第8章 价值函数方法 (Value Function Methods)

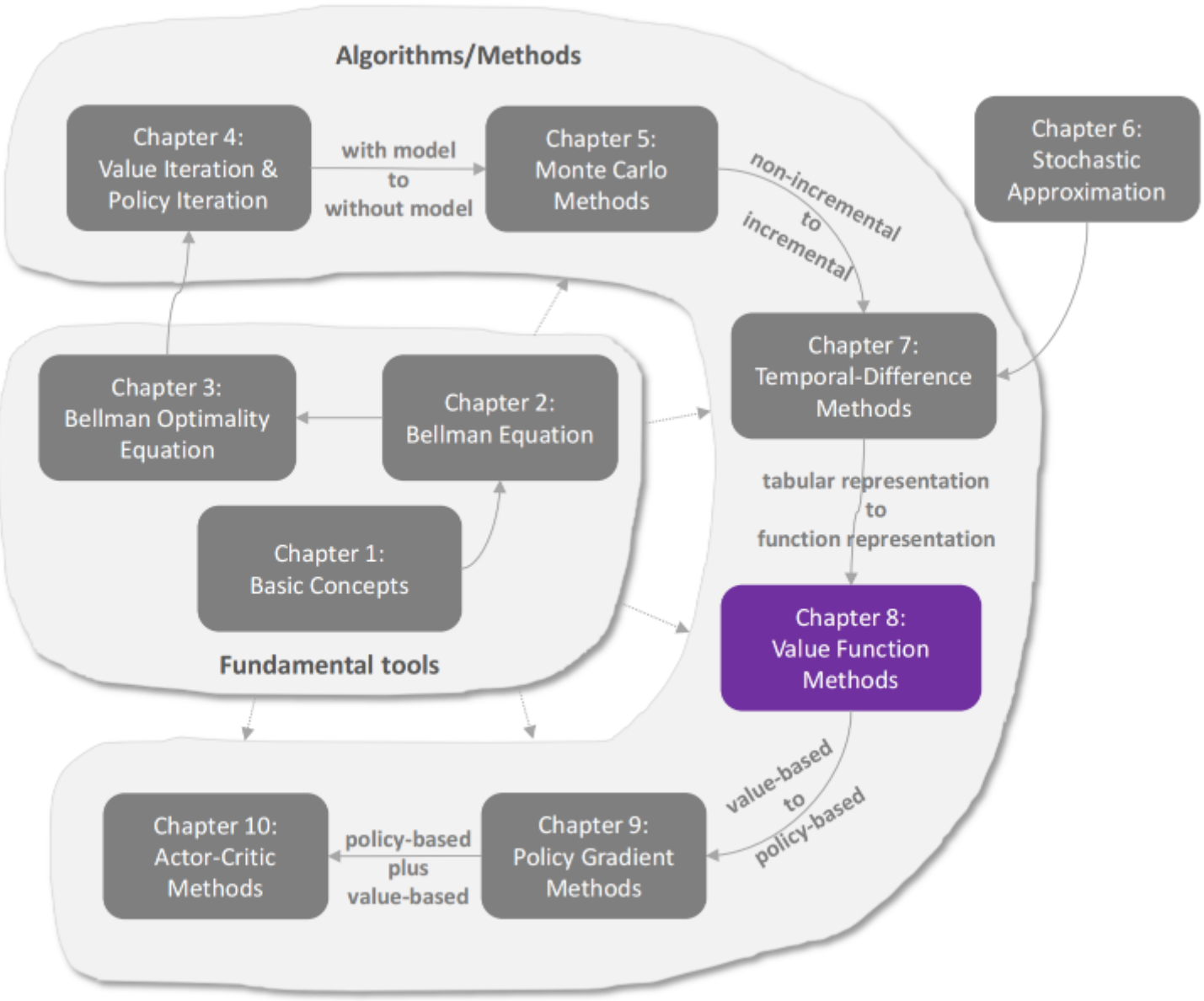


Figure 8.1: Where we are in this book.

图 8.1：我们在本书中的位置。

在本章中，我们将继续学习时序差分 (temporal-difference) 学习算法。然而，我们将使用一种不同的方法来表示状态/动作价值。到目前为止，本书中的状态/动作价值一直是用 ***表格 (tables)*** 来表示的。表格方法直观易懂，但在处理大规模状态或动作空间时效率低下。为了解决这个问题，本章介绍了 ***价值函数 (value function)*** 方法，它已成为表示价值的标准方式。这也是将人工神经网络作为函数逼近器 (function approximators) 引入强化学习的地方。价值函数的思想也可以扩展到 ***策略函数 (policy function)***，这将在第 9 章中介绍。

8.1 价值表示：从表格到函数 (Value representation: From table to function)

接下来，我们用一个例子来演示表格方法 (tabular method) 和函数逼近方法 (function approximation method) 之间的区别。

假设有 n 个状态 $\{s_i\}_{i=1}^n$，其状态价值为 $\{v_\pi(s_i)\}_{i=1}^n$。这里，π 是一个给定策略。令 $\{\hat{v}(s_i)\}_{i=1}^n$ 表示真实状态价值的估计值。如果我们使用表格方法，估计值可以维护在下表中。该表可以作为数组或向量存储在内存中。要检索或更新任何值，我们可以直接读取或重写表中的相应条目。

状态 (State)	$s1$	$s2$	...	s_n
估计值 (Estimated value)	$\hat{v}(s1)$	$\hat{v}(s2)$	$\cdots$	$\hat{v}(s_n)$

接下来我们展示上表中的值可以通过一个函数来逼近。具体来说，$\{(s_i, \hat{v}(s_i))\}_{i=1}^n$ 显示为图 8.2 中的 n 个点。这些点可以通过一条曲线来拟合或逼近。最简单的曲线是一条直线，可以描述为

$$\hat{v}(s,w) = as + b = \underbrace{[s,1]}_{\phi^T(s)} \underbrace{\begin{bmatrix} a \\ b \end{bmatrix}}_{w} = \phi^T(s)w. \quad (8.1)$$

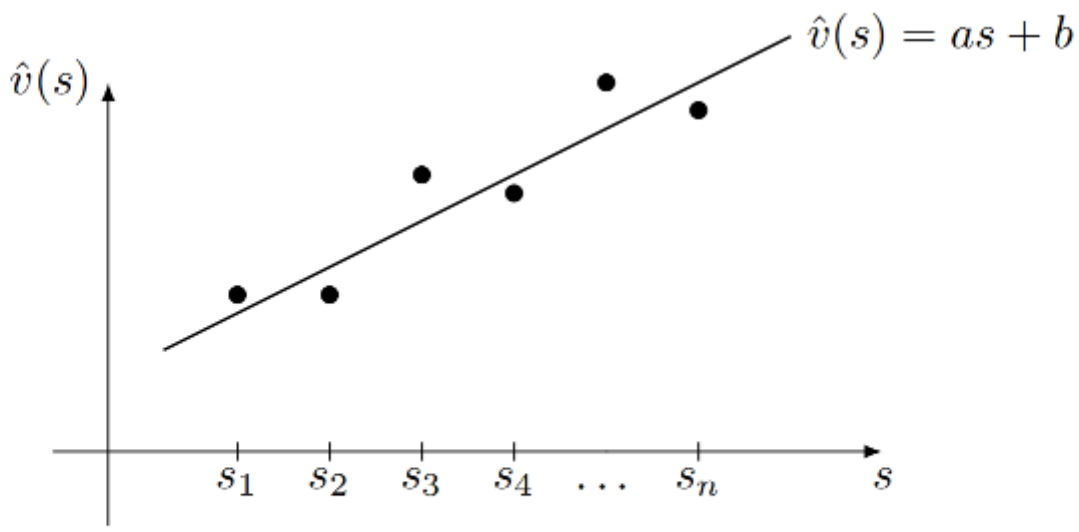


Figure 8.2: An illustration of the function approximation method. The x-axis and y-axis correspond to s and $\hat{v}(s)$, respectively.

图 8.2：函数逼近方法的图示。 x 轴和 y 轴分别对应于 s 和 $\hat{v}(s)$。

这里，$\hat{v}(s,w)$ 是一个用于逼近 $v_\pi(s)$ 的函数。**它由状态 s 和参数向量 $w \in \mathbb{R}^2$ 共同决定。** $\hat{v}(s,w)$ 有时也被写作 $\hat{v}_w(s)$。这里，$\phi(s) \in \mathbb{R}^2$ 被称为 s 的 ***特征向量 (feature vector)***。

表格方法和函数逼近方法之间第一个显著的区别在于它们如何检索和更新一个值。

- **如何 *检索 (retrieve)* 一个值：** 当价值由表格表示时，如果我们想要检索一个值，我们可以直接读取表中的相应条目。然而，当数值由函数表示时，检索数值会稍微复杂一点。具体来说，我们需要将状态索引 s 输入到函数中并计算函数值（图 8.3）。对于 (8.1) 中的例子，**我们首先需要计算特征向量 $\phi(s)$，然后计算 $\phi^T(s)w$。如果函数是人工神经网络，则需要进行从输入到输出的前向传播 (forward propagation)。**

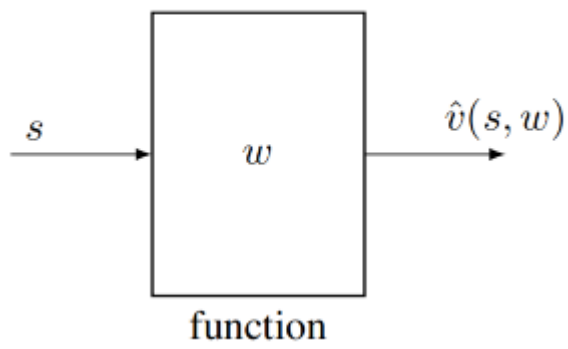


图 8.3：使用函数逼近方法时检索 s 的值的过程图示。

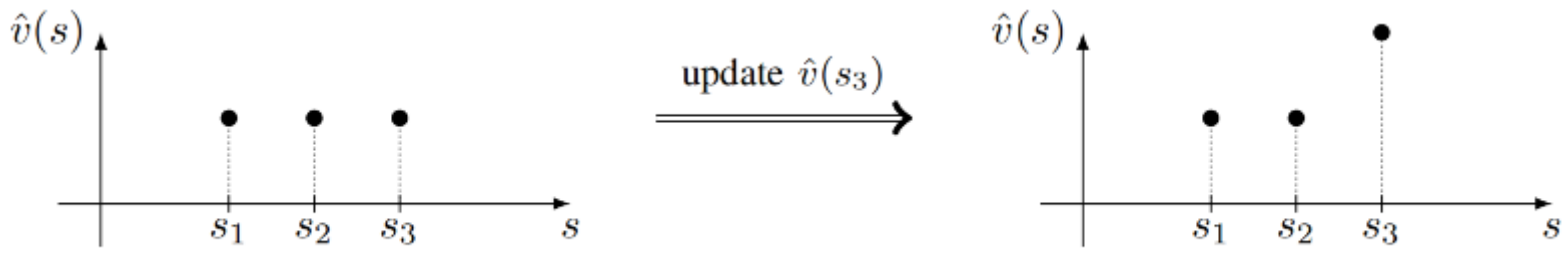
由于状态价值的检索方式，函数逼近方法在存储方面更加高效。具体来说，虽然表格方法需要存储 n 个值，但我们现在只需要存储一个低维参数向量 w。因此，存储效率可以显著提高。然而，这种好处*不是*免费的。它伴随着代价：**状态价值可能无法被函数准确地表示**。例如，一条直线无法准确拟合图 8.2 中的点。这就是为什么这种方法被称为逼近 (approximation)。从根本上说，当我们使用低维向量来表示高维数据集时，肯定会丢失一些信息。因此，**函数逼近方法通过牺牲精度来提高存储效率**。

- **如何 *更新 (update)* 一个值：** 当数值由表格表示时，如果我们想更新一个值，我们可以直接重写表格中的相应条目。然而，当数值由函数表示时，更新数值的方式完全不同。具体来说，我们必须更新 w 来间接地改变数值。如何更新 w 以找到最优状态价值将在后面详细讨论。

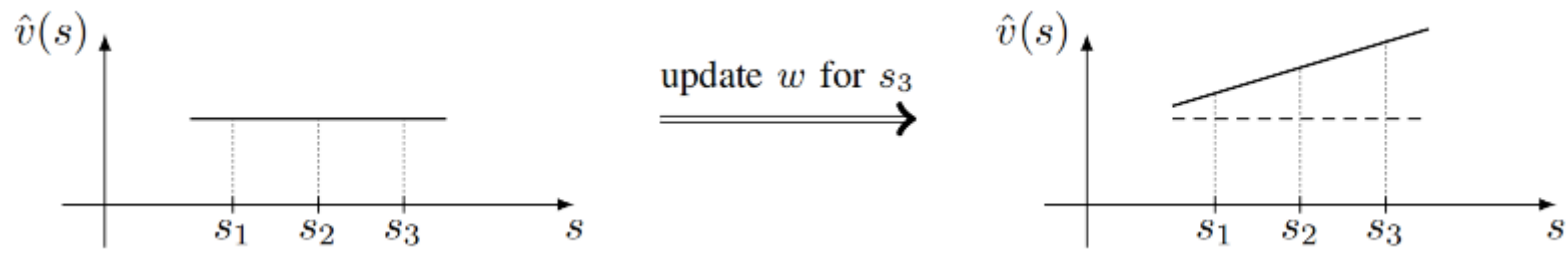
由于状态价值的更新方式，函数逼近方法还有另一个优点：它的 *泛化能力 (generalization ability)* 比表格方法更强。原因如下。当使用表格方法时，如果相应的状态在一个回合中被访问，我们可以更新该值。未被访问的状态的值无法更新。**然而，当使用函数逼近方法时，我们需要更新 w 来更新某个状态的值。w 的更新也会影响其他一些状态的值，即使这些状态没有被访问过。因此，一个状态的经验样本可以泛化以帮助估计其他一些状态的值。**

上述分析如图 8.4 所示，其中有三个状态 $\{s_1, s_2, s_3\}$。

假设我们有一个针对 s_3 的经验样本，并且想要更新 $\hat{v}(s_3)$。当使用表格方法时，我们只能更新 $\hat{v}(s_3)$ 而不会改变 $\hat{v}(s_1)$ 或 $\hat{v}(s_2)$，如图 8.4(a) 所示。当使用函数逼近方法时，更新 w 不仅可以更新 $\hat{v}(s_3)$，还会改变 $\hat{v}(s_1)$ 和 $\hat{v}(s_2)$，如图 8.4(b) 所示。因此，s_3 的经验样本可以帮助更新其邻近状态的价值。



(a) 表格方法： 当 $\hat{v}(s_3)$ 更新时，其他值保持不变。



(b) 函数逼近方法： 当我们通过改变 w 来更新 $\hat{v}(s_3)$ 时，邻近状态的价值也会改变。

图 8.4：关于如何更新状态价值的图示。

我们可以使用比直线具有更强逼近能力的更复杂的函数。例如，考虑一个二阶多项式：

$$\hat{v}(s, w) = as^2 + bs + c = \underbrace{[s^2, s, 1]}_{\phi^T(s)} \underbrace{\begin{bmatrix} a \\ b \\ c \end{bmatrix}}_{w} = \phi^T(s)w. \quad (8.2)$$

我们甚至可以使用更高阶的多项式曲线来拟合这些点。随着曲线阶数的增加，逼近精度可以提高，但参数向量的维度也会增加，从而需要更多的存储和计算资源。

注意，无论是 (8.1) 还是 (8.2) 中的 $\hat{v}(s, w)$ 关于 w 都是 *线性 (linear)* 的（尽管它关于 s 可能是非线性的）。这种类型的方法被称为 *线性函数逼近 (linear function approximation)*，它是最简单的函数逼近方法。为了实现线性函数逼近，我们需要选择一个合适的特征向量 $\phi(s)$。也就是说，我们必须决定，例如，是应该使用一阶直线还是二阶曲线来拟合这些点。**选择合适的特征向量并非易事。它需要关于给定任务的先验知识：我们对任务理解得越好，就能选择出越好的特征向量。**例如，如果我们知道图 8.2 中的点大致位于一个直线，我们可以使用一条直线来拟合这些点。然而，这种先验知识在实践中通常是未知的。如果我们没有任何先验知识，一个流行的解决方案是使用人工神经网络作为非线性函数逼近器。

另一个重要的问题是如何找到最优参数向量。如果我们知道 $\{v_\pi(s_i)\}_{i=1}^n$，这就变成了一个最小二乘问题 (least-squares problem)。最优参数可以通过优化以下目标函数获得：

$$\begin{aligned} J_1 &= \sum_{i=1}^{n} (\hat{v}(s_i, w) - v_\pi(s_i))^2 = \sum_{i=1}^{n} (\phi^T(s_i)w - v_\pi(s_i))^2 \\ &= \left\| \begin{bmatrix} \phi^T(s_1) \\ \vdots \\ \phi^T(s_n) \end{bmatrix} w - \begin{bmatrix} v_\pi(s_1) \\ \vdots \\ v_\pi(s_n) \end{bmatrix} \right\|^2 \doteq \|\Phi w - v_\pi\|^2, \end{aligned}$$

其中

$$\Phi \doteq \begin{bmatrix} \phi^T(s_1) \\ \vdots \\ \phi^T(s_n) \end{bmatrix} \in \mathbb{R}^{n \times 2}, \quad v_\pi \doteq \begin{bmatrix} v_\pi(s_1) \\ \vdots \\ v_\pi(s_n) \end{bmatrix} \in \mathbb{R}^n.$$

