

# 第五章 SARSA 算法

上一章介绍了 Q 学习 (Q-learning) 的表格形式、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 TD 的一种。本章介绍 SARSA，它是另一种 TD 算法，它的目的是学习动作价值函数  $Q_\pi(s, a)$ 。（Q 学习算法是学习  $Q_*$ ，注意两者区别。）

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

## 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  $3 \times 4$  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  $\pi$  发生变化，表格  $Q_\pi$  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  $Q_\pi$  表示成表格形式。

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

**推导表格形式的 SARSA 学习算法：**首先复习一下贝尔曼方程：

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

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  $\tilde{q}(s_t, a_t)$ 。 $\tilde{q}(s_t, a_t)$  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

$$a_{t+1} \sim \pi(\cdot \mid s_{t+1}).$$

用观测到的  $r_t$ 、 $s_{t+1}$ 、 $a_{t+1}$  对期望做蒙特卡洛近似，得到：

$$r_t + \gamma \cdot Q_\pi(s_{t+1}, a_{t+1}). \quad (5.1)$$

- 进一步把公式 (5.1) 中的  $Q_\pi$  近似成  $\tilde{q}$ ，得到

$$\hat{y}_t \triangleq r_t + \gamma \cdot \tilde{q}(s_{t+1}, a_{t+1}).$$

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。

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

$$\tilde{q}(s_t, a_t) \leftarrow \tilde{q}(s_t, a_t) - \alpha \cdot \tilde{q}(s_t, a_t) + \alpha \cdot \hat{y}_t.$$

这样可以使得  $\tilde{q}(s_t, a_t)$  更接近  $\hat{y}_t$ 。SARSA 是 State-Action-Reward-State-Action 的缩写，原因是 SARSA 算法用到了这个五元组： $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ 。

**训练流程：**设当前表格为  $\tilde{q}_{\text{now}}$ ，当前策略为  $\pi_{\text{now}}$ 。每一轮更新表格中的一个元素，把更新之后的表格记作  $\tilde{q}_{\text{new}}$ 。

- 观测到当前状态  $s_t$ ，根据当前策略做抽样： $a_t \sim \pi_{\text{now}}(\cdot | s_t)$ 。
- 把表格  $\tilde{q}_{\text{now}}$  中第  $(s_t, a_t)$  位置上的元素记作：

$$\hat{q}_t = \tilde{q}_{\text{now}}(s_t, a_t).$$

- 智能体执行动作  $a_t$  之后，观测到奖励  $r_t$  和新的状态  $s_{t+1}$ 。
- 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，智能体不予以执行。
- 把表格  $\tilde{q}_{\text{now}}$  中第  $(s_{t+1}, \tilde{a}_{t+1})$  位置上的元素记作：

$$\hat{q}_{t+1} = \tilde{q}_{\text{now}}(s_{t+1}, \tilde{a}_{t+1}).$$

- 计算 TD 目标和 TD 误差：

$$\hat{y}_t = r_t + \gamma \cdot \hat{q}_{t+1}, \quad \delta_j = \hat{q}_t - \hat{y}_t.$$

- 更新表格中  $(s_t, a_t)$  位置上的元素：

$$\tilde{q}_{\text{new}}(s_t, a_t) \leftarrow \tilde{q}_{\text{now}}(s_t, a_t) - \alpha \cdot \delta_t.$$

- 用某种算法更新策略函数。（算法取决于学习策略函数的目标是什么，而与 SARSA 算法无关。SARSA 学到的  $\tilde{q}$  只是评价策略、状态、动作的好坏。）

**Q 学习与 SARSA 的对比：**Q 学习不依赖于  $\pi$ ，而 SARSA 依赖于  $\pi$ 。因此，Q 学习属于异策略，可以用经验回放。而 SARSA 属于同策略，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  $Q_*$  的近似。因为  $Q_*$  与  $\pi$  无关，所以在理想情况下，不论收集经验用的行为策略  $\pi$  是什么，都不影响 Q 学习得到的  $\tilde{Q}$ 。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  $\tilde{q}$ ，作为动作价值函数  $Q_\pi$  的近似。 $Q_\pi$  与一个策略  $\pi$  相对应：用不同的策略  $\pi$ ，对应  $Q_\pi$  就会不同；策略  $\pi$  越好， $Q_\pi$  的值越大。 $Q_\pi$  显然依赖于  $\pi$ ；作为  $Q_\pi$  的近似， $\tilde{q}$  也依赖于  $\pi$ ，这导致 SARSA 算法不允许使用经验回放。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**  $\pi_{\text{old}}$  收集到的。四元组的  $r_j$  是用  $\pi_{\text{old}}$  做决策得到的产物， $r_j$  与 **当前策略**  $\pi_{\text{now}}$  及其对应的价值  $Q_{\pi_{\text{now}}}$  对应不上。因此，不能用过时的  $r_j$  帮助训练当前的表格  $\tilde{q}_{\text{now}}$ 。SARSA 属于**同策略** (On-policy)，要求**行为策略** (收集经验用的策略) 与**目标策略** (当前需要更新的策略) 必须相同。

