

第七章 时序差分方法

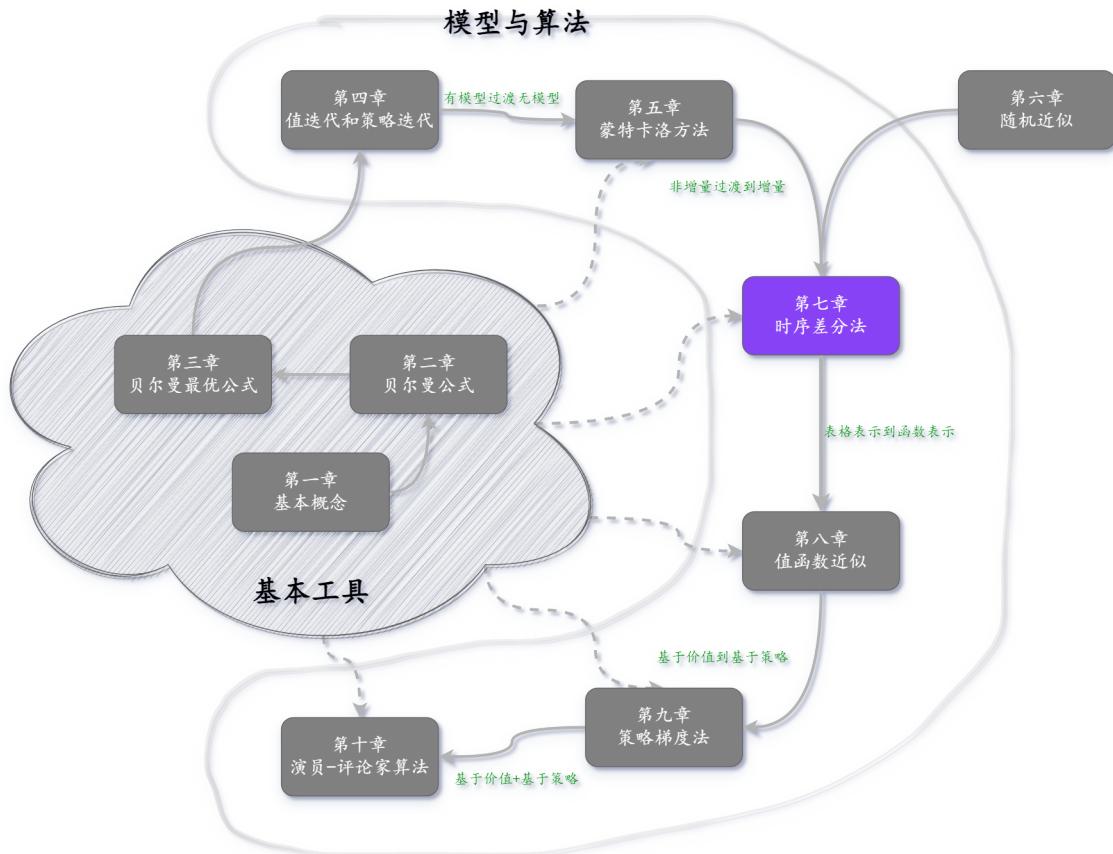


图 7.1

本章介绍强化学习的**时序差分**（Temporal - Difference，简称 TD）方法。与蒙特卡罗（MC）学习类似，TD 方法也是无模型的，但由于其增量形式而具有一些优势。有了第 6 章的准备，读者在看到 TD 学习算法时就不会感到惊慌了。事实上，TD 学习算法可以被视为求解贝尔曼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理清它们之间的关系。

- ✓ 7.1 节介绍了最基本的 TD 算法，它可以估计给定策略的状态值。在学习其他 TD 算法之前，首先了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值。该算法可以与策略提升步骤相结合来找到最佳策略。通过将状态值估计替换为动作值估计，可以从 7.1 节中的 TD 算法轻松推演出 Sarsa 算法。
- ✓ 7.3 节介绍了 n 步 Sarsa 算法，它是 Sarsa 算法的推广。将证明 Sarsa 和 MC 学习是 n 步 Sarsa 的两个特殊情况。
- ✓ 7.4 节介绍了 Q - learning 算法，它是最经典的强化学习算法之一。其他 TD 算法旨在求解给定策略的贝尔曼方程，而 Q 学习旨在直接求解贝尔曼最优方程以获得最优策略。
- ✓ 7.5 节对本章介绍的 TD 算法进行了比较，并提供了统一的观点。

7.1 状态值的 TD 学习

TD 学习通常指的是一大类强化学习算法。例如，本章介绍的所有算法都属于 TD 学习的范围。然而，本节中的 TD 学习特指估计状态值的经典算法。

7.1.1 算法说明

给定策略 π ，我们的目标是估计所有 $s \in \mathcal{S}$ 的 $v_\pi(s)$ 。假设我们根据 π 生成一些经验样本 $(s_0, r_1, s_1, \dots, s_t, r_{t+1}, s_{t+1}, \dots)$ 。这里， t 表示时间步长。以下 TD 算法可以使用这些样本来估计状态值：

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

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

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

需要注意的是，在时间 t ，仅更新访问状态 s_t 的值。未访问状态 $s \neq s_t$ 的值保持不变，如 (7.2) 所示。为了简单起见，方程 (7.2) 经常被省略，但应该记住它，因为如果没有这个方程，算法在数学上将是不完整的。

第一次看到 TD 学习算法的读者可能会奇怪为什么要这样设计。事实上，它可以被看作是求解贝尔曼方程的一种特殊的随机逼近算法。要看到这一点，首先回想一下状态值的定义是

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

我们可以将 (7.3) 重写为

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

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个表达式。有时称为贝尔曼期望方程。

TD 算法可以通过应用 Robbins-Monro 算法（第 6 章）求解 (7.4) 中的贝尔曼方程来导出。有兴趣的读者可以查看框 7.1 中的详细信息。

□ 7.1 TD 算法的推导

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

对于状态 s_t ，我们定义一个函数为

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

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 $v_\pi(s_t)$ 。由于我们可以得到 r_{t+1} 和 s_{t+1} , 即 R_{t+1} 和 S_{t+1} 的样本, 因此我们可以得到的 $g(v_\pi(s_t))$ 的噪声观测值为