可以验证，该最小二乘问题的最优解为

$$w^* = (\Phi^T \Phi)^{-1} \Phi v_\pi.$$

关于最小二乘问题的更多信息可以在 [47, Section 3.3] 和 [48, Section 5.14] 中找到。

本节介绍的曲线拟合示例阐述了价值函数逼近的基本思想。这个思想将在下一节中被正式介绍。

8.2 基于函数逼近的状态价值 TD 学习 (TD learning of state values based on function approximation)

在本节中，我们将展示如何将**函数逼近方法**整合到 TD 学习中，以估计给定策略的状态价值。该算法将在 8.3 节中被扩展以学习动作价值和最优策略。

本节包含相当多的小节和许多连贯的内容。在深入细节之前，我们最好先预览一下这些内容。

- 函数逼近方法被公式化为一个 *优化问题 (optimization problem)*。该问题的 *目标函数 (objective function)*在 8.2.1 节中介绍。用于优化该目标函数的 TD 学习算法在 8.2.2 节中介绍。
- **为了应用 TD 学习算法，我们需要选择合适的特征向量 (feature vectors)。** 8.2.3 节讨论了这个问题。
- **8.2.4 节给出了示例来演示 TD 算法以及不同特征向量的影响。**
- **8.2.5 节给出了 TD 算法的理论分析。** 该小节的数学性较强。读者可以根据自己的兴趣选择性阅读。

8.2.1 目标函数 (Objective function)

令 $v_\pi(s)$ 和 $\hat{v}(s,w)$ 分别为 $s \in \mathcal{S}$ 的**真实状态价值和近似状态价值**。待求解的问题是找到一个 *最优* w，使得 $\hat{v}(s,w)$ 对于每个 s 都能最好地逼近 $v_\pi(s)$。具体来说，目标函数为

(更直接的说，我们试图找到一个函数来近似真实状态价值函数, 这个过程就是Policy Evaluation)

$$J(w) = \mathbb{E}[(v_\pi(S) - \hat{v}(S,w))^2], \quad (8.3)$$

其中期望是关于随机变量 $S \in \mathcal{S}$ 计算的。虽然 S 是一个随机变量，但它的概率分布是什么？这个问题对于理解该目标函数至关重要。定义 S 的概率分布有几种方式。

- **第一种方式是使用 *均匀分布* (uniform distribution)。** 即通过将每个状态的概率设为 $1/n$，将所有状态视为 *同等重要* **(equally important)**。在这种情况下，(8.3) 中的目标函数变为

$$J(w) = \frac{1}{n}\sum_{s\in\mathcal{S}}(v_\pi(s) - \hat{v}(s,w))^2, \quad (8.4)$$

即所有状态的逼近误差的平均值。**然而，这种方式没有考虑给定策略下马尔可夫过程的真实动态。由于某些状态可能很少被策略访问，因此将所有状态视为同等重要可能是不合理的。**

- **第二种方式，也是本章的重点，是使用 *平稳分布* (stationary distribution)。** 平稳分布描述了马尔可夫决策过程的 *长期* **(long-term)** 行为。**更具体地说，在智能体执行给定策略足够长的时间后，智能体位于任何状态的概率都可以由该平稳分布来描述。**感兴趣的读者可以查看方框 8.1 中的详细信息。
- 令 $\{d_\pi(s)\}_{s\in\mathcal{S}}$ 表示策略 π 下马尔可夫过程的平稳分布。即，智能体在很长一段时间后访问 s 的概率为 $d_\pi(s)$。根据定义，$\sum_{s\in\mathcal{S}} d_\pi(s) = 1$。那么，(8.3) 中的目标函数可以重写为:

$$J(w) = \sum_{s\in\mathcal{S}} d_\pi(s)(v_\pi(s) - \hat{v}(s,w))^2, \quad (8.5)$$

这是逼近误差的加权平均值。被访问概率较高的状态被赋予更大的权重。

值得注意的是，$d_\pi(s)$ 的值很难获得，因为它需要知道状态转移概率矩阵 P_π （见方框 8.1）。**幸运的是，正如我们在下一小节中所展示的，我们并不需要计算 $d_\pi(s)$ 的具体值来最小化该目标函数。**此外，我们在介绍 (8.4) 和 (8.5) 时假设状态数量是 ***有限的 (finite)***。当状态空间是连续时，我们可以用积分代替求和。

✏️ 方框 8.1：马尔可夫决策过程的平稳分布 (Stationary distribution of a Markov decision process)

分析平稳分布的关键工具是 $P_\pi \in \mathbb{R}^{n\times n}$，即给定策略 π 下的概率转移矩阵。如果状态索引为 $s_1, \ldots, s_n$，则 $[P_\pi]_{ij}$ 定义为智能体从 s_i 移动到 s_j 的概率。P_π 的定义可以在 2.6 节中找到。

- **$P_\pi^k (k = 1, 2, 3, \ldots)$ 的解释 (Interpretation of P_π^k)。**

  首先，有必要考察 P_π^k 的解释。智能体使用确切的 k 步从 s_i 转移到 s_j 的概率记为

$$p_{ij}^{(k)} = \Pr(S_{t_k} = j | S_{t_0} = i),$$

其中 t_0 和 t_k 分别是初始时刻和第 k 个时刻。根据 P_π 的定义，我们有

$$[P_\pi]_{ij} = p_{ij}^{(1)},$$

这意味着 $[P_\pi]_{ij}$ 是使用 *单步 (single step)* 从 s_i 转移到 s_j 的概率。其次，考虑 P_π^2。可以验证

$$[P_\pi^2]_{ij} = [P_\pi P_\pi]_{ij} = \sum_{q=1}^{n} [P_\pi]_{iq} [P_\pi]_{qj}.$$

由于 $[P_\pi]_{iq}[P_\pi]_{qj}$ 是从 s_i 转移到 s_q 然后从 s_q 转移到 s_j 的联合概率，我们知道 $[P_\pi^2]_{ij}$ 是从 s_i 转移到 s_j 的概率, 使用 *恰好两步 (exactly two steps)*。即

$$[P_\pi^2]_{ij} = p_{ij}^{(2)}.$$

类似地，我们知道

$$[P_\pi^k]_{ij} = p_{ij}^{(k)},$$

这意味着 $[P_\pi^k]_{ij}$ 是使用恰好 k 步从 s_i 转移到 s_j 的概率。

- **平稳分布的定义 (Definition of stationary distributions)。**

令 $d_0 \in \mathbb{R}^n$ 为表示**初始时间步状态概率分布的向量**。例如，如果 s 总是被选为起始状态，则 $d_0(s) = 1$ 而 d_0 的其他项为 0。令 $d_k \in \mathbb{R}^n$ 为从 d_0 开始恰好 k 步后获得的概率分布向量。那么，我们有

$$d_k(s_i) = \sum_{j=1}^{n} d_0(s_j)[P_\pi^k]_{ji}, \quad i = 1, 2, \dots \quad (8.6)$$

该方程表明，**智能体在第 k 步访问 s_i 的概率**等于智能体使用恰好 k 步从 $\{s_j\}_{j=1}^n$ 转移到 s_i 的概率之和。(8.6) 的矩阵-向量形式为

$$d_k^T = d_0^T P_\pi^k. \quad (8.7)$$

当我们考虑马尔可夫过程的 **长期 (long-term)** 行为时，在某些条件下成立

$$\lim_{k\to\infty} P_\pi^k = \mathbf{1}_n d_\pi^T, \quad (8.8)$$

其中 $\mathbf{1}_n = [1, \dots, 1]^T \in \mathbb{R}^n$，$\mathbf{1}_n d_\pi^T$ 是一个常数矩阵，其所有行都等于 d_π^T。(8.8) 成立的条件将在稍后讨论。将 (8.8) 代入 (8.7) 可得

$$\lim_{k\to\infty} d_k^T = d_0^T \lim_{k\to\infty} P_\pi^k = d_0^T \mathbf{1}_n d_\pi^T = d_\pi^T, \quad (8.9)$$

其中最后一个等式成立是因为 $d_0^T \mathbf{1}_n = 1$。

方程 (8.9) 意味着状态分布 d_k 收敛于一个常数值 d_π，这被称为 *极限分布 (limiting distribution)*。极限分布取决于系统模型和策略 π。有趣的是，它独立于初始分布 d_0。也就是说，**无论智能体从哪个状态开始，经过足够长的时间后，智能体的概率分布总是可以用极限分布来描述。**

d_π 的值可以通过以下方式计算。对 $d_k^T = d_{k-1}^T P_\pi$ 两边取极限（单步转移的递推式），得到 $\lim_{k\to\infty} d_k^T = \lim_{k\to\infty} d_{k-1}^T P_\pi$，因此

$$d_\pi^T = d_\pi^T P_\pi. \quad (8.10)$$

结果是，d_π 是 P_π 对应于特征值 1 的左特征向量。(8.10) 的解被称为平稳分布。它满足 $\sum_{s\in\mathcal{S}} d_\pi(s) = 1$ 且对于所有 $s \in \mathcal{S}$，$d_\pi(s) > 0$。为什么 $d_\pi(s) > 0$（而不是 $d_\pi(s) \geq 0$）的原因将在后面解释。

- **平稳分布唯一性的条件 (Conditions for the uniqueness of stationary distributions)。**

(8.10) 的解 d_π 通常被称为 *平稳分布 (stationary distribution)*，而 (8.9) 中的分布 d_π 通常被称为 *极限分布 (limiting distribution)*。注意 (8.9) 蕴含了 (8.10)，但反之未必成立。**具有唯一平稳（或极限）分布的一类通用马尔可夫过程是 *不可约 (irreducible)*（或 *正则 (regular)*）马尔可夫过程。**下面给出一些必要的定义。更多细节可以在 [49, Chapter IV] 中找到。

如果存在一个有限整数 k 使得 $[P_\pi^k]_{ij} > 0$，则称状态 s_j 从状态 s_i 是 *可达的 (accessible)*，这意味着从 s_i 开始的智能体可以在有限次转移后到达 s_j。

- 如果两个状态 s_i 和 s_j 是相互可达的，则称这两个状态 **连通 (communicate)**。
- 如果一个马尔可夫过程的所有状态都互相连通，则称该过程为 ***不可约的 (irreducible)***。换句话说，从任意状态开始的智能体都可以在有限步数内到达任何其他状态。数学上，这表明对于任意 s_i 和 s_j，存在 $k \geq 1$ 使得 $[P_\pi^k]_{ij} > 0$（k 的值可能因 i, j 而异）。
- **如果存在 $k \geq 1$ 使得对于所有 i, j 都有 $[P_\pi^k]_{ij} > 0$，则称马尔可夫过程为 *正则的 (regular)***。等价地，存在 $k \geq 1$ 使得 $P_\pi^k > 0$，其中 $>$ 是按元素比较的。结果是，任何状态都有可能在最多 k 步内从任何其他状态到达。**正则马尔可夫过程也是不可约的，但反之不成立。**然而，如果一个马尔可夫过程是不可约的且存在 i 使得 $[P_\pi]_{ii} > 0$，那么它也是正则的。此外，如果 $P_\pi^k > 0$，那么对于任何 $k' \geq k$，都有 $P_\pi^{k'} > 0$，因为 $P_\pi \geq 0$。由此根据 (8.9) 可知，对于每个 s，都有 $d_\pi(s) > 0$。

- **可能导致唯一平稳分布的策略 (Policies that may lead to unique stationary distributions)。**

一旦给定策略，马尔可夫决策过程就变成了马尔可夫过程，其长期行为由给定的策略和系统模型共同决定。那么，一个重要的问题是什么样的策略可以导致正则马尔可夫过程？通常，答案是 *探索性策略 (exploratory policies)*，例如 ϵ -贪婪策略，可以带来正则的马尔可夫过程。这是因为探索性策略在任何状态下采取任何动作的概率都为正。结果是，当系统模型允许时，状态之间可以互相连通。

- **图 8.5 给出了一个示例来说明平稳分布。** 本例中的策略是 $\epsilon = 0.5$ 的 ϵ -贪婪策略。状态索引为 s_1, s_2, s_3, s_4，分别对应网格中的左上、右上、左下和右下单元格。
- 我们比较两种计算平稳分布的方法。第一种方法是求解 (8.10) 以获得 d_π 的理论值。第二种方法是数值估计 d_π：我们从任意初始状态开始，通过遵循给定策略生成足够长的回合。然后，d_π 可以通过该回合中访问每个状态的次数与回合总长度之比来估计。回合越长，估计结果越准确。接下来我们比较理论结果和估计结果。

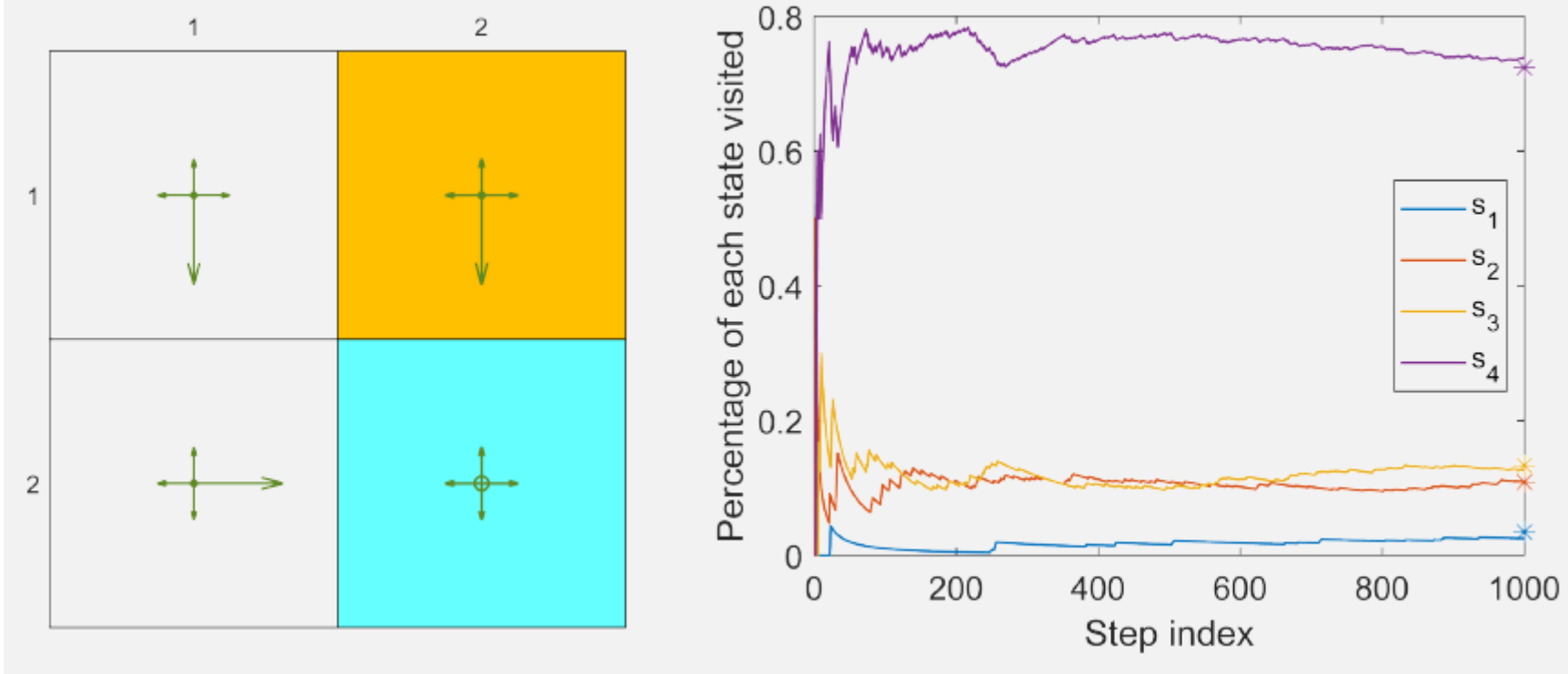


图 8.5： $\epsilon = 0.5$ **的** ϵ **-贪婪策略的长期行为。** 右图中的星号代表 d_π 元素的理论值。

- d_π **的理论值 (**$Theoretical value of d_\pi$**)：** 可以验证，由该策略诱导的马尔可夫过程既是不可约的又是正则的。这是由于以下原因。首先，由于所有状态都相互连通，因此所得的马尔可夫过程是不可约的。其次，由于每个状态都可以转移到自身，所得的...