Q 学习	近似 $Q_*$	异策略	可以使用经验回放
SARSA	近似 $Q_\pi$	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  $Q_\pi$ ，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  $q(s, a; \mathbf{w})$  来近似  $Q_\pi(s, a)$ ；理想情况下，

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

神经网络  $q(s, a; \mathbf{w})$  被称为价值网络 (Value Network)，其中的  $\mathbf{w}$  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  $\mathbf{w}$  需要通过智能体与环境的交互来学习。首先随机初始化  $\mathbf{w}$ ，然后用 SARSA 算法更新  $\mathbf{w}$ 。

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)，那么可以用卷积层处理  $s$  (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  $\mathcal{A} = \{\text{左, 右, 上}\}$ ，价值网络的输出是

$$q(s, \text{左}; \mathbf{w}) = 219,$$

$$q(s, \text{右}; \mathbf{w}) = -73,$$

$$q(s, \text{上}; \mathbf{w}) = 580.$$

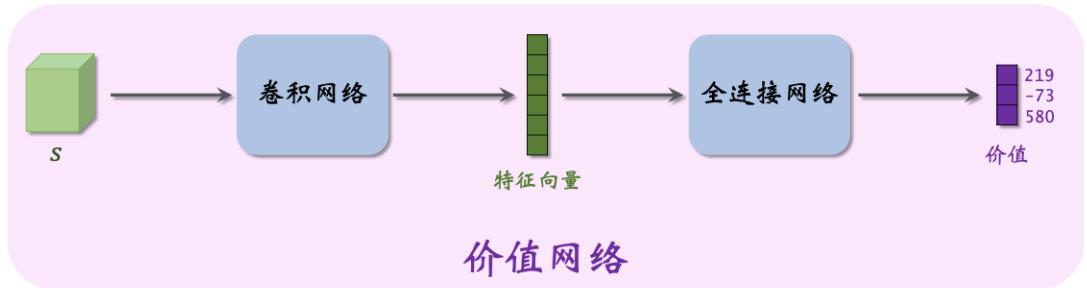


图 5.3：价值网络  $q(s, a; \mathbf{w})$  的结构。输入是状态  $s$ ；输出是每个动作的价值。

**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  $a_{t+1} \sim \pi(\cdot | s_{t+1})$ 。定义 TD 目标：

$$\hat{y}_t \triangleq r_t + \gamma \cdot q(s_{t+1}, a_{t+1}; \mathbf{w}),$$

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

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

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

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

做一次梯度下降更新  $\mathbf{w}$ :

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

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

**训练流程:** 设当前价值网络的参数为  $\mathbf{w}_{\text{now}}$ ，当前策略为  $\pi_{\text{now}}$ 。每一轮训练用五元组  $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$  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：

$$\hat{q}_t = q(s_t, a_t; \mathbf{w}_{\text{now}}).$$

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：

$$\hat{q}_{t+1} = q(s_{t+1}, \tilde{a}_{t+1}; \mathbf{w}_{\text{now}}).$$

6. 计算 TD 目标和 TD 误差：

$$\hat{y}_t = r_t + \gamma \cdot \hat{q}_{t+1}, \quad \delta_j = \hat{q}_t - \hat{y}_t.$$

7. 对价值网络  $q$  做反向传播，计算  $q$  关于  $\mathbf{w}$  的梯度： $\nabla_{\mathbf{w}} q(s_t, a_t; \mathbf{w}_{\text{now}})$ 。
8. 更新价值网络参数：

$$\mathbf{w}_{\text{new}} \leftarrow \mathbf{w}_{\text{now}} - \alpha \cdot \delta_t \cdot \nabla_{\mathbf{w}} q(s_t, a_t; \mathbf{w}_{\text{now}}).$$

