第五章 SARSA 算法

上一章介绍了 Q 学习的表格形式和神经网络形式（即 DQN）。TD 算法是一大类算法的总称。上一章用的 Q 学习是一种 TD 算法，Q 学习的目的是学习最优动作价值函数 $Q_\star$。本章介绍 SARSA，它也是一种 TD 算法，SARSA 的目的是学习动作价值函数 $Q_\pi(s,a)$。

虽然传统的强化学习用 Q_π 作为确定性的策略控制智能体，但是现在 Q_π 通常被用于评价策略的好坏，而非用于控制智能体。Q_π 常与策略函数 π 结合使用，被称作 Actor-Critic（演员—评委）方法。策略函数 π 控制智能体，因此被看做“演员”；而 Q_π 评价 π 的表现，帮助改进 π，因此 Q_π 被看做“评委”。Actor-Critic 通常用 SARSA 训练“评委”Q_π。在后面策略学习的章节会详细介绍 Actor-Critic 方法。

5.1 表格形式的 SARSA

假设状态空间 $\mathcal{S}$ 和动作空间 $\mathcal{A}$ 都是有限集，即集合中元素数量有限。比如，$\mathcal{S}$ 中一共有 3 种状态，$\mathcal{A}$ 中一共有 4 种动作。那么动作价值函数 $Q_\pi(s,a)$ 可以表示为一个 3×4 的表格，比如右边的表格。该表格与一个策略函数 $\pi(a|s)$ 相关联；如果 π 发生变化，表格 Q_π 也会发生变化。

	第 1 种动作	第 2 种动作	第 3 种动作	第 4 种动作
第 1 种状态	380	-95	20	173
第 2 种状态	-7	64	-195	210
第 3 种状态	152	72	413	-80

图 5.1: 动作价值函数 Q_π 表示成表格形式。

我们用表格 q 近似 Q_π。该如何通过智能体与环境的交互来学习表格 q 呢？首先初始化 q，可以让它是全零的表格。然后用表格形式的 SARSA 算法更新 q，每次更新表格的一个元素。最终 q 收敛到 Q_π。

推导表格形式的 SARSA 学习算法： SARSA 算法由下面的贝尔曼方程推导出：

$$Q_\pi(s_t,a_t) \;=\; \mathbb{E}_{S_{t+1},A_{t+1}}\Big[R_t+\gamma\cdot Q_\pi(S_{t+1},A_{t+1})\,\Big|\,S_t=s_t,A_t=a_t\Big]$$

贝尔曼方程的证明见附录 A。我们对贝尔曼方程左右两边做近似：

- 方程左边的 $Q_\pi(s_t,a_t)$ 可以近似成 $q(s_t,a_t)$。$q(s_t,a_t)$ 是表格在 t 时刻对 $Q_\pi(s_t,a_t)$ 做出的估计。
- 方程右边的期望是关于下一时刻状态 S_{t+1} 和动作 A_{t+1} 求的。给定当前状态 s_t，智能体执行动作 a_t，环境会给出奖励 r_t 和新的状态 s_{t+1}。然后基于 s_{t+1} 做随机抽样，得到新的动作

  $$\tilde{a}_{t+1} \;\sim\; \pi\big(\cdot\,\big|\,s_{t+1}\big).$$

  用观测到的 r_t、s_{t+1} 和计算出的 $\tilde{a}_{t+1}$ 对期望做蒙特卡洛近似，得到：

  $$r_t+\gamma\cdot Q_\pi\big(s_{t+1},\tilde{a}_{t+1}\big). \tag{5.1}$$

- 进一步把公式 (5.1) 中的 Q_π 近似成 q，得到

$$\boxed{\widehat{y}_t \ \triangleq\ r_t + \gamma \cdot q\big(s_{t+1}, \tilde{a}_{t+1}\big).}$$

把它称作 TD 目标。它是表格在 $t+1$ 时刻对 $Q_\pi(s_t, a_t)$ 做出的估计。

$q(s_t, a_t)$ 和 $\widehat{y}_t$ 都是对动作价值 $Q_\pi(s_t, a_t)$ 的估计。由于 $\widehat{y}_t$ 部分基于真实观测到的奖励 r_t，我们认为 $\widehat{y}_t$ 是更可靠的估计，所以鼓励 $q(s_t, a_t)$ 趋近 $\widehat{y}_t$。更新表格 (s_t, a_t) 位置上的元素：

$$\boxed{q\big(s_t, a_t\big) \ \leftarrow\ \big(1-\alpha\big) \cdot q\big(s_t, a_t\big) \ +\ \alpha \cdot \widehat{y}_t.}$$