马尔可夫过程是正则的。从图 8.5 中可以看出

$$P_\pi^T = \begin{bmatrix} 0.3 & 0.1 & 0.1 & 0 \\ 0.1 & 0.3 & 0 & 0.1 \\ 0.6 & 0 & 0.3 & 0.1 \\ 0 & 0.6 & 0.6 & 0.8 \end{bmatrix}.$$

P_π^T 的特征值计算结果为 $\{-0.0449, 0.3, 0.4449, 1\}$。$P_\pi^T$ 对应于特征值 1 的单位长度（右）特征向量为 $[0.0463, 0.1455, 0.1785, 0.9720]^T$。对该向量进行缩放，使其所有元素之和等于 1，我们得到 d_π 的理论值如下：

$$d_\pi = \begin{bmatrix} 0.0345 \\ 0.1084 \\ 0.1330 \\ 0.7241 \end{bmatrix}.$$

d_π 的第 i 个元素对应于长期运行中智能体访问 s_i 的概率。

- **d_π 的估计值 (Estimated value of d_π)：** 接下来我们通过在仿真中执行该策略足够多的步数来验证 d_π 的理论值。具体来说，我们选择 s_1 作为起始状态，并遵循该策略运行 1,000 步。过程中每个状态的访问比例在图 8.5 中给出。可以看出，经过数百步后，这些比例收敛到了 d_π 的理论值。

8.2.2 优化算法 (Optimization algorithms)

为了最小化 (8.3) 中的目标函数 $J(w)$，我们可以使用梯度下降算法：

$$w_{k+1} = w_k - \alpha_k \nabla_w J(w_k),$$

其中

$$\begin{aligned} \nabla_w J(w_k) &= \nabla_w \mathbb{E}[(v_\pi(S) - \hat{v}(S, w_k))^2] \\ &= \mathbb{E}[\nabla_w (v_\pi(S) - \hat{v}(S, w_k))^2] \\ &= 2\mathbb{E}[(v_\pi(S) - \hat{v}(S, w_k))(-\nabla_w \hat{v}(S, w_k))] \\ &= -2\mathbb{E}[(v_\pi(S) - \hat{v}(S, w_k))\nabla_w \hat{v}(S, w_k)]. \end{aligned}$$

因此，梯度下降算法为

$$w_{k+1} = w_k + 2\alpha_k \mathbb{E}[(v_\pi(S) - \hat{v}(S, w_k))\nabla_w \hat{v}(S, w_k)], \quad (8.11)$$

其中 α_k 前面的系数 2 可以合并到 α_k 中而不失一般性。(8.11) 中的算法需要计算期望。本着随机梯度下降 (stochastic gradient descent) 的精神，我们可以用随机梯度代替真实梯度。那么，(8.11) 变为

$$w_{t+1} = w_t + \alpha_t (v_\pi(s_t) - \hat{v}(s_t, w_t))\nabla_w \hat{v}(s_t, w_t), \quad (8.12)$$

其中 s_t 是时刻 t 的 S 的一个样本。

值得注意的是，(8.12) 是不可实现的，因为它需要真实状态价值 v_π，而该值是未知的且必须被估计。我们可以用一个近似值来代替 $v_\pi(s_t)$ 以使算法可实现。可以使用以下两种方法来实现这一点。

- **蒙特卡洛方法 (Monte Carlo method)：** 假设我们有一个回合 $(s_0, r_1, s_1, r_2, \dots)$。**令 g_t 为从 s_t 开始的折扣回报。那么，g_t 可以用作 $v_\pi(s_t)$ 的近似值。**(8.12) 中的算法变为

$$w_{t+1} = w_t + \alpha_t (g_t - \hat{v}(s_t, w_t))\nabla_w \hat{v}(s_t, w_t).$$

  这就是带有函数逼近的蒙特卡洛学习算法。

- **时序差分方法 (Temporal-difference method)：** 本着 TD 学习的精神，$r_{t+1} + \gamma \hat{v}(s_{t+1}, w_t)$ 可以用作 $v_\pi(s_t)$ 的近似值。(8.12) 中的算法变为

$$w_{t+1} = w_t + \alpha_t [r_{t+1} + \gamma \hat{v}(s_{t+1}, w_t) - \hat{v}(s_t, w_t)]\nabla_w \hat{v}(s_t, w_t). \quad (8.13)$$

这就是带有函数逼近的 TD 学习算法。该算法总结在算法 8.1 中。

理解 (8.13) 中的 TD 算法对于学习本章中的其他算法非常重要。值得注意的是，(8.13) 只能学习给定策略的 *状态价值*。它将在 8.3.1 和 8.3.2 节中扩展为可以学习 *动作价值* 的算法。

8.2.3 函数逼近器的选择 (Selection of function approximators)

为了应用 (8.13) 中的 TD 算法，我们需要选择合适的 $\hat{v}(s, w)$。有两种方法可以做到这一点。第一种是使用人工神经网络作为 *非线性函数* 逼近器。神经网络的输入是状态，输出是 $\hat{v}(s, w)$，网络参数是 w。第二种是简单地使用 *线性函数*：

$$\hat{v}(s, w) = \phi^T(s) w,$$

算法 8.1：带有函数逼近的状态价值 TD 学习 (TD learning of state values with function approximation)

初始化： 一个关于 w 可微的函数 $\hat{v}(s, w)$。初始参数 w_0。

目标： 学习给定策略 π 的真实状态价值。

对于 由 π 生成的每个回合 $\{(s_t, r_{t+1}, s_{t+1})\}_t$，做

对于 每个样本 (s_t, r_{t+1}, s_{t+1})，做

在一般情况下，$w_{t+1} = w_t + \alpha_t[r_{t+1} + \gamma \hat{v}(s_{t+1}, w_t) - \hat{v}(s_t, w_t)]\nabla_w \hat{v}(s_t, w_t)$

在线性情况下，$w_{t+1} = w_t + \alpha_t[r_{t+1} + \gamma \phi^T(s_{t+1})w_t - \phi^T(s_t)w_t]\phi(s_t)$

其中 $\phi(s) \in \mathbb{R}^m$ 是 s 的特征向量。$\phi(s)$ 和 w 的长度等于 m，这通常远小于状态的数量。在线性情况下，梯度为

$$\nabla_w \hat{v}(s, w) = \phi(s),$$

将其代入 (8.13) 可得

$$w_{t+1} = w_t + \alpha_t[r_{t+1} + \gamma \phi^T(s_{t+1})w_t - \phi^T(s_t)w_t]\phi(s_t). \quad (8.14)$$

这就是带有线性函数逼近的 TD 学习算法。我们将它简称为 *TD-Linear*。

线性情况在理论上比非线性情况更容易理解。然而，它的逼近能力是有限的。为复杂的任务选择合适的特征向量也绝非易事。相比之下，人工神经网络作为黑盒通用非线性逼近器，可以逼近价值，使用起来更加友好。

尽管如此，研究线性情况仍然是有意义的。更好地理解线性情况可以帮助读者更好地掌握函数逼近方法的思想。此外，线性情况足以解决本书中考虑的简单网格世界任务。更重要的是，线性情况依然很强大，因为表格方法可以被视为一种特殊的线性情况。更多信息可以在方框 8.2 中找到。

方框 8.2：表格型 TD 学习是 TD-Linear 的一个特例 (Tabular TD learning is a special case of TD-Linear)

接下来我们展示第 7 章中的表格型 TD 算法 (7.1) 是 TD-Linear 算法 (8.14) 的一个特例。

考虑以下针对任意 $s \in \mathcal{S}$ 的特殊特征向量：

$$\phi(s) = e_s \in \mathbb{R}^n,$$

其中 e_s 是一个向量，其对应于 s 的项等于 1，而其他项等于 0。在这种情况下，

$$\hat{v}(s, w) = e_s^T w = w(s),$$

其中 $w(s)$ 是 w 中对应于 s 的元素。将上述方程代入 (8.14) 可得

$$w_{t+1} = w_t + \alpha_t(r_{t+1} + \gamma w_t(s_{t+1}) - w_t(s_t))e_{s_t}.$$

由于 e_{s_t} 的定义，上述方程仅更新元素 $w_t(s_t)$。受此启发，在方程两边同乘 $e_{s_t}^T$ 可得

$$w_{t+1}(s_t) = w_t(s_t) + \alpha_t(r_{t+1} + \gamma w_t(s_{t+1}) - w_t(s_t)),$$

这正是 (7.1) 中的表格型 TD 算法。

总之，通过选择特征向量 $\phi(s) = e_s$，TD-Linear 算法就变成了表格型 TD 算法。

8.2.4 演示示例 (Illustrative examples)

接下来我们展示一些示例，以演示如何使用 (8.14) 中的 TD-Linear 算法来估计给定策略的状态价值。同时，我们演示如何选择特征向量。

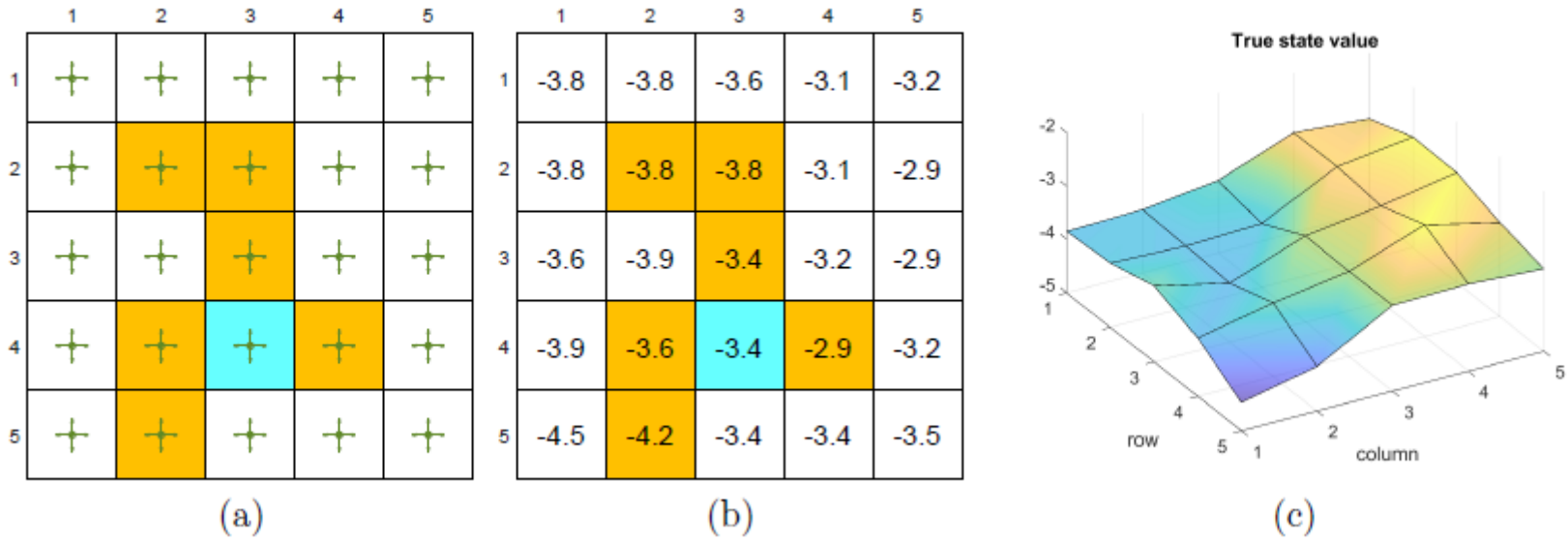


图 8.6：(a) 待评估的策略。(b) 真实状态价值以表格形式表示。(c) 真实状态价值以三维曲面形式表示。

网格世界示例如图 8.6 所示。给定的策略在某个状态下以 0.2 的概率采取任何动作。我们的目标是估计该策略下的状态价值。总共有 25 个状态价值。真实状态价值如图 8.6(b) 所示。真实状态价值在图 8.6(c) 中被可视化为一个三维曲面。

接下来我们展示可以使用少于 25 个参数来逼近这些状态价值。仿真设置如下。由给定策略生成 500 个回合。每个回合包含 500 步，并从一个遵循均匀分布随机选择的状态-动作对开始。此外，在每次仿真试验中，参数向量 w 都是随机初始化的，使得每个元素都取自均值为 0、标准差为 1 的标准正态分布。我们设定 $r_{\text{forbidden}} = r_{\text{boundary}} = -1$，$r_{\text{target}} = 1$，且 $\gamma = 0.9$。

为了实现 TD-Linear 算法，我们需要首先选择特征向量 $\phi(s)$。如下所示，有不同的做法。

- **第一类特征向量是基于多项式的 (polynomials)。** 在网格世界示例中，状态 s 对应于一个 2D 位置。令 x 和 y 分别表示 s 的列索引和行索引。为了避免数值问题，我们将 x 和 y 归一化，使它们的值位于区间 $[-1, +1]$ 内。这里稍微滥用一下符号，归一化后的值也用 x 和 y 表示。那么，最简单的特征向量是

$$\phi(s) = \begin{bmatrix} x \\ y \end{bmatrix} \in \mathbb{R}^2.$$

在这种情况下，我们有

$$\hat{v}(s, w) = \phi^T(s) w = [x, y] \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = w_1 x + w_2 y.$$

当 w 给定时，$\hat{v}(s, w) = w_1 x + w_2 y$ 表示一个通过原点的 2D 平面。由于状态价值的曲面可能不经过原点，我们需要向 2D 平面引入一个 *偏置 (bias)* 以更好地逼近状态价值。为此，我们考虑以下 3D 特征向量：

$$\phi(s) = \begin{bmatrix} 1 \\ x \\ y \end{bmatrix} \in \mathbb{R}^3. \quad (8.15)$$

在这种情况下，近似的状态价值为

$$\hat{v}(s, w) = \phi^T(s) w = [1, x, y] \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = w_1 + w_2 x + w_3 y.$$

当 w 给定时，$\hat{v}(s, w)$ 对应于一个可能不经过原点的平面。值得注意的是，$\phi(s)$ 也可以定义为 $\phi(s) = [x, y, 1]^T$，元素的顺序并不重要。

当我们使用 (8.15) 中的特征向量时的估计结果展示在图 8.7(a)。可以看出，估计的状态价值形成了一个 2D 平面。虽然随着使用更多回合，估计误差会收敛，但由于 2D 平面的逼近能力有限，误差无法减小到零。

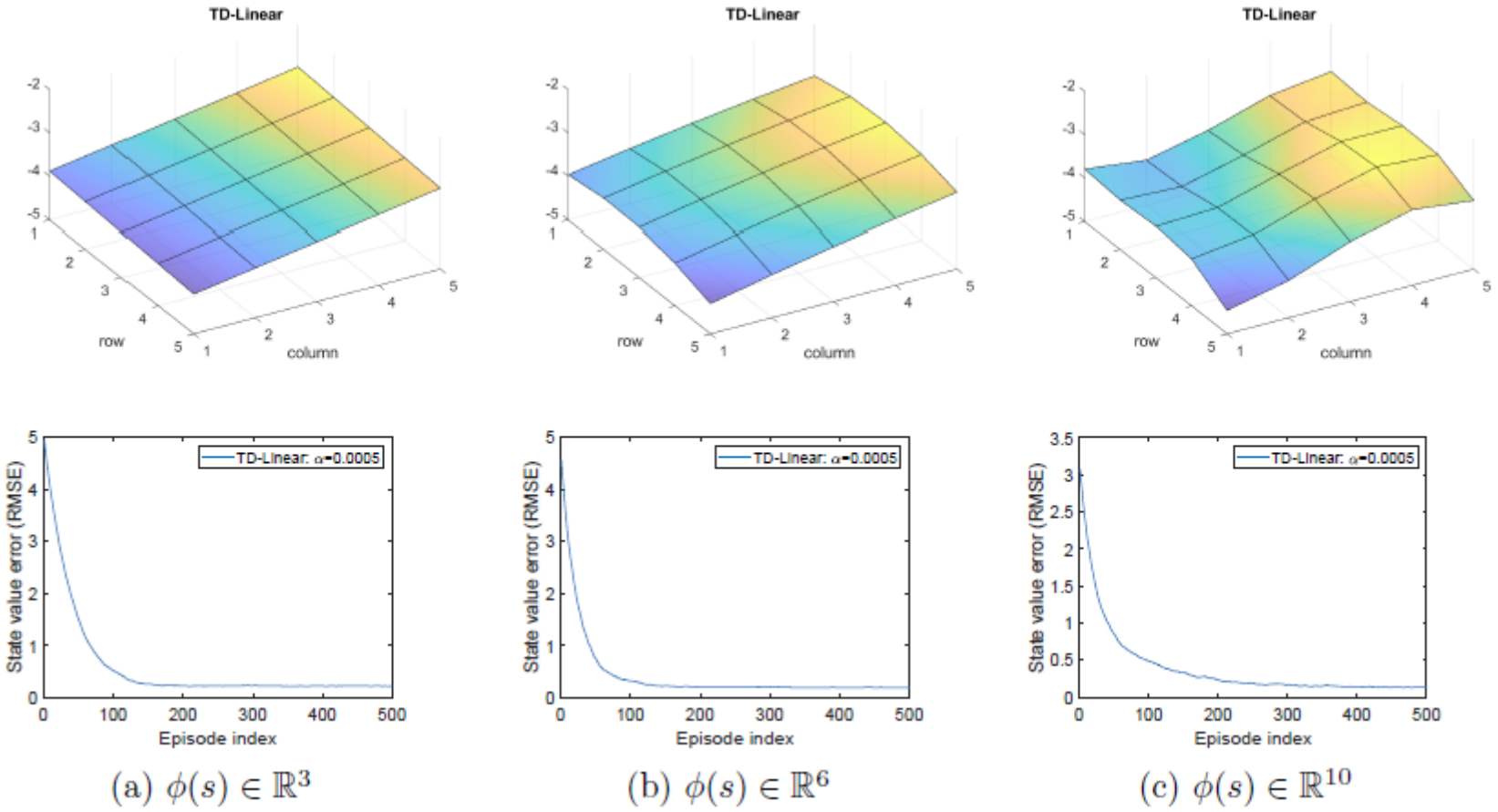
为了增强逼近能力，我们可以增加特征向量的维数。为此，考虑

$$\phi(s) = [1, x, y, x^2, y^2, xy]^T \in \mathbb{R}^6. \quad (8.16)$$

在这种情况下，$\hat{v}(s, w) = \phi^T(s) w = w_1 + w_2 x + w_3 y + w_4 x^2 + w_5 y^2 + w_6 xy$，它对应于一个二次 3D 曲面。我们可以进一步增加特征向量的维数：