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

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

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

其中 $v_t(s_t)$ 是 $v_\pi(s_t)$ 在时间 t 的估计, $\alpha_t(s_t)$ 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。唯一的区别是, (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) 的设计目的只是通过假设其他状态的动作值已知来估计 s_t 的动作值。如果我们想估计所有状态的状态值, 那么右侧的 $v_\pi(s_{t+1})$ 应该替换为 $v_t(s_{t+1})$ 。那么, (7.5) 与 (7.1) 完全相同。然而, 这样的替换还能保证收敛吗? 答案是肯定的, 这将在后面的定理 7.1 中得到证明。

7.1.2 性质分析

TD 算法的一些重要属性讨论如下。首先, 我们更仔细地研究 TD 算法的表达式。特别地, (7.1) 可以描述为

$$\underbrace{v_{t+1}(s_t)}_{\text{new estimate}} = \underbrace{v_t(s_t)}_{\text{current estimate}} - \alpha_t(s_t) \underbrace{\left[v_t(s_t) + \underbrace{(r_{t+1} + \gamma v_t(s_{t+1}))}_{\text{TD target } \bar{v}_t} \right]}_{\text{TD errors } \delta_t},\tag{7.6}$$

其中

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

称为 **TD 目标 (TD Target)** , 且

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

称为 **TD 误差 (TD Error)** , 可以看出, 新的估计 $v_{t+1}(s_t)$ 是当前估计 $v_t(s_t)$ 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 是算法尝试将 $v(s_t)$ 逼近到的**目标值 (Target Value)**。为了看出这一点, 从 (7.6) 两边减去 \bar{v}_t 可以得到

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

上式两边取绝对值得

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

由于 $\alpha_t(s_t)$ 是一个很小的正数，因此我们有 $0 < 1 - \alpha_t(s_t) < 1$ 。然后得出

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

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，该算法在数学上将 $v_t(s_t)$ 推向 \bar{v}_t 。这就是为什么 \bar{v}_t 被称为 TD 目标。

- TD 误差的解释是什么？

首先，这个误差称为**时序差分 (Temporal-Difference)**，因为 $\delta_t = v_t(s_t) - (r_{t+1} + \gamma v_t(s_{t+1}))$ 反映了两个时间步 t 和 $t+1$ 之间的差异。其次，当状态值估计准确时，TD 误差在期望意义上为零。为了看到，当 $v_t = v_\pi$ 时，TD 误差的期望值是

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

因此，TD 误差不仅反映了两个时间步长之间的差异，更重要的是反映了估计 v_t 和真实状态值 v_π 之间的差异。

在更抽象的层面上，TD 误差可以被解释为创新，它表明了从经验样本 (s_t, r_{t+1}, s_{t+1}) 中获得的新信息。TD 学习的基本思想是基于新获得的信息来修正我们当前对状态值的估计。在许多估计问题中，如卡尔曼滤波^{1 2}，创新是根本。

其次，(7.1) 中的 TD 算法只能估计给定策略的状态值。要找到最优策略，还需要进一步计算动作值，然后进行策略改进。这将在第 7.2 节中介绍。尽管如此，本节中介绍的 TD 算法对于理解本章中的其他算法是非常基本和重要的。

第三，虽然 TD 学习和 MC 学习都是无模型的，但它们的优势和劣势是什么？答案汇总在表 7.1 中。

TD Learning	MC Learning
在线 (Online) ：TD 学习是在线学习。它可以在接收到经验样本后立即更新状态/动作值	离线 (Offline) ：MC 学习就是离线学习。它必须等到一回合完全收集完毕。这是因为它必须计算这一回合的折扣回报。
持续任务 (Continuing Tasks) ：由于 TD 学习是在线的，它既可以处理间歇性任务，也可以处理持续任务。持续执行的任务可能没有终止状态。	间歇性任务 (Episodic Tasks) ：由于 MC 学习是离线的，它只能处理在有限步骤后终止的间歇性任务。
自举 (Bootstrapping) ：TD 学习是自举的，因为状态/动作值的更新依赖于该值的先前估计。因此，TD 学习需要对这些值进行初步猜测。	非自举 (Non-bootstrapping) ：MC 不是自举的，因为它可以直接估计状态/动作值，而不需要初始猜测。

TD Learning	MC Learning
低估计方差 (Low Estimation Variance) : TD 的估计方差比 MC 低, 因为它涉及的随机变量更少。例如, 为了估计动作值 $q_\pi(s_t, a_t)$, Sarsa 只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 的估计方差较大, 因为涉及的随机变量较多。例如, 要估计 $q_\pi(s_t, a_t)$, 我们需要 $R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$ 的样本, 假设每回合的长度为 L , 假设每个状态的动作数是 $ \mathcal{A} $ 。然后, $ \mathcal{A} ^L$ 可能会出现一些遵循软策略的插曲。如果我们只用几个回合来估计, 估计的方差很高也就不足为奇了。

表 7.1 TD 学习与 MC 学习的比较

7.1.3 收敛分析

下面 (7.1) 中给出 TD 算法的收敛性分析。

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 $\sum_t \alpha_t(s) = \infty$ 并且 $\sum_t \alpha_t^2(s) < \infty$, 对于所有的 $s \in \mathcal{S}$, 则对于所有 $s \in \mathcal{S}$, 当 $t \rightarrow \infty$ 时, $v_t(s)$ 几乎肯定收敛于 $v_\pi(s)$ 。

下面给出一些关于 α_t 的评论。首先, $\sum_t \alpha_t(s) = \infty$, $\sum_t \alpha_t^2(s) < \infty$ 的条件必须对所有 $s \in \mathcal{S}$ 都有效。注意, 在时间 t , 如果正在访问 s , 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。这需要探索开始的条件或探索策略, 以便每个状态-动作对都可能被访问多次。第二, 在实践中, 学习率 α_t 经常被选为一个很小的正常数。在这种情况下, $\sum_t \alpha_t^2(s) < \infty$ 的条件不再有效。当 α 为常数时, 仍然可以证明算法在期望意义下收敛³。

□ 7.2 定理 7.1 的证明:

在第六章中, 我们基于定理 6.3 证明了收敛。要做到这一点, 我们首先需要构造一个像定理 6.3 中那样的随机过程。考虑任意状态的 $s \in \mathcal{S}$ 。在时间 t , 它从 (7.1) 中的 TD 算法推导出