9. 用某种算法更新策略函数。

### 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:

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

公式中只用到一个奖励  $r_t$ , 这样得到的  $\hat{y}_t$  叫做“单步 TD 目标”。多步 TD 目标用  $m > 1$  个奖励。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:

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

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  $Q_\pi$  是回报  $U$  的期望。我们可以利用上面的等式, 按照贝尔曼方程的证明 (见附录 A) 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,  $\forall k = 1, \dots, n$ 。那么

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

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

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

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

**多步 TD 目标:** 我们用价值网络  $q(s, a; \mathbf{w})$  近似动作价值函数  $Q_\pi(s, a)$ , 用蒙特卡洛近似期望。具体这样做近似:

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

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

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

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:

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

- 进单步用  $q(s_{t+m}, a_{t+m}; \mathbf{w})$  近似  $Q_\pi(s_{t+m}, a_{t+m})$ , 得到:

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

把  $\hat{y}_t$  称作  $m$  步 TD 目标。

$\hat{q}_t = q(s_t, a_t; \mathbf{w})$  和  $\hat{y}_t$  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。 $\hat{q}_t$  是纯粹的预测, 而  $\hat{y}_t$  则基于  $m$  组实际观测, 因此  $\hat{y}_t$  比  $\hat{q}_t$  更可靠。我们鼓励  $\hat{q}_t$  接近  $\hat{y}_t$ 。设损失函数为

$$L(\mathbf{w}) \triangleq \frac{1}{2} [q(s_t, a_t; \mathbf{w}) - \hat{y}_t]^2. \quad (5.2)$$

做单步梯度下降更新价值网络参数  $\mathbf{w}$ :

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \cdot (\hat{q}_t - \hat{y}_t) \cdot \nabla_{\mathbf{w}} q(s_t, a_t; \mathbf{w}).$$

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

- 用策略网络  $\pi_{\text{now}}$  控制智能体与环境交互, 完成一个回合, 得到轨迹:

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

- 对于所有的  $t = 1, \dots, n - m$ , 计算

$$\hat{q}_t = q(s_t, a_t; \mathbf{w}_{\text{now}}).$$

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

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

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

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

- 更新价值网络参数:

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

- 用某种算法更新策略函数。

## 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  $u_t$ 。

$$\begin{array}{ccc} \text{单步 TD 目标: } & \xleftarrow{m=1} & m\text{ 步 TD 目标: } \\ \hat{y}_t = r_t + \gamma \hat{q}_{t+1}. & & \hat{y}_t = \sum_{i=0}^{m-1} \gamma^i r_{t+i} + \gamma^m \hat{q}_{t+m}. \\ (\text{自举}) & & \end{array} \quad \xrightarrow{m=n-t+1} \quad \begin{array}{c} \text{观测到的回报:} \\ u_t = \sum_{i=0}^{n-t} \gamma^i r_{t+i}. \\ (\text{蒙特卡洛}) \end{array}$$

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

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

然后做一次梯度下降更新  $\mathbf{w}$ ：

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \cdot \nabla_{\mathbf{w}} L(\mathbf{w}),$$

这样可以让价值网络的预测  $q(s_t, a_t; \mathbf{w})$  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  $S_{t+1}, A_{t+1}, \dots, S_n, A_n$  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：

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

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

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

TD 目标  $\hat{y}_t$  的一部分是价值网络做出的估计  $\gamma \cdot q(s_{t+1}, a_{t+1}; \mathbf{w})$ ，然后 SARSA 让  $q(s_t, a_t; \mathbf{w})$  去拟合  $\hat{y}_t$ 。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”。<sup>1</sup>

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  $S_{t+1}, A_{t+1}, \dots, S_n, A_n$ 。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。假如  $q(s_{t+1}, a_{t+1}; \mathbf{w})$  是对  $Q_\pi(s_{t+1}, a_{t+1})$  的低估（或高估），那么  $\hat{y}_t$  则是对  $Q_\pi(s_t, a_t)$  的低估（或高估）；用  $\hat{y}_t$  作为目标，会让  $q(s_t, a_t; \mathbf{w})$  也倾向于低估（或高估）。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  $\hat{y}_t$  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， $\hat{y}_t$  不会偏离期望太远。但是  $\hat{y}_t$  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。