这样可以使得 $q(s_t, a_t)$ 更接近 $\widehat{y}_t$。SARSA 是 State-Action-Reward-State-Action 的缩写，原因是 SARSA 算法用到了这个五元组：$(s_t, a_t, r_t, s_{t+1}, \tilde{a}_{t+1})$。SARSA 算法学到的 q 依赖于策略 π，这是因为五元组中的 $\tilde{a}_{t+1}$ 是根据 $\pi(\cdot \,|\, s_{t+1})$ 抽样得到的。

训练流程： 设当前表格为 q_{now}，当前策略为 π_{now}。每一轮更新表格中的一个元素，把更新之后的表格记作 q_{new}。

1. 观测到当前状态 s_t，根据当前策略做抽样：$a_t \sim \pi_{\text{now}}(\cdot \,|\, s_t)$。
2. 把表格 q_{now} 中第 (s_t, a_t) 位置上的元素记作：
$$\widehat{q}_t \ =\ q_{\text{now}}\big(s_t, a_t\big).$$
3. 智能体执行动作 a_t 之后，观测到奖励 r_t 和新的状态 s_{t+1}。
4. 根据当前策略做抽样：$\tilde{a}_{t+1} \sim \pi_{\text{now}}(\cdot \,|\, s_{t+1})$。注意，$\tilde{a}_{t+1}$ 只是假想的动作，智能体不予执行。
5. 把表格 q_{now} 中第 $(s_{t+1}, \tilde{a}_{t+1})$ 位置上的元素记作：
$$\widehat{q}_{t+1} \ =\ q_{\text{now}}\big(s_{t+1}, \tilde{a}_{t+1}\big).$$
6. 计算 TD 目标和 TD 误差：
$$\widehat{y}_t \ =\ r_t + \gamma \cdot \widehat{q}_{t+1}, \qquad \delta_j \ =\ \widehat{q}_t - \widehat{y}_t.$$
7. 更新表格中 (s_t, a_t) 位置上的元素：
$$q_{\text{new}}\big(s_t, a_t\big) \ \leftarrow\ \big(1-\alpha\big) \cdot q_{\text{now}}\big(s_t, a_t\big) \ -\ \alpha \cdot \delta_t.$$
8. 用某种算法更新策略函数。该算法与 SARSA 算法无关。

Q 学习与 SARSA 的对比： Q 学习不依赖于 π，因此 Q 学习属于**异策略** (Off-policy)，可以用经验回放。而 SARSA 依赖于 π，因此 SARSA 属于**同策略** (On-policy)，不能用经验回放。两种算法的对比如图 5.2 所示。

Q 学习的目标是学到表格 $\tilde{Q}$，作为最优动作价值函数 $Q_\star$ 的近似。因为 $Q_\star$ 与 π 无关，所以在理想情况下，不论收集经验用的行为策略 π 是什么，都不影响 Q 学习得到的 Q。因此，Q 学习属于异策略 (Off-policy)，允许行为策略区别于目标策略。Q 学习允许使用经验回放，可以重复利用过时的经验。

SARSA 算法的目标是学到表格 q，作为动作价值函数 Q_π 的近似。Q_π 与一个策略 π

相对应；用不同的策略 π，对应 Q_π 就会不同；策略 π 越好，Q_π 的值越大。经验回放数组里的经验 (s_j, a_j, r_j, s_{j+1}) 是过时的行为策略 π_{old} 收集到的，与当前策略 π_{now} 及其对应的价值 $Q_{\pi_{\text{now}}}$ 对应不上。想要学习 Q_π 的话，必须要用与当前策略 π_{now} 收集到的经验，而不能用过时的 π_{old} 收集到的经验。这就是为什么 SARSA 不能用经验回放。