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

或者

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

估计误差定义为

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

其中 $v_\pi(s)$ 是 s 在策略 π 下的状态价值。从 (7.7) 的两边减去 $v_\pi(s)$ 得出

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

从 (7.8) 的两边减去 $v_\pi(s)$ 得出

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

除 $\alpha_t(s) = 0$ 和 $\eta_t(s) = 0$ 外，其表达式与 (7.9) 相同。因此，不管是否 $s = s_t$ ，我们得到如下统一表达式：

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

这就是定理 6.3 中的过程。我们的目标是证明定理 6.3 中的三个条件都满足，因此过程收敛。

第一个条件如定理 7.1 中所假定的那样有效。接下来，我们证明第二个条件是有效的。也就是说，对所有的 $s \in \mathcal{S}$ ，都存在

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

这里， \mathcal{H}_t 表示历史信息（见定理 6.3 中定义）。由于马尔可夫性质，

$$\eta_t(s) = r_{t+1} + \gamma v_t(s_{t+1}) - v_\pi(s)$$

或 $\eta_t(s) = 0$ ，一旦给出 s ，就不依赖于历史信息。结果是 $\mathbb{E}[\eta_t(s)|\mathcal{H}_t] = \mathbb{E}[\eta_t(s)]$ 。对于 $s \neq s_t$ ，我们有 $\eta_t(s) = 0$ 。然后，很容易看到

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

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 $v_\pi(s_t) = \mathbb{E}[r_{t+1} + \gamma v_\pi(s_{t+1})|s_t]$ ，上面的等式意味着

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

由此得出，

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

因此，在时间 t ，我们从 (7.10) 和 (7.11) 得知：对所有的 $s \in \mathcal{S}$ ，无论是否 $s = s_t$ ，都有 $|\mathbb{E}[\eta_t(s)]| \leq \gamma \|\Delta_t(s)\|_\infty$ 。因此，

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

这是定理 7.1 中的第二个条件。最后，关于第三个条件，对于 $s = s_t$ ，我们有

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

对于 $s \neq s_t$ ，我们有 $\text{var}[\eta_t(s)|\mathcal{H}_t] = 0$ 。由于 r_{t+1} 是有界的，所以第三个条件可以很容易地证明。

7.2 动作值的 TD 学习：SARSA

7.1 节中介绍的 TD 算法只能估计状态值（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值。估计动作值很重要，因为它可以与策略改进步骤相结合，以了解最优策略。

7.2.1 算法说明

在给定策略 π 的情况下，我们的目标是估计动作值。假设我们有一些根据 π 生成的经验样本：

$$(s_0, a_0, r_1, s_1, a_1, \dots, s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, \dots).$$

我们可以使用 SARSA 算法来估计动作值：

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

其中 $t = 0, 1, 2, \dots$ ，且 $\alpha_t(s_t, a_t)$ 是学习率。这里， $q_t(s_t, a_t)$ 是 $q_\pi(s_t, a_t)$ 的估计。在时间 t ，只有 (s_t, a_t) 的 q 值被更新，而其他的 q 值保持不变。

下面讨论了 SARSA 算法的一些重要性质。

- 为什么这种算法被称为 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 state-action-reward-state-action 的缩写。
- 为什么 SARSA 是这样设计的？人们可能已经注意到，SARSA 类似于 (7.1) 中的 TD 算法。实际上，用动作值估计代替状态值估计，可以很容易地从 TD 算法中得到 SARSA 算法。
- SARSA 在数学上是做什么的？与 (7.1) 中的 TD 算法类似，SARSA 是求解给定策略的贝尔曼方程的随机近似算法：

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

方程 (7.13) 是用动作值表示的贝尔曼方程。框 7.3 中给出了证明。

□ 7.3 (7.13) 是贝尔曼方程

如第 2.8.2 节所述，用动作值表示的贝尔曼方程为

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

这个等式建立了动作值之间的关系。由于

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

(7.14) 可重写为

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

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

- SARSA 是否收敛？由于 SARSA 是 (7.1) 中 TD 算法的动作值版本，因此其收敛结果类似于定理 7.1，如下所示。

定理 7.2 SARSA 的收敛

给定策略 π ，通过 (7.12) 中的 SARSA 算法，如果 $\sum_t \alpha_t(s, a) = \infty$ 并且对于所有的 (s, a) ， $\sum_t \alpha_t^2(s, a) < \infty$ ，当对所有的 (s, a) ， $t \rightarrow \infty$ ，则 $q_t(s, a)$ 几乎必然收敛到动作值 $q_{\pi}(s, a)$ 。

□ 证明：

这个证明类似于定理 7.1 的证明，这里省略。 $\sum_t \alpha_t(s, a) = \infty$ 和 $\sum_t \alpha_t^2(s, a) < \infty$ 的条件应该对所有 (s, a) 有效。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。在时间 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 实现最优策略学习

(7.12) 中的 SARSA 算法只能估计给定策略的动作值。要找到最优策略，我们可以将其与策略改进步骤结合起来。这种组合通常也称为 SARSA，其实现过程在算法 7.1 中给出。

♥ 算法 7.1：基于 SARSA 的最优策略学习

初始化： 对于所有 (s, a) 和所有的 $t, \epsilon \in (0, 1)$ ， $\alpha_t(s, a) = \alpha > 0$ 。初始化 $q_0(s, a)$ ，适用于所有 (s, a) 。初始从 q_0 派生的 ϵ -贪婪策略 π_0 。

目标： 学习能够将智能体从初始状态 s_0 引导到目标状态的最优策略。

For 每个回合，**do**

根据 $\pi_0(s_0)$ 在 s_0 生成 a_0

If s_t ($t = 0, 1, 2, \dots$) 不是目标状态, **do**

收集给定 (s_t, a_t) 的经验样本 $(r_{t+1}, s_{t+1}, a_{t+1})$: 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:

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

更新 s_t 的策略:

$$\pi_{t+1}(a|s_t) = 1 - \frac{\epsilon}{|\mathcal{A}(s_t)|}(|\mathcal{A}| - 1), \text{ If } a = \operatorname{argmax}_a q_{t+1}(s_t, a) \text{ Else } \pi_{t+1}(a|s_t) = \frac{\epsilon}{|\mathcal{A}(s_t)|}$$

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

如算法 7.1 所示, 每次迭代有两个步骤。第一步是更新访问状态动作对的 q 值。第二步是将策略更新为“ ϵ -贪婪”策略。 q 值更新步骤仅更新在时间 t 访问的单个状态动作对。之后, 立即更新 s_t 的策略。因此, 在更新策略之前, 我们不能很好地评估给定的策略。这是基于广义策略迭代的思想。此外, 在策略更新后, 该策略将立即用于生成下一个体验样本。这里的策略是 ϵ -贪婪的, 所以它是探索性的。