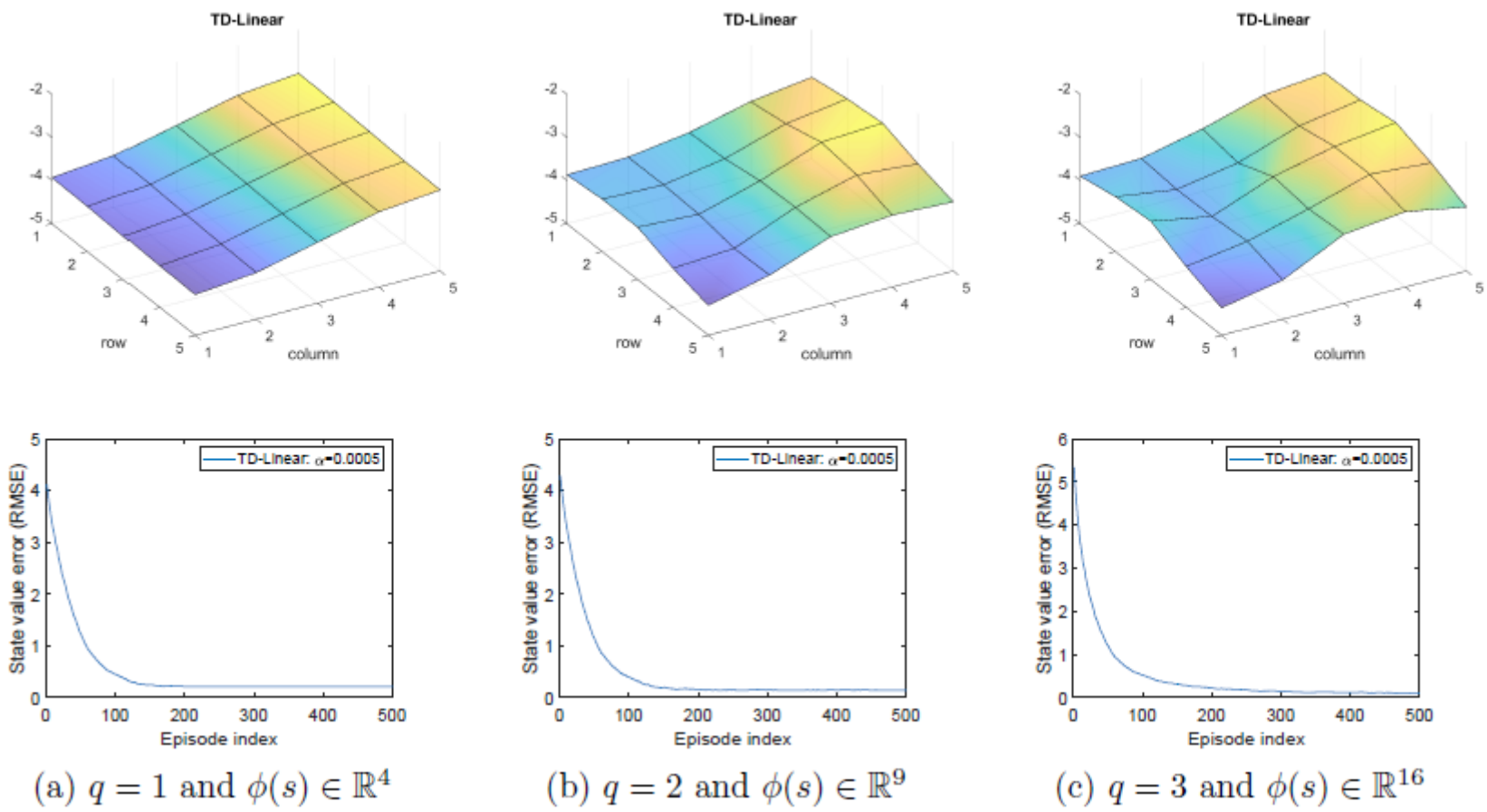
$$\phi(s) = [1, x, y, x^2, y^2, xy, x^3, y^3, x^2 y, xy^2]^T \in \mathbb{R}^{10}. \quad (8.17)$$

当我们使用 (8.16) 和 (8.17) 中的特征向量时的估计结果分别显示在图 8.7(b)-(c) 中。可以看出，特征向量越长，状态价值的逼近就越准确。然而，在这三种情况下，估计误差都无法收敛到零，因为这些线性逼近器的逼近能力仍然是有限的。



(a) $\phi(s) \in \mathbb{R}^3$ (b) $\phi(s) \in \mathbb{R}^6$ (c) $\phi(s) \in \mathbb{R}^{10}$

图 8.7：使用 (8.15)、(8.16) 和 (8.17) 中的多项式特征获得的 TD-Linear 估计结果。



(a) $q = 1$ and $\phi(s) \in \mathbb{R}^4$ (b) $q = 2$ and $\phi(s) \in \mathbb{R}^9$ (c) $q = 3$ and $\phi(s) \in \mathbb{R}^{16}$

图 8.8：使用 (8.18) 中的傅里叶特征获得的 TD-Linear 估计结果。 (a) $q = 1$ 且 $\phi(s) \in \mathbb{R}^4$。(b) $q = 2$ 且 $\phi(s) \in \mathbb{R}^9$。(c) $q = 3$ 且 $\phi(s) \in \mathbb{R}^{16}$。

除了多项式特征向量外，还有许多其他类型的特征可用，例如傅里叶基 (Fourier basis) 和瓦片编码 (tile coding) [3, Chapter 9]。

首先，x 和 y 的值每个状态都被归一化到区间 $[0, 1]$ 内。所得的特征向量为

$$\phi(s) = \begin{bmatrix} \vdots \\ \cos(\pi(c_1 x + c_2 y)) \\ \vdots \end{bmatrix} \in \mathbb{R}^{(q+1)^2}, \quad (8.18)$$

其中 π 表示圆周率 3.1415...，而不是策略。这里，c_1 或 c_2 可以被设为 $\{0,1,\dots,q\}$ 中的任意整数，其中 q 是用户指定的整数。结果是，数对 (c_1,c_2) 有 $(q+1)^2$ 种可能的取值。因此，$\phi(s)$ 的维数是 $(q+1)^2$。例如，在 $q=1$ 的情况下，特征向量为

$$\phi(s)=\begin{bmatrix}\cos(\pi(0x+0y))\\ \cos(\pi(0x+1y))\\ \cos(\pi(1x+0y))\\ \cos(\pi(1x+1y))\end{bmatrix}=\begin{bmatrix}1\\ \cos(\pi y)\\ \cos(\pi x)\\ \cos(\pi(x+y))\end{bmatrix}\in\mathbb{R}^4.$$

当我们使用 $q=1,2,3$ 的傅里叶特征时获得的估计结果如图 8.8 所示。在这三种情况下，特征向量的维数分别为 4、9 和 16。可以看出，特征向量的维数越高，状态价值的逼近就越准确。

8.2.5 理论分析 (Theoretical analysis)

到目前为止，我们已经完成了带有函数逼近的 TD 学习的描述。这个故事始于 (8.3) 中的目标函数。为了优化这个目标函数，我们引入了 (8.12) 中的随机算法。后来，算法中未知的真实价值函数被一个近似值所替代，从而导致了 (8.13) 中的 TD 算法。虽然这个故事有助于理解价值函数逼近的基本思想，但它在数学上并不严谨。例如，(8.13) 中的算法实际上并没有最小化 (8.3) 中的目标函数。

接下来我们给出 (8.13) 中 TD 算法的理论分析，以揭示为什么该算法有效以及它解决了什么数学问题。由于一般的非线性逼近器难以分析，这部分只考虑线性情况。建议读者根据自己的兴趣选择性阅读，因为这部分数学强度较大。

收敛性分析 (Convergence analysis)

为了研究 (8.13) 的收敛性质，我们首先考虑以下确定性算法：

$$w_{t+1}=w_t+\alpha_t\mathbb{E}[(r_{t+1}+\gamma\phi^T(s_{t+1})w_t-\phi^T(s_t)w_t)\phi(s_t)],\quad(8.19)$$

其中期望是关于随机变量 s_t,s_{t+1},r_{t+1} 计算的。假设 s_t 的分布是平稳分布 d_π。(8.19) 中的算法是确定性的，因为计算期望后随机变量 s_t,s_{t+1},r_{t+1} 都消失了。

我们为什么要考虑这个确定性算法？首先，这个确定性算法的收敛性更容易分析（尽管并非易事）。其次，更重要的是，这个确定性算法的收敛性蕴含了 (8.13) 中随机 TD 算法的收敛性。这是因为 (8.13) 可以被视为 (8.19) 的随机梯度下降 (SGD) 实现。因此，我们只需要研究确定性算法的收敛性质。

$$w_{t+1}=w_t+\alpha_t[r_{t+1}+\gamma\hat{v}(s_{t+1},w_t)-\hat{v}(s_t,w_t)]\nabla_w\hat{v}(s_t,w_t).\quad(8.13)$$

虽然 (8.19) 的表达式乍一看可能很复杂，但它可以被大大简化。为此，定义

$$\Phi=\begin{bmatrix}\vdots\\ \phi^T(s)\\ \vdots\end{bmatrix}\in\mathbb{R}^{n\times m},\quad D=\begin{bmatrix}\ddots & & \\ & d_\pi(s) & \\ & & \ddots\end{bmatrix}\in\mathbb{R}^{n\times n},\quad(8.20)$$

其中 Φ 是包含所有特征向量的矩阵，D 是其对角线元素为平稳分布的对角矩阵。这两个矩阵将被频繁使用。

引理 8.1. (8.19) 中的期望可以重写为

$$\mathbb{E}\left[\left(r_{t+1}+\gamma\phi^T(s_{t+1})w_t-\phi^T(s_t)w_t\right)\phi(s_t)\right]=b-Aw_t,$$

其中

$$A\doteq\Phi^TD(I-\gamma P_\pi)\Phi\in\mathbb{R}^{m\times m},$$
$$b\doteq\Phi^TDr_\pi\in\mathbb{R}^m.\quad(8.21)$$

这里，P_π,r_π 是贝尔曼方程 $v_\pi=r_\pi+\gamma P_\pi v_\pi$ 中的两项，而 I 是具有适当维度的单位矩阵。

证明过程在方框 8.3 中给出。利用引理 8.1 中的表达式，(8.19) 中的确定性算法可以重写为

$$w_{t+1}=w_t+\alpha_t(b-Aw_t),\quad(8.22)$$

这是一个简单的确定性过程。其收敛性分析如下。

首先，w_t 的收敛值是什么？假设当 $t\to\infty$ 时 w_t 收敛到一个常数值 w^*，那么 (8.22) 意味着 $w^*=w^*+\alpha_\infty(b-Aw^*)$，这表明 $b-Aw^*=0$，因此

$$w^*=A^{-1}b.$$

关于这个收敛值，有几点说明如下。

- A **是否可逆？** 答案是肯定的。事实上，A 不仅可逆，而且是正定 (positive definite) 的。也就是说，对于任何具有适当维度的非零向量 x，都有 $x^TAx>0$。证明在方框 8.4 中给出。

- $w^* = A^{-1}b$ ***的解释是什么？它实际上是最小化 投影贝尔曼误差 (projected Bellman error) 的最优解。详细信息将在 8.2.5 节后面介绍。***
- **表格方法是一个特例。** 一个有趣的结果是，当 w 的维数等于 $n = |\mathcal{S}|$ 且 $\phi(s) = [0, \dots, 1, \dots, 0]^T$ （其中对应于 s 的条目为 1）时，我们有

$$w^* = A^{-1}b = v_\pi. \quad (8.23)$$

- 该方程表明，待学习的参数向量实际上就是真实的状态价值。**这个结论与我们在方框 8.2 中介绍的“表格型 TD 算法是 TD-Linear 算法的一个特例”这一事实是一致的。**(8.23) 的证明如下。可以验证在这种情况下 $\Phi = I$，因此 $A = \Phi^T D(I - \gamma P_\pi)\Phi = D(I - \gamma P_\pi)$ 且 $b = \Phi^T D r_\pi = D r_\pi$。因此，

$$w^* = A^{-1}b = (I - \gamma P_\pi)^{-1} D^{-1} D r_\pi = (I - \gamma P_\pi)^{-1} r_\pi = v_\pi \text{。}$$

其次，我们证明当 $t \to \infty$ 时 (8.22) 中的 w_t 收敛到 $w^* = A^{-1}b$。由于 (8.22) 是一个简单的确定性过程，可以通过多种方式证明。我们给出以下两种证明。

- **证明 1：** 定义收敛误差为 $\delta_t \doteq w_t - w^*$。我们只需要证明 δ_t 收敛于零。为此，将 $w_t = \delta_t + w^*$ 代入 (8.22) 可得

$$\delta_{t+1} = \delta_t - \alpha_t A \delta_t = (I - \alpha_t A)\delta_t.$$

由此可得

$$\delta_{t+1} = (I - \alpha_t A) \cdots (I - \alpha_0 A)\delta_0.$$

考虑对于所有 t 都有 $\alpha_t = \alpha$ 的简单情况。那么，我们有

$$\|\delta_{t+1}\|_2 \le \|I - \alpha A\|_2^{t+1} \|\delta_0\|_2.$$

$\alpha > 0$ 足够小时，我们有 $\|I - \alpha A\|_2 < 1$，因此当 $t \to \infty$ 时 $\delta_t \to 0$。$\|I - \alpha A\|_2 < 1$ 成立的原因在于 A 是正定的，因此对于任何 x 都有 $x^T(I - \alpha A)x < 1$。

- **证明 2：** 考虑 $g(w) \doteq b - Aw$。由于 w^* 是 $g(w) = 0$ 的根，该任务实际上是一个求根问题。(8.22) 中的算法实际上是一个 Robbins-Monro (RM) 算法。虽然最初的 RM 算法是为随机过程设计的，但它也可以应用于确定性情况。**RM 算法的收敛性可以阐明 $w_{t+1} = w_t + \alpha_t(b - Aw_t)$ 的收敛性。**即，当 $\sum_t \alpha_t = \infty$ 且 $\sum_t \alpha_t^2 < \infty$ 时，w_t 收敛于 w^*。

方框 8.3：引理 8.1 的证明 (Proof of Lemma 8.1)

利用全期望公式 (law of total expectation)，我们有

$$\begin{aligned}
&\mathbb{E}\left[r_{t+1}\phi(s_t) + \phi(s_t)(\gamma\phi^T(s_{t+1}) - \phi^T(s_t))w_t\right] \\
&= \sum_{s\in\mathcal{S}} d_\pi(s)\mathbb{E}\left[r_{t+1}\phi(s_t) + \phi(s_t)(\gamma\phi^T(s_{t+1}) - \phi^T(s_t))w_t \middle| s_t = s\right] \\
&= \sum_{s\in\mathcal{S}} d_\pi(s)\mathbb{E}\left[r_{t+1}\phi(s_t) \middle| s_t = s\right] + \sum_{s\in\mathcal{S}} d_\pi(s)\mathbb{E}\left[\phi(s_t)(\gamma\phi^T(s_{t+1}) - \phi^T(s_t))w_t \middle| s_t = s\right]. \quad (8.24)
\end{aligned}$$

这里，假设 s_t 服从平稳分布 d_π。

首先，考虑 (8.24) 中的第一项。注意到

$$\mathbb{E}[r_{t+1}\phi(s_t)|s_t = s] = \phi(s)\mathbb{E}[r_{t+1}|s_t = s] = \phi(s)r_\pi(s),$$

其中 $r_\pi(s) = \sum_a \pi(a|s) \sum_r r p(r|s,a)$。那么，(8.24) 中的第一项可以重写为

$$\sum_{s\in\mathcal{S}} d_\pi(s)\mathbb{E}[r_{t+1}\phi(s_t)|s_t = s] = \sum_{s\in\mathcal{S}} d_\pi(s)\phi(s)r_\pi(s) = \Phi^T D r_\pi, \quad (8.25)$$

其中 $r_\pi = [\dots, r_\pi(s), \dots]^T \in \mathbb{R}^n$。

其次，考虑 (8.24) 中的第二项。由于

$$\begin{aligned}
&\mathbb{E}[\phi(s_t)(\gamma\phi^T(s_{t+1}) - \phi^T(s_t))w_t|s_t = s] \\
&= \mathbb{E}[\phi(s_t)\gamma\phi^T(s_{t+1})w_t|s_t = s] - \mathbb{E}[\phi(s_t)\phi^T(s_t)w_t|s_t = s] \\
&= -\phi(s)\phi^T(s)w_t + \gamma\phi(s)\mathbb{E}[\phi^T(s_{t+1})|s_t = s]w_t \\
&= -\phi(s)\phi^T(s)w_t + \gamma\phi(s)\sum_{s'\in\mathcal{S}} p(s'|s)\phi^T(s')w_t,
\end{aligned}$$