Q 学习	近似 $Q_\star$	异策略	可以使用经验回放
SARSA	近似 Q_π	同策略	不能使用经验回放

图 5.2: Q 学习与 SARSA 的对比。

5.2 神经网络形式的 SARSA

价值网络： 如果状态空间 $\mathcal{S}$ 是无限集，那么我们无法用一张表格表示 Q_π，否则表格的行数是无穷。一种可行的方案是用一个神经网络 $q(s,a;\boldsymbol{w})$ 来近似 $Q_\pi(s,a)$；理想情况下，

$$q(s,a;\boldsymbol{w}) \;=\; Q_\pi(s,a), \qquad \forall s\in\mathcal{S},\ a\in\mathcal{A}.$$

神经网络 $q(s,a;\boldsymbol{w})$ 被称为价值网络 (Value Network)，其中的 $\boldsymbol{w}$ 表示神经网络中可训练的参数。神经网络的结构是人预先设定的（比如有多少层，每一层的宽度是多少），而参数 $\boldsymbol{w}$ 需要通过智能体与环境的交互来学习。首先随机初始化 $\boldsymbol{w}$，然后用 SARSA 算法更新 $\boldsymbol{w}$。

神经网络的结构见图 5.3。价值网络的输入是状态 s；如果 s 是矩阵或张量 (Tensor)，那么可以用卷积网络处理 s（如图 5.3）；如果 s 是向量，那么可以用全连接层处理 s。价值网络的输出是每个动作的价值。动作空间 $\mathcal{A}$ 中有多少种动作，则价值网络的输出就是多少维的向量，向量每个元素对应一个动作。举个例子，动作空间是 $\mathcal{A}=\{左,右,上\}$，价值网络的输出是

$$\begin{aligned} q(s,\ 左;\ \boldsymbol{w}) &= 219, \\ q(s,\ 右;\ \boldsymbol{w}) &= -73, \\ q(s,\ 上;\ \boldsymbol{w}) &= 580. \end{aligned}$$

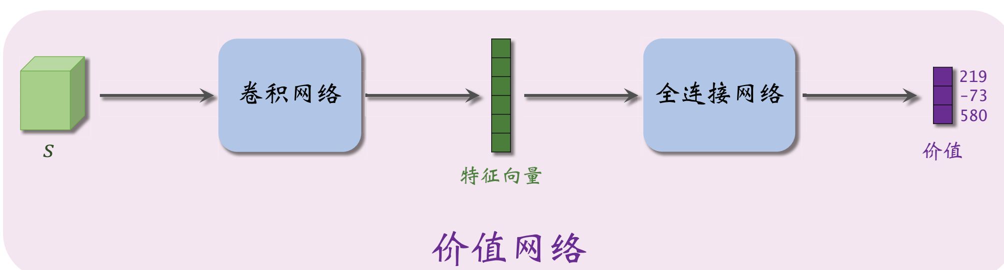

图 5.3: 价值网络 $q(s,a;\boldsymbol{w})$ 的结构。输入是状态 s；输出是每个动作的价值。

算法推导： 给定当前状态 s_t，智能体执行动作 a_t，环境会给出奖励 r_t 和新的状态 s_{t+1}。然后基于 s_{t+1} 做随机抽样，得到新的动作 $\tilde{a}_{t+1}\sim\pi(\cdot\,|\,s_{t+1})$。定义 TD 目标：

$$\widehat{y}_t \;\triangleq\; r_t+\gamma\cdot q(s_{t+1},\tilde{a}_{t+1};\ \boldsymbol{w}),$$

我们鼓励 $q(s_t,a_t;\boldsymbol{w})$ 接近 TD 目标，所以定义损失函数：

$$L(\boldsymbol{w}) \;\triangleq\; \frac{1}{2}\Big[q(s_t,a_t;\boldsymbol{w}) \;-\; \widehat{y}_t\Big]^2.$$

损失函数的变量是 $\boldsymbol{w}$，而 $\widehat{y}_t$ 被视为常数（尽管 $\widehat{y}_t$ 也依赖于参数 $\boldsymbol{w}$，但这一点被忽略掉）。设 $\widehat{q}_t=q(s_t,a_t;\boldsymbol{w})$。损失函数关于 $\boldsymbol{w}$ 的梯度是：

$$\nabla_{\boldsymbol{w}}\, L(\boldsymbol{w}) \;=\; \underbrace{(\widehat{q}_t \;-\; \widehat{y}_t)}_{\text{TD 误差 }\delta_t}\cdot\nabla_{\boldsymbol{w}}\, q(s_t,a_t;\ \boldsymbol{w}).$$