图 7.2 中显示了一个模拟示例来演示 SARSA 算法。与我们在本书中看到的所有任务不同, 这里的任务旨在找到从特定起始状态到目标状态的最佳路径。它并不旨在为所有状态找到最佳策略。这项任务在实践中经常遇到, 其中起始状态 (如家庭) 和目标状态 (如工作场所) 是固定的, 我们只需要找到连接它们的最佳路径。这个任务相对简单, 因为我们只需要探索接近路径的状态, 而不需要探索所有状态。然而, 如果我们不探索所有状态, 最终路径可能是局部最优的, 而不是全局最优的。

Tip

图 7.2 是一个演示 SARSA 的例子。所有回合都从左上角开始, 并在达到目标状态 (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。奖励设置为: $r_{\text{target}} = 0$, $r_{\text{forbidden}} = r_{\text{boundary}} = -10$, $r_{\text{other}} = -1$ 。学习率为 $\alpha = 0.1$, ϵ 的值为 0.1。图 7.2 左图显示了该算法获得的最终策略。右图显示了每一回合的总回报和长度。

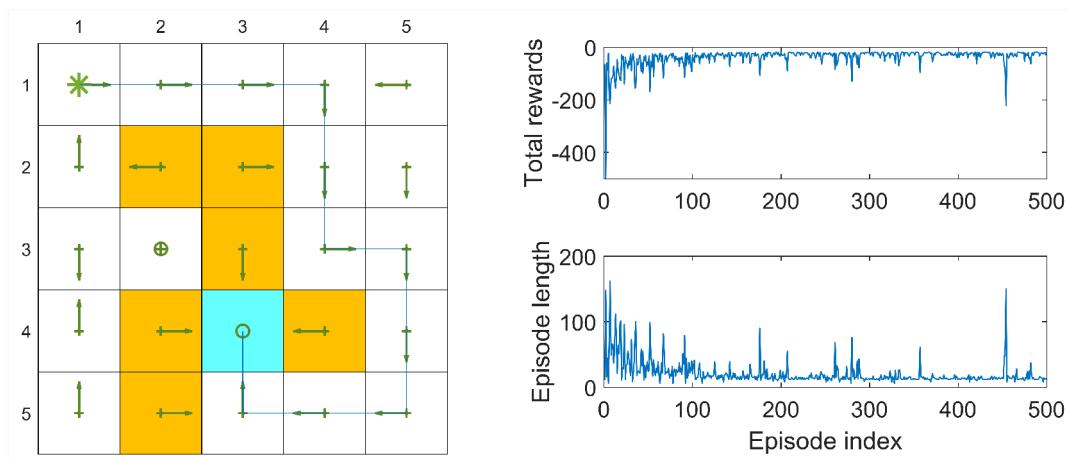


图 7.2

下面讨论模拟设置和模拟结果:

- **模拟设置 (Simulation Setup)**：在本例中，所有回合从左上角状态开始，在目标状态结束。奖励设置为： $r_{\text{target}} = 0$, $r_{\text{forbidden}} = -10$, $r_{\text{boundary}} = -10$, $r_{\text{other}} = -1$ 。此外，对于所有 t , $\alpha_t(s, a) = 0.1$, $\epsilon = 0.1$ 。对动作值的初始猜测是对所有的 (s, a) , 设置 $q_0(s, a) = 0$ 。初始策略服从均匀分布：对所有的 s, a , $\pi_0(a|s) = 0.2$ 。
- **学习策略 (Learned Policy)**：图 7.2 中的左图显示了 SARSA 学习的最终策略。由此可见，这一策略可以从起始状态成功引导到目标状态。然而，其他一些状态的策略可能并不是最优的。这是因为其他状态没有得到很好的探索。
- **每回合的总奖励 (Total Reward of Each Episode)**：图 7.2 中右上角的子图显示了每回合的总奖励。在这里，总奖励是所有即时奖励的非折扣总和。由此可见，每一回合的总奖励是逐步递增的。这是因为最初的策略不好，因此经常获得负回报。随着策略变得更好，总奖励也会增加。
- **每回合的长度 (Length of Each Episode)**：图 7.2 中右下角的子图显示每回合的长度逐渐下降。这是因为最初的策略并不好，在达到目标之前可能会走很多弯路。随着策略变得更好，轨迹的长度也会变短。值得注意的是，一回合的长度可能会突然增加（例如，第 460 回合），相应的总奖励也会急剧下降。这是因为这个策略是贪婪的，而且它有可能采取非最优行动。解决这一问题的一种方法是使用衰减 ϵ ，其值逐渐收敛到零。

最后，SARSA 也有一些变体，如**期望 (Expected) SARSA**。感兴趣的读者可参考框 7.4。

7.4 Expected SARSA

在给定策略 π 的情况下，其动作值可以由期望 SARSA 来评估，期望 SARSA 是 SARSA 的变体。期望 SARSA 算法是

$$\begin{aligned} q_{t+1}(s_t, a_t) &= q_t(s_t, a_t) - \alpha_t(s_t, a_t) \left[q_t(s_t, a_t) - (r_{t+1} + \gamma \mathbb{E}[q_t(s_{t+1}, A)]) \right], \\ q_{t+1}(s, a) &= q_t(s, a), \quad \text{for all } (s, a) \neq (s_t, a_t), \end{aligned}$$

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 算法的表达式非常相似。他们只是在 TD 目标方面有所不同。特别是，在期望 SARSA 中，TD 目标是 $r_{t+1} + \gamma \mathbb{E}[q_t(s_{t+1}, A)]$ ，而 SARSA 的目标是 $r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1})$ 。由于该算法涉及一个期望值，因此被称为期望 SARSA。虽然计算期望值可能会略微增加计算复杂性，但它在减少估计方差的意义上是有益的，因为它将 SARSA 中的随机变量从 $\{s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}\}$ 减少到 $\{s_t, a_t, r_{t+1}, s_{t+1}\}$ 。

与 (7.1) 中的 TD 学习算法类似，期望 SARSA 可以看作是求解以下方程的随机逼近算法：

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

乍一看，上面的等式可能看起来很奇怪。事实上，它是贝尔曼方程的另一种表达形式。为此，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), 得到