(8.24) 中的第二项变为

$$\begin{aligned}
&\sum_{s\in\mathcal{S}} d_\pi(s)\mathbb{E}[\phi(s_t)(\gamma\phi^T(s_{t+1})-\phi^T(s_t))w_t|s_t=s] \\
&=\sum_{s\in\mathcal{S}} d_\pi(s)\left[-\phi(s)\phi^T(s)w_t+\gamma\phi(s)\sum_{s'\in\mathcal{S}}p(s'|s)\phi^T(s')w_t\right] \\
&=\sum_{s\in\mathcal{S}} d_\pi(s)\phi(s)\left[-\phi(s)+\gamma\sum_{s'\in\mathcal{S}}p(s'|s)\phi(s')\right]^T w_t \\
&=\Phi^T D(-\Phi+\gamma P_\pi\Phi)w_t \\
&=-\Phi^T D(I-\gamma P_\pi)\Phi w_t. \quad (8.26)
\end{aligned}$$

结合 (8.25) 和 (8.26) 可得

$$\begin{aligned}
\mathbb{E}[(r_{t+1}+\gamma\phi^T(s_{t+1})w_t-\phi^T(s_t)w_t)\phi(s_t)] &= \Phi^T D r_\pi - \Phi^T D(I-\gamma P_\pi)\Phi w_t \\
&\doteq b - A w_t, \quad (8.27)
\end{aligned}$$

其中 $b\doteq\Phi^T D r_\pi$ 且 $A\doteq\Phi^T D(I-\gamma P_\pi)\Phi$。

方框 8.4：证明 $A=\Phi^T D(I-\gamma P_\pi)\Phi$ 是可逆且正定的 (Proving that A is invertible and positive definite)

如果对于任意具有适当维度的非零向量 x 都有 $x^T Ax>0$，则矩阵 A 是正定的。如果 A 是正定（或负定）的，记为 $A\succ 0$（或 $A\prec 0$）。**这里，$\succ$ 和 $\prec$ 应与表示按元素比较的 $>$ 和 $<$ 区分开来。注意 A 可能不是对称的。虽然正定矩阵通常指的是对称矩阵，但非对称矩阵也可以是正定的。**

接下来我们证明 $A\succ 0$，因此 A 是可逆的。证明 $A\succ 0$ 的思路是证明

$$D(I-\gamma P_\pi)\doteq M\succ 0. \quad (8.28)$$

显然，$M\succ 0$ 意味着 $A=\Phi^T M\Phi\succ 0$，因为 Φ 是一个列满秩的高矩阵（假设特征向量被选为线性无关的）。注意

$$M=\frac{M+M^T}{2}+\frac{M-M^T}{2}.$$

由于 $M-M^T$ 是反对称的，因此对于任意 x，都有 $x^T(M-M^T)x=0$，我们要知道 $M\succ 0$ 当且仅当 $M+M^T\succ 0$。为了证明 $M+M^T\succ 0$，我们应用一个事实：严格对角占优矩阵是正定的 [4]。

首先，成立

$$(M+M^T)\mathbf{1}_n>0, \quad (8.29)$$

其中 $\mathbf{1}_n=[1,\dots,1]^T\in\mathbb{R}^n$。(8.29) 的证明如下。由于 $P_\pi\mathbf{1}_n=\mathbf{1}_n$，我们有 $M\mathbf{1}_n=D(I-\gamma P_\pi)\mathbf{1}_n=D(\mathbf{1}_n-\gamma\mathbf{1}_n)=(1-\gamma)d_\pi$。此外，$M^T\mathbf{1}_n=(I-\gamma P_\pi^T)D\mathbf{1}_n=(I-\gamma P_\pi^T)d_\pi=(1-\gamma)d_\pi$，其中最后一个等式成立是因为 $P_\pi^T d_\pi=d_\pi$。总之，我们有

$$(M+M^T)\mathbf{1}_n=2(1-\gamma)d_\pi.$$

由于 d_π 的所有项都是正的（见方框 8.1），我们有 $(M+M^T)\mathbf{1}_n>0$。

其次，(8.29) 的按元素形式是

$$\sum_{j=1}^n [M+M^T]_{ij}>0, \quad i=1,\dots,n,$$

这可以进一步写为

$$[M+M^T]_{ii}+\sum_{j\neq i}[M+M^T]_{ij}>0.$$

根据 (8.28) 中 M 的表达式可以验证，M 的对角线元素是正的，而 M 的非对角线元素是非正的（$D=\begin{bmatrix}\ddots & & \\ & d_\pi(s) & \\ & & \ddots\end{bmatrix}$，把（8.28）展开，就可以证明）。因此，上述不等式可以重写为

$$|[M+M^T]_{ii}| > \sum_{j\neq i} |[M+M^T]_{ij}|.$$

上述不等式表明，$M+M^T$ 中第 i 个对角线元素的绝对值大于同一行中非对角线元素的绝对值之和。因此，$M+M^T$ 是严格对角占优的，证明完成。

TD 学习最小化投影贝尔曼误差 (TD learning minimizes the projected Bellman error)

虽然我们已经展示了 TD-Linear 算法收敛到 $w^* = A^{-1}b$，接下来我们展示 w^* 是最小化 *投影贝尔曼误差 (projected Bellman error)* 的最优解。为此，我们回顾三个目标函数。

- **第一个目标函数是**

$$J_E(w) = \mathbb{E}[(v_\pi(S) - \hat{v}(S,w))^2],$$

这在 (8.3) 中已经介绍过。根据期望的定义，$J_E(w)$ 可以重写为矩阵-向量形式

$$J_E(w) = \sum d_\pi(\hat{v}(w) - v_\pi)^2 = \|\hat{v}(w) - v_\pi\|_D^2,$$

其中 v_π 是真实状态价值向量，$\hat{v}(w)$ 是近似值。这里，$\|\cdot\|_D^2$ 是加权范数：$\|x\|_D^2 = x^T Dx = \|D^{1/2}x\|_2^2$，其中 D 在 (8.20) 中给出。

这是我们在谈论函数逼近时能想到的最简单的目标函数。然而，**它依赖于真实状态价值，而这是未知的**。为了获得一个可实现的算法，我们必须考虑其他目标函数，如**贝尔曼误差和投影贝尔曼误差 [50-54]**。

- **第二个目标函数是贝尔曼误差 (Bellman error)。** 特别是，**由于 v_π 满足贝尔曼方程 $v_\pi = r_\pi + \gamma P_\pi v_\pi$，我们期望估计值 $\hat{v}(w)$ 也应尽可能地满足该方程。因此，*贝尔曼误差* 为**

$$J_{BE}(w) = \|\hat{v}(w) - (r_\pi + \gamma P_\pi \hat{v}(w))\|_D^2 \doteq \|\hat{v}(w) - T_\pi(\hat{v}(w))\|_D^2. \quad (8.30)$$

这里，$T_\pi(\cdot)$ **是贝尔曼算子**。特别是，对于任意向量 $x \in \mathbb{R}^n$，贝尔曼算子定义为

$$T_\pi(x) \doteq r_\pi + \gamma P_\pi x.$$

最小化贝尔曼误差是一个标准的最小二乘问题。这里省略了求解的细节。

第三，值得注意的是，由于逼近器的逼近能力有限，(8.30) 中的 $J_{BE}(w)$ 可能无法被最小化到 *零 (zero)*。 相比之下，**一个可以被最小化到零的目标函数是 *投影贝尔曼误差 (projected Bellman error)*：**

$$J_{PBE}(w) = \|\hat{v}(w) - MT_\pi(\hat{v}(w))\|_D^2,$$

其中 $M \in \mathbb{R}^{n\times n}$ 是正交投影矩阵，它将任何向量几何投影到所有近似值的空间上。

事实上，(8.13) 中的 **TD 学习算法旨在最小化投影贝尔曼误差** J_{PBE}，而不是 J_E 或 J_{BE}。原因如下。为了简单起见，考虑线性情况，其中 $\hat{v}(w) = \Phi w$。这里，Φ 定义在 (8.20) 中。Φ 的值域 (range space) 是所有可能的线性近似的集合。那么，

$$M = \Phi(\Phi^T D\Phi)^{-1}\Phi^T D \in \mathbb{R}^{n\times n}$$

简单来说，矩阵 $M = \Phi(\Phi^T D\Phi)^{-1}\Phi^T D$ 之所以长这样，是因为它是**加权最小二乘法 (Weighted Least Squares)** 下的**投影矩阵**。

这里的关键在于那个加权矩阵 D（对角线上是平稳分布 d_π）。如果去掉 D（即设 $D = I$），这就变成了我们在线性代数中常见的标准投影矩阵（Hat Matrix）。

下面我为您逐步推导这个 M 是如何从“最小化加权误差”这个目标中推导出来的：

1. 设定目标：寻找最近的近似

假设我们有一个任意向量 y（**在你的文中，这个 y 是贝尔曼算子的结果 $T_\pi(\hat{v})$**），我们想在特征空间 Φ 中找到一个线性近似 $\hat{y} = \Phi w$，使得这个近似与原向量 y 的距离最近。

在强化学习的这个语境下，“距离”不是欧几里得距离，而是由状态访问概率 D 加权的距离：

$$J(w) = \|y - \Phi w\|_D^2 = (y - \Phi w)^T D(y - \Phi w)$$

2. 求解最优参数 w

为了让误差最小，我们需要对 w 求导并令其为 0：

$$\begin{aligned}\nabla_w J(w) &= \nabla_w \left((y^T - w^T\Phi^T)D(y-\Phi w)\right)\\ &= \nabla_w \left(y^T Dy - 2w^T\Phi^T Dy + w^T\Phi^T D\Phi w\right)\end{aligned}$$

对 w 求偏导（注意 $\Phi^T D\Phi$ 是对称阵）：

$$\frac{\partial J}{\partial w} = -2\Phi^T Dy + 2(\Phi^T D\Phi)w$$

令导数为 0：

$$\Phi^T D\Phi w = \Phi^T Dy$$

求解 w （假设 Φ 列满秩且 D 正定，则 $\Phi^T D\Phi$ 可逆）：

$$w^* = (\Phi^T D\Phi)^{-1}\Phi^T Dy$$

3. 构造投影矩阵 M

我们刚刚求出了最优的系数 w^*，但投影矩阵 M 的作用是将原向量 y 映射回近似向量 $\hat{y}$。

$$\hat{y} = \Phi w^*$$

将刚才求出的 w^* 代入：

$$\hat{y} = \Phi \underbrace{\left[(\Phi^T D\Phi)^{-1}\Phi^T D\right]}_{w^*} y$$

由此可见，将向量 y 变换为向量 $\hat{y}$ 的矩阵就是：

$$M = \Phi(\Phi^T D\Phi)^{-1}\Phi^T D$$

是将任何向量几何投影到 Φ 的值域上的投影矩阵。**由于 $\hat{v}(w)$ 在 Φ 的值域中，我们总是可以找到一个 w 值使得 $J_{PBE}(w)$ 最小化为零**。可以证明，最小化 $J_{PBE}(w)$ 的解是 $w^* = A^{-1}b$。即

$$w^* = A^{-1}b = \arg\min_w J_{PBE}(w),$$

证明在方框 8.5 中给出。

方框 8.5：证明 $w^* = A^{-1}b$ 最小化 $J_{PBE}(w)$ (Showing that $w^* = A^{-1}b$ minimizes $J_{PBE}(w)$)

我们接下来展示 $w^* = A^{-1}b$ 是最小化 $J_{PBE}(w)$ 的最优解。由于 $J_{PBE}(w) = 0 \iff \hat{v}(w) - MT_\pi(\hat{v}(w)) = 0$，我们只需要研究以下方程的根

$$\hat{v}(w) = MT_\pi(\hat{v}(w)).$$

在线性情况下，将 $\hat{v}(w) = \Phi w$ 和 M 的表达式代入上述方程可得

$$\Phi w = \Phi(\Phi^T D\Phi)^{-1}\Phi^T D(r_\pi + \gamma P_\pi \Phi w). \quad (8.31)$$

由于 Φ 具有列满秩 (full column rank)，因此对于任意 x, y，我们有 $\Phi x = \Phi y \iff x = y$。因此，(8.31) 蕴含

$$\begin{aligned}w &= (\Phi^T D\Phi)^{-1}\Phi^T D(r_\pi + \gamma P_\pi \Phi w)\\ &\iff \Phi^T D(r_\pi + \gamma P_\pi \Phi w) = (\Phi^T D\Phi)w\\ &\iff \Phi^T Dr_\pi + \Phi^T DP_\pi \Phi w = (\Phi^T D\Phi)w\\ &\iff \Phi^T Dr_\pi = \Phi^T D(I - \gamma P_\pi)\Phi w\\ &\iff w = (\Phi^T D(I-\gamma P_\pi)\Phi)^{-1}\Phi^T Dr_\pi = A^{-1}b,\end{aligned}$$

其中 A, b 在 (8.21) 中给出。因此，$w^* = A^{-1}b$ 是最小化 $J_{PBE}(w)$ 的最优解。

由于 TD 算法旨在最小化 J_{PBE} 而不是 J_E，一个自然的问题是估计值 $\hat{v}(w)$ 与真实状态价值 v_π 有多接近。在线性情况下，最小化投影贝尔曼误差的估计值是 $\hat{v}(w^*) = \Phi w^*$。它与真实状态价值 v_π 的偏差满足

$$\|\hat{v}(w^*) - v_\pi\|_D = \|\Phi w^* - v_\pi\|_D \le \frac{1}{1-\gamma}\min_w \|\hat{v}(w) - v_\pi\|_D = \frac{1}{1-\gamma}\min_w \sqrt{J_E(w)}. \quad (8.32)$$

该不等式的证明在方框 8.6 中给出。不等式 (8.32) 表明，Φw^* 与 v_π 之间的差异被 $J_E(w)$ 的最小值从上方界定（即有一个上界）。然而，这个界是松的 (loose)，特别是当 γ 接近 1 时。因此它主要具有理论价值。

方框 8.6：(8.32) 中误差界的证明 (Proof of the error bound in (8.32))

注意到

$$\begin{aligned}\|\Phi w^* - v_\pi\|_D &= \|\Phi w^* - Mv_\pi + Mv_\pi - v_\pi\|_D \\ &\le \|\Phi w^* - Mv_\pi\|_D + \|Mv_\pi - v_\pi\|_D \\ &= \|MT_\pi(\Phi w^*) - MT_\pi(v_\pi)\|_D + \|Mv_\pi - v_\pi\|_D, \quad (8.33)\end{aligned}$$

其中最后一个等式是由于 $\Phi w^* = MT_\pi(\Phi w^*)$ 以及 $v_\pi = T_\pi(v_\pi)$。代入

$$MT_\pi(\Phi w^*) - MT_\pi(v_\pi) = M(r_\pi + \gamma P_\pi \Phi w^*) - M(r_\pi + \gamma P_\pi v_\pi) = \gamma M P_\pi(\Phi w^* - v_\pi)$$

代入 (8.33) 可得

$$\begin{aligned}\|\Phi w^* - v_\pi\|_D &\le \|\gamma M P_\pi(\Phi w^* - v_\pi)\|_D + \|Mv_\pi - v_\pi\|_D \\ &\le \gamma \|M\|_D \|P_\pi(\Phi w^* - v_\pi)\|_D + \|Mv_\pi - v_\pi\|_D \\ &= \gamma \|P_\pi(\Phi w^* - v_\pi)\|_D + \|Mv_\pi - v_\pi\|_D \quad (\text{因为 } \|M\|_D = 1) \\ &\le \gamma \|\Phi w^* - v_\pi\|_D + \|Mv_\pi - v_\pi\|_D. \quad (\text{因为对于所有 } x \text{ 都有 } \|P_\pi x\|_D \le \|x\|_D)\end{aligned}$$

关于 $\|M\|_D = 1$ 和 $\|P_\pi x\|_D \le \|x\|_D$ 的证明推迟到本方框的最后。

整理上述不等式可得

$$\begin{aligned}\|\Phi w^* - v_\pi\|_D &\le \frac{1}{1-\gamma}\|Mv_\pi - v_\pi\|_D \\ &= \frac{1}{1-\gamma}\min_w \|\hat{v}(w) - v_\pi\|_D,\end{aligned}$$

其中最后一个等式成立是因为 $\|Mv_\pi - v_\pi\|_D$ 是 v_π 与其在所有可能近似值的空间上的正交投影之间的误差。因此，它是 v_π 与任意 $\hat{v}(w)$ 之间误差的最小值。

接下来我们证明上述证明中已经用到的一些有用事实。

- **矩阵加权范数的性质 (Properties of matrix weighted norms)**。
  - 根据定义，$\|x\|_D = \sqrt{x^T D x} = \|D^{1/2}x\|_2$。
  - 诱导矩阵范数为 $\|A\|_D = \max_{x \ne 0} \|Ax\|_D / \|x\|_D = \|D^{1/2} A D^{-1/2}\|_2$。
  - 对于具有适当维度的矩阵 A, B，我们有 $\|ABx\|_D \le \|A\|_D \|B\|_D \|x\|_D$。
  - 为了看出这一点:

$$\begin{aligned}\|ABx\|_D = \|D^{1/2}ABx\|_2 = \|D^{1/2}AD^{-1/2}D^{1/2}BD^{-1/2}D^{1/2}x\|_2 \le \\ \|D^{1/2}AD^{-1/2}\|_2 \|D^{1/2}BD^{-1/2}\|_2 \|D^{1/2}x\|_2 = \|A\|_D \|B\|_D \|x\|_D\end{aligned}$$

