在的状态**。

从 s_t 出发，你会观察到 r_{t+1} 并到达 s_{t+1}

- v_{t+1} **的下标** $t+1$

v_{t+1} 表示：**在做第** $t+1$ **次更新之前/之后（具体看写法约定）你的价值函数估计**。

但在这类在线算法中它们**按设计同步：走一步环境，就更新一次**，因此作者干脆用同一个 t。

假设我们有一些遵循 π 生成的经验样本 $(s_0, r_1, s_1, \ldots, s_t, r_{t+1}, s_{t+1}, \ldots)$。这里，$t$ 表示时间步。下面的 TD 算法可以使用这些样本来估计状态价值：

$$v_{t+1}(s_t) = v_t(s_t) - \alpha_t(s_t)\left[v_t(s_t) - (r_{t+1} + \gamma v_t(s_{t+1}))\right], \quad (7.1)$$

$$v_{t+1}(s) = v_t(s), \quad \text{对于所有 } s \neq s_t, \quad (7.2)$$

其中 $t = 0, 1, 2, \ldots$。这里，$v_t(s_t)$ 是在时刻 t 对 $v_\pi(s_t)$ 的估计值；$\alpha_t(s_t)$ 是时刻 t 针对 s_t 的学习率。

值得注意的是，在时刻 t，只有被访问的状态 s_t 的价值被更新。如式 (7.2) 所示，未访问状态 $s \neq s_t$ 的价值保持不变。**为了简单起见，式 (7.2) 经常被省略，但应当记住它，因为没有这个方程，算法在数学上是不完整的。**

初次接触 TD 学习算法的读者可能会好奇它为什么是这样设计的。事实上，它可以被视为一种用于求解贝尔曼方程的特殊随机逼近算法。为了理解这一点，首先回顾状态价值的定义是

$$v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma G_{t+1} | S_t = s], \quad s \in \mathcal{S}. \quad (7.3)$$

我们可以将 (7.3) 重写为***贝尔曼期望方程 (Bellman expectation equation)***

$$v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s], \quad s \in \mathcal{S}. \quad (7.4)$$

这是因为 $\mathbb{E}[G_{t+1}|S_t = s] = \sum_a \pi(a|s) \sum_{s'} p(s'|s,a) v_\pi(s') = \mathbb{E}[v_\pi(S_{t+1})|S_t = s]$。方程 (7.4) 是贝尔曼方程的另一种表达形式。它有时被称为 *贝尔曼期望方程 (Bellman expectation equation)*。

TD 算法可以通过应用 Robbins-Monro 算法（第 6 章）求解 (7.4) 中的贝尔曼方程推导而来。感兴趣的读者可以在方框 7.1 中查看详细信息。

💡 方框 7.1：TD 算法的推导 (Derivation of the TD algorithm)

接下来我们展示 (7.1) 中的 TD 算法可以通过应用 Robbins-Monro 算法求解 (7.4) 获得。

对于状态 s_t，我们定义一个函数为(这个式子**就是从状态价值函数和 return（回报）定义推出来的**，理论上它应该等于 0)

$$g(v_\pi(s_t)) \doteq v_\pi(s_t) - \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s_t].$$

那么，(7.4) 等价于

$$g(v_\pi(s_t)) = 0.$$

我们的目标是使用 Robbins-Monro 算法求解上述方程以获得 $v_\pi(s_t)$。由于我们可以获得 r_{t+1} 和 s_{t+1}，它们分别是 R_{t+1} 和 S_{t+1} 的样本，我们可以获得的 $g(v_\pi(s_t))$ 的含噪观测是

$$
\begin{aligned}
\tilde{g}(v_\pi(s_t)) &= v_\pi(s_t) - [r_{t+1} + \gamma v_\pi(s_{t+1})] \\
&= \underbrace{(v_\pi(s_t) - \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s_t])}_{g(v_\pi(s_t))} \\
&\quad + \underbrace{(\mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s_t] - [r_{t+1} + \gamma v_\pi(s_{t+1})])}_{\eta}.
\end{aligned}
$$

因此，用于求解 $g(v_\pi(s_t)) = 0$ 的 Robbins-Monro 算法（第 6.2 节）为：

$$
\begin{aligned}
v_{t+1}(s_t) &= v_t(s_t) - \alpha_t(s_t)\tilde{g}(v_t(s_t)) \\
&= v_t(s_t) - \alpha_t(s_t)\left(v_t(s_t) - [r_{t+1} + \gamma v_\pi(s_{t+1})]\right), \quad (7.5)
\end{aligned}
$$

其中 $v_t(s_t)$ 是时刻 t 对 $v_\pi(s_t)$ 的估计值，$\alpha_t(s_t)$ 是学习率。

(7.5) 中的算法与 (7.1) 中的 TD 算法具有相似的表达式。唯一的区别在于 (7.5) 的右侧包含 $v_\pi(s_{t+1})$，而 (7.1) 包含 $v_t(s_{t+1})$。**这是因为 (7.5) 的设计初衷是在假设其他状态的价值已知的情况下，仅估计 s_t 的状态价值**。如果我们想要估计所有状态的状态价值，那么右侧的 $v_\pi(s_{t+1})$ 应该被替换为 $v_t(s_{t+1})$。这样，(7.5) 就与 (7.1) 完全相同了。**然而，这样的替换还能保证收敛吗？答案是肯定的，这将在稍后的定理 7.1 中得到证明。**

7.1.2 属性分析 (Property analysis)

接下来讨论 TD 算法的一些重要属性。

首先，我们更仔细地检查 TD 算法的表达式。具体来说，(7.1) 可以描述为

$$
\underbrace{v_{t+1}(s_t)}_{\text{新估计值}} = \underbrace{v_t(s_t)}_{\text{当前估计值}} - \underbrace{\alpha_t(s_t)[v_t(s_t) - \underbrace{(r_{t+1} + \gamma v_t(s_{t+1}))}_{\text{TD 目标}\bar{v}_t\text{(可以理解为从定义出发的理论值)}}]}_{\text{TD 误差 } \delta_t}, \quad (7.6)
$$

其中

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

被称为 **TD 目标 (TD target)**，而

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

被称为 *TD 误差 (TD error)*。

$$
\bar{v}_t\text{(可以理解为从定义出发的理论值)}
$$

可以看出，新估计值 $v_{t+1}(s_t)$ 是当前估计值 $v_t(s_t)$ 和 TD 误差 δ_t 的组合。

- **为什么 $\bar{v}_t$ 被称为 TD 目标?**
- 这是因为 $\bar{v}_t$ 是算法试图将 $v(s_t)$ 驱动到的 **目标值 (target value)**。为了看清这一点，从 (7.6) 两边同时减去 $\bar{v}_t$ 可得

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

对上述方程两边取绝对值可得

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

由于 $\alpha_t(s_t)$ 是一个很小的正数，我们有 $0 < 1 - \alpha_t(s_t) < 1$。由此可得

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

上述不等式很重要，因为它表明新值 $v_{t+1}(s_t)$ 比旧值 $v_t(s_t)$ 更接近 $\bar{v}_t$。因此，该算法在数学上驱动 $v_t(s_t)$ 朝向 $\bar{v}_t$。这就是为什么 $\bar{v}_t$ 被称为 TD 目标。

- **TD 误差 (TD error) 的解释是什么?**

首先，这个误差被称为 ***时序差分 (temporal-difference)***，因为 $\delta_t = v_t(s_t) - (r_{t+1} + \gamma v_t(s_{t+1}))$ 反映了 t 和 $t+1$ 两个时间步之间的差异。**其次，当状态价值估计准确时，TD 误差在期望意义下为零。**为了看到这一点，当 $v_t = v_\pi$ 时，TD 误差的期望值为

$$\begin{aligned}\mathbb{E}[\delta_t|S_t=s_t] &= \mathbb{E}[v_\pi(S_t)-(R_{t+1}+\gamma v_\pi(S_{t+1}))|S_t=s_t] \\ &= v_\pi(s_t)-\mathbb{E}[R_{t+1}+\gamma v_\pi(S_{t+1})|S_t=s_t] \\ &= 0. \qquad (\text{由于 } (7.3))\end{aligned}$$

因此，TD 误差不仅反映了两个时间步之间的差异，更重要的是，它反映了**估计值** v_t 与**真实状态价值** v_π 之间的差异。

在更抽象的层面上，TD 误差可以被解释为 **新信息 *(innovation)***，它表示**从经验样本** (s_t, r_{t+1}, s_{t+1}) **中获得的新信息**。TD 学习的基本思想就是基于新获得的信息来修正我们当前对状态价值的估计。新信息在许多估计问题中都是基础，例如卡尔曼滤波 (Kalman filtering) [33, 34]。

- 第二，式 (7.1) 中的 TD 算法只能估计 ***给定策略*** **的状态价值。为了找到最优策略，我们仍然需要进一步计算动作价值，然后进行策略改进**。这将在 7.2 节中介绍。尽管如此，本节介绍的 TD 算法是非常基础的，对于理解本章中的其他算法非常重要。
- 第三，虽然 TD 学习和 MC 学习都是无模型 (model-free) 的，但它们的优缺点是什么？答案总结在表 7.1 中。

TD 学习 (TD learning)	MC 学习 (MC learning)
增量式 (Incremental)： TD 学习是增量式的。**它可以在接收到一个经验样本后立即更新状态/动作价值。**	**非增量式 (Non-incremental)：** MC 学习是非增量式的。它必须等到一个回合 (episode) 完全收集完毕。这是因为它必须计算该回合的折扣回报。
持续性任务 (Continuing tasks)： 由于 TD 学习是增量式的，它可以处理回合制任务和持续性任务。持续性任务可能没有终止状态。	**回合制任务 (Episodic tasks)：** 由于 MC 学习是非增量式的，它只能处理回合制任务，即回合在有限步数后终止的任务。
自举 (Bootstrapping)： TD 学习进行自举，因为状态/动作价值的更新依赖于该价值的先前估计。结果是，**TD 学习需要价值的初始猜测。**	**非自举 (Non-bootstrapping)：** MC 不进行自举，因为它可以直接估计状态/动作价值而无需初始猜测。
低估计方差 (Low estimation variance)： TD 的估计方差低于 MC，因为它涉及较少的随机变量。例如，为了估计动作价值 $q_\pi(s_t,a_t)$，Sarsa 仅需要三个随机变量的样本：$R_{t+1}, S_{t+1}, A_{t+1}$。	**高估计方差 (High estimation variance)：** MC 方法的估计方差较高，因为它涉及许多随机变量。例如，为了估计动作价值 $q_\pi(s_t,a_t)$，我们需要 $R_{t+1}+\gamma R_{t+2}+\gamma^2 R_{t+3}+\dots$ 的样本。假设每个回合的长度为 L。假设每个状态拥有的动作数量相同，均为 $\lvert\mathcal{A}\rvert$。那么，遵循一个软策略（soft policy）可能产生的回合总共有 $\lvert\mathcal{A}\rvert^L$ 种。如果我们仅使用少量的回合样本来进行估计，那么估计方差较高也就不足为奇了。