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

这显然是贝尔曼方程。

期望 SARSA 的实现类似于 SARSA。更多细节参见文献^{4 5 6}。

7.3 动作值的 TD 学习: N-Step SARSA

本节介绍 N-Step SARSA, 它是 SARSA 的扩展。我们将看到 SARSA 和 MC 学习是 N-Step SARSA 的两个极端情况。

回想一下, 动作值的定义为

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

其中 G_t 是折扣回报, 满足

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

事实上, G_t 也可以分解成不同的形式:

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

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

将 $G_t^{(n)}$ 的不同分解代入 (7.16) 中的 $q_\pi(s, a)$ 会产生不同的算法。

- 当 $n = 1$ 时, 我们有

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

求解该方程的相应随机近似算法为

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

即 (7.12) 中的 SARSA 算法。

- 当 $n = \infty$ 时, 我们有

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

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 的样本。事实上，这就是 MC 学习算法，它利用从 (s_t, a_t) 开始的一个回合的折扣回报来近似 (s_t, a_t) 的动作值。

- 对于 n 是一般值，我们有

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

求解上式对应的算法为

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

该算法称为 N-Step SARSA。

综上所述，N-Step SARSA 是一种更通用的算法，因为当 $n = 1$ 时它成为（一步）SARSA 算法，当 $n = \infty$ （通过设置 $\alpha_t = 1$ ）时，它成为 MC 学习算法。

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

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

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

由于 N-Step SARSA 将 SARSA 和 MC 学习作为两个极端情况，因此 N-Step SARSA 的性能介于 SARSA 和 MC 学习之间也就不足为奇了。特别是，如果 n 选择一个大数，则 N-Step SARSA 接近 MC 学习：估计具有相对较高的方差但偏差较小。如果 n 选得较小，则 N-Step SARSA 接近 SARSA：估计值有相对较大的偏差，但方差较小。最后，这里提出的 N-Step 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。实现与 SARSA 类似，此处不再赘述。

7.4 最优动作值的 TD 学习：Q-learning

在本节中，我们介绍 Q-learning 算法，它是最经典的强化学习算法之一^{7 8}。回想一下，SARSA 只能估计给定策略的动作值，并且必须与策略改进步骤相结合才能找到最佳策略。相比之下，Q-learning 可以直接估计最优动作值并找到最优策略。

7.4.1 算法说明

Q-learning 算法为

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

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

其中 $t = 0, 1, 2, \dots$ 。这里， $q_t(s_t, a_t)$ 是 (s_t, a_t) 的最优动作值的估计， $\alpha_t(s_t, a_t)$ 是 (s_t, a_t) 的学习率。

Q-learning 的表达方式与 SARSA 类似。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 TD 目标是 $r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1})$ 。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，它在数学上有什么作用？Q-learning 是一种随机近似算法，用于求解以下方程：

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

这是用动作值表示的贝尔曼最优方程。框 7.5 给出了证明。Q-learning 的收敛性分析与定理 7.1 类似，此处不再赘述。更多信息可以在相关文献^{9 10} 中找到。

□ 7.5 证明 (7.19) 是贝尔曼最优方程

根据期望的定义，(7.19) 可以重写为

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

取方程两边的最大值得出

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

令 $v(s) \doteq \max_{a \in \mathcal{A}(s)} q(s, a)$ ，我们可以将上式改写为

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

这显然是第 3 章中介绍的用状态值表示的贝尔曼最优方程。

7.4.2 离线策略 VS 在线策略

接下来我们介绍两个重要的概念：**在线策略学习（on-policy learning）** 和**离线策略学习（off-policy learning）**。与其他 TD 算法相比，Q-learning 的特殊之处在于 Q-learning 是离线策略，而其他算法是在线策略。

任何强化学习任务中都存在两种策略：**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，这样的学习过程称为在线策略。否则，当它们不同时，学习过程称为离线策略。

离线策略学习的优点是它可以根据其他策略生成的经验样本来学习最优策略，例如，这些策略可能是由操作员执行的策略。作为一个重要案例，行为政策可以选择具有探索性的。例如，如果我们想估计所有状态-动作对的动作值，我们必须生成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，因此探索能力是有限的。相比之下，如果我们能够使用具有较强探索能力的策略来生成回合，然后使用离线策略学习来学习最优策略，那么学习效率将会显著提高。

为了确定算法是在线策略还是离线策略，我们可以分析两个方面。首先是算法要解决的数学问题。第二个是算法所需的经验样本。

- **SARSA 是在线策略**

理由如下：SARSA 在每次迭代中都有两个步骤。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 就是行为策略。第二步是根据 π 的估计值获得改进的策略。因此， π 是不断更新的目标策略，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，我们可以考虑算法所需的数据样本。SARSA 每次迭代所需的数据样本包括 $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ 。这些样本是如何生成的，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 和在 s_{t+1} 处生成 a_{t+1} 的策略。SARSA 算法旨在估计表示为 π_T 的策略的 (s_t, a_t) 的动作值，该策略是目标策略，因为它在每次迭代中都会基于估计值进行改进。事实上， π_T 与 π_b 相同，因为 π_T 的评估依赖于样本 $(r_{t+1}, s_{t+1}, a_{t+1})$ ，其中 a_{t+1} 是在 π_b 之后生成的。换句话说，SARSA 评估的策略是用于生成样本的策略。

- **Q-Learning 是离线策略**

根本原因是 Q-learning 是求解贝尔曼最优方程的算法，而 SARSA 是求解给定策略的贝尔曼方程的算法。求解贝尔曼方程可以评估相关策略，而求解贝尔曼最优方程可以直接生成最优值和最优策略。

特别地，Q-learning 每次迭代所需的数据样本为 $(s_t, a_t, r_{t+1}, s_{t+1})$ 。这些样本是如何生成的，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 处生成的策略。Q-learning 算法旨在估计 (s_t, a_t) 的最优动作值。该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 ，因为它是通过系统模型（或通过与环境交互）控制的。因此， (s_t, a_t) 的最优动作值的估计不涉及 π_b ，我们可以使用任何 π_b 来在 s_t 处生成 a_t 。而且，这里的目标策略 π_T 是根据估计的最优值得到的贪婪策略（算法 7.3）。行为策略不必与 π_T 相同。

