

5

蒙特卡洛方法

在这一章中，我们考虑第一类实用的估计价值函数并寻找最优策略的算法。与第 4 章不同，在这里我们并不假设拥有完备的环境知识。蒙特卡洛算法仅仅需要经验，即从真实或者模拟的环境交互中采样得到的状态、动作、收益的序列。从真实经验中进行学习是非常好的，因为它不需要关于环境动态变化规律的先验知识，却依然能够达到最优的行为。从模拟经验中学习也是同样有效的，尽管这时需要一个模型，但这个模型只需要能够生成状态转移的一些样本，而不需要像动态规划 (DP) 算法那样生成所有可能转移的概率分布。在绝大多数情况下，虽然很难得到显式的分布，但从希望得到的分布进行采样却很容易。

蒙特卡洛算法通过平均样本的回报来解决强化学习问题。为了保证能够得到具有良好定义的回报，这里我们只定义用于分幕式任务的蒙特卡洛算法。在分幕式任务中，我们假设一段经验可以被分为若干个幕，并且无论选取怎样的动作整个幕一定会终止。价值估计以及策略改进在整个幕结束时才进行。因此蒙特卡洛算法是逐幕做出改进的，而非在每一步 (在线) 都有改进。通常，术语“蒙特卡洛”泛指任何包含大量随机成分的估计方法。在这里我们用它特指那些对完整的回报取平均的算法 (而非在下一章中介绍的从部分回报中学习的算法)。

第 2 章中的赌博机算法采样并平均每个动作的收益，蒙特卡洛算法与之类似，采样并平均每一个“状态-动作”二元组的回报。这里主要的区别在于，我们现在有多个状态，每一个状态都类似于一个不同的赌博机问题 (如关联搜索或者上下文相关的赌博机)，并且这些不同的赌博机问题是相互关联的。换言之，在某个状态采取动作之后的回报取决于在同一个幕内后来的状态中采取的动作。由于所有动作的选择都在随时刻演进而不断地学习，所以从较早状态的视角来看，这个问题是非平稳的。

为了处理其非平稳性，我们采用了类似于第 4 章中研究 DP 时提到的广义策略迭

代 (GPI) 算法的思想。当时我们从马尔可夫决策过程的知识中计算价值函数，而现在我们从马尔可夫决策过程采样样本的经验回报中学习价值函数。价值函数和其对应的策略依然以同样的方式 (GPI) 保证其最优性。与在 DP 的章节中一样，我们首先考虑预测问题 (对任意一个固定的策略 π 计算其 v_π 与 q_π)，然后再考虑策略的改进问题，最后讨论控制问题以及通过 GPI 所得到的解。DP 中的所有思想都会拓展到蒙特卡洛方法中，唯一的区别在于，这里我们只有通过采样获得的经验。

5.1 蒙特卡洛预测

我们首先考虑如何在给定一个策略的情况下，用蒙特卡洛算法来学习其状态价值函数。我们已经知道一个状态的价值是从该状态开始的期望回报，即未来的折扣收益累积值的期望。那么一个显而易见的方法是根据经验进行估计，即对所有经过这个状态之后产生的回报进行平均。随着越来越多的回报被观察到，平均值就会收敛于期望值。这一想法是所有蒙特卡洛算法的基础。

特别是，假设给定在策略 π 下途经状态 s 的多幕数据，我们想估计策略 π 下状态 s 的价值函数 $v_\pi(s)$ 。在给定的某一幕中，每次状态 s 的出现都称为对 s 的一次访问。当然，在同一幕中， s 可能会被多次访问到。在这种情况下，我们称第一次访问为 s 的首次访问。首次访问型 MC 算法用 s 的所有首次访问的回报的平均值估计 $v_\pi(s)$ ，而每次访问型 MC 算法则使用所有访问的回报的平均值。这两种蒙特卡洛 (MC) 算法十分相似但却有着不同的理论基础。“首次访问型 MC 算法”从 20 世纪 40 年代开始就被人们进行了大量的研究，是我们这一章主要关注的算法。“每次访问型 MC 算法”则能更自然地扩展到函数近似与资格迹方法，我们将分别在第 9 章和第 12 章中介绍这两个概念。下框中展示的是“首次访问型 MC 算法”的伪代码。除了无须检查 S_t 是否在当前幕的早期时段出现过之外，“每次访问型 MC 算法”的流程与此相同。

当 s 的访问次数 (或首次访问次数) 趋向无穷时，首次访问型 MC 和每次访问型 MC 均会收敛到 $v_\pi(s)$ 。对于首次访问型 MC 来说，这个结论是显然的。算法中的每个回报值都是对 $v_\pi(s)$ 的一个独立同分布的估计，且估计的方差是有限的。根据大数定理，这一平均值的序列会收敛到它们的期望值。每次平均都是一个无偏估计，其误差的标准差以 $1/\sqrt{n}$ 衰减，这里的 n 是被平均的回报值的个数。在每次访问型 MC 中，这个结论就没有这么显然，但它也会二阶收敛到 $v_\pi(s)$ (Singh 和 Sutton, 1996)。