表 7.1：TD 学习与 MC 学习的比较。

7.1.3 收敛性分析 (Convergence analysis)

TD 算法 (7.1) 的收敛性分析如下。

定理 7.1 (TD 学习的收敛性)。*给定策略 π，通过 (7.1) 中的 TD 算法，如果对于所有 $s\in\mathcal{S}$ 满足 $\sum_t \alpha_t(s)=\infty$ 和 $\sum_t \alpha_t^2(s)<\infty$，则当 $t\to\infty$ 时，$v_t(s)$ 几乎处处收敛于 $v_\pi(s)$。*

下面给出关于 α_t 的一些注记。

- 首先，条件 $\sum_t \alpha_t(s)=\infty$ 和 $\sum_t \alpha_t^2(s)<\infty$ 必须对所有 $s\in\mathcal{S}$ 有效。注意，在时刻 t，如果 s 被访问，则 $\alpha_t(s)>0$，否则 $\alpha_t(s)=0$。

条件 $\sum_t \alpha_t(s) = \infty$ 要求状态 s 被访问无限次（或足够多次）。这需要探索性出发 (exploring starts) 条件或探索性策略，以便每个“状态-动作”对都有可能被访问多次。

- 其次，学习率 α_t 通常被选择为一个很小的正*常数 (constant)*。在这种情况下，$\sum_t \alpha_t^2(s) < \infty$ 的条件不再成立。当 α 为常数时，仍然可以证明算法在期望意义下收敛 [24, Section 1.5]。

方框 7.2：定理 7.1 的证明 (Proof of Theorem 7.1)

我们基于第 6 章中的定理 6.3 来证明收敛性。为此，我们需要首先构造一个如定理 6.3 中那样的随机过程。考虑任意状态 $s \in \mathcal{S}$。在时刻 t ，由 (7.1) 中的 TD 算法可得

$$v_{t+1}(s) = v_t(s) - \alpha_t(s)\left(v_t(s) - (r_{t+1} + \gamma v_t(s_{t+1}))\right), \quad \text{如果 } s = s_t, \quad (7.7)$$

或

$$v_{t+1}(s) = v_t(s), \quad \text{如果 } s \neq s_t. \quad (7.8)$$

估计误差定义为

$$\Delta_t(s) \doteq v_t(s) - v_\pi(s),$$

其中 $v_\pi(s)$ 是策略 π 下 s 的状态价值。从 (7.7) 两边同时减去 $v_\pi(s)$ 可得

$$\begin{aligned}\Delta_{t+1}(s) &= (1-\alpha_t(s))\Delta_t(s) + \alpha_t(s)\underbrace{(r_{t+1} + \gamma v_t(s_{t+1}) - v_\pi(s))}_{\eta_t(s)} \\ &= (1-\alpha_t(s))\Delta_t(s) + \alpha_t(s)\eta_t(s), \quad s = s_t. \qquad (7.9)\end{aligned}$$

从 (7.8) 两边同时减去 $v_\pi(s)$ 可得

$$\Delta_{t+1}(s) = \Delta_t(s) = (1-\alpha_t(s))\Delta_t(s) + \alpha_t(s)\eta_t(s), \quad s \neq s_t,$$

其表达式与 (7.9) 相同，只是 $\alpha_t(s) = 0$ 且 $\eta_t(s) = 0$。**因此，无论是否 $s = s_t$ ，我们都获得以下统一表达式：**

$$\Delta_{t+1}(s) = (1-\alpha_t(s))\Delta_t(s) + \alpha_t(s)\eta_t(s).$$

这就是定理 6.3 中的过程。**我们的目标是证明定理 6.3 中的三个条件都得到满足，因此该过程收敛。**

第一个条件根据定理 7.1 中的假设是成立的。

接下来我们要证明第二个条件也是成立的。即，对于所有 $s \in \mathcal{S}$ ， $\|\mathbb{E}[\eta_t(s)|\mathcal{H}_t]\|_\infty \leq \gamma\|\Delta_t(s)\|_\infty$。这里， $\mathcal{H}_t$ 表示历史信息（见定理 6.3 中的定义）。

由于马尔可夫性质， $\eta_t(s) = r_{t+1} + \gamma v_t(s_{t+1}) - v_\pi(s)$ 或 $\eta_t(s) = 0$,一旦 s 给定，便不再依赖于历史信息。结果是，我们有 $\mathbb{E}[\eta_t(s)|\mathcal{H}_t] = \mathbb{E}[\eta_t(s)]$。对于 $s \neq s_t$ ，我们有 $\eta_t(s) = 0$。那么，显而易见的是

$$|\mathbb{E}[\eta_t(s)]| = 0 \leq \gamma\|\Delta_t(s)\|_\infty. \quad (7.10)$$

对于 $s = s_t$ ，我们有

$$\begin{aligned}\mathbb{E}[\eta_t(s)] &= \mathbb{E}[\eta_t(s_t)] \\ &= \mathbb{E}[r_{t+1} + \gamma v_t(s_{t+1}) - v_\pi(s_t)|s_t] \\ &= \mathbb{E}[r_{t+1} + \gamma v_t(s_{t+1})|s_t] - v_\pi(s_t).\end{aligned}$$

由于 $v_\pi(s_t) = \mathbb{E}[r_{t+1} + \gamma v_\pi(s_{t+1})|s_t]$ ，上述方程意味着

$$\begin{aligned}\mathbb{E}[\eta_t(s)] &= \gamma\mathbb{E}[v_t(s_{t+1}) - v_\pi(s_{t+1})|s_t] \\ &= \gamma\sum_{s' \in \mathcal{S}} p(s'|s_t)[v_t(s') - v_\pi(s')].\end{aligned}$$

由此可得

$$
\begin{aligned}
|\mathbb{E}[\eta_t(s)]| &= \left|\gamma \sum_{s' \in \mathcal{S}} p(s'|s_t)[v_t(s') - v_\pi(s')]\right| \\
&\leq \gamma \sum_{s' \in \mathcal{S}} p(s'|s_t) \max_{s' \in \mathcal{S}} |v_t(s') - v_\pi(s')| \\
&= \gamma \max_{s' \in \mathcal{S}} |v_t(s') - v_\pi(s')| \\
&= \gamma \|v_t(s') - v_\pi(s')\|_\infty \\
&= \gamma \|\Delta_t(s)\|_\infty. \quad (7.11)
\end{aligned}
$$

因此，在时刻 t ，我们由 (7.10) 和 (7.11) 可知，无论是否 $s = s_t$ ，对于所有 $s \in \mathcal{S}$ 都有 $|\mathbb{E}[\eta_t(s)]| \leq \gamma \|\Delta_t(s)\|_\infty$ 。于是，

$$
\|\mathbb{E}[\eta_t(s)]\|_\infty \leq \gamma \|\Delta_t(s)\|_\infty,
$$

这正是定理 6.3 中的第二个条件。

最后，关于第三个条件，对于 $s = s_t$ 我们有

$$
\mathrm{var}[\eta_t(s)|\mathcal{H}_t] = \mathrm{var}[r_{t+1} + \gamma v_t(s_{t+1}) - v_\pi(s_t)|s_t] = \mathrm{var}[r_{t+1} + \gamma v_t(s_{t+1})|s_t]
$$

而对于 $s \neq s_t$ 有 $\mathrm{var}[\eta_t(s)|\mathcal{H}_t] = 0$ 。由于 r_{t+1} 是有界的，第三个条件可以毫无困难地被证明。

上述证明受到 [32] 的启发。

7.2 动作价值的 TD 学习：Sarsa (TD learning of action values: Sarsa)

第 7.1 节介绍的 TD 算法只能估计 *状态价值 (state values)*。本节介绍另一种称为 Sarsa 的 TD 算法，它可以直接估计 *动作价值 (action values)*。估计动作价值很重要，因为它可以与策略改进步骤相结合来学习最优策略。

7.2.1 算法描述 (Algorithm description)

给定一个策略 π ，我们的目标是估计动作价值。假设我们有一些遵循 π 生成的经验样本：$(s_0, a_0, r_1, s_1, a_1, \ldots, s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, \ldots)$ 。我们可以使用下面的 *Sarsa* 算法来估计动作价值：

$$
\begin{aligned}
q_{t+1}(s_t, a_t) &= q_t(s_t, a_t) - \alpha_t(s_t, a_t)\left[q_t(s_t, a_t) - (r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1}))\right], \quad (7.12) \\
q_{t+1}(s, a) &= q_t(s, a), \quad \text{对于所有 } (s, a) \neq (s_t, a_t),
\end{aligned}
$$

其中 $t = 0, 1, 2, \ldots$ 且 $\alpha_t(s_t, a_t)$ 是学习率。这里， $q_t(s_t, a_t)$ 是在时刻 t 对 $q_\pi(s_t, a_t)$ 的估计值。

在时刻 t ，只有 (s_t, a_t) 的 q 值被更新，而其他项的 q 值保持不变。

下面讨论 Sarsa 算法的一些重要属性。

- **为什么这个算法被称为“Sarsa”？** 这是因为算法的每次迭代都需要 $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ 。

  Sarsa 是状态-动作-奖励-状态-动作 (state-action-reward-state-action) 的缩写。Sarsa 算法最早由 [35] 提出，其名称由 [3] 创造。

- **为什么 Sarsa 被设计成这样？** 人们可能已经注意到 Sarsa 与 (7.1) 中的 TD 算法相似。事实上，通过将状态价值估计替换为动作价值估计，可以很容易地从 TD 算法中获得 Sarsa。

- **Sarsa 在数学上做什么？** 类似于 (7.1) 中的 TD 算法，Sarsa 是一种用于求解给定策略的贝尔曼方程的随机逼近算法：

$$
q_\pi(s, a) = \mathbb{E}[R + \gamma q_\pi(S', A')|s, a], \quad \text{对所有 } (s, a). \quad (7.13)
$$