- **MC 学习是在线策略**，原因与 SARSA 类似。待评估和改进的目标策略与生成样本的行为策略相同。

另一个可能与在线策略/离线策略混淆的概念是**在线/离线学习（Online/Offline Learning）**。在线学习是指一旦获得经验样本，就更新价值和策略的情况。离线学习是指所有经验样本收集完毕后才能进行更新的情况。例如，TD 学习是在线的，而 MC 学习是离线的。像 SARSA 这样的在线策略学习算法必须在线工作，因为更新的策略必须用于生成新的经验样本。像 Q-learning 这样的离线策略学习算法可以在线或离线工作。它可以在收到经验样本后或在收集所有体验样本后更新值和策略。

♡ 算法 7.2：通过 Q-learning 进行最优策略学习（on-policy 版本）

初始化：对于所有 (s, a) 和所有的 $t, \epsilon \in (0, 1)$, $\alpha_t(s, a) = \alpha > 0$ 。初始化 $q_0(s, a)$, 适用于所有 (s, a) 。初始从 q_0 派生的 ϵ -贪婪策略 π_0 。

目标：学习能够将智能体从初始状态 s_0 引导到目标状态的最优路径。

For 每个回合, **do**

If s_t ($t = 0, 1, 2, \dots$) 不是目标状态, **do**

 收集给定 s_t 的经验样本 (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:

$$\pi_{t+1}(a|s_t) = 1 - \frac{\epsilon}{|\mathcal{A}(s_t)|} (|\mathcal{A}(s_t)| - 1), \text{ If } a = \operatorname{argmax}_a q_{t+1}(s_t, a) \text{ Else } \pi_{t+1}(a|s_t) = \frac{\epsilon}{|\mathcal{A}(s_t)|}$$

♡ 算法 7.3：通过 Q-learning 进行最优策略学习（off-policy 版本）

初始化：初始化 $q_0(s, a)$, 适用于所有 (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 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 。

For 每个回合 $\{s_0, a_0, r_1, s_1, a_1, r_2, \dots\}$ 由 π_b 生成, **do**

For 回合的每一步 $t = 0, 1, 2, \dots$, **do**

 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:

$$\pi_{T,t+1}(a|s_t) = 1, \text{ If } a = \operatorname{argmax}_a q_{t+1}(s_t, a) \text{ Else } \pi_{T,t+1}(a|s_t) = 0$$

♀ Tip

图 7.3 是 Q-learning 算法的示例。所有回合都从左上角状态开始，并在达到目标状态后终止。目的是找到从起始状态到目标状态的最优路径。奖励设置为 $r_{\text{target}} = 0$, $r_{\text{forbidden}} = r_{\text{boundary}} = -10$, $r_{\text{other}} = -1$ 。学习率为 $\alpha = 0.1$, ϵ 的值为 0.1。左图为算法得到的最终策略。右图显示了每回合的总奖励和长度。

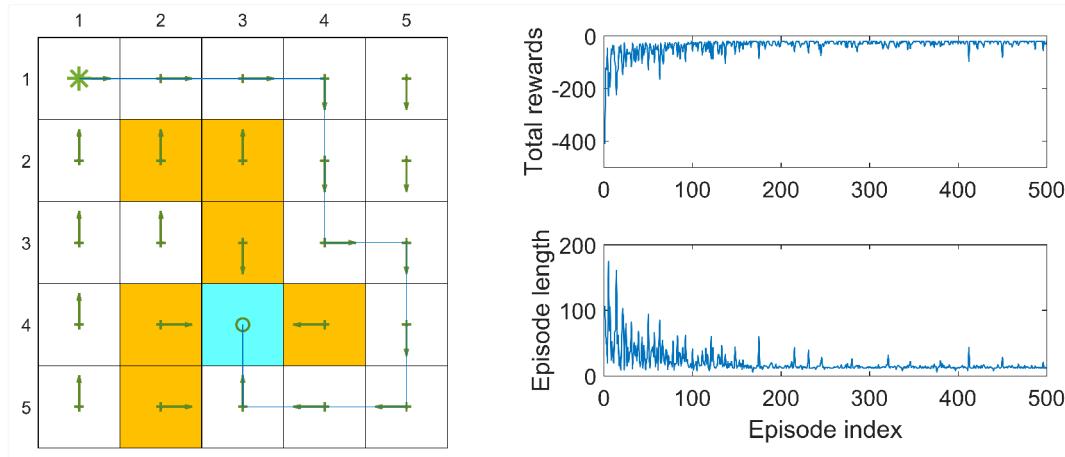


图 7.3

7.4.3 实现

由于 Q-learning 是离线策略的，因此它可以以在线策略或离线策略的方式实现。

Q-learning 的在线策略版本如算法 7.2 所示。这一实现与算法 7.1 中的 SARSA 类似。这里，行为策略与目标策略相同，都是 ϵ -贪婪策略。离线策略版本如算法 7.3 所示。行为策略 π_b 可以是任何策略，只要它能产生足够的经验样本即可。当 π_b 具有探索性时通常是有利的。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，因此不需要探索性。此外，这里提出的 Q-learning 的离线策略版本是离线实现的：首先收集所有经验样本，然后进行处理。可以修改为在线：一旦收到样品，可以立即更新值和策略。

7.4.4 说明性示例

接下来我们将通过示例来演示 Q-learning。

第一个示例如图 7.3 所示。它演示了在线策略 Q-learning。这里的目标是找到从起始状态到目标状态的最佳路径。图 7.3 的绿框给出了设置。可以看出，Q-learning 最终能够找到一条最优路径。在学习过程中，每回合的长度减少，而每回合的总奖励增加。

第二组示例如图 7.4 和图 7.5 所示。他们展示了离线策略 Q-learning。这里的目标是为所有状态找到最优策略。奖励设置为 $r_{\text{target}} = 1$, $r_{\text{forbidden}} = r_{\text{boundary}} = -1$ 。折扣率为 $\gamma = 0.9$ 。学习率为 $\alpha = 0.1$ 。

Tip

图 7.4 是通过 Q-learning 演示离线策略学习的示例。最优策略和最优状态值分别如 (a) 和 (b) 所示。行为策略和生成的回合分别如 (c) 和 (d) 所示。估计策略和估计误差演变分别如 (e) 和 (f) 所示。具有不同初始值的情况如 (g) 和 (h) 所示。