首次访问型 MC 预测算法，用于估计 $V \approx v_\pi$

输入：待评估的策略 π

初始化：

对所有 $s \in \mathcal{S}$, 任意初始化 $V(s) \in \mathbb{R}$

对所有 $s \in \mathcal{S}$, $Returns(s) \leftarrow$ 空列表

无限循环 (对每幕)：

根据 π 生成一幕序列： $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

对本幕中的每一步进行循环， $t = T-1, T-2, \dots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

除非 S_t 在 S_0, S_1, \dots, S_{t-1} 中已出现过：

将 G 加入 $Returns(S_t)$

$V(S_t) \leftarrow \text{average}(Returns(S_t))$

下面我们通过一个例子来讲解如何应用蒙特卡洛算法。

例 5.1 二十一点 二十一点是一种流行于赌场的游戏，其目标是使得你的扑克牌点数之和在不超过 21 的情况下越大越好。所有的人头牌 (J、Q、K) 的点数为 10，A 既可当作 1 也可当作 11。假设每一个玩家都独立地与庄家进行比赛。游戏开始时会给各玩家与庄家发两张牌。庄家的牌一张正面朝上一张背面朝上。玩家直接获得 21 点 (一张 A 与一张 10) 的情况称之为天和。此时玩家直接获胜，除非庄家也是天和，那就是平局。如果玩家不是天和，那么他可以一张一张地继续要牌，直到他主动停止 (停牌) 或者牌的点数和超过 21 点 (爆牌)。如果玩家爆牌了就算输掉比赛。如果玩家选择停牌，就轮到庄家行动。庄家根据一个固定的策略进行游戏：他一直要牌，直到点数等于或超过 17 时停牌。如果庄家爆牌，那么玩家获胜，否则根据谁的点数更靠近 21 决定胜负或平局。

二十一点是一个典型的分幕式有限马尔可夫决策过程。可以将每一局看作一幕。胜、负、平局分别获得收益 +1、-1 和 0。每局游戏进行中的收益都为 0，并且不打折扣 ($\gamma = 1$)；所以最终的收益即为整个游戏的回报。玩家的动作是“要牌”或“停牌”。状态则取决于玩家的牌与庄家显示的牌。我们假设所有的牌来自无穷多的一组牌（即每次取出的牌都会再放回牌堆），因此就没有必要记下已经抽了哪些牌。如果玩家手上有一张 A，可以视作 11 且不爆牌，那么称这张 A 为可用的。此时这张牌总会被视作 11，因为如果视作 1 的话，点数总和必定小于等于 11，而这时玩家就无须进行选择，可以直接要牌，因为抽牌几乎一定会更优。所以，玩家做出的选择只会依赖于三个变量：他手牌的总和 (12~21)，庄家显示的牌 (A~10)，以及他是否有可用的 A。这样共计就有 200 个状态。

考虑下面这种策略, 玩家在手牌点数之和小于 20 时要牌, 否则停牌。为了通过蒙特卡洛法计算这种策略的状态价值函数, 需要根据该策略模拟很多次二十一点游戏, 并且计算每一个状态的回报的平均值。我们得到了如图 5.1 所示的状态价值函数。有可用的 A 的状态的估计会更不确定、不规律, 因为这样的状态更加罕见。无论是哪种情况, 在大约 500 000 局游戏后, 价值函数都能很好地近似。

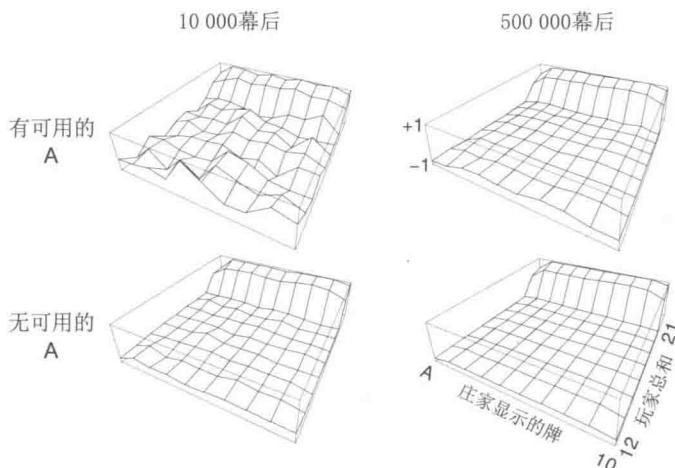


图 5.1 当玩二十一点游戏的策略为只在点数和为 20 或 21 停牌时, 通过蒙特卡洛策略评估计算得到的近似状态价值函数 ■

练习 5.1 考虑图 5.1 右侧的图。为什么估计的价值函数在最后两行突然增高? 为什么它在最靠左侧的一列降低? 为什么上方图中靠前的值比下方图中对应位置要高? □