- **证明** $\|M\|_D = 1$。这成立是因为 $\|M\|_D = \|\Phi(\Phi^T D\Phi)^{-1}\Phi^T D\|_D = \|D^{1/2}\Phi(\Phi^T D\Phi)^{-1}\Phi^T D D^{-1/2}\|_2 = 1$，其中最后一个等式有效是因为 L_2 范数中的矩阵是一个正交投影矩阵，而任何正交投影矩阵的 L_2 范数都等于 1。
- **证明对于任意 $x \in \mathbb{R}^n$，$\|P_\pi x\|_D \le \|x\|_D$**。首先，

$$\|P_\pi x\|_D^2 = x^T P_\pi^T D P_\pi x = \sum_{i,j} x_i [P_\pi^T D P_\pi]_{ij} x_j = \sum_{i,j} x_i \left(\sum_k [P_\pi^T]_{ik}[D]_{kk}[P_\pi]_{kj}\right) x_j.$$

整理上述方程可得

$$\begin{aligned}\|P_\pi x\|_D^2 &= \sum_k [D]_{kk} \left(\sum_i [P_\pi]_{ki} x_i\right)^2 \\ &\le \sum_k [D]_{kk} \left(\sum_i [P_\pi]_{ki} x_i^2\right) \quad (\text{根据詹森不等式 [55, 56]}) \\ &= \sum_i \left(\sum_k [D]_{kk}[P_\pi]_{ki}\right) x_i^2 \\ &= \sum_i [D]_{ii} x_i^2 \quad (\text{由于 } d_\pi^T P_\pi = d_\pi^T) \\ &= \|x\|_D^2.\end{aligned}$$

最小二乘 TD (Least-squares TD)

接下来我们介绍一种称为 *最小二乘 TD* (least-squares TD, LSTD) 的算法 [57]。像 TD-Linear 算法一样，LSTD 旨在最小化投影贝尔曼误差。然而，它比 TD-Linear 算法具有一些优势。

回顾一下，最小化投影贝尔曼误差的最优参数是 $w^* = A^{-1}b$，其中 $A = \Phi^T D(I - \gamma P_\pi)\Phi$ 且 $b = \Phi^T D r_\pi$。事实上，由 (8.27) 可知，A 和 b 也可以写成

$$A = \mathbb{E}\left[\phi(s_t)(\phi(s_t) - \gamma\phi(s_{t+1}))^T\right],$$
$$b = \mathbb{E}[r_{t+1}\phi(s_t)].$$

$$\begin{aligned}\mathbb{E}[(r_{t+1} + \gamma\phi^T(s_{t+1})w_t - \phi^T(s_t)w_t)\phi(s_t)] &= \Phi^T D r_\pi - \Phi^T D(I - \gamma P_\pi)\Phi w_t \\ &\doteq b - A w_t, \quad (8.27)\end{aligned}$$

其中 $b \doteq \Phi^T D r_\pi$ 且 $A \doteq \Phi^T D(I - \gamma P_\pi)\Phi$。

上述两个方程表明 A 和 b 是 s_t, s_{t+1}, r_{t+1} 的期望。LSTD 的 *思想* 很简单：如果我们能利用随机样本直接获得 A 和 b 的估计值（记为 $\hat{A}$ 和 $\hat{b}$），那么最优参数可以直接估计为 $w^* \approx \hat{A}^{-1}\hat{b}$。

具体来说，假设 $(s_0, r_1, s_1, \ldots, s_t, r_{t+1}, s_{t+1}, \ldots)$ 是通过遵循给定策略 π 获得的轨迹。令 $\hat{A}_t$ 和 $\hat{b}_t$ 分别为时刻 t 时 A 和 b 的估计值。它们通过样本的平均值来计算：

$$\hat{A}_t = \sum_{k=0}^{t-1} \phi(s_k)(\phi(s_k) - \gamma\phi(s_{k+1}))^T,$$
$$\hat{b}_t = \sum_{k=0}^{t-1} r_{k+1}\phi(s_k). \quad (8.34)$$

那么，估计的参数为

$$w_t = \hat{A}_t^{-1}\hat{b}_t.$$

读者可能会想，(8.34) 式的右边是否缺少了一个系数 $1/t$。事实上，为了简洁起见，我们省略了它，因为即使省略了它，w_t 的值也保持不变。由于 $\hat{A}_t$ 可能不可逆，特别是在 t 较小时，通常会给 $\hat{A}_t$ 加上一个小的常数矩阵 σI（其中 I 是单位矩阵，σ 是一个小的正数）来使其可逆。

LSTD 的 *优势* 在于它能更有效地利用经验样本，并且比 TD 方法收敛得更快。这是因为该算法是专门基于最优解表达式的知识而设计的。我们对问题的理解越深，就能设计出越好的算法。

LSTD 的 *劣势* 如下。

- 首先，它只能估计状态价值。相比之下，TD 算法可以扩展到估计动作价值，如下一节所示。
- 此外，虽然 TD 算法允许使用非线性逼近器，但 LSTD 不允许。这是因为该算法是专门基于 w^* 的表达式（线性假设下）设计的。
- 其次，LSTD 的计算成本高于 TD，因为 LSTD 在每一步更新中都要更新一个 $m \times m$ 的矩阵，而 TD 更新的是一个 m 维向量。
- 更重要的是，在每一步中，LSTD 需要计算 $\hat{A}_t$ 的逆矩阵，其计算复杂度为 $O(m^3)$。解决这个问题的常用方法是直接更新 $\hat{A}_t$ 的逆矩阵，而不是更新 $\hat{A}_t$。具体来说，$\hat{A}_{t+1}$ 可以递归计算如下：

$$\begin{aligned}\hat{A}_{t+1} &= \sum_{k=0}^{t} \phi(s_k)(\phi(s_k) - \gamma\phi(s_{k+1}))^T \\ &= \sum_{k=0}^{t-1} \phi(s_k)(\phi(s_k) - \gamma\phi(s_{k+1}))^T + \phi(s_t)(\phi(s_t) - \gamma\phi(s_{t+1}))^T \\ &= \hat{A}_t + \phi(s_t)(\phi(s_t) - \gamma\phi(s_{t+1}))^T.\end{aligned}$$

上述表达式将 $\hat{A}_{t+1}$ 分解为两个矩阵之和。其逆矩阵可以计算如下 [58]：

$$\begin{aligned}\hat{A}_{t+1}^{-1} &= \left(\hat{A}_t + \phi(s_t)(\phi(s_t) - \gamma\phi(s_{t+1}))^T\right)^{-1} \\ &= \hat{A}_t^{-1} + \frac{\hat{A}_t^{-1}\phi(s_t)(\phi(s_t) - \gamma\phi(s_{t+1}))^T \hat{A}_t^{-1}}{1 + (\phi(s_t) - \gamma\phi(s_{t+1}))^T \hat{A}_t^{-1}\phi(s_t)}.\end{aligned}$$

因此，我们可以直接存储和更新 $\hat{A}_t^{-1}$ 以避免计算矩阵逆。这种递归算法不需要步长。然而，它需要设置 $\hat{A}_0^{-1}$ 的初始值。这种递归算法的初始值可以选择为 $\hat{A}_0^{-1} = \sigma I$，其中 σ 是一个正数。关于递归最小二乘法的优秀教程可以在 [59] 中找到。

8.3 基于函数逼近的动作价值 TD 学习 (TD learning of action values based on function approximation)

虽然 8.2 节介绍了 *状态价值估计 (state value estimation)* 的问题，但本节将介绍如何估计 *动作价值 (action values)*。表格型 Sarsa 和表格型 Q-learning 算法将被扩展到价值函数逼近的情况。读者将会看到，这种扩展是非常直接的。

8.3.1 带有函数逼近的 Sarsa (Sarsa with function approximation)

带有函数逼近的 Sarsa 算法可以通过将状态价值替换为动作价值，从而很容易地从 (8.13) 中获得。具体来说，假设 $q_\pi(s,a)$ 由 $\hat{q}(s,a,w)$ 逼近。将 (8.13) 中的 $\hat{v}(s,w)$ 替换为 $\hat{q}(s,a,w)$ 可得

$$w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma\hat{q}(s_{t+1},a_{t+1},w_t) - \hat{q}(s_t,a_t,w_t)\right]\nabla_w \hat{q}(s_t,a_t,w_t). \quad (8.35)$$

对 (8.35) 的分析与对 (8.13) 的分析类似，在此省略。当使用线性函数时，我们有

$$\hat{q}(s,a,w) = \phi^T(s,a)w,$$

其中 $\phi(s,a)$ 是特征向量。在这种情况下，$\nabla_w \hat{q}(s,a,w) = \phi(s,a)$。

(8.35) 中的价值估计步骤可以与策略改进步骤相结合，以学习最优策略。该过程总结在算法 8.2 中。

- 值得注意的是，准确估计给定策略的动作价值需要运行 (8.35) 足够多次。
- 然而，在切换到策略改进步骤之前，(8.35) 仅执行一次。这与表格型 Sarsa 算法类似。
- 此外，算法 8.2 中的实现旨在解决从预先指定的起始状态寻找通往目标状态的良好路径的任务。结果是，它可能无法找到针对 *每个* 状态的最优策略。然而，如果有足够多的经验数据可用，该实现过程可以很容易地调整为寻找针对每个状态的最优策略。

图 8.9 展示了一个演示示例。在该示例中，**任务是找到一个当从左上角状态开始时能够引导智能体到达目标的良好策略。总回报和每个回合的长度都逐渐收敛到稳定值。**在该示例中，线性特征向量被选为 5 阶傅里叶函数。傅里叶特征向量的表达式已在 (8.18) 中给出。

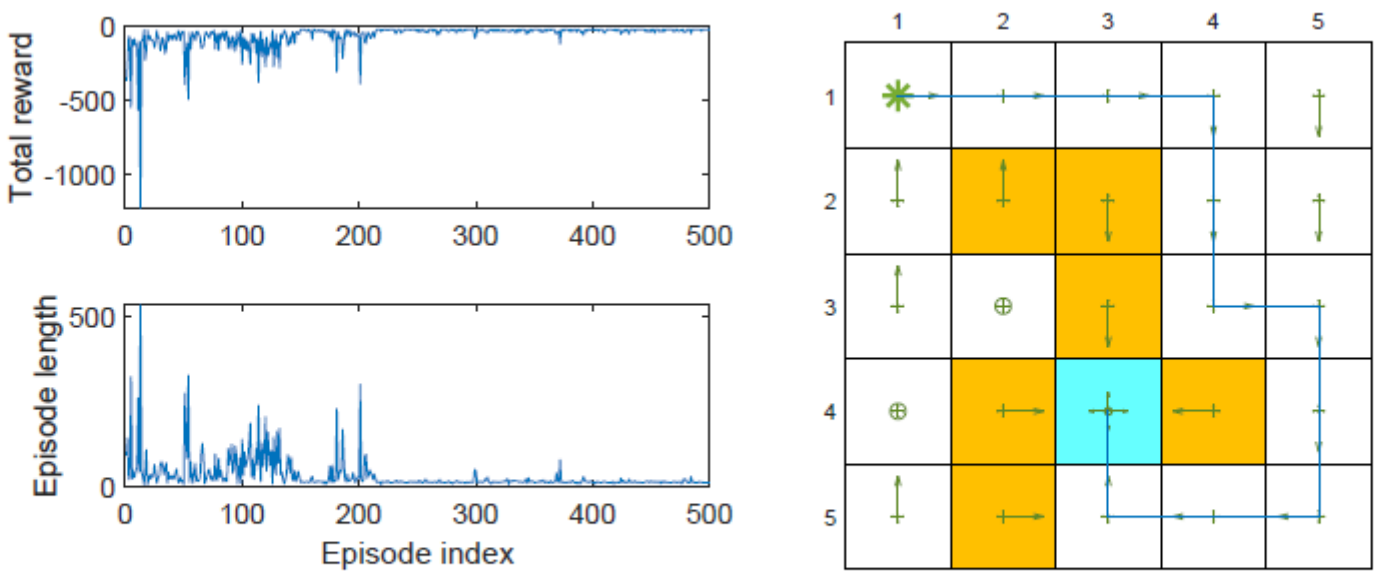


Figure 8.9: Sarsa with linear function approximation. Here, $\gamma = 0.9$, $\epsilon = 0.1$, $r_{\text{boundary}} = r_{\text{forbidden}} = -10$, $r_{\text{target}} = 1$, and $\alpha = 0.001$.

图 8.9：带有线性函数逼近的 Sarsa。 这里，$\gamma = 0.9, \epsilon = 0.1, r_{\text{boundary}} = r_{\text{forbidden}} = -10, r_{\text{target}} = 1$ 且 $\alpha = 0.001$。

算法 8.2：带有函数逼近的 Sarsa (Sarsa with function approximation)

初始化：初始参数 w_0。初始策略 π_0。对所有 t，$\alpha_t = \alpha > 0$。$\epsilon \in (0,1)$。

目标： 学习一个最优策略，能够引导智能体从初始状态 s_0 到达目标状态。

对于 每个回合，做

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，做

给定 (s_t, a_t) 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 q 值 (Update q-value)：(**之前这里是更新表格的值，现在变为更新带参模型的值**)

$$w_{t+1} = w_t + \alpha_t[r_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}, w_t) - \hat{q}(s_t, a_t, w_t)]\nabla_w \hat{q}(s_t, a_t, w_t)$$

更新策略 (Update policy)：

$$\pi_{t+1}(a|s_t) = 1 - \frac{\epsilon}{|\mathcal{A}(s_t)|}(|\mathcal{A}(s_t)| - 1) \text{ 如果 } a = \arg\max_{a \in \mathcal{A}(s_t)} \hat{q}(s_t, a, w_{t+1})$$

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

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

8.3.2 带有函数逼近的 Q 学习 (Q-learning with function approximation)

表格型 Q 学习也可以扩展到函数逼近的情况。更新规则为

$$w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma \max_{a \in \mathcal{A}(s_{t+1})} \hat{q}(s_{t+1}, a, w_t) - \hat{q}(s_t, a_t, w_t) \right] \nabla_w \hat{q}(s_t, a_t, w_t). \quad (8.36)$$

上述更新规则与 (8.35) 类似，只是 (8.35) 中的 $\hat{q}(s_{t+1}, a_{t+1}, w_t)$ 被替换为了 $\max_{a \in \mathcal{A}(s_{t+1})} \hat{q}(s_{t+1}, a, w_t)$ 。

与表格情况类似，(8.36) 可以以 ***on-policy*** 或 ***off-policy*** 的方式实现。算法 8.3 给出了一个 ***on-policy*** 版本。图 8.10 展示了一个演示 ***on-policy*** 版本的示例。在该示例中，任务是找到一个能够引导智能体从左上角状态到达目标状态的良好策略。

算法 8.3：带有函数逼近的 Q 学习（ *on-policy* ）(Q-learning with function approximation (on-policy version))

初始化： 初始参数 w_0 。初始策略 π_0 。对所有 t ， $\alpha_t = \alpha > 0$ 。$\epsilon \in (0,1)$ 。

目标： 学习一条能够引导智能体从初始状态 s_0 到达目标状态的最优路径。

对于 每个回合，做

如果 $s_t(t=0,1,2,\dots)$ 不是目标状态，做

给定 s_t 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 q 值 (Update q-value)：

$$w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma \max_{a \in \mathcal{A}(s_{t+1})} \hat{q}(s_{t+1}, a, w_t) - \hat{q}(s_t, a_t, w_t) \right] \nabla_w \hat{q}(s_t, a_t, w_t)$$

更新策略 (Update policy)：

$$\pi_{t+1}(a|s_t) = 1 - \frac{\epsilon}{|\mathcal{A}(s_t)|}(|\mathcal{A}(s_t)| - 1) \text{ 如果 } a = \arg\max_{a \in \mathcal{A}(s_t)} \hat{q}(s_t, a, w_{t+1})$$

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

可以看出，带有线性函数逼近的 Q 学习可以成功学习到最优策略。这里使用了 5 阶线性傅里叶基函数。我们将在 8.4 节介绍深度 Q 学习时演示off-policy 版本。

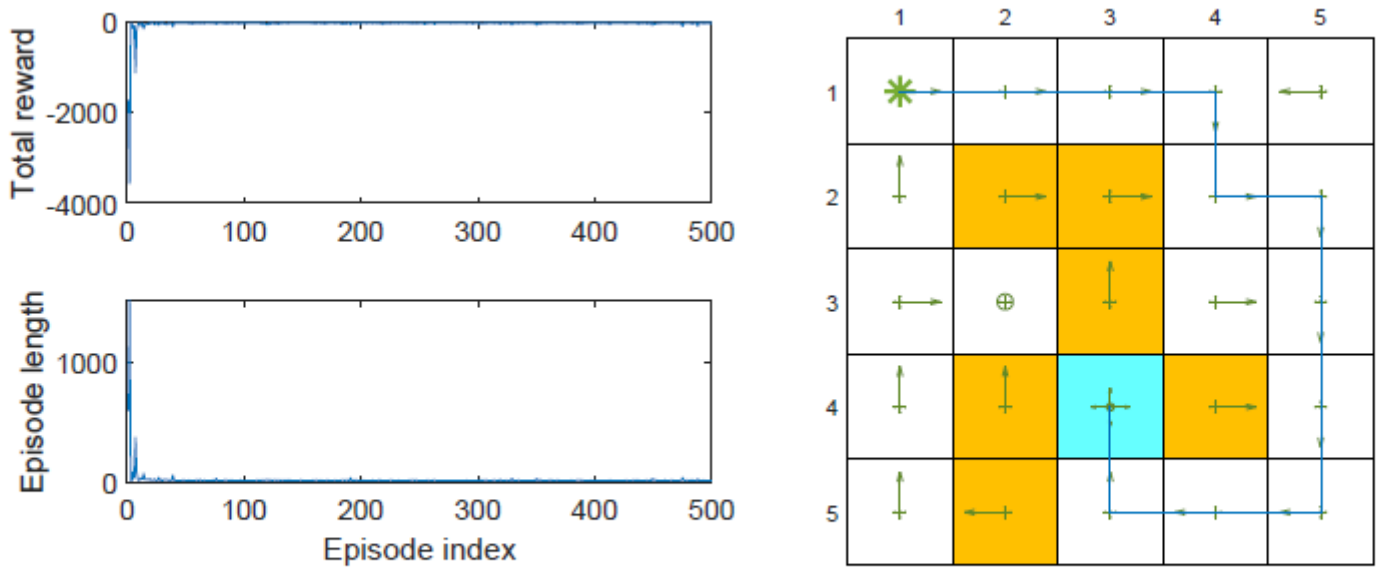


Figure 8.10: Q-learning with linear function approximation. Here, $\gamma = 0.9$, $\epsilon = 0.1$, $r_{\text{boundary}} = r_{\text{forbidden}} = -10$, $r_{\text{target}} = 1$, and $\alpha = 0.001$.