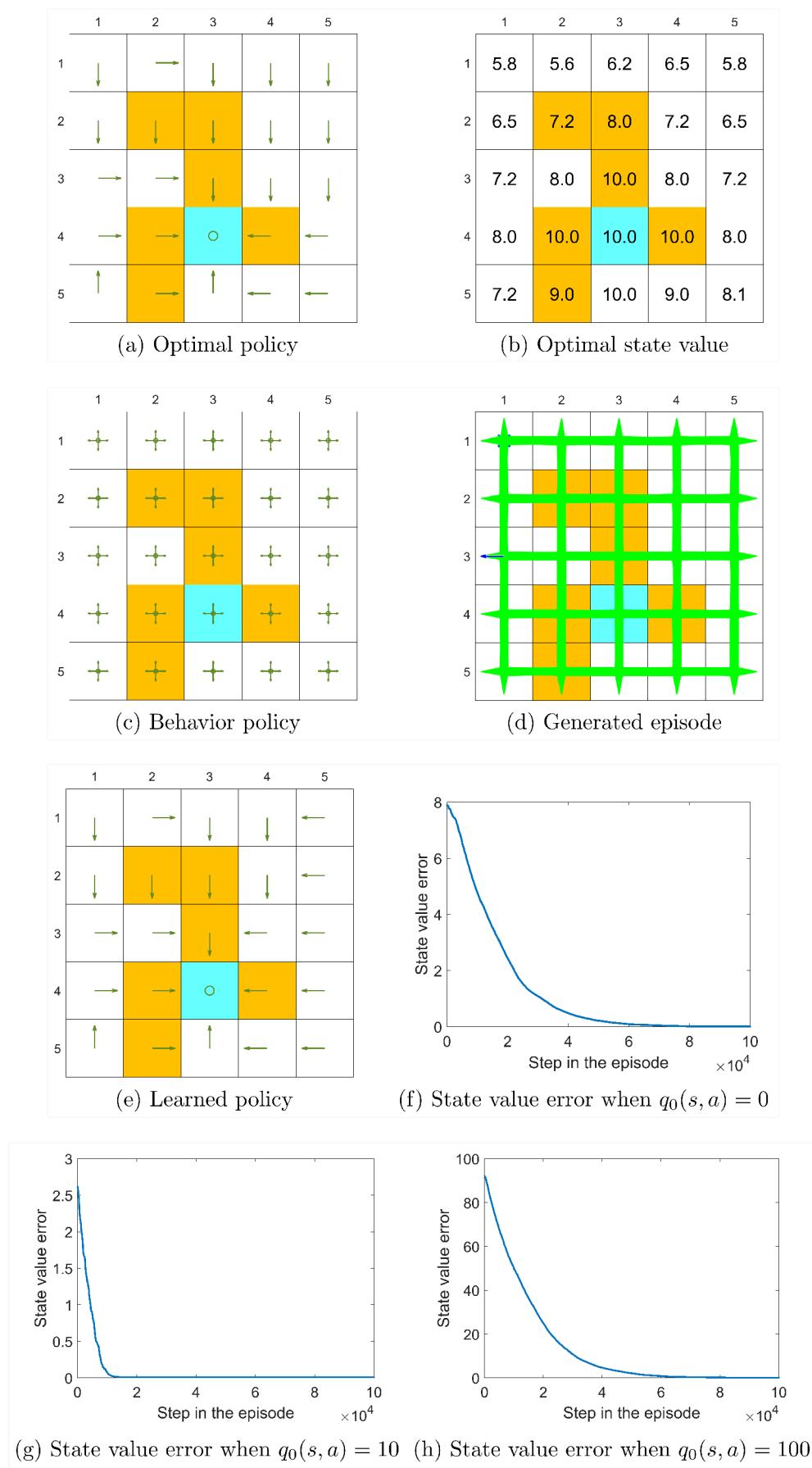


图 7.4

💡 Tip

图 7.5 是当行为策略不具有探索性时, Q-learning 的性能会下降。左栏中的数字显示了行为策略。中间一列的数字显示了遵循相应行为策略生成的回合。每个示例中的回合都有 100,000 个步骤。右列中的数字显示了估计状态值的均方根误差的变化。