  方程 (7.13) 是以**动作价值表示的贝尔曼方程**。方框 7.3 中给出了证明。

📍 方框 7.3：证明 (7.13) 是贝尔曼方程 (Showing that (7.13) is the Bellman equation)

正如 2.8.2 节所介绍的，以动作价值表示的贝尔曼方程为

$$
\begin{aligned}
q_\pi(s, a) &= \sum_r r p(r|s, a) + \gamma \sum_{s'} \sum_{a'} q_\pi(s', a') p(s', a'|s, a) \pi(a'|s') \\
&= \sum_r r p(r|s, a) + \gamma \sum_{s'} p(s'|s, a) \sum_{a'} q_\pi(s', a') \pi(a'|s'). \quad (7.14)
\end{aligned}
$$

$$q_\pi(s,a)=\sum_{r\in\mathcal{R}}p(r|s,a)r+\gamma\sum_{s'\in\mathcal{S}}p(s'|s,a)\sum_{a'\in\mathcal{A}(s')}\pi(a'|s')q_\pi(s',a'),$$ (2.8.2节的公式)

该方程建立了动作价值之间的关系。由于

$$\begin{aligned}p(s',a'|s,a)&=p(s'|s,a)p(a'|s',s,a)\\&=p(s'|s,a)p(a'|s')\quad\text{(由于条件独立性)}\\&\doteq p(s'|s,a)\pi(a'|s'),\end{aligned}$$

(7.14) 可以重写为

$$q_\pi(s,a)=\sum_r rp(r|s,a)+\gamma\sum_{s'}\sum_{a'}q_\pi(s',a')p(s',a'|s,a).$$

根据期望值的定义，上述方程等价于 (7.13)。因此，(7.13) 就是贝尔曼方程。

- **Sarsa 收敛吗?** 由于 Sarsa 是 (7.1) 中 TD 算法的动作价值版本，其收敛结果类似于定理 7.1，如下所示。

定理 7.2 (Sarsa 的收敛性)。*给定策略 π，通过 (7.12) 中的 Sarsa 算法，如果对于所有 (s,a) 满足 $\sum_t\alpha_t(s,a)=\infty$ 和 $\sum_t\alpha_t^2(s,a)<\infty$，则当 $t\to\infty$ 时，$q_t(s,a)$ 几乎处处收敛于动作价值 $q_\pi(s,a)$。*

证明过程与定理 7.1 类似，在此省略。

- $\sum_t\alpha_t(s,a)=\infty$ 和 $\sum_t\alpha_t^2(s,a)<\infty$ 的条件必须对所有 (s,a) 成立。特别地，$\sum_t\alpha_t(s,a)=\infty$ 要求每个状态-动作对必须被访问无限次（或足够多次）。
- 在时刻 t，如果 $(s,a)=(s_t,a_t)$，则 $\alpha_t(s,a)>0$；否则 $\alpha_t(s,a)=0$。

7.2.2 通过 Sarsa 学习最优策略 (Optimal policy learning via Sarsa)

(7.12) 中的 Sarsa 算法只能估计给定策略的动作价值。为了找到最优策略，我们可以将其与策略改进步骤相结合。这种组合通常也被称为 Sarsa，其实现过程在算法 7.1 中给出。

✏️ **算法 7.1：通过 Sarsa 进行最优策略学习 (Optimal policy learning by Sarsa)**

注意：Sarsa是一个在线算法，需要和环境交互获得状态，动作，从而更新每个状态下的策略分布（概率分布）

初始化：对于所有 (s,a) 和所有 t，$\alpha_t(s,a)=\alpha>0$。$\epsilon\in(0,1)$。初始化所有 (s,a) 的 $q_0(s,a)$。源自 q_0 的初始 ϵ -贪婪策略 π_0。

目标：**学习一个最优策略，该策略可以将智能体从初始状态 s_0 引导至目标状态。**

对于每个episode，Do

在 s_0 处根据 $\pi_0(s_0)$ 生成 a_0

如果 $s_t(t=0,1,2,\dots)$ 不是目标状态，Do

在给定 (s_t,a_t) 的情况下收集经验样本 $(r_{t+1},s_{t+1},a_{t+1})$

通过与环境交互生成 r_{t+1},s_{t+1}；根据 $\pi_t(s_{t+1})$ 生成 a_{t+1}。

更新 (s_t,a_t) 的 q 值：

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

更新 s_t 的策略（更新概率分布）：

$$\pi_{t+1}(a|s_t)=1-\frac{\epsilon}{|\mathcal{A}(s_t)|}(|\mathcal{A}(s_t)|-1)\text{ 如果 }a=\arg\max_a q_{t+1}(s_t,a)$$

$$\pi_{t+1}(a|s_t)=\frac{\epsilon}{|\mathcal{A}(s_t)|}\text{ 其他情况}$$

$s_t\leftarrow s_{t+1},a_t\leftarrow a_{t+1}$

如算法 7.1 所示，每次迭代有两个步骤。第一步是更新被访问的状态-动作对的 q 值。

第二步是将策略更新为 ϵ -贪婪策略。q 值更新步骤仅更新时刻 t 被访问的单个状态-动作对。

随后， s_t 的策略立即被更新。因此，我们在更新策略之前并没有充分评估给定的策略。这是基于 *广义策略迭代 (generalized policy iteration)* 的思想。**此外，在策略更新后，该策略立即被用于生成下一个经验样本。这里的策略是 ϵ -贪婪的，因此它是探索性的。**

图 7.2 展示了一个仿真示例来演示 Sarsa 算法。与我们在本书中见过的所有任务不同，这里的任务旨在寻找从特定起始状态到目标状态的最优路径。它并不旨在寻找所有状态的最优策略。这种任务在实践中经常遇到，其中起始状态（例如，家）和目标状态（例如，工作场所）是固定的，我们只需要找到连接它们的最优路径。这个任务相对简单，因为我们只需要探索靠近路径的状态，而不需要探索所有状态。然而，如果我们不探索所有状态，最终的路径可能是 *局部* 最优的，而不是 *全局* 最优的。

仿真设置和仿真结果讨论如下。

- **仿真设置 (Simulation setup)：** 在此示例中，所有回合都从左上角状态开始，并在目标状态终止。奖励设置如下： $r_{\text{target}} = 0$, $r_{\text{forbidden}} = r_{\text{boundary}} = -10$, 且 $r_{\text{other}} = -1$ 。此外，对于所有 t ， $a_t(s,a) = 0.1$ ，且 $\epsilon = 0.1$ 。动作价值的初始猜测值为 $q_0(s,a) = 0$ （对所有 (s,a) ）。初始策略具有均匀分布：对于所有 s, a ， $\pi_0(a|s) = 0.2$ 。

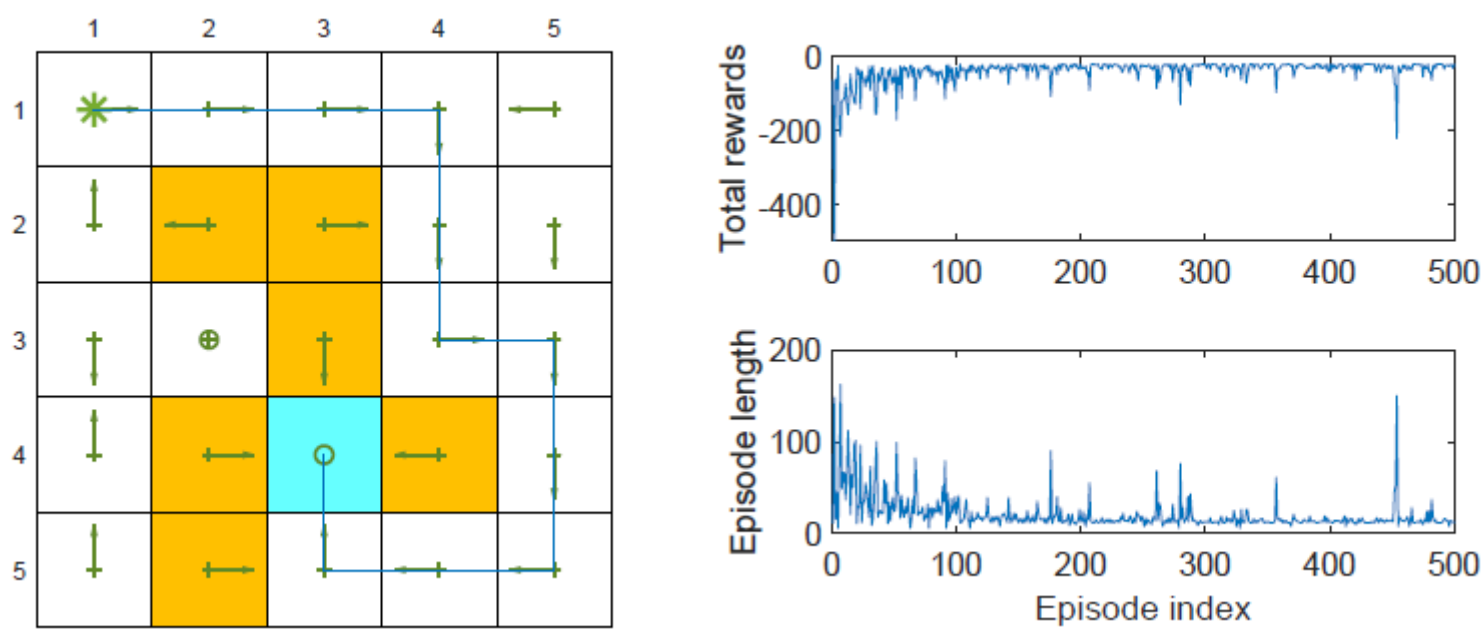


图 7.2：演示 Sarsa 算法的一个示例。所有回合都从左上角的状态开始，并在到达目标状态（蓝色单元格）时终止。目标是找到一条从起始状态到目标状态的最优路径。奖励设置如下： $r_{\text{target}} = 0$, $r_{\text{forbidden}} = r_{\text{boundary}} = -10$, 且 $r_{\text{other}} = -1$ 。学习率 $\alpha = 0.1$ ， ϵ 的值为 0.1。左图显示了算法获得的最终策略。右边的图显示了每个回合的总奖励和长度。