做一次梯度下降更新 $\boldsymbol{w}$:

$$\boldsymbol{w} \;\leftarrow\; \boldsymbol{w} \;-\; \alpha \cdot \delta_t \cdot \nabla_{\boldsymbol{w}}\, q\big(s_t, a_t;\, \boldsymbol{w}\big).$$

这样可以使得 $q(s_t, a_t; \boldsymbol{w})$ 更接近 $\widehat{y}_t$。此处的 α 是学习率，需要手动调。

训练流程： 设当前价值网络的参数为 $\boldsymbol{w}_{\text{now}}$，当前策略为 π_{now}。每一轮训练用五元组 $(s_t, a_t, r_t, s_{t+1}, \tilde{a}_{t+1})$ 对价值网络参数做一次更新。

1. 观测到当前状态 s_t，根据当前策略做抽样：$a_t \sim \pi_{\text{now}}(\cdot \,|\, s_t)$。
2. 用价值网络计算 (s_t, a_t) 的价值：
$$\widehat{q}_t \;=\; q\big(s_t, a_t;\, \boldsymbol{w}_{\text{now}}\big).$$
3. 智能体执行动作 a_t 之后，观测到奖励 r_t 和新的状态 s_{t+1}。
4. 根据当前策略做抽样：$\tilde{a}_{t+1} \sim \pi_{\text{now}}(\cdot \,|s_{t+1})$。注意，$\tilde{a}_{t+1}$ 只是假想的动作，智能体不予执行。
5. 用价值网络计算 $(s_{t+1}, \tilde{a}_{t+1})$ 的价值：
$$\widehat{q}_{t+1} \;=\; q\big(s_{t+1}, \tilde{a}_{t+1};\, \boldsymbol{w}_{\text{now}}\big).$$
6. 计算 TD 目标和 TD 误差：
$$\widehat{y}_t \;=\; r_t + \gamma \cdot \widehat{q}_{t+1}, \qquad \delta_j \;=\; \widehat{q}_t - \widehat{y}_t.$$
7. 对价值网络 q 做反向传播，计算 q 关于 $\boldsymbol{w}$ 的梯度：$\nabla_{\boldsymbol{w}} q\big(s_t, a_t;\, \boldsymbol{w}_{\text{now}}\big)$。
8. 更新价值网络参数：
$$\boldsymbol{w}_{\text{new}} \;\leftarrow\; \boldsymbol{w}_{\text{now}} \;-\; \alpha \cdot \delta_t \cdot \nabla_{\boldsymbol{w}} q\big(s_t, a_t;\, \boldsymbol{w}_{\text{now}}\big).$$
9. 用某种算法更新策略函数。该算法与 SARSA 算法无关。

5.3 多步 TD 目标

首先回顾一下 SARSA 算法。给定五元组 $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$，SARSA 计算 TD 目标：

$$\widehat{y}_t \;=\; r_t \;+\; \gamma \cdot q\big(s_{t+1}, a_{t+1};\, \boldsymbol{w}\big).$$

公式中只用到一个奖励 r_t，这样得到的 $\widehat{y}_t$ 叫做单步 TD 目标。多步 TD 目标用 m 个奖励，可以视作单步 TD 目标的推广。下面我们推导多步 TD 目标。

数学推导： 设一局游戏的长度为 n。根据定义，t 时刻的回报 U_t 是 t 时刻之后的所有奖励的加权和：

$$U_t \;=\; R_t \;+\; \gamma R_{t+1} \;+\; \gamma^2 R_{t+2} \;+\; \cdots \;+\; \gamma^{n-t} R_n.$$

同理，$t+m$ 时刻的回报可以写成：

$$U_{t+m} \;=\; R_{t+m} \;+\; \gamma R_{t+m+1} \;+\; \gamma^2 R_{t+m+2} \;+\; \cdots \;+\; \gamma^{n-t-m} R_n.$$

下面我们推导两个回报的关系。把 U_t 写成：

$$\begin{aligned} U_t \;&=\; \Big(R_t + \gamma R_{t+1} + \cdots + \gamma^{m-1} R_{t+m-1}\Big) \;+\; \Big(\gamma^m R_{t+m} + \cdots + \gamma^{n-t} R_n\Big) \\ &=\; \left(\sum_{i=0}^{m-1} \gamma^i R_{t+i}\right) \;+\; \gamma^m \underbrace{\Big(R_{t+m} + \gamma R_{t+m+1} + \cdots + \gamma^{n-t-m} R_n\Big)}_{\text{等于 } U_{t+m}}. \end{aligned}$$