图 8.10：带有线性函数逼近的 Q 学习。 这里，$\gamma = 0.9, \epsilon = 0.1, r_{\text{boundary}} = r_{\text{forbidden}} = -10, r_{\text{target}} = 1$ 且 $\alpha = 0.001$。

读者可能在算法 8.2 和算法 8.3 中注意到，虽然价值被表示为函数，但策略 $\pi(a|s)$ 仍然被表示为表格。因此，它仍然假设状态和动作的数量是有限的。在第 9 章中，我们将看到策略也可以被表示为函数，从而使得连续的状态和动作空间可以被处理。

8.4 深度 Q-learning (Deep Q-learning)

我们可以将深度神经网络集成到 Q-learning中，从而获得一种称为 *深度 Q-learning* 或 *深度 Q 网络 (deep Q-network, DQN)* 的方法 [22, 60, 61]。深度Q-learning是最早且最成功的深度强化学习算法之一。值得注意的是，神经网络不一定要很深。对于诸如我们的网格世界示例之类的简单任务，具有一个或两个隐藏层的浅层网络可能就足够了。

深度 Q 学习可以被视为 (8.36) 中算法的扩展。然而，其数学公式和实现技术有很大不同，值得特别关注。

8.4.1 算法描述 (Algorithm description)

在数学上，深度 Q-learning 旨在最小化以下目标函数：

$$J = \mathbb{E}\left[\left(R + \gamma \max_{a \in \mathcal{A}(S')} \hat{q}(S', a, w) - \hat{q}(S, A, w)\right)^2\right], \quad (8.37)$$

其中 (S, A, R, S') 分别是表示状态、动作、即时奖励和下一个状态的随机变量。该目标函数可以看作是贝尔曼最优误差的平方。这是因为

$$q(s, a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a \in \mathcal{A}(S_{t+1})} q(S_{t+1}, a) \middle| S_t = s, A_t = a\right], \quad \text{对所有 } s, a$$

是贝尔曼最优方程（证明在方框 7.5 中给出）。因此，当 $\hat{q}(S, A, w)$ 能够准确逼近最优动作价值时，$R + \gamma \max_{a \in \mathcal{A}(S')} \hat{q}(S', a, w) - \hat{q}(S, A, w)$ 在期望意义上应等于零。

为了最小化 (8.37) 中的目标函数，我们可以使用梯度下降算法。为此，我们需要计算 J 关于 w 的梯度。

值得注意的是，参数 w 不仅出现在 $\hat{q}(S, A, w)$ 中，还出现在 $y \doteq R + \gamma \max_{a \in \mathcal{A}(S')} \hat{q}(S', a, w)$ 中。

因此，计算梯度并非易事。为了简单起见，假设 y 中的 w 值是固定的（在短时间内），这样梯度的计算就变得容易多了。具体来说，我们引入两个网络：一个是表示 $\hat{q}(s, a, w)$ 的 *主网络 (main network)*，另一个是 *目标网络 (target network)* $\hat{q}(s, a, w_T)$。在这种情况下，目标函数变为

$$J = \mathbb{E}\left[\left(R + \gamma \max_{a \in \mathcal{A}(S')} \hat{q}(S', a, w_T) - \hat{q}(S, A, w)\right)^2\right],$$

其中 w_T 是目标网络的参数。当 w_T 固定时，J 的梯度为

$$\nabla_w J = -\mathbb{E}\left[\left(R + \gamma \max_{a \in \mathcal{A}(S')} \hat{q}(S', a, w_T) - \hat{q}(S, A, w)\right) \nabla_w \hat{q}(S, A, w)\right], \quad (8.38)$$

其中为了不失一般性省略了一些常数系数。

为了使用 (8.38) 中的梯度来最小化目标函数，我们需要注意以下技术。

- **第一种技术是使用两个网络，即主网络 (main network) 和目标网络 (target network)，** 正如我们在计算 (8.38) 中的梯度时所提到的。实现细节解释如下。令 w 和 w_T 分别表示主网络和目标网络的参数。它们最初被设置为相同的值。
  - 在每次迭代中，我们从回放缓冲区（回放缓冲区将在稍后解释）中提取一个小批量 (mini-batch) 的样本 $\{(s, a, r, s')\}$。主网络的输入是 s 和 a。**输出 $y = \hat{q}(s, a, w)$ 是估计的 q 值**。输出的目标值是 $y_T \doteq r + \gamma \max_{a \in \mathcal{A}(s')} \hat{q}(s', a, w_T)$。更新主网络以最小化样本 $\{(s, a, r, s')\}$ 上的 TD 误差（也称为损失函数） $\sum (y - y_T)^2$。
  - 更新主网络中的 w 并不显式地使用 (8.38) 中的梯度。相反，它依赖于现有的神经网络训练软件工具。因此，我们需要一个小批量的样本来训练网络，而不是根据 (8.38) 使用单个样本来更新主网络。这是深度强化学习算法与非深度强化学习算法之间的一个显著区别。
  - 主网络在每次迭代中都会更新。**相比之下，目标网络每隔一定数量的迭代次数才会被设置为与主网络相同**，以满足在计算 (8.38) 中的梯度时 w_T 是固定的这一假设。
- **第二种技术是 *经验回放 (experience replay)* [22, 60, 62]。** 也就是说，在我们收集了一些经验样本后，我们并不按照收集它们的顺序来使用这些样本。相反，我们将它们存储在一个称为 *回放缓冲区 (replay buffer)* 的数据集中。具体来说，令 (s, a, r, s') 为一个经验样本，$\mathcal{B} \doteq \{(s, a, r, s')\}$ 为回放缓冲区。每次我们更新主网络时，我们可以从回放缓冲区中提取一个小批量的经验样本。**样本的提取（称为 *经验回放*）应遵循 *均匀分布 (uniform distribution)*。**
  - 为什么经验回放在深度Q-learning中是必要的，以及为什么回放必须遵循均匀分布？ 答案在于 (8.37) 中的目标函数。具体来说，为了良好地定义目标函数，我们必须指定 S, A, R, S' 的概率分布。一旦给定 (S, A)，R 和 S' 的分布就由系统模型决定了。描述状态-动作对 (S, A) 分布的最简单方法是假设它是 *均匀分布* 的。
  - 然而，在实践中，状态-动作样本可能并非均匀分布，因为它们是根据行为策略生成的样本序列。必须打破序列中样本之间的相关性，以满足均匀分布的假设。为此，我们可以使用经验回放技术，通过从回放缓冲区中均匀地提取样本。这就是为什么经验回放是必要的以及经验回放必须遵循均匀分布的数学原因。随机采样的一个好处是每个经验样本可以被多次重复使用，这可以提高数据效率。当我们的数据量有限时，这一点尤为重要。

深度 Q-learning的实现过程总结在算法 8.3 中。该实现是异策略 (off-policy) 的。如果需要，它也可以调整为同策略 (on-policy)。

算法 8.3：深度 Q-learning（off-policy）(Deep Q-learning (off-policy version))

初始化： 一个主网络和一个具有相同初始参数的目标网络。

目标： 学习一个最优目标网络，利用给定的行为策略 π_b 生成的经验样本来逼近 最优 动作价值。

将由 π_b 生成的经验样本存储在回放缓冲区 $\mathcal{B} = \{(s, a, r, s')\}$ 中

对于 每次迭代，做

从 $\mathcal{B}$ 中均匀抽取一个小批量的样本

对于每个样本 (s, a, r, s')，计算目标值为 $y_T = r + \gamma \max_{a \in \mathcal{A}(s')} \hat{q}(s', a, w_T)$

其中 w_T 是目标网络的参数

利用该小批量样本更新主网络以最小化 $(y_T - \hat{q}(s, a, w))^2$

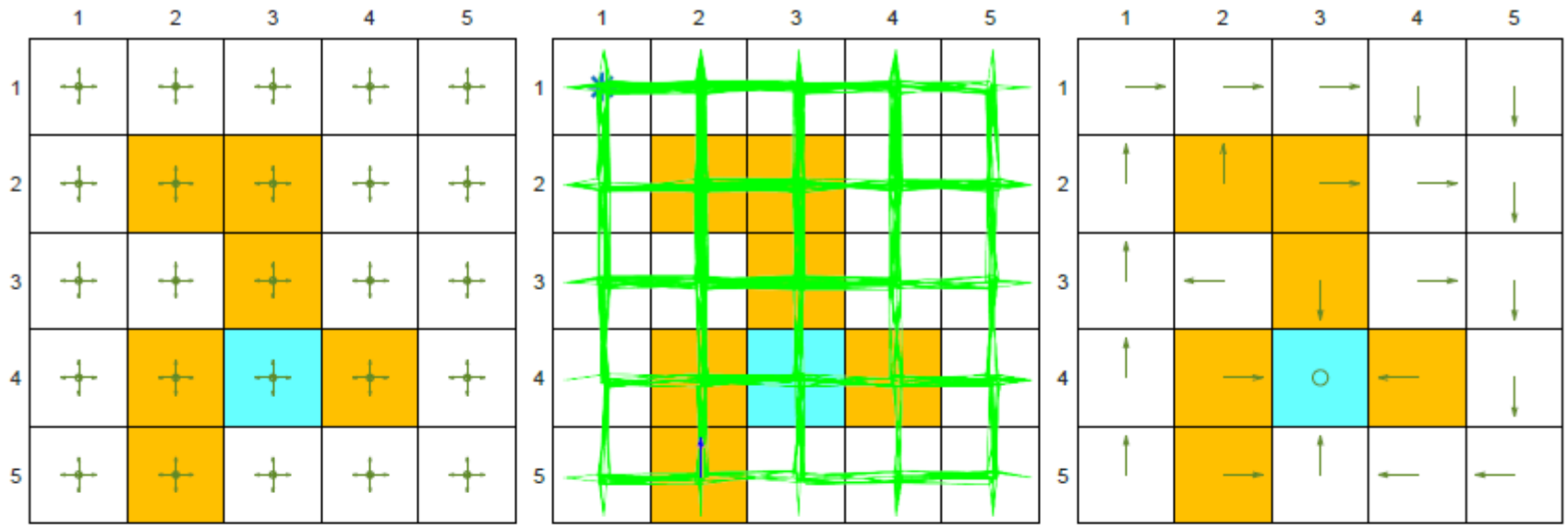
每 C 次迭代设 $w_T = w$

8.4.2 演示示例 (Illustrative examples)

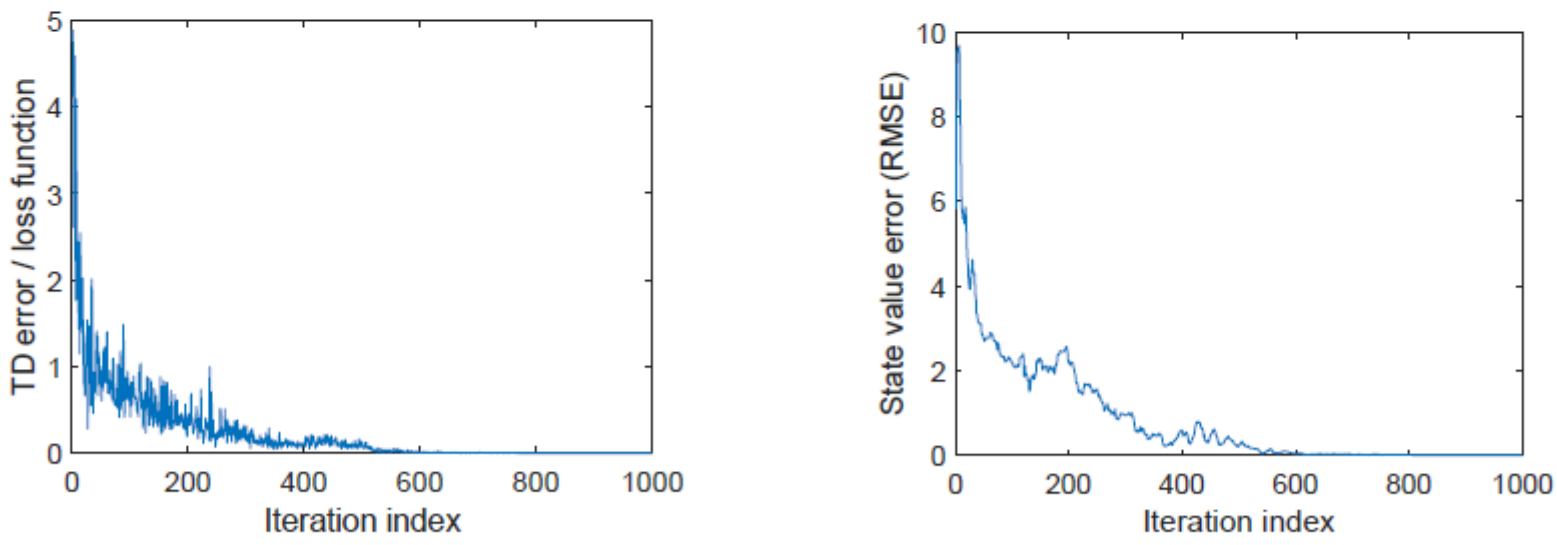
图 8.11 给出了一个示例来演示算法 8.3。**该示例旨在学习 *每个* 状态-动作对的最优动作价值。一旦获得了最优动作价值，就可以立即获得最优贪婪策略。**

单个回合由图 8.11(a) 所示的行为策略生成。这个行为策略是探索性的，因为它在任何状态下采取任何动作的概率都是相同的。如图 8.11(b) 所示，**该回合只有 1,000 步。**尽管只有 1,000 步，但由于行为策略强大的探索能力，几乎所有的状态-动作对都在该回合中被访问过。回放缓冲区是一组包含 1,000 个经验样本的集合。小批量大小为 100，这意味着我们每次获取样本时都会从回放缓冲区中均匀抽取 100 个样本。

主网络和目标网络具有相同的结构：一个包含 100 个神经元的单隐藏层神经网络（层数和神经元数量可以调整）。该神经网络有三个输入和一个输出。前两个输入是状态的归一化行索引和列索引。第三个输入是归一化动作索引。这里，“归一化”是指将值转换到区间 $[0, 1]$ 内。网络的输出是估计价值。我们之所以将输入设计为状态的行和列而不是状态索引，是因为我们知道状态对应于网格中的二维位置。在设计网络时，我们使用的关于状态的信息越多，网络的性能就越好。此外，神经网络也可以设计成其他方式。例如，它可以有两个输入和五个输出，其中两个输入是状态的归一化行索引和列索引，输出是该输入状态的五个估计动作价值 [22]。



(a) The behavior policy. (b) An episode with 1,000 steps. (c) The final learned policy.



(d) The loss function converges to zero. (e) The value error converges to zero.

Figure 8.11: Optimal policy learning via deep Q-learning. Here, $\gamma = 0.9$, $r_{\text{boundary}} = r_{\text{forbidden}} = -10$, and $r_{\text{target}} = 1$. The batch size is 100.