- **学习到的策略 (Learned policy)：** 图 7.2 中的左图显示了 Sarsa 学习到的最终策略。可以看出，该策略可以成功地将智能体从起始状态引导至目标状态。然而，其他一些状态的策略可能不是最优的。这是因为其他状态没有得到充分的探索。
- **每个回合的总奖励 (Total reward of each episode)：** 图 7.2 中的右上子图显示了每个回合的总奖励。这里，总奖励是所有即时奖励的非折扣总和。可以看出，每个回合的总奖励逐渐增加。这是因为初始策略不好，因此经常获得负奖励。随着策略变得更好，总奖励也随之增加。
- **每个回合的长度 (Length of each episode)：** 图 7.2 中的右下子图显示每个回合的长度逐渐下降。这是因为初始策略不好，在到达目标之前可能会绕很多弯路。随着策略变得更好，轨迹的长度变得更短。值得注意的是，回合的长度可能会突然增加（例如第 460 个回合），相应的总奖励也会急剧下降。这是因为策略是 ϵ -贪婪的，它有机会采取非最优动作。解决这个问题的一种方法是使用衰减的 ϵ ，其值逐渐收敛于零。

🎉 额外的例子，ChatGPT生成的

用一个 5 步 episode 跑一遍 Sarsa（最直观版）

假设一个 episode 正好走 5 次转移（从 t=0 到 t=4），到达终止状态 s_5 。

你可以把它想成你拿着一张表 $q(\text{state}, \text{action})$ ，边走边填。

Episode 开始

- 你在起点状态： s_0
- 用当前策略 π_0 选动作： $a_0 \sim \pi_0(\cdot \mid s_0)$

第 0 步更新（$t=0$）

1. 与环境交互：执行 a_0，得到 r_1 和新状态 s_1
2. 在 s_1 用当前策略选下一个动作：$a_1 \sim \pi_0(\cdot \mid s_1)$
3. 更新 Q 表里这一格 (s0,a0)(s_0,a_0)(s0,a0)：
4. $q_1(s_0,a_0) = q_0(s_0,a_0) + \alpha_0(s_0,a_0)\Big(r_1 + \gamma q_0(s_1,a_1) - q_0(s_0,a_0)\Big)$
5. 更新策略在状态 s_0 上（变成对 $q_1(s_0,\cdot)$ 的 ϵ -贪婪）得到 $\pi_1(\cdot \mid s_0)$
6. 指针前移：$s_0 \leftarrow s_1$，$a_0 \leftarrow a_1$

> 注意：这里的 q_1 是整张表的新版本，只是只改了 (s_0,a_0) 这一格。

第 1 步更新（$t=1$）

同理：

- 你现在“当前状态动作”是 (s_1,a_1)
- 执行动作得到 r_2, s_2
- 选 $a_2 \sim \pi_1(\cdot \mid s_2)$
- 更新 (s_1,a_1)：
- $q_2(s_1,a_1) = q_1(s_1,a_1) + \alpha_1(\cdot)\Big(r_2 + \gamma q_1(s_2,a_2) - q_1(s_1,a_1)\Big)$
- 更新 $\pi_2(\cdot \mid s_1)$
- 前移到 (s_2,a_2)

第 2 步（$t=2$）

更新 (s_2,a_2)，得到 q_3，更新 $\pi_3(\cdot \mid s_2)$，前移。

第 3 步（$t=3$）

更新 (s_3,a_3)，得到 q_4，更新 $\pi_4(\cdot \mid s_3)$，前移。

第 4 步（t=4t=4t=4）终止前最后一次

- 执行 a_4 得到 r_5 和终止状态 s_5
- 终止状态通常不再选 a_5（或定义 $q(\text{terminal},\cdot)=0$）
- 更新：
- $q_5(s_4,a_4) = q_4(s_4,a_4) + \alpha_4(\cdot)\Big(r_5 + \gamma \cdot 0 - q_4(s_4,a_4)\Big)$
- 更新 $\pi_5(\cdot \mid s_4)$
- episode 结束

最后，Sarsa 也有一些变体，例如期望 Sarsa (Expected Sarsa)。感兴趣的读者可以查看方框 7.4。

🍰 方框 7.4：期望 Sarsa (Expected Sarsa)

给定一个策略 π，其动作价值可以通过期望 Sarsa (Expected Sarsa) 来评估，这是 Sarsa 的一个变体。期望 Sarsa 算法为

$$q_{t+1}(s_t,a_t) = q_t(s_t,a_t) - \alpha_t(s_t,a_t)\left[q_t(s_t,a_t) - \left(r_{t+1} + \gamma \mathbb{E}[q_t(s_{t+1},A)]\right)\right],$$

$$q_{t+1}(s,a) = q_t(s,a), \quad \text{对于所有 } (s,a) \neq (s_t,a_t),$$

其中

$$\mathbb{E}[q_t(s_{t+1},A)] = \sum_a \pi_t(a|s_{t+1}) q_t(s_{t+1},a) \doteq v_t(s_{t+1})$$

是策略 π_t 下 $q_t(s_{t+1}, a)$ 的期望值。期望 Sarsa 算法的表达式与 Sarsa 非常相似。它们仅在 TD 目标方面有所不同。具体而言，期望 Sarsa 中的 TD 目标是 $r_{t+1} + \gamma \mathbb{E}[q_t(s_{t+1}, A)]$，而在 Sarsa 中是 $r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1})$。由于该算法涉及期望值，因此被称为期望 Sarsa。尽管计算期望值可能会略微增加计算复杂度，但在减少估计方差方面是有益的，因为它将 Sarsa 中的随机变量从

$$\{s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}\} \text{ 减少到了 } \{s_t, a_t, r_{t+1}, s_{t+1}\}。$$

与 (7.1) 中的 TD 学习算法类似，期望 Sarsa 可以被视为用于求解以下方程的随机逼近算法：

$$q_\pi(s, a) = \mathbb{E}\left[R_{t+1} + \gamma \mathbb{E}[q_\pi(S_{t+1}, A_{t+1}) | S_{t+1}] | S_t = s, A_t = a\right], \quad \text{对于所有 } s, a. \quad (7.15)$$

上述方程乍一看可能很奇怪。事实上，它是贝尔曼方程的另一种表达形式。为了看清这一点，将

$$\mathbb{E}[q_\pi(S_{t+1}, A_{t+1}) | S_{t+1}] = \sum_{A'} q_\pi(S_{t+1}, A') \pi(A' | S_{t+1}) = v_\pi(S_{t+1})$$

代入 (7.15) 可得

$$q_\pi(s, a) = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a],$$

这显然就是贝尔曼方程。

期望 Sarsa 的实现与 Sarsa 类似。更多细节可以在 [3, 36, 37] 中找到。

7.3 动作价值的 TD 学习：n-步 Sarsa (TD learning of action values: n-step Sarsa)

本节介绍 **n-步 Sarsa (n-step Sarsa)**，它是 Sarsa 的一种推广。我们将看到 Sarsa 和 MC 学习是 n-步 Sarsa 的两个极端情况。

回顾动作价值的定义为

$$q_\pi(s, a) = \mathbb{E}[G_t | S_t = s, A_t = a], \quad (7.16)$$

其中 G_t 是满足以下条件的折扣回报

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$

事实上，G_t 也可以分解为不同的形式：

$$\begin{aligned}
\text{Sarsa} \longleftarrow \quad & G_t^{(1)} = R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}), \\
& G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi(S_{t+2}, A_{t+2}), \\
& \vdots \\
n\text{-step Sarsa} \longleftarrow \quad & G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^n q_\pi(S_{t+n}, A_{t+n}), \\
& \vdots \\
\text{MC} \longleftarrow \quad & G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} \dots
\end{aligned}$$

必须注意的是 $G_t = G_t^{(1)} = G_t^{(2)} = G_t^{(n)} = G_t^{(\infty)}$，其中的上标仅表示 G_t 的不同分解结构。

将 $G_t^{(n)}$ 的不同分解形式代入 (7.16) 中的 $q_\pi(s, a)$ 会得到不同的算法。

- 当 $n = 1$ 时，我们有

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

  用于求解该方程的相应随机逼近算法是

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

  这正是 (7.12) 中的 Sarsa 算法。

- 当 $n = \infty$ 时，我们有

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

用于求解该方程的相应算法为：

$$q_{t+1}(s_t, a_t) = g_t \doteq r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots,$$

其中 g_t 是 G_t 的一个样本。事实上，这就是 MC 学习算法，它使用从 (s_t, a_t) 开始的回合的折扣回报来近似 (s_t, a_t) 的动作价值。

- 对于 n 的一般取值，我们有

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

  用于求解上述方程的相应算法为

$$q_{t+1}(s_t,a_t) = q_t(s_t,a_t) - \alpha_t(s_t,a_t)\left[q_t(s_t,a_t) - (r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^n q_t(s_{t+n},a_{t+n}))\right]. \quad (7.17)$$

该算法被称为 n *-步 Sarsa*。

总之，n -步 Sarsa 是一个更通用的算法，因为当 $n = 1$ 时它变为（单步）Sarsa 算法，而当 $n = \infty$（通过设定 $\alpha_t = 1$）时它变为 MC 学习算法。

为了实现 (7.17) 中的 n -步 Sarsa 算法，我们需要经验样本 $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, \ldots, r_{t+n}, s_{t+n}, a_{t+n})$。由于在时刻 t 尚未收集到 $(r_{t+n}, s_{t+n}, a_{t+n})$，我们必须等到时刻 $t+n$ 才能更新 (s_t, a_t) 的 q 值。为此，(7.17) 可以重写为

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

其中 $q_{t+n}(s_t, a_t)$ 是在时刻 $t+n$ 对 $q_\pi(s_t, a_t)$ 的估计值。

> 🌟 **注：n-step Sarsa 不是采样n次，而是推迟更新，当交互达到n步后开始更新**

由于 n -步 Sarsa 包含了 Sarsa 和 MC 学习这两个极端情况，因此 n -步 Sarsa 的性能介于 Sarsa 和 MC 学习之间也就不足为奇了。

特别地，如果 n 选择为一个较大的数，n -步 Sarsa 接近于 MC 学习：**估计值具有相对较高的方差（需要纳入多步的随机性）但偏差较小（总体被平均了）**。如果 n 选择得较小，n -步 Sarsa 接近于 Sarsa：**估计值具有相对较大的偏差但方差较小（需要纳入几步的随机性）**。

最后，**这里介绍的 n -步 Sarsa 算法仅用于策略评估。**它必须与策略改进步骤相结合才能学习最优策略。其实现与 Sarsa 类似，在此省略。感兴趣的读者可以查看 [3, Chapter 7] 以了解多步 TD 学习的详细分析。