因此，回报可以写成这种形式：

$$U_t \;=\; \left(\sum_{i=0}^{m-1} \gamma^i R_{t+i}\right) \;+\; \gamma^m U_{t+m}.$$

动作价值函数 $Q_\pi(s_t, a_t)$ 是回报 U_t 的期望，而 $Q_\pi(s_{t+m}, a_{t+m})$ 是回报 U_{t+m}。利用上面的等式，再按照贝尔曼方程的证明（见附录 A），不难得出下面的定理：

定理 5.1

设 R_k 是 S_k、A_k、S_{k+1} 的函数，$\forall\, k = 1, \cdots, n$。那么

$$\underbrace{Q_\pi\big(s_t, a_t\big)}_{U_t \text{ 的期望}} \;=\; \mathbb{E}\left[\left(\sum_{i=0}^{m-1} \gamma^i R_{t+i}\right) \;+\; \gamma^m \cdot \underbrace{Q_\pi\big(S_{t+m}, A_{t+m}\big)}_{U_{t+m} \text{ 的期望}} \,\middle|\, S_t = s_t, A_t = a_t\right].$$

公式中的期望是关于随机变量 $S_{t+1}, A_{t+1}, \cdots, S_{t+m}, A_{t+m}$ 求的。

注 回报 U_t 的随机性来自于 t 到 n 时刻的状态和动作：

$$S_t, A_t, \quad S_{t+1}, A_{t+1}, \cdots, S_{t+m}, A_{t+m}, \quad S_{t+m+1}, A_{t+m+1}, \cdots, S_n, A_n.$$

定理中把 $S_t = s_t$ 和 $A_t = a_t$ 看做是观测值，用期望消掉 $S_{t+1}, A_{t+1}, \cdots, S_{t+m}, A_{t+m}$，而 $Q_\pi(S_{t+m}, A_{t+m})$ 则消掉了剩余的随机变量 $S_{t+m+1}, A_{t+m+1}, \cdots, S_n, A_n$。

多步 TD 目标： 我们对定理 5.1 中的期望做蒙特卡洛近似，然后再用价值网络 $q(s, a; \boldsymbol{w})$ 近似动作价值函数 $Q_\pi(s, a)$。具体做法如下：

- 在 t 时刻，价值网络做出预测 $\widehat{q}_t = q(s_t, a_t; \boldsymbol{w})$，它是对 $Q_\pi(s_t, a_t)$ 的估计。

- 已知当前状态 s_t，用策略 π 控制智能体与环境交互 m 次，得到轨迹

$$r_t, \quad s_{t+1}, a_{t+1}, r_{t+1}, \quad \cdots, \quad s_{t+m-1}, a_{t+m-1}, r_{t+m-1}, \quad s_{t+m}, a_{t+m}.$$

在 $t+m$ 时刻，用观测到的轨迹对定理 5.1 中的期望做蒙特卡洛近似，把近似的结果记作：

$$\left(\sum_{i=0}^{m-1} \gamma^i r_{t+i}\right) \ + \ \gamma^m \cdot Q_\pi\big(s_{t+m}, a_{t+m}\big).$$

- 进一步用 $q(s_{t+m}, a_{t+m}; \boldsymbol{w})$ 近似 $Q_\pi(s_{t+m}, a_{t+m})$，得到：

$$\boxed{\widehat{y}_t \ = \ \left(\sum_{i=0}^{m-1} \gamma^i r_{t+i}\right) \ + \ \gamma^m \cdot q\big(s_{t+m}, a_{t+m}; \boldsymbol{w}\big).}$$

把 $\widehat{y}_t$ 称作 m 步 TD 目标。

$\widehat{q}_t = q(s_t, a_t; \boldsymbol{w})$ 和 $\widehat{y}_t$ 分别是价值网络在 t 时刻和 $t+m$ 时刻做出的预测，两者都是对 $Q_\pi(s_t, a_t)$ 的估计值。$\widehat{q}_t$ 是纯粹的预测，而 $\widehat{y}_t$ 则基于 m 组实际观测，因此 $\widehat{y}_t$ 比 $\widehat{q}_t$ 更可靠。我们鼓励 $\widehat{q}_t$ 接近 $\widehat{y}_t$。设损失函数为

$$L(\boldsymbol{w}) \ \triangleq \ \frac{1}{2}\Big[q\big(s_t, a_t; \boldsymbol{w}\big) \ - \ \widehat{y}_t\Big]^2. \tag{5.2}$$