多步 TD 目标  $\hat{y}_t = (\sum_{i=0}^{m-1} \gamma^i r_{t+i}) + \gamma^m \cdot q(s_{t+m}, a_{t+m}; \mathbf{w})$  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  $\gamma^m \cdot q(s_{t+m}, a_{t+m}; \mathbf{w})$  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 目标效果会优于单步 TD 目标、优于回报。

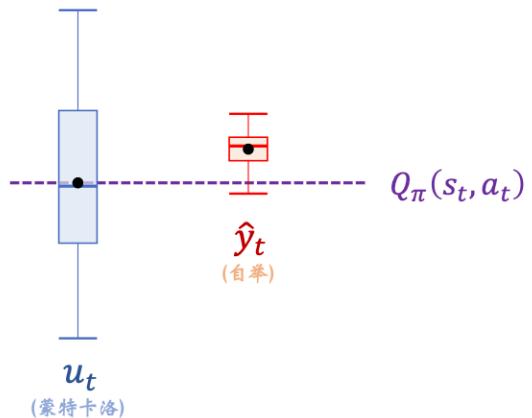


图 5.5:  $u_t$  和  $\hat{y}_t$  的箱型图 (Boxplot) 示意。

<sup>1</sup>严格地说，TD 目标  $\hat{y}_t$  中既有自举的成分，也有蒙特卡洛的成分。TD 目标中的  $\gamma \cdot q(s_{t+1}, a_{t+1}; \mathbf{w})$  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。

## 相关文献

Q 学习算法首先由 Watkins 在他 1989 年的博士论文 [59] 中提出。Watkins 和 Dayan 发表在 1992 年的论文 [58] 分析了 Q 学习的收敛。1994 年的论文 [26, 51] 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 [39]，但名字不叫 SARSA。SARSA 的名字是 Sutton 在 1996 年起的 [47]。

多步 TD 目标也是 Watkins 1989 年的博士论文 [59] 提出的。Sutton 和 Barto 的书 [48] 对多步 TD 目标有详细介绍和分析。近年来有不少论文（比如 [32, 56, 25]）表明多步 TD 目标非常有用。

## 参考文献

- [1] M. S. Abdulla and S. Bhatnagar. Reinforcement learning based algorithms for average cost markov decision processes. *Discrete Event Dynamic Systems*, 17(1):23–52, 2007.
- [2] L. V. Allis et al. *Searching for solutions in games and artificial intelligence*. Ponsen & Looijen Wageningen, 1994.
- [3] L. Baird. Residual algorithms: reinforcement learning with function approximation. In *Machine Learning Proceedings 1995*, pages 30–37. Elsevier, 1995.
- [4] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846, 1983.
- [5] P. Baudiš and J.-l. Gailly. Pachi: State of the art open source go program. In *Advances in computer games*, pages 24–38. Springer, 2011.
- [6] M. G. Bellemare, W. Dabney, and R. Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2017.
- [7] D. P. Bertsekas. *Constrained optimization and Lagrange multiplier methods*. Academic press, 2014.
- [8] S. Bhatnagar and S. Kumar. A simultaneous perturbation stochastic approximation-based actor-critic algorithm for markov decision processes. *IEEE Transactions on Automatic Control*, 49(4):592–598, 2004.
- [9] S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee. Natural actor-critic algorithms. *Automatica*, 45(11):2471–2482, 2009.
- [10] B. Bouzy and B. Helmstetter. Monte-Carlo go developments. In *Advances in computer games*, pages 159–174. Springer, 2004.
- [11] S. Boyd, S. P. Boyd, and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.
- [12] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfschagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.
- [13] M. Buro. From simple features to sophisticated evaluation functions. In *International Conference on Computers and Games*, pages 126–145. Springer, 1998.
- [14] M. Campbell, A. J. Hoane Jr, and F.-h. Hsu. Deep blue. *Artificial intelligence*, 134(1-2):57–83, 2002.
- [15] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck. Monte-Carlo tree search: A new framework for game AI. In *AIIDE*, 2008.
- [16] G. Chaslot, J.-T. Saito, B. Bouzy, J. Uiterwijk, and H. J. Van Den Herik. Monte-Carlo strategies for computer Go. In *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence, Namur, Belgium*, 2006.
- [17] G. M. J.-B. C. Chaslot. *Monte-Carlo tree search*. Maastricht University, 2010.
- [18] A. R. Conn, N. I. Gould, and P. L. Toint. *Trust region methods*. SIAM, 2000.
- [19] R. Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006.
- [20] R. Coulom. Computing “elo ratings” of move patterns in the game of Go. *ICGA journal*, 30(4):198–208, 2007.
- [21] M. Enzenberger, M. Müller, B. Arneson, and R. Segal. Fuego: an open-source framework for board games and go engine based on monte carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):259–270, 2010.
- [22] M. Fortunato, M. G. Azar, B. Piot, J. Menick, M. Hessel, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, et al. Noisy networks for exploration. In *International Conference on Learning Representations (ICLR)*, 2018.
- [23] R. Hafner and M. Riedmiller. Reinforcement learning in feedback control. *Machine learning*, 84(1-2):137–169, 2011.
- [24] M. Hausknecht and P. Stone. Deep recurrent Q-learning for partially observable MDPs. In *AAAI Fall Symposium on Sequential Decision Making for Intelligent Agents*, 2015.