7.4 最优动作价值的 TD 学习：Q-learning (TD learning of optimal action values: Q-learning)

在本节中，我们介绍 Q-learning 算法，这是最经典的强化学习算法之一 [38,39]。回顾一下，Sarsa 只能估计 *给定策略* 的动作价值，**并且必须与策略改进步骤相结合才能找到最优策略。**

相比之下，Q-learning 可以直接估计 *最优* 动作价值并找到最优策略。

7.4.1 算法描述 (Algorithm description)

Q-learning 算法如下：

$$q_{t+1}(s_t,a_t) = q_t(s_t,a_t) - \alpha_t(s_t,a_t)\left[q_t(s_t,a_t) - \left(r_{t+1} + \gamma \max_{a \in \mathcal{A}(s_{t+1})} q_t(s_{t+1},a)\right)\right], \quad (7.18)$$

$$q_{t+1}(s,a) = q_t(s,a), \quad \text{对于所有 } (s,a) \neq (s_t,a_t),$$

> 🐵 $q_{t+1}(s_t,a_t) = q_t(s_t,a_t) - \alpha_t(s_t,a_t)\left[q_t(s_t,a_t) - (r_{t+1} + \gamma q_t(s_{t+1},a_{t+1}))\right], \quad (7.12)$ 这是Sarsa

其中 $t = 0, 1, 2, \ldots$。这里，$q_t(s_t, a_t)$ 是时刻 t 对 (s_t, a_t) 的 ***最优动作价值* 的估计值**；$\alpha_t(s_t, a_t)$ 是 (s_t, a_t) 的学习率。

Q-learning 的表达式与 Sarsa 的表达式相似。它们仅在 TD 目标方面有所不同：Q-learning 的 TD 目标是 $r_{t+1} + \gamma \max_a q_t(s_{t+1}, a)$，而 Sarsa 的是 $r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1})$。此外，给定 (s_t, a_t)，Sarsa 在每次迭代中都需要 $(r_{t+1}, s_{t+1}, a_{t+1})$，而 Q-learning 仅需要 (r_{t+1}, s_{t+1})。

为什么 Q-learning 被设计成 (7.18) 中的表达式，它在数学上是在做什么？Q-learning 是一种用于求解以下方程的随机逼近算法：

$$q(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \max_a q(S_{t+1},a) \middle| S_t = s, A_t = a\right]. \quad (7.19)$$

这是以动作价值表示的贝尔曼最优方程。证明在方框 7.5 中给出。Q-learning 的收敛性分析与定理 7.1 类似，在此省略。更多信息可以在 [32, 39] 中找到。

👍 **方框 7.5：证明 (7.19) 是贝尔曼最优方程 (Showing that (7.19) is the Bellman optimality equation)**

根据期望的定义，(7.19) 可以重写为

$$q(s,a) = \sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a) \max_{a \in \mathcal{A}(s')} q(s',a).$$

对方程两边取最大值可得：

$$\max_{a \in \mathcal{A}(s)} q(s,a) = \max_{a \in \mathcal{A}(s)} \left[\sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a) \max_{a \in \mathcal{A}(s')} q(s',a) \right].$$

通过令 $v(s) \doteq \max_{a \in \mathcal{A}(s)} q(s,a)$，我们可以将上述方程重写为

$$\begin{aligned} v(s) &= \max_{a \in \mathcal{A}(s)} \left[\sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a)v(s') \right] \\ &= \max_{\pi} \sum_{a \in \mathcal{A}(s)} \pi(a|s) \left[\sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a)v(s') \right], \end{aligned}$$

这显然就是第 3 章介绍的以状态价值表示的贝尔曼最优方程。

7.4.2 Off-policy vs on-policy)

接下来我们介绍两个重要的概念：*on-policy learning* 和 *off-policy learning*。Q-learning 与其他 TD 算法相比略显特殊之处在于，Q-learning 是***off-policy learning***的，而其他算法是***on-policy learning*** 的。

任何强化学习任务中都存在两个策略：**行为策略 (behavior policy)** 和 **目标策略 (target policy)**。行为策略是用于生成经验样本的策略。目标策略是不断更新以收敛到最优策略的策略。当行为策略与目标策略相同时，这种学习过程称为 **on-policy**。否则，当它们不同时，学习过程称为**off-policy**。

off-policy学习的优势在于，**它可以基于其他策略生成的经验样本来学习最优策略**，例如，这些样本可以是由人类操作员执行的策略生成的。作为一个重要的例子，行为策略可以选择为 *探索性 (exploratory)* 的。例如，如果我们想要估计所有状态-动作对的动作价值，我们必须生成能够足够多次访问每个状态-动作对的回合。虽然 Sarsa 使用 ϵ -贪婪策略来维持一定的探索能力，但 ϵ 的值通常很小，因此探索能力有限。相比之下，如果我们能使用具有强探索能力的策略来生成回合，然后使用**off-policy**学习来学习最优策略，学习效率将显著提高。

要确定一个算法是**on-policy**还是**off-policy**，我们可以考察两个方面。第一是算法旨在解决的数学问题。第二是算法所需的经验样本。

- Sarsa 是**on-policy**的。

原因如下。Sarsa 在每次迭代中有两个步骤。

  - 第一步是通过求解贝尔曼方程来评估策略 π。为此，我们需要由 π 生成的样本。因此，π 是行为策略。
  - 第二步是基于 π 的估计值获得改进的策略。结果是，π 是目标策略，它不断更新并最终收敛到最优策略。因此，行为策略和目标策略是相同的。

从另一个角度来看，我们可以检查算法所需的样本。Sarsa 在每次迭代中所需的样本包括 $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$。这些样本的生成方式如下图所示：

$$s_t \xrightarrow{\pi_b} a_t \xrightarrow{\text{model}} r_{t+1}, s_{t+1} \xrightarrow{\pi_b} a_{t+1}$$

可以看出，行为策略 π_b 是在 s_t 处生成 a_t 并在 s_{t+1} 处生成 a_{t+1} 的策略。

Sarsa 算法旨在估计记为 π_T 的策略的 *动作价值* (s_t, a_t)，π_T 是目标策略，因为它在每次迭代中都会根据估计值进行改进。事实上，π_T 与 π_b 相同，**因为对 π_T 的评估依赖于样本 $(r_{t+1}, s_{t+1}, a_{t+1})$，其中 a_{t+1} 是遵循 π_b 生成的。**换句话说，Sarsa 评估的策略正是用于生成样本的策略。

- **Q-learning 是off-policy的。**
  - **根本原因是 Q-learning 是求解 *贝尔曼最优方程 (Bellman optimality equation)* 的算法**，而 Sarsa 是求解给定策略的 *贝尔曼方程 (Bellman equation)* 的算法。虽然求解贝尔曼方程可以评估相关联的策略，**但求解贝尔曼最优方程可以直接生成最优价值和最优策略。**
  - 特别地，Q-learning 在每次迭代中所需的样本是 $(s_t, a_t, r_{t+1}, s_{t+1})$。这些样本的生成方式如下图所示：

$$s_t \xrightarrow{\pi_b} a_t \xrightarrow{\text{model}} r_{t+1}, s_{t+1}$$

  - 可以看出，行为策略 π_b 是在 s_t 处生成 a_t 的策略。Q-learning 算法旨在估计 (s_t, a_t) 的**最优动作价值**。这个估计过程依赖于样本 (r_{t+1}, s_{t+1})。生成 (r_{t+1}, s_{t+1}) 的过程不涉及 π_b，因为它是由系统模型（或通过与环境交互）控制的。因此，对 (s_t, a_t) 的最优动作价值的估计不涉及 π_b，我们可以使用任何 π_b 在 s_t 处生成 a_t。**此外，这里的目标策略 π_T 是基于估计的最优价值获得的贪婪策略（算法 7.3）**。行为策略不必与 π_T 相同。
- **MC 学习是on-policy 的。** 原因是类似于 Sarsa。要评估和改进的目标策略与生成样本的行为策略是相同的。

另一个容易与**on-policy /off-policy** 混淆的概念是 *在线/离线 (online/offline)*。在线学习指的是智能体在与环境交互的同时更新价值和策略的情况。离线学习指的是智能体使用预先收集的经验数据更新价值和策略，而不与环境交互的情况。**如果一个算法是on-policy的，那么它可以以在线方式实现，但不能使用由其他策略生成的预先收集的数据。如果一个算法是off-policy 的，那么它既可以以在线方式实现，也可以以离线方式实现。**

算法 7.2：通过 Q-learning 进行最优策略学习 (on-policy version) (Optimal policy learning via Q-learning (c))

初始化：对于所有 (s, a) 和所有 t，$\alpha_t(s, a) = \alpha > 0$。$\epsilon \in (0, 1)$。源自 q_0 的初始 ϵ -贪婪策略 π_0。初始 $q_0(s, a)$ 对所有 (s, a)。

目标：学习一个最优策略，该策略可以将智能体从初始状态 s_0 引导至目标状态。

对于每个回合 (episode)，做

如果 $s_t (t = 0, 1, 2, \dots)$ 不是目标状态，做

在给定 s_t 的情况下收集经验样本 (a_t, r_{t+1}, s_{t+1})：

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}。

更新 (s_t, a_t) 的 q 值：

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

更新 s_t 的策略：

$$\pi_{t+1}(a|s_t) = 1 - \frac{\epsilon}{|\mathcal{A}(s_t)|}(|\mathcal{A}(s_t)| - 1) \text{ 如果 } a = \arg\max_a q_{t+1}(s_t, a)$$

$$\pi_{t+1}(a|s_t) = \frac{\epsilon}{|\mathcal{A}(s_t)|} \text{ 其他情况}$$

算法 7.3：通过 Q-learning 进行最优策略学习 (off-policy version) (Optimal policy learning via Q-learning (off-policy version))

初始化：所有 (s, a) 的初始猜测 $q_0(s, a)$。所有 (s, a) 的行为策略 $\pi_b(a|s)$。对于所有 (s, a) 和所有 t，$\alpha_t(s, a) = \alpha > 0$。