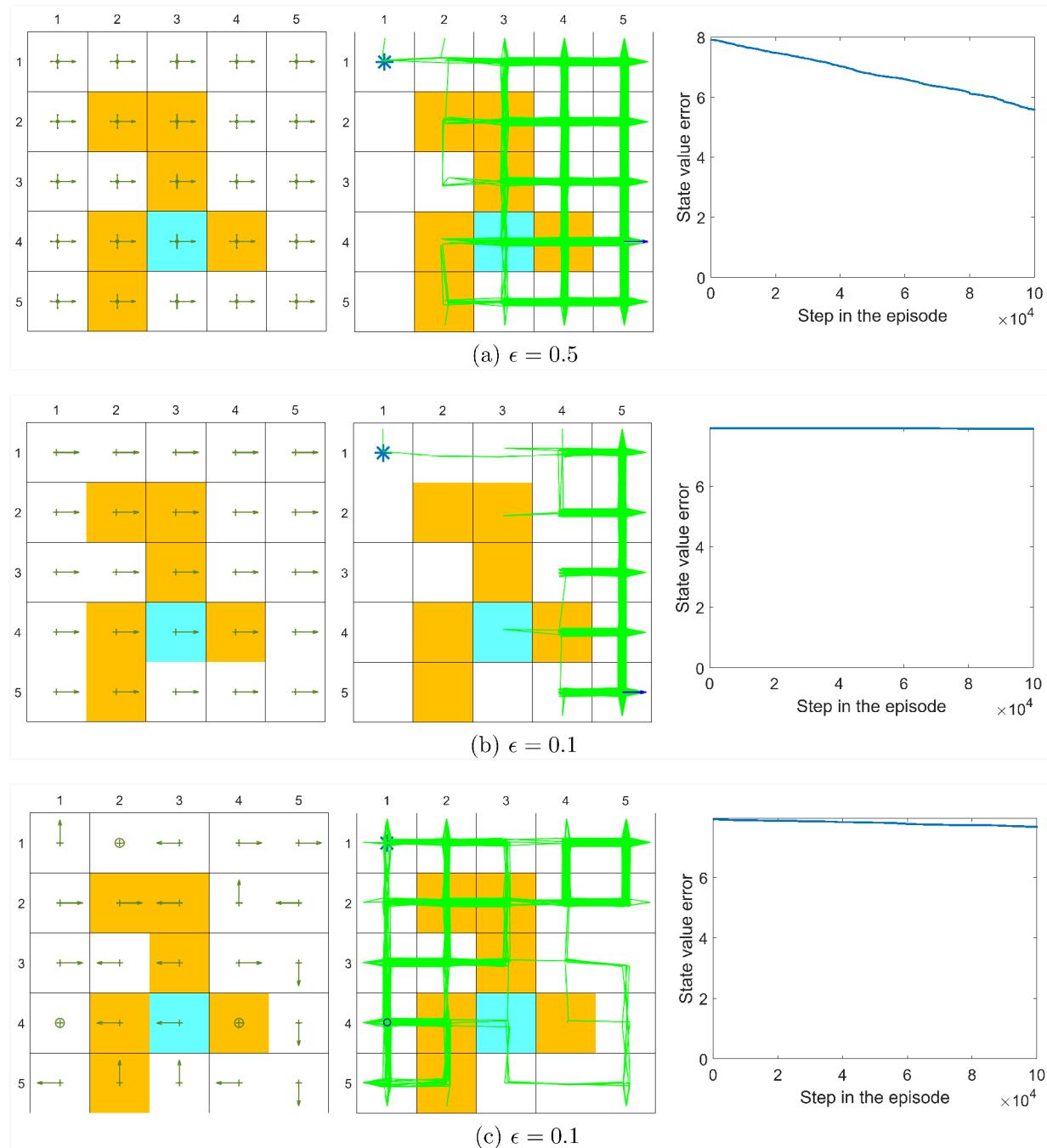


图 7.5

- 基准真值 (Ground Truth) :** 为了验证 Q-learning 的有效性, 我们首先需要知道最优策略和最优状态值的基本事实。这里, 基准真值是通过基于模型的策略迭代算法获得的。基准真值如图 7.4(a) 和 (b) 所示。
- 经验样本 (Experience Samples) :** 行为策略具有均匀分布: 在任何状态下采取任何动作的概率为 0.2 (图 7.4(c))。生成具有 100,000 个步骤的单个回合 (图 7.4(d))。由于行为策略良好的探索能力, 回合可以多次访问每个状态-动作对。

- **学习结果 (Learned Results)**：基于行为策略生成的回合，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，因为估计状态值误差（均方根误差）收敛到零，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) 中的策略并不完全相同。事实上，存在多个具有相同最优状态值的最优策略。
- **不同的初始值 (Different Initial Values)**：由于 Q-learning 是自举的，算法的性能取决于对动作值的初始猜测。如图 7.4(g) 所示，当初始猜测接近真实值时，估计在大约 10,000 步内收敛。否则，收敛需要更多步骤（图 7.4(h)）。尽管如此，这些数据表明，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)**：当行为策略不是探索性的时，学习表现会显着下降。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) 中的统一策略可以被视为 $\epsilon = 1$ 的 ϵ -贪婪）。结果表明，当 ϵ 从 1 减小到 0.5，再减小到 0.1 时，学习速度显着下降。这是因为策略的探索能力较弱，经验样本不足。

7.5 统一框架

到目前为止，我们已经介绍了 SARSA、N-Step SARSA 和 Q-learning 等不同的 TD 算法。在本节中，我们介绍一个统一的框架来容纳所有这些算法和 MC 学习。

Tip

表 7.2 是 TD 算法的统一框架。这里，BE 和 BOE 分别表示贝尔曼方程和贝尔曼最优方程。

算法	(7.20) 中 TD 目标 \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	$\bar{q}_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots$

算法	待解方程
SARSA	BE: $q_\pi(s, a) = \mathbb{E}[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) S_t = s, A_t = a]$
N-Step SARSA	BE: $q_\pi(s, a) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^n q_\pi(S_{t+n}, A_{t+n}) S_t = s, A_t = a]$
Q-learning	BOE: $q_\pi(s, a) = \mathbb{E}[R_{t+1} + \gamma \max_a q(S_{t+1}, a) S_t = s, A_t = a]$
Monte Carlo	BE: $q_\pi(s, a) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots S_t = s, A_t = a]$

表 7.2

特别地，TD 算法（用于动作值估计）可以用统一的表达式来表示：

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

其中 \bar{q}_t 是 TD 目标。不同的 TD 算法有不同的 \bar{q}_t 。总结见表 7.2。MC 学习算法可以被视为 (7.20) 的特例：我们设置 $\alpha_t(s_t, a_t) = 1$ ，然后 (7.20) 变为 $q_{t+1}(s_t, a_t) = \bar{q}_t$ 。

算法 (7.20) 可以看作是求解随机近似算法的统一方程 $q(s, a) = \mathbb{E}[\bar{q}_t | s, a]$ 。这个方程有不同的表达式和不同的 \bar{q}_t 。表 7.2 总结了这些表达式。可以看出，除了 Q-learning 旨在求解贝尔曼最优方程外，其他所有算法都以求解贝尔曼方程为目标。

-
1. T. Kailath, A. H. Sayed, and B. Hassibi, Linear estimation. Prentice Hall, 2000. [↵](#)
 2. C. K. Chui and G. Chen, Kalman filtering. Springer, 2017. [↵](#)
 3. H.-F. Chen, Stochastic approximation and its applications, vol. 64. Springer Science & Business Media, 2006. [↵](#)
 4. R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction (2nd Edition). MIT Press, 2018. [↵](#)
 5. H. Van Seijen, H. Van Hasselt, S. Whiteson, and M. Wiering, “A theoretical and empirical analysis of Expected Sarsa,” in IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning, pp. 177–184, 2009. [↵](#)
 6. M. Ganger, E. Duryea, and W. Hu, “Double Sarsa and double expected Sarsa with shallow and deep learning,” Journal of Data Analysis and Information Processing, vol. 4, no. 4, pp. 159–176, 2016. [↵](#)
 7. C. J. C. H. Watkins, Learning from delayed rewards. PhD thesis, King’s College, 1989. [↵](#)
 8. C. J. Watkins and P. Dayan, “Q-learning,” Machine learning, vol. 8, no. 3-4, pp. 279–292, 1992. [↵](#)
 9. T. Jaakkola, M. I. Jordan, and S. P. Singh, “On the convergence of stochastic iterative dynamic programming algorithms,” Neural Computation, vol. 6, no. 6, pp. 1185–1201, 1994. [↵](#)
 10. C. J. Watkins and P. Dayan, “Q-learning,” Machine learning, vol. 8, no. 3-4, pp. 279–292, 1992. [↵](#)