- [25] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.
- [26] T. Jaakkola, M. I. Jordan, and S. P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural computation*, 6(6):1185–1201, 1994.
- [27] L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.
- [28] V. R. Konda and J. N. Tsitsiklis. Actor-critic algorithms. In *Advances in Neural Information Processing Systems (NIPS)*, 2000.
- [29] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2016.
- [30] L.-J. Lin. Reinforcement learning for robots using neural networks. Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993.
- [31] P. Marbach and J. N. Tsitsiklis. Simulation-based optimization of markov reward processes: Implementation issues. In *Proceedings of the 38th IEEE Conference on Decision and Control*, 1999.
- [32] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2016.
- [33] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.
- [34] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.
- [35] M. Müller. Computer go. *Artificial Intelligence*, 134(1-2):145–179, 2002.
- [36] J. Nocedal and S. Wright. *Numerical optimization*. Springer Science & Business Media, 2006.
- [37] D. V. Prokhorov and D. C. Wunsch. Adaptive critic designs. *IEEE transactions on Neural Networks*, 8(5):997–1007, 1997.
- [38] T. Rashid, M. Samvelyan, C. Schroeder, G. Farquhar, J. Foerster, and S. Whiteson. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2018.
- [39] G. A. Rummery and M. Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.
- [40] J. Schaeffer, N. Burch, Y. Björnsson, A. Kishimoto, M. Müller, R. Lake, P. Lu, and S. Sutphen. Checkers is solved. *science*, 317(5844):1518–1522, 2007.
- [41] J. Schaeffer, J. Culberson, N. Treloar, B. Knight, P. Lu, and D. Szafron. A world championship caliber checkers program. *Artificial Intelligence*, 53(2-3):273–289, 1992.
- [42] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. In *International Conference on Learning Representations (ICLR)*, 2015.
- [43] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International Conference on Machine Learning (ICML)*, 2015.
- [44] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.
- [45] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *International Conference on Machine Learning (ICML)*, 2014.
- [46] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.
- [47] R. S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In

- Advances in Neural Information Processing Systems (NIPS)*, 1996.
- [48] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
  - [49] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, 2000.
  - [50] G. Tesauro and G. R. Galperin. On-line policy improvement using monte-carlo search. In *Advances in Neural Information Processing Systems*, pages 1068–1074, 1997.
  - [51] J. N. Tsitsiklis. Asynchronous stochastic approximation and Q-learning. *Machine learning*, 16(3):185–202, 1994.
  - [52] J. N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE transactions on automatic control*, 42(5):674–690, 1997.
  - [53] H. J. Van Den Herik, J. W. Uiterwijk, and J. Van Rijswijck. Games solved: Now and in the future. *Artificial Intelligence*, 134(1-2):277–311, 2002.
  - [54] H. van Hasselt. Double q-learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2010.
  - [55] H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.
  - [56] H. van Seijen. Effective multi-step temporal-difference learning for non-linear function approximation. *arXiv preprint arXiv:1608.05151*, 2016.
  - [57] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas. Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2016.
  - [58] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.
  - [59] C. J. C. H. Watkins. Learning from delayed rewards. 1989.
  - [60] R. J. Williams. *Reinforcement-learning connectionist systems*. College of Computer Science, Northeastern University, 1987.
  - [61] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.
  - [62] Z. Yang, Y. Chen, M. Hong, and Z. Wang. Provably global convergence of actor-critic: A case for linear quadratic regulator with ergodic cost. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 8353–8365, 2019.