目标：从由 π_b 生成的经验样本中学习所有状态的最优目标策略 π_T。

对于每个由 π_b 生成的回合 $\{s_0, a_0, r_1, s_1, a_1, r_2, \dots\}$，做

对于回合中的每一步 $t = 0, 1, 2, \dots$，做

更新 (s_t, a_t) 的 q 值：

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

更新 s_t 的目标策略：

$$\pi_{T,t+1}(a|s_t) = 1 \text{ 如果 } a = \arg\max_a q_{t+1}(s_t, a)$$

$$\pi_{T,t+1}(a|s_t) = 0 \text{ 其他情况}$$

🏖️ **如何更新目标策略的方法** $a = \arg\max_a q_{t+1}(s_t, a)$

离散动作：把所有动作都喂进去算一遍，取最大那个（跟Q表格一样简单）

连续动作：没法遍历所有动作，通常不用纯 Q-learning 的 $\arg\max$，而改用 **actor-critic**（比如 DDPG/TD3/SAC 这一类）：再训练一个策略网络 $\pi_\phi(s)$ 来近似“让 Q 最大的动作”，就不需要显式求 $\arg\max$ 了。

7.4.3 实现 (Implementation)

由于 Q-learning 是off-policy 的，它既可以以on-policy方式实现，也可以以off-policy方式实现。

Q-learning 的同策略版本如算法 7.2 所示。这种实现类似于算法 7.1 中的 Sarsa 实现。在这里，行为策略与目标策略相同，均为 ϵ -贪婪策略。

Q-learning 的off-policy版本如算法 7.3 所示。行为策略 π_b 可以是任何策略，只要它能生成足够的经验样本。当 π_b 具有探索性时通常是有利的。在这里，目标策略 π_T 是贪婪的而不是 ϵ -贪婪的，因为它不用于生成样本，因此不需要具有探索性。此外，这里展示的 Q-learning 的off-policy版本是离线 (offline) 实现的：即首先收集所有经验样本，然后再进行处理。

7.4.4 演示示例 (Illustrative examples)

接下来我们展示演示 Q-learning 的示例。

第一个示例见图 7.3。它演示了On-policy Q-learning。这里的 *目标* 是找到一条从起始状态到目标状态的最优路径。设置细节见图 7.3 的标题。可以看出，Q-learning 最终可以找到一条最优路径。在学习过程中，每个回合的长度减少，而每个回合的总奖励增加。

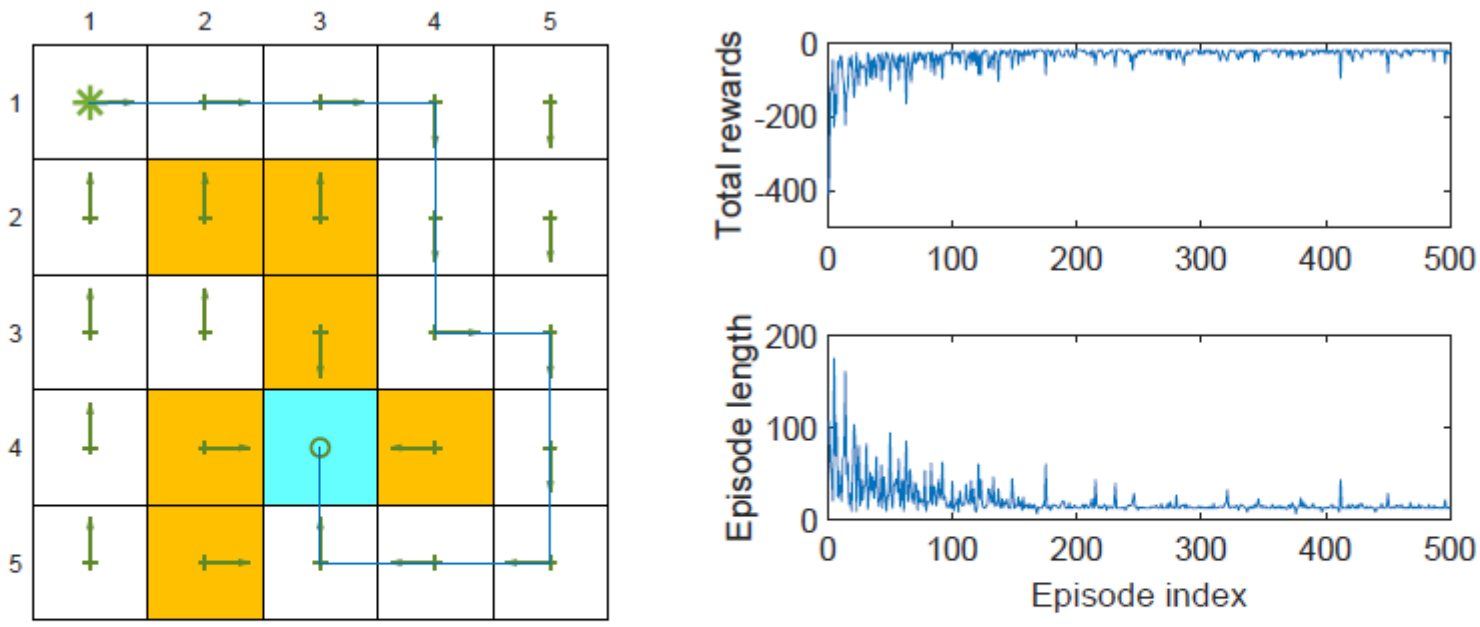


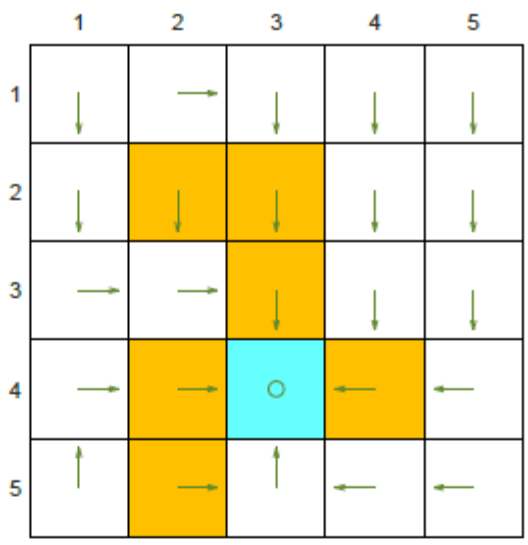
Figure 7.3: An example for demonstrating Q-learning. All the episodes start from the top-left state and terminate after reaching the target state. The aim is to find an optimal path from the starting state to the target state. The reward settings are $r_{\text{target}} = 0$, $r_{\text{forbidden}} = r_{\text{boundary}} = -10$, and $r_{\text{other}} = -1$. The learning rate is $\alpha = 0.1$ and the value of ϵ is 0.1. The left figure shows the final policy obtained by the algorithm. The right figure shows the total reward and length of every episode.

图 7.3：演示 Q-learning 的一个示例。所有回合都从左上角状态开始，并在到达目标状态时终止。目标是找到一条从起始状态到目标状态的最优路径。奖励设置如下：$r_{\text{target}} = 0$，$r_{\text{forbidden}} = r_{\text{boundary}} = -10$，且 $r_{\text{other}} = -1$。学习率 $\alpha = 0.1$，ϵ 的值为 0.1。左图显示了算法获得的最终策略。右图显示了每个回合的总奖励和长度。

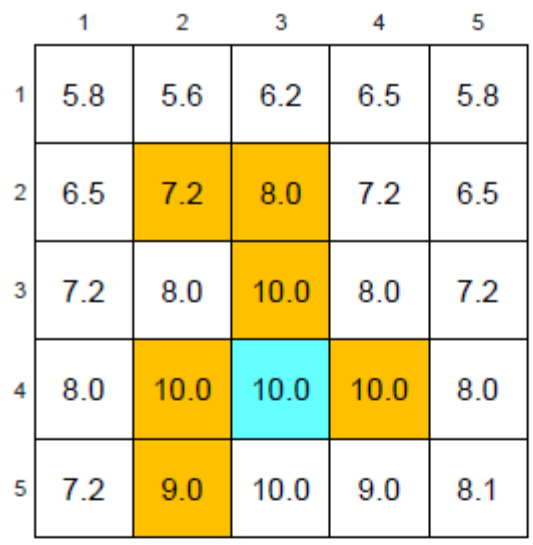
第二组示例见图 7.4 和图 7.5。它们演示了Off-policy Q-learning。这里的 *目标* 是找到所有状态的最优策略。奖励设置为 $r_{\text{boundary}} = r_{\text{forbidden}} = -1$，且 $r_{\text{target}} = 1$。折扣率是 $\gamma = 0.9$。学习率是 $\alpha = 0.1$。

- **真值 (Ground truth)：** 为了验证 Q-learning 的有效性，我们需要首先知道最优策略和最优状态价值的真值。在这里，真值是通过基于模型的策略迭代算法获得的。真值在图 7.4(a) 和 (b) 中给出。

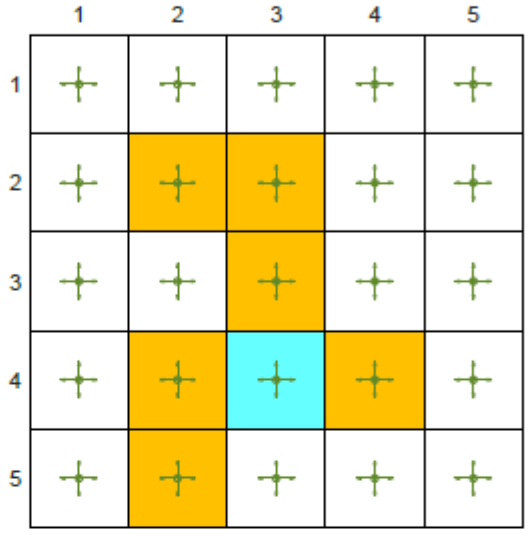
- **经验样本 (Experience samples)：** 行为策略服从均匀分布：在任何状态下采取任何动作的概率均为 0.2（图 7.4(c)）。生成了一个包含 100,000 步的单个回合（图 7.4(d)）。由于行为策略具有良好的探索能力，该回合多次访问了每个状态-动作对。
- **学习结果 (Learned results)：** 基于由行为策略生成的那个回合，Q-learning 学习到的最终目标策略如图 7.4(e) 所示。该策略是最优的，因为估计的状态价值误差（均方根误差）收敛于零，如图 7.4(f) 所示。此外，人们可能会注意到学习到的最优策略与图 7.4(a) 中的不完全相同。事实上，存在多个具有相同最优状态价值的最优策略。
- **不同的初始值 (Different initial values)：** 由于 Q-learning 进行自举 (bootstraps)，算法的性能取决于动作价值的初始猜测。如图 7.4(g) 所示，当初始猜测接近真实值时，估计值在大约 10,000 步内收敛。否则，收敛需要更多的步数（图 7.4(h)）。尽管如此，这些图表明即使初始值不准确，Q-learning 仍然可以快速收敛。
- **不同的行为策略 (Different behavior policies)：** 当行为策略不具有探索性时，学习性能会显著下降。例如，考虑图 7.5 中所示的行为策略。它们是 $\epsilon = 0.5$ 或 0.1 的 ϵ -贪婪策略（图 7.4(c) 中的均匀策略可以看作是 $\epsilon = 1$ 的 ϵ -贪婪策略）。结果表明，当 ϵ 从 1 减小到 0.5 再到 0.1 时，学习速度显著下降。这是因为策略的探索能力变弱，导致经验样本不足。