做单步梯度下降更新价值网络参数 $\boldsymbol{w}$：

$$\boldsymbol{w} \ \leftarrow \ \boldsymbol{w} - \alpha \cdot \Big(\widehat{q}_t \ - \ \widehat{y}_t\Big) \cdot \nabla_{\boldsymbol{w}} q\big(s_t, a_t; \boldsymbol{w}\big).$$

训练流程： 设当前价值网络的参数为 $\boldsymbol{w}_{\text{now}}$，当前策略为 π_{now}。执行以下步骤更新价值网络和策略。

1. 用策略网络 π_{now} 控制智能体与环境交互，完成一个回合，得到轨迹：

$$s_1, a_1, r_1, \quad s_2, a_2, r_2, \quad \cdots, \quad s_n, a_n, r_n.$$

2. 对于所有的 $t = 1, \cdots, n-m$，计算

$$\widehat{q}_t \ = \ q\big(s_t, a_t; \boldsymbol{w}_{\text{now}}\big).$$

3. 对于所有的 $t = 1, \cdots, n-m$，计算多步 TD 目标和 TD 误差：

$$\widehat{y}_t \ = \ \sum_{i=0}^{m-1} \gamma^i r_{t+i} \ + \ \gamma^m \widehat{q}_{t+m}, \qquad \delta_t \ = \ \widehat{q}_t - \widehat{y}_t.$$

4. 对于所有的 $t = 1, \cdots, n-m$，对价值网络 q 做反向传播，计算 q 关于 $\boldsymbol{w}$ 的梯度：

$$\nabla_{\boldsymbol{w}} q\big(s_t, a_t; \ \boldsymbol{w}_{\text{now}}\big).$$

5. 更新价值网络参数：

$$\boldsymbol{w}_{\text{new}}\big(s_t, a_t\big) \ \leftarrow \ \boldsymbol{w}_{\text{now}}\big(s_t, a_t\big) \ - \ \alpha \cdot \sum_{t=1}^{n-m} \delta_t \cdot \nabla_{\boldsymbol{w}} q\big(s_t, a_t; \ \boldsymbol{w}_{\text{now}}\big).$$

6. 用某种算法更新策略函数 π。该算法与 SARSA 算法无关。

5.4 蒙特卡洛与自举

上一节介绍了多步 TD 目标。单步 TD 目标、回报是多步 TD 目标的两种特例。如下图所示，如果设 $m = 1$，那么多步 TD 目标变成单步 TD 目标；如果设 $m = n - t + 1$，那么多步 TD 目标变成实际观测的回报 u_t。

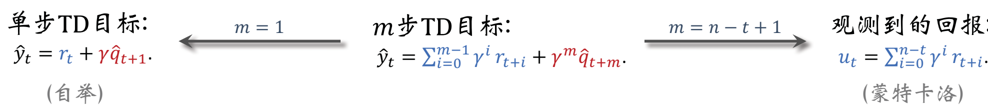

图 5.4: 单步 TD 目标、多步 TD 目标、回报的关系。

5.4.1 蒙特卡洛

训练价值网络 $q(s, a; \boldsymbol{w})$ 的时候，我们可以将一局游戏进行到底，观测到所有的奖励 $r_1, \cdots, r_n$，然后计算回报 $u_t = \sum_{i=0}^{n-t} \gamma^i r_{t+i}$，拿 u_t 作为目标，鼓励价值网络 $q(s_t, a_t; \boldsymbol{w})$ 接近 u_t。定义损失函数：

$$L(\boldsymbol{w}) \;=\; \frac{1}{2}\Big[q\big(s_t, a_t; \boldsymbol{w}\big) \;-\; u_t\Big]^2.$$

然后做一次梯度下降更新 $\boldsymbol{w}$：

$$\boldsymbol{w} \;\leftarrow\; \boldsymbol{w} - \alpha \cdot \nabla_{\boldsymbol{w}} L\big(\boldsymbol{w}\big),$$