(a) 行为策略 (The behavior policy)。 (b) 一个包含 1,000 步的回合 (An episode with 1,000 steps)。 (c) 最终学习到的策略 (The final learned policy)。

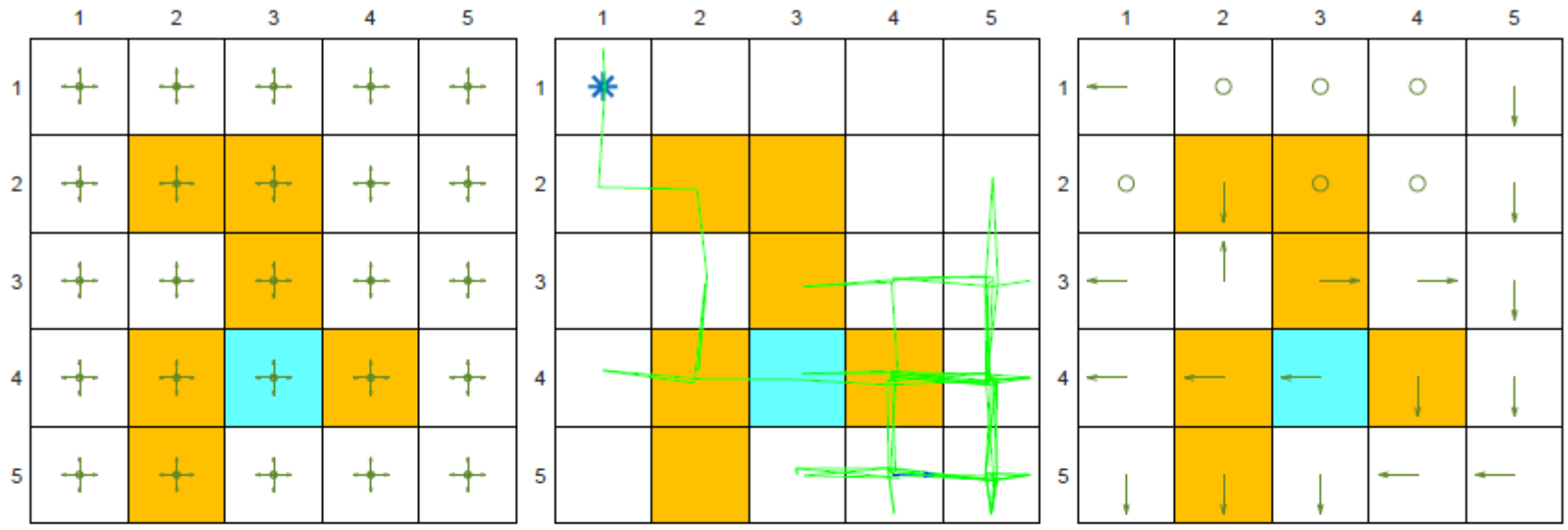
(d) 损失函数收敛到零 (The loss function converges to zero)。 (e) 价值误差收敛到零 (The value error converges to zero)。

图 8.11：通过深度 Q-learning进行最优策略学习。 这里，$\gamma = 0.9, r_{\text{boundary}} = r_{\text{forbidden}} = -10$ 且 $r_{\text{target}} = 1$。批量大小 (batch size) 为 100。

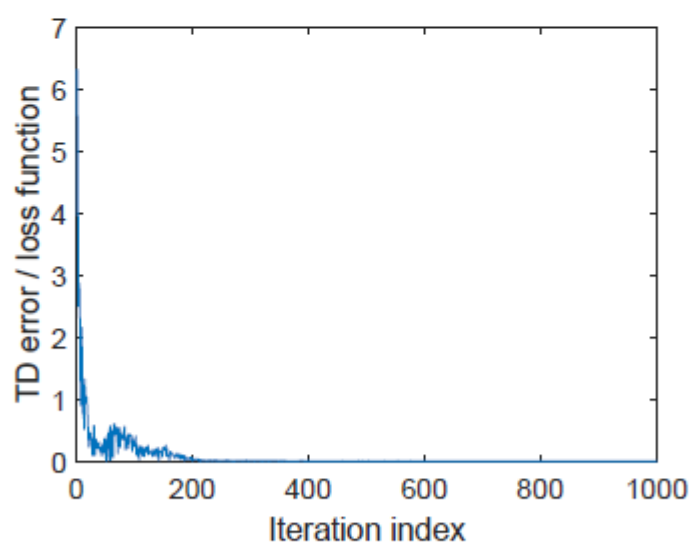
如图 8.11(d) 所示，损失函数（定义为每个小批量的平均平方 TD 误差）收敛到零，这意味着网络能够很好地拟合训练样本。如图 8.11(e) 所示，状态价值估计误差也收敛到零，表明最优动作价值的估计变得足够准确。随后，相应的贪婪策略就是最优的。

这个例子展示了深度 Q-learning的高效率。特别是，这里仅需一个 1,000 步的短回合就足以获得最优策略。相比之下，如图 7.4 所示，表格型 Q-learning需要一个 100,000 步的回合。**效率高的一个原因是函数逼近方法具有很强的 *泛化能力 (generalization ability)*。另一个原因是经验样本可以被 *重复使用 (repeatedly used)*。**

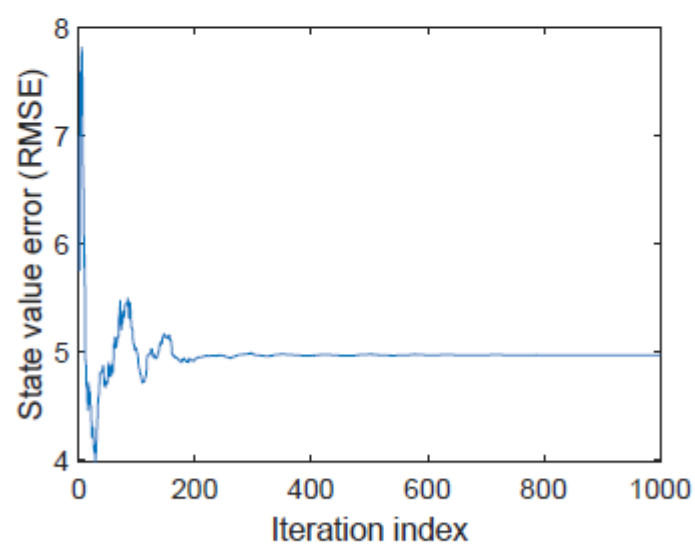
接下来，我们通过考虑经验样本较少的场景来特意挑战深度 Q-learning算法。图 **8.12 展示了一个只有 100 步的回合示例。**在这个例子中，虽然网络在某种意义上仍然可以被很好地训练, 即损失函数收敛到零，状态估计误差也无法收敛到零。这意味着网络可以正确地拟合给定的经验样本，但经验样本太少，无法准确估计最优动作价值。（*注：这就是典型的过拟合现象。*）



(a) The behavior policy. (b) An episode with 100 steps. (c) The final learned policy.



(d) The loss function converges to zero.



(e) The value error does not converge to zero.

Figure 8.12: Optimal policy learning via deep Q-learning. Here, $\gamma = 0.9$, $r_{\text{boundary}} = r_{\text{forbidden}} = -10$, and $r_{\text{target}} = 1$. The batch size is 50.

(a) 行为策略 (The behavior policy)。 (b) 一个包含 100 步的回合 (An episode with 100 steps)。 (c) 最终学习到的策略 (The final learned policy)。

(d) 损失函数收敛到零 (The loss function converges to zero)。 (e) 价值误差没有收敛到零 (The value error does not converge to zero)。

图 8.12：通过深度 Q-learning进行最优策略学习。 这里，$\gamma = 0.9, r_{\text{boundary}} = r_{\text{forbidden}} = -10$ 且 $r_{\text{target}} = 1$。批量大小为 50。

8.5 总结 (Summary)

本章继续介绍 TD 学习算法。然而，它从表格方法转向了函数逼近方法。理解函数逼近方法的关键在于认识到它是一个优化问题。最简单的目标函数是真实状态价值与估计价值之间的平方误差。还有其他的目标函数，如贝尔曼误差 (Bellman error) 和投影贝尔曼误差 (projected Bellman error)。我们已经展示了 TD-Linear 算法实际上最小化的是投影贝尔曼误差。我们介绍了几种优化算法，如带有价值逼近的 Sarsa 和 Q-learning。

价值函数逼近方法之所以重要，原因之一是它允许将人工神经网络与强化学习相结合。例如，深度 Q-learning是最成功的深度强化学习算法之一。

虽然神经网络已被广泛用作非线性函数逼近器，但本章对线性函数的情况进行了全面的介绍。完全理解线性情况对于更好地理解非线性情况非常重要。感兴趣的读者可以参考 [63] 以获取关于带有函数逼近的 TD 学习算法的深入分析。关于深度 Q 学习的更多理论讨论可以在 [61] 中找到。

本章介绍了一个名为 *平稳分布 (stationary distribution)* 的重要概念。平稳分布在定义价值函数逼近方法中的适当目标函数方面起着重要作用。当我们使用函数来逼近策略时，它在第 9 章中也起着关键作用。关于该主题的精彩介绍可以在 [49, Chapter IV] 中找

到。本章的内容在很大程度上依赖于矩阵分析。有些结果在使用时没有解释。关于矩阵分析和线性代数的优秀参考文献可以在 [4, 48] 中找到。

8.6 问与答 (Q&A)

- **问：表格方法和函数逼近方法之间有什么区别?**

答： 一个重要的区别在于价值是如何被更新和检索的。

  - **如何 *检索 (retrieve)* 一个价值：** 当价值由表格表示时，如果我们想要检索一个价值，我们可以直接读取表格中相应的条目。然而，当价值由函数表示时，我们需要将状态索引 s 输入到函数中并计算函数值。如果函数是一个人工神经网络，则需要一个从输入到输出的前向传播过程。
  - **如何 *更新 (update)* 一个价值：** 当价值由表格表示时，如果我们想要更新一个价值，我们可以直接重写表格中相应的条目。然而，当价值由函数表示时，我们必须更新函数参数来间接地改变价值。

- **问：函数逼近方法相比表格方法有什么优势?**

答： 由于检索状态价值的方式不同，函数逼近方法在存储方面更有效率。特别是，表格方法需要存储 $|\mathcal{S}|$ 个值，而函数逼近方法只需要存储一个维度通常远小于 $|\mathcal{S}|$ 的参数向量。

由于更新状态价值的方式不同，函数逼近方法还有另一个优点：它的 *泛化能力 (generalization ability)* 比表格方法更强。原因如下。在表格方法中，更新一个状态价值不会改变其他状态的价值。然而，在函数逼近方法中，更新函数参数会影响许多状态的价值。

因此，一个状态的经验样本可以泛化以帮助估计其他状态的价值。

- **问：我们可以统一表格方法和函数逼近方法吗?**

答： 可以。表格方法可以被视为函数逼近方法的一个特例。相关细节可以在方框 8.2 中找到。

- **问：什么是平稳分布 (stationary distribution)，为什么它很重要?**

答： **平稳分布描述了马尔可夫决策过程的长期行为。**具体来说，**在智能体执行给定策略足够长的时间后，智能体访问某个状态的概率可以用这个平稳分布来描述。**更多信息可以在方框 8.1 中找到。这个概念之所以出现在本章，是因为它对于定义一个有效的目标函数是必要的。特别是，目标函数涉及状态的概率分布，而这通常被选择为平稳分布。平稳分布不仅对价值逼近方法很重要，对于将在第 9 章介绍的策略梯度方法也很重要。

- **问：线性函数逼近方法的优缺点是什么?**

答： 线性函数逼近是理论性质可以被彻底分析的最简单情况。然而，该方法的逼近能力是有限的。**为复杂任务选择合适的特征向量也并非易事。**相比之下，人工神经网络可以用作黑盒通用非线性逼近器来逼近价值，使用起来更加友好。尽管如此，研究线性情况对于更好地掌握函数逼近方法的思想仍然是有意义的。此外，线性情况也是强大的，因为表格方法可以被视为一种特殊的线性情况（方框 8.2）。

- **问：为什么深度 Q-learning需要经验回放 (experience replay)?**

答： 原因在于 (8.37) 中的目标函数。特别是，为了良好地定义目标函数，我们必须指定 S, A, R, S' 的概率分布。一旦给定 (S, A)，R 和 S' 的分布就由系统模型决定了。描述状态-动作对 (S, A) 分布的最简单方法是假设它是 *均匀分布 (uniformly distributed)* 的。然而，在实践中，状态-动作样本可能并非均匀分布，因为它们是根据行为策略生成的序列。为了满足均匀分布的假设，必须 *打破序列中样本之间的相关性 (break the correlation)*。为此，我们可以使用经验回放技术，通过从回放缓冲区中均匀地抽取样本。经验回放的一个好处是每个经验样本可以被多次使用，这可以提高数据效率。

- **问：表格型 Q-learning可以使用经验回放 (experience replay) 吗?**

答： 虽然表格型 Q 学习不需要经验回放，但它也可以使用经验回放而不会遇到问题。这是因为由于 Q 学习的off-policy 属性，它对样本的获取方式没有任何要求。使用经验回放的一个好处是样本可以被重复使用，从而更加高效。

-

- **问：当人工神经网络被用作非线性函数逼近器时，我们应该如何更新其参数?**

答： 必须注意，我们不应该（例如使用 (8.36)）直接更新参数向量。相反，我们应该遵循网络训练过程来更新参数。该过程可以基于神经网络训练工具包来实现，这些工具包目前已经成熟并广泛可用。

- **问：为什么深度 Q-learning需要两个网络?**

  答： 根本原因是为了简化 (8.37) 中梯度的计算。由于 w 不仅出现在 $\hat{q}(S,A,w)$ 中，还出现在 $R+\gamma\max_{a\in\mathcal{A}(s')}\hat{q}(S',a,w)$ 中，因此计算关于 w 的梯度并非易事。一方面，如果我们固定 $R+\gamma\max_{a\in\mathcal{A}(s')}\hat{q}(S',a,w)$ 中的 w，则可以像 (8.38) 中那样轻松计算梯度。这个梯度表明应该维护两个网络。主网络的参数在每次迭代中都会更新。**目标网络的参数在一定时期内是固定的。另一方面，目标网络的参数不能永远固定。它应该每隔一定数量的迭代次数更新一次。**

- 问：这里的解释着重于解释了模型更新的问题，但是在目前的pytorch框架下，这部分原因是不是被解决，为什么大家还是这样用

这是一个非常深刻且直击 Reinforcement Learning (RL) 本质的问题。

简单直接的回答是：**在 PyTorch 中，计算包含目标项的完整梯度在技术上是完全可行的（Autograd 可以轻松做到），但这并不是因为框架无法处理才引入双网络。大家至今仍使用双网络（Target Network），是因为这关乎 RL 的收敛理论和训练稳定性，而非单纯的计算难度。**

那段文字的解释其实将“数学上的不稳定”稍微混淆为了“计算梯度的困难”。

以下是详细的深度解析，解释为什么 PyTorch 即使能算，我们也不让它算，以及为什么要用双网络。

---

1. 核心误区：是“算不出”还是“不能算”？

在 PyTorch 中，如果你写出如下 Loss：

$$L(w)=\Big((R+\gamma\max_{a'}Q(S',a';w))-Q(S,A;w)\Big)^2$$

如果你不使用 `.detach()`，PyTorch 会毫不犹豫地通过链式法则计算 w 在目标项（Target）中的梯度。

但是，Q-learning（以及大多数 TD 方法）在理论上属于“半梯度方法”（Semi-gradient Methods）。

这意味着在数学定义上，我们强制将目标值视为一个常数（Ground Truth），而不是一个变量。我们希望当前的 Q 值去逼近目标值，而不是让目标值也随着 Q 值的变化而移动。

如果你让梯度流过目标项（即计算全梯度），你实际上是在优化一个不同的目标函数（类似于 Residual Gradient 方法），这通常会导致学习效率极低甚至无法收敛。

2. 为什么必须用双网络（Target Network）？

即便我们使用 `.detach()` 切断了梯度，让一个网络既做预测又做目标（单网络）依然面临巨大的**稳定性问题**。这就是 DQN 引入 Target Network 的真正原因：

A. 移动目标问题（Moving Target Problem）

想象你在射箭（更新参数 w），靶子（Target Value）是你手中弓箭位置的函数。

- **单网络情况：** 每次你调整姿势（更新 w）想射中靶心，靶子的位置也立刻因为 w 的改变而发生了瞬移。**这会导致你一直在追逐一个不断变化的目标，网络容易发生震荡（Oscillation）甚至发散。**
- **双网络情况：** Target Network 相当于把靶子固定住一段时间（比如 1000 步）。你在这 1000 步里只管瞄准这个固定的靶子射。等练好了，再把靶子挪到新的位置（同步参数）。这极大地稳定了训练过程。

B. 数据相关性与自举放大的误差

- **自举（Bootstrapping）：** Q-learning 是用估计值去更新估计值。如果使用同一个网络，网络对 $$Q(S, A)$ $的高估（Overestimation）会直接通过目标公式 $R+\gamma\max Q(S',a')$ 传递并进一步放大。
- **解耦： 使用一个参数较旧的、更新缓慢的 Target Network，可以打断这种误差的快速正反馈循环。**

3. PyTorch 中的实际体现

在 PyTorch 的 DQN 代码实现中，这一逻辑体现得非常明显。并不是框架限制了我们，而是我们主动选择了这种写法。

错误的做法（数学上计算了全梯度，PyTorch 能跑但逻辑不对）：

Python

```
# 这在 PyTorch 中是可以运行的，但这是错误的 RL 算法
target = reward + gamma * model(next_state).max(1)[0]
prediction = model(state)
loss = (target - prediction) ** 2
loss.backward() # 梯度会同时流向 prediction 和 target 中的 model
```

单网络的正确做法（半梯度，但极其不稳定）：

Python

```
# 使用 .detach() 切断梯度，强行把 target 视为常数
target = reward + gamma * model(next_state).max(1)[0].detach()
prediction = model(state)
loss = (target - prediction) ** 2
loss.backward()
```

双网络的做法（DQN 标准做法，既切断梯度又稳定目标）：

Python

```
# 使用完全独立的 target_model 来计算目标with torch.no_grad(): # 显式告诉 PyTorch 这里不需要算梯度
    target = reward + gamma * target_model(next_state).max(1)[0]

prediction = model(state) # 主网络
loss = (target - prediction) ** 2
loss.backward() # 梯度只更新 model，不更新 target_model
```