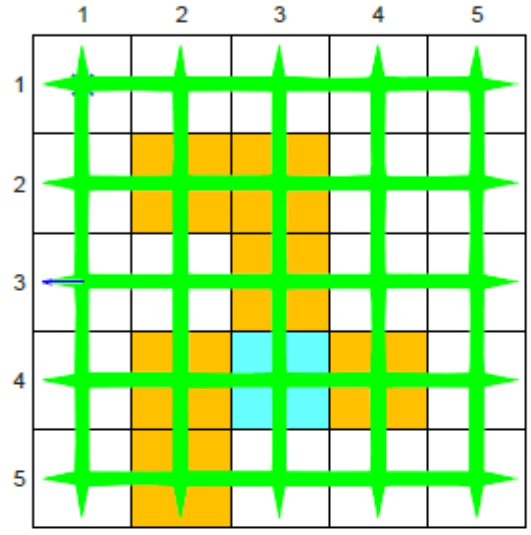
(a) Optimal policy



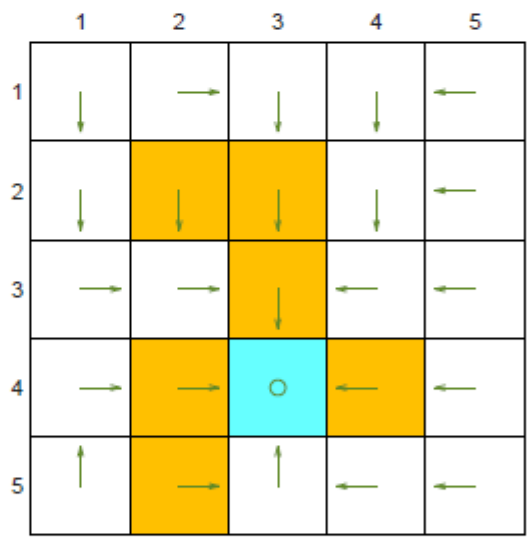
(b) Optimal state value



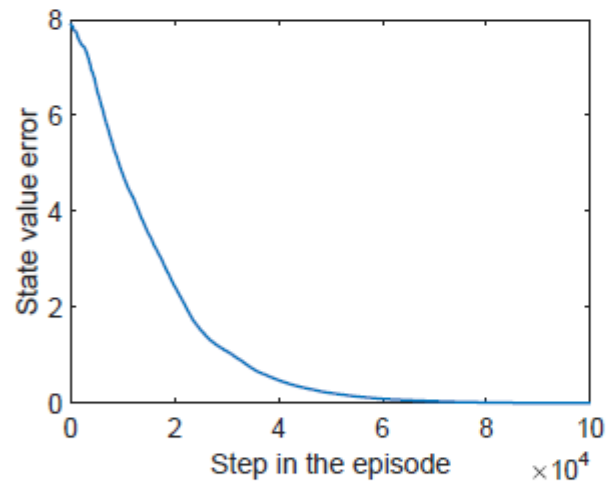
(c) Behavior policy



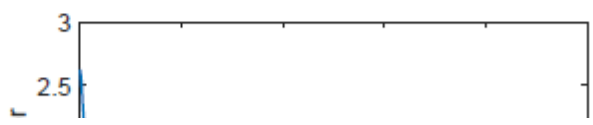
(d) Generated episode

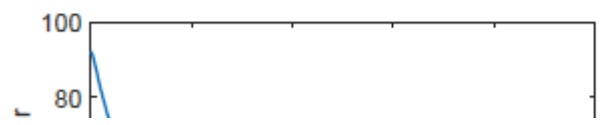


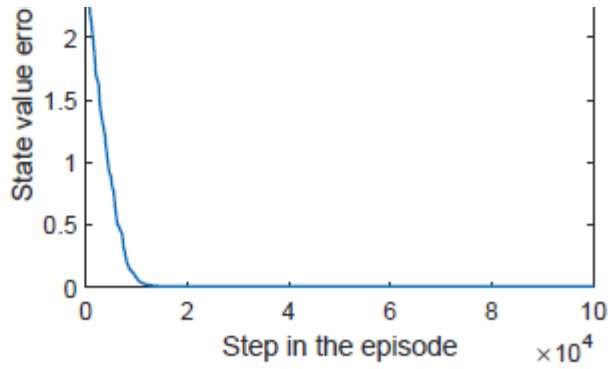
(e) Learned policy



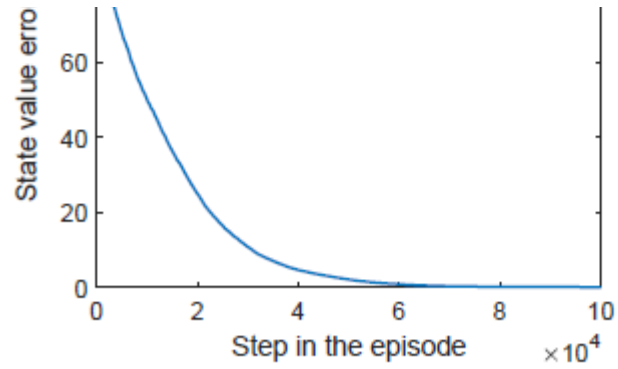
(f) State value error when $q_0(s, a) = 0$