这样可以让价值网络的预测 $q(s_t, a_t; \boldsymbol{w})$ 更接近 u_t。这种训练价值网络的方法不是 TD。

在强化学习中，训练价值网络的时候以 u_t 作为目标，这种方式被称作“**蒙特卡洛**”。原因非常显然：动作价值函数可以写作 $Q_\pi(s_t, a_t) = \mathbb{E}[U_t | S_t = s_t, A_t = a_t]$，而我们用实际观测 u_t 去近似期望，这就是典型的蒙特卡洛近似。

蒙特卡洛的好处是**无偏性**：u_t 是 $Q_\pi(s_t, a_t)$ 的无偏估计。由于 u_t 的无偏性，拿 u_t 作为目标训练价值网络，得到的价值网络也是无偏的。

蒙特卡洛的坏处是**方差大**。随机变量 U_t 依赖于 $S_{t+1}, A_{t+1}, \cdots, S_n, A_n$ 这些随机变量，其中不确定性很大。观测值 u_t 虽然是 U_t 的无偏估计，但可能实际上离 $\mathbb{E}[U_t]$ 很远。因此，拿 u_t 作为目标训练价值网络，收敛会很慢。

5.4.2 自举

在介绍价值学习的自举之前，先解释一下什么叫自举。大家可能经常在强化学习和统计学的文章里见到 Bootstrapping 这个词。它的字面意思是“拔自己的鞋带，把自己举起来”。所以 Bootstrapping 翻译成“自举”，即自己把自己举起来。自举听起来很荒谬。即使你“力拔山兮气盖世”，你也没办法拔自己的鞋带，把自己举起来。自举乍看起来很荒唐，但是在统计和机器学习是可以做到自举的；Bootstrapping 方法在统计和机器学习里面非常常用。

在强化学习中，“自举”的意思是“用一个估算去更新同类的估算”，类似于“自己

把自己给举起来”。SARSA 使用的单步 TD 目标定义为：

$$\widehat{y}_t \;=\; r_t + \underbrace{\gamma \cdot q\big(s_{t+1}, a_{t+1}; \boldsymbol{w}\big)}_{\text{价值网络做出的估计}}$$

SARSA 鼓励 $q(s_t, a_t; \boldsymbol{w})$ 接近 $\widehat{y}_t$，所以定义损失函数

$$L(\boldsymbol{w}) \;=\; \frac{1}{2}\Big[\underbrace{q\big(s_t, a_t; \boldsymbol{w}\big) - \widehat{y}_t}_{\text{让价值网络拟合 } \widehat{y}_t}\Big]^2$$

TD 目标 $\widehat{y}_t$ 的一部分是价值网络做出的估计 $\gamma \cdot q(s_{t+1}, a_{t+1}; \boldsymbol{w})$，然后 SARSA 让 $q(s_t, a_t; \boldsymbol{w})$ 去拟合 $\widehat{y}_t$。这就是用价值网络自己做出的估计去更新价值网络自己，这属于“自举”。[1]

自举的好处是**方差小**。单步 TD 目标的随机性只来自于 S_{t+1} 和 A_{t+1}，而回报 U_t 的随机性来自于 $S_{t+1}, A_{t+1}, \cdots, S_n, A_n$。很显然，单步 TD 目标的随机性较小，因此方差较小。用自举的训练价值网络，收敛比较快。

自举的坏处是**有偏差**。价值网络 $q(s, a; \boldsymbol{w})$ 是对动作价值 $Q_\pi(s, a)$ 的近似；最理想的情况下，$q(s, a; \boldsymbol{w}) = Q_\pi(s, a)$，$\forall\, s, a$。假如碰巧 $q(s_{j+1}, a_{j+1}; \boldsymbol{w})$ 低估（或高估）真实价值 $Q_\pi(s_{j+1}, a_{j+1})$，则会发生下面的情况：

$$\begin{aligned}
& q\big(s_{j+1}, a_{j+1}; \boldsymbol{w}\big) && \text{低估（或高估）} && Q_\pi\big(s_{j+1}, a_{j+1}\big) \\
\Longrightarrow\quad & \widehat{y}_j && \text{低估（或高估）} && Q_\pi\big(s_j, a_j\big) \\
\Longrightarrow\quad & q\big(s_j, a_j; \boldsymbol{w}\big) && \text{低估（或高估）} && Q_\pi\big(s_j, a_j\big).
\end{aligned}$$