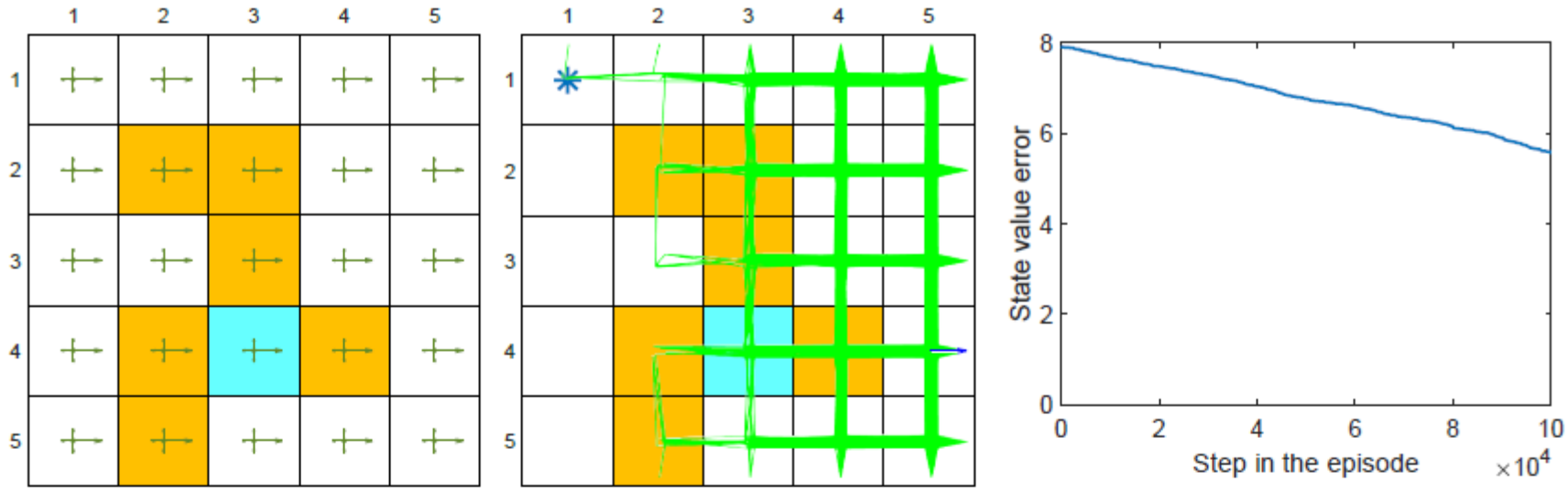


(g) State value error when $q_0(s,a) = 10$

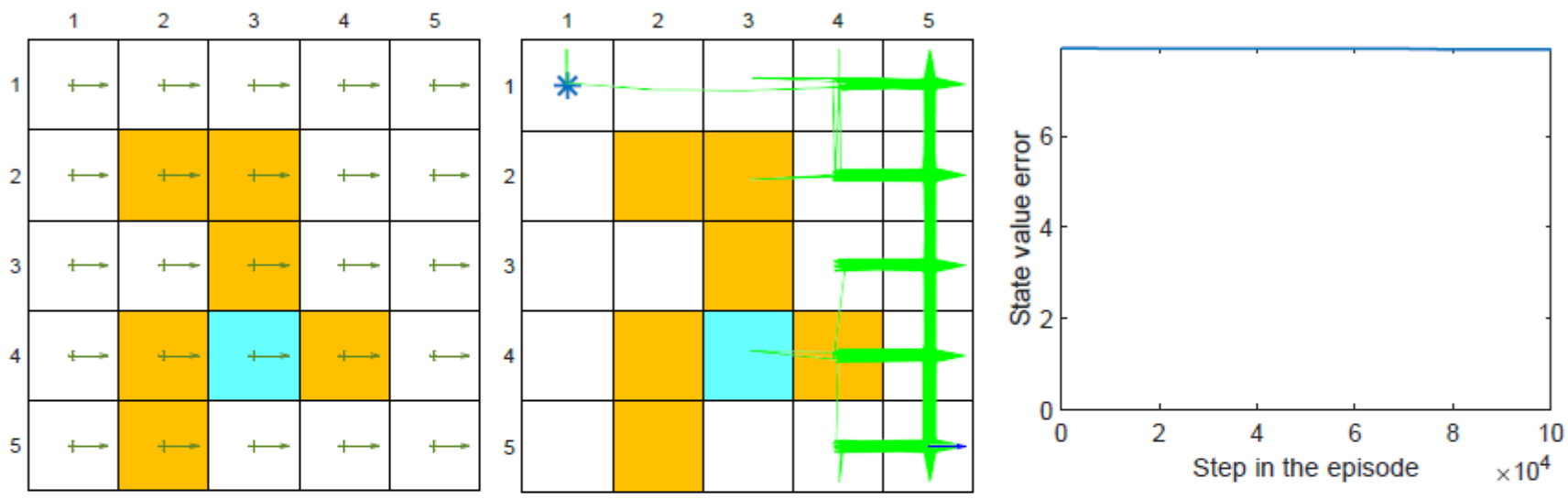


(h) State value error when $q_0(s,a) = 100$

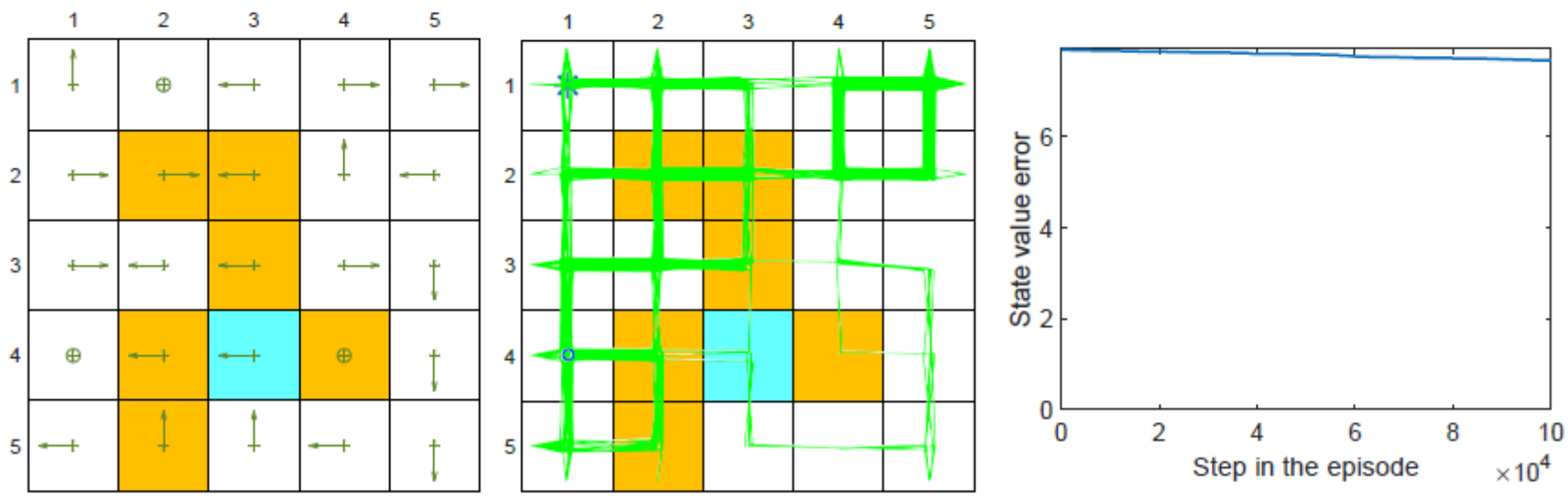
Figure 7.4: Examples for demonstrating off-policy learning via Q-learning. The optimal policy and optimal state values are shown in (a) and (b), respectively. The behavior policy and the generated episode are shown in (c) and (d), respectively. The estimated policy and the estimation error evolution are shown in (e) and (f), respectively. The cases with different initial values are shown in (g) and (h).



(a) $\epsilon = 0.5$



(b) $\epsilon = 0.1$



(c) $\epsilon = 0.1$

Figure 7.5: The performance of Q-learning drops when the behavior policy is not exploratory. The figures in the left column show the behavior policies. The figures in the middle column show the generated episodes following the corresponding behavior policies. The episode in each example has 100,000 steps. The figures in the right column show the evolution of the root-mean-square error of the estimated state values.

7.5 统一视角 (A unified viewpoint)

到目前为止，我们已经介绍了不同的 TD 算法，如 Sarsa、 n -步 Sarsa 和 Q-learning。在本节中，我们引入一个统一的框架来容纳所有这些算法以及 MC 学习。

特别是，TD 算法（用于动作价值估计）可以用一个统一的表达式来表示：

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)[q_t(s_t, a_t) - \bar{q}_t], \quad (7.20)$$

算法 (Algorithm)	TD 目标 $\bar{q}_t$ 在 (7.20) 中的表达式
Sarsa	$\bar{q}_t = r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1})$

n -步 Sarsa (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	$\bar{q}_t = r_{t+1} + \gamma \max_a q_t(s_{t+1}, a)$
蒙特卡洛 (Monte Carlo)	$\bar{q}_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots$

算法 (Algorithm)	**待求解方程 (Equation to be solved)**
Sarsa	BE: $q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \vert S_t = s, A_t = a]$
n -步 Sarsa (n-step Sarsa)	BE: $q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^n q_\pi(S_{t+n}, A_{t+n}) \vert S_t = s, A_t = a]$
Q-learning	BOE: $q(s,a) = \mathbb{E}[R_{t+1} + \gamma \max_a q(S_{t+1}, a) \vert S_t = s, A_t = a]$
蒙特卡洛 (Monte Carlo)	BE: $q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \vert S_t = s, A_t = a]$

表 7.2：TD 算法的统一视角。 这里，BE 和 BOE 分别表示贝尔曼方程和贝尔曼最优方程。

其中 $\bar{q}_t$ 是 *TD 目标 (TD target)*。不同的 TD 算法具有不同的 $\bar{q}_t$ 。总结见表 7.2。MC 学习算法可以被视为 (7.20) 的一个特例：我们可以设定 $\alpha_t(s_t, a_t) = 1$ ，那么 (7.20) 就变成了 $q_{t+1}(s_t, a_t) = \bar{q}_t$ 。

算法 (7.20) 可以被视为用于求解统一方程 $q(s,a) = \mathbb{E}[\bar{q}_t \vert s, a]$ 的随机逼近算法。该方程对于不同的 $\bar{q}_t$ 有不同的表达式。这些表达式总结在表 7.2 中。可以看出，除了 Q-learning 旨在求解贝尔曼最优方程外，所有其他算法都旨在求解贝尔曼方程。

7.6 总结 (Summary)

本章介绍了一类重要的强化学习算法，称为 TD 学习。我们介绍的具体算法包括 Sarsa、n -步 Sarsa 和 Q-learning。所有这些算法都可以被视为用于求解贝尔曼方程或贝尔曼最优方程的随机逼近算法。

本章介绍的 TD 算法，除 Q-learning 外，均用于评估给定策略。即根据一些经验样本来估计给定策略的状态/动作价值。结合策略改进，它们可以用于学习最优策略。此外，这些算法是同策略 (on-policy) 的：目标策略被用作行为策略来生成经验样本。

与其他 TD 算法相比，Q-learning 稍显特殊，因为它是off-policy的。在 Q-learning 中，目标策略可以与行为策略不同。**Q-learning 之所以是off-policy的，其根本原因在于 Q-learning 旨在求解贝尔曼最优方程，而不是给定策略的贝尔曼方程。**

值得一提的是，有一些方法可以将on-policy算法转换为off-policy。重要**性采样 (Importance sampling) 是一种广泛使用的方法 [3, 40]**，将在第 10 章中介绍。最后，本章介绍的 TD 算法还有一些变体和扩展 [41-45]。例如，TD(λ) 方法为 TD 学习提供了一个更通用和统一的框架。更多信息可以在 [3, 20, 46] 中找到。

7.7 问与答 (Q&A)

- 问：TD 学习中的术语“TD”是什么意思?

  答： 每个 TD 算法都有一个 TD 误差，它表示新样本与当前估计值之间的差异。由于这种差异是在不同时间步之间计算的，因此被称为时序差分 (temporal-difference)。

- 问：TD 学习中的术语“学习 (learning)”是什么意思?

  答： 从数学角度来看，**“学习”仅仅意味着“估计 (estimation)”**。即根据**一些样本估计状态/动作价值，然后基于估计的价值获得策略。**

- 问：既然 Sarsa 只能估计给定策略的动作价值，它是如何用于学习最优策略的?

  答： 为了获得最优策略，价值估计过程应该与策略改进过程相互作用。也就是说，在价值更新后，相应的策略也应该更新。然后，更新后的策略生成新的样本，这些样本可再次用于估计价值。这就是广义策略迭代 (generalized policy iteration) 的思想。

- 问：为什么 Sarsa 将策略更新为 ϵ -贪婪策略?

  答： 这是因为该策略也被用于生成用于价值估计的样本。因此，它应该具有探索性以生成足够的经验样本。

- 问：虽然定理 7.1 和 7.2 要求学习率 α_t 逐渐收敛于零，但为什么在实践中它通常被设置为一个小的常数?

答： 根本原因是待评估的策略在不断变化（或称为非平稳的）。特别是，像 Sarsa 这样的 TD 学习算法旨在估计给定策略的动作价值。如果策略是固定的，使用衰减的学习率是可以接受的。然而，在最优策略学习过程中，Sarsa 旨在评估的策略在每次迭代后都在变化。在这种情况下，我们需要一个恒定的学习率；否则，衰减的学习率可能太小而无法有效地评估策略。虽然恒定学习率的一个缺点是价值估计最终可能会波动，但只要恒定学习率足够小，这种波动是可以忽略不计的。

- 问：我们应该学习所有状态的最优策略，还是只学习一部分状态的最优策略？

  答： 这取决于任务。人们可能会注意到本章中考虑的一些任务（例如，图 7.2）并不要求找到所有状态的最优策略。相反，它们只需要找到一条从给定起始状态到目标状态的最优路径。此类任务对数据的要求不高，因为智能体不需要足够多次地访问每个状态-动作对。然而，必须注意的是，获得的路径并不能保证是最优的。这是因为如果没有充分探索所有的状态-动作对，可能会错过更好的路径。尽管如此，只要有足够的数据，我们仍然可以找到一条好的或局部最优的路径。

- 问：为什么 Q-learning 是off-policy 的，而本章中的所有其他 TD 算法都是 on-policy 的？

  答： 根本原因是 Q-learning 旨在求解贝尔曼最优方程，而其他 TD 算法旨在求解给定策略的贝尔曼方程。详情请见 7.4.2 节。

- 问：为什么 Q-learning 的异策略版本将策略更新为贪婪策略而不是 ϵ -贪婪策略？

  答： 这是因为目标策略不需要用于生成经验样本。因此，它不需要具有探索性。