也就是说，自举会让偏差从 (s_{t+1}, a_{t+1}) 传播到 (s_t, a_t)。第 6.2 节详细讨论自举造成的偏差以及解决方案。

5.4.3 蒙特卡洛和自举的对比

在价值学习中，用实际观测的回报 u_t 作为目标的方法被称为蒙特卡洛，即图 5.5 中的蓝色的箱型图。u_t 是 $Q_\pi(s_t, a_t)$ 的无偏估计，即 U_t 的期望等于 $Q_\pi(s_t, a_t)$。但是它的方差很大，也就是说实际观测到的 u_t 可能离 $Q_\pi(s_t, a_t)$ 很远。

用单步 TD 目标 $\widehat{y}_t$ 作为目标的方法称为自举，即图 5.5 中的红色的箱型图。自举的好处在于方差小，$\widehat{y}_t$ 不会偏离期望太远。但是 $\widehat{y}_t$ 往往是有偏的，它的期望往往不等于 $Q_\pi(s_t, a_t)$。用自举训练出的价值网络往往有系统性的偏差（低估或者高估）。实践中，自举通常比蒙特卡洛收敛更快，这就是为什么训练 DQN 和价值网络通常用 TD 算法。

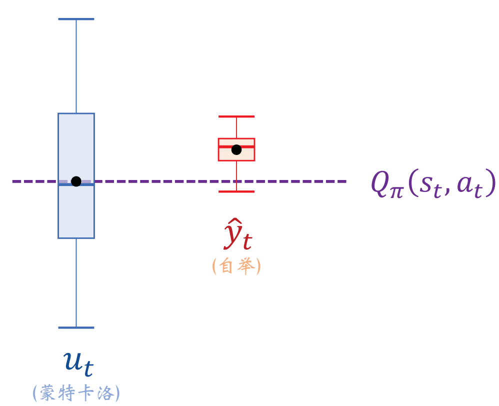

图 5.5: u_t 和 $\widehat{y}_t$ 的箱型图 (Boxplot) 示意。

[1]严格地说，TD 目标 $\widehat{y}_t$ 中既有自举的成分，也有蒙特卡洛的成分。TD 目标中的 $\gamma \cdot q(s_{t+1}, a_{t+1}; \boldsymbol{w})$ 是自举，因为它拿价值网络自己的估计作为目标。TD 目标中的 r_t 是实际观测，它是对 $\mathbb{E}[R_t]$ 的蒙特卡洛。

如图 5.4 所示，多步 TD 目标 $\widehat{y}_t = \left(\sum_{i=0}^{m-1} \gamma^i r_{t+i}\right) + \gamma^m \cdot q\left(s_{t+m}, a_{t+m}; \boldsymbol{w}\right)$ 介于蒙特卡洛和自举之间。多步 TD 目标有很大的蒙特卡洛成分，其中的 $\sum_{i=0}^{m-1} \gamma^i r_{t+i}$ 基于 m 个实际观测到的奖励。多步 TD 目标也有自举的成分，其中的 $\gamma^m \cdot q\left(s_{t+m}, a_{t+m}; \boldsymbol{w}\right)$ 是用价值网络自己算出来的。如果把 m 设置得比较好，可以在方差和偏差之间找到好的平衡，使得多步 TD 目标效果优于单步 TD 目标、也优于回报 u_t。

第五章 相关文献

Q 学习算法首先由 Watkins 在他 1989 年的博士论文 [116] 中提出。Watkins 和 Dayan 发表在 1992 年的论文 [115] 分析了 Q 学习的收敛。1994 年的论文 [54, 105] 改进了 Q 学习算法的收敛分析。

SARSA 算法比 Q 学习提出得晚。SARSA 首先由 Rummery 和 Niranjan 于 1994 年提出 [82]，但名字不叫 SARSA。SARSA 的名字是 Sutton 在 1996 年起的 [96]。

多步 TD 目标也是 Watkins 1989 年的博士论文 [116] 提出的。Sutton 和 Barto 的书 [97] 对多步 TD 目标有详细介绍和分析。近年来有不少论文（比如 [69, 110, 47]）表明多步 TD 目标非常有用。