练习 5.2 假如我们在解决二十一点问题时使用的不是首次访问型蒙特卡洛算法而是每次访问型蒙特卡洛算法, 那么你觉得结果会非常不一样吗? 为什么? □

即使我们完全了解这个任务的环境知识, 使用 DP 方法来计算价值函数也不容易。因为 DP 需要知道下一个时刻的概率分布, 它需要知道表示环境动态的四元函数 p , 而这在二十一点问题中是很难得到的。比如, 假设玩家手牌的点数为 14, 然后选择停牌。如果将玩家最后得到 +1 的收益的概率视作庄家显示的牌的函数, 那么该如何进行求解? 在 DP 之前必须计算得到所有的这些概率, 然而这些计算通常既复杂又容易出错。相反, 通过蒙特卡洛算法生成一些游戏的样本则非常容易, 且这种情况很常见。蒙特卡洛算法只需利用若干幕采样序列就能计算的特性是一个巨大的优势, 即便我们已知整个环境的动态特性也是如此。

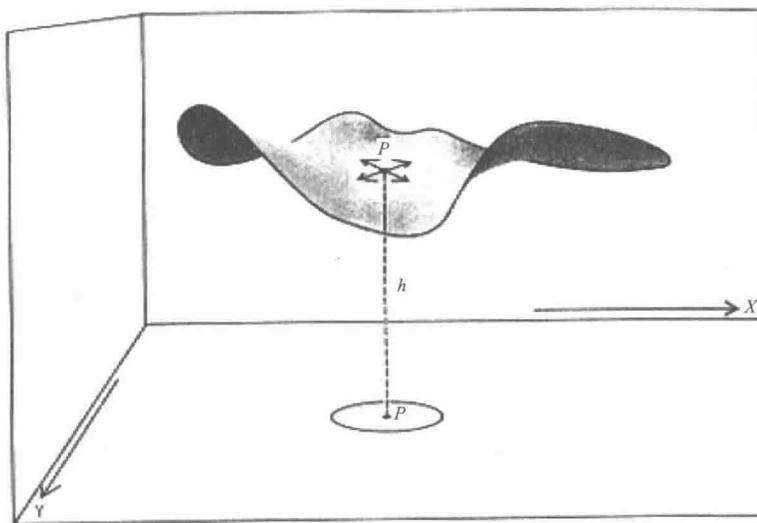
我们是否能够将回溯图的思想应用到蒙特卡洛算法中呢? 回溯图的基本思想是顶部为

待更新的根节点，下面则是中间与叶子节点，它们的收益和近似值都会在更新时用到。

如右图所示，在蒙特卡洛算法中估计 v_π 时，根为一个状态节点，然后往下是某幕样本序列一直到终止状态为止的完整轨迹，其中包含该幕中的全部状态转移。DP 的回溯图（第 57 页）显示了所有可能的转移，而蒙特卡洛算法则仅仅显示在当前幕中采样到的那些转移。DP 的回溯图仅仅包含一步转移，而蒙特卡洛算法则包含了到这一幕结束为止的所有转移。回溯图上的这些区别清楚地体现了这两种算法之间的本质区别。

蒙特卡洛算法的一个重要的事实是：对于每个状态的估计是独立的。它对于一个状态的估计完全不依赖于对其他状态的估计，这与 DP 完全不同。换言之，蒙特卡洛算法并没有使用我们在之前章节中描述的自举思想。

特别需要注意的是，计算一个状态的代价与状态的个数是无关的。这使得蒙特卡洛算法适合在仅仅需要获得一个或者一个子集的状态的价值函数时使用。我们可以从这个特定的状态开始采样生成一些幕序列，然后获取回报的平均值，而完全不需要考虑其他的状态。这是蒙特卡洛方法相比于 DP 的第三个优势（另外两个优势是：可以从实际经历和模拟经历中学习）。



线框上的肥皂泡

原图来自 Hersh 和 Griego (1969)。转载许可。版权所有 (1969) Scientific American, a division of Nature America, Inc. 保留所有权利。

例 5.2 肥皂泡 假设一个闭合的线框浸泡在肥皂水中，沿着线框的边缘形成了一个肥皂

泡。如果线框的几何形状不规则但是已知，我们该如何计算肥皂泡表面的形状呢？这个形状满足任意一点受到相邻部分施加的外力和为零（否则形状会发生改变）的条件。这意味着在表面上的任意一点，其高度等于它邻域的其他点的高度的平均值。此外，表面的边界必须恰好与线框重合。这个问题通常的解法是将整个表面网格化，然后通过迭代计算每个网格的高度。在线框上的网格的高度固定为线框的高度，其他的点则调整为四个相邻网格高度的平均值。就像 DP 的策略迭代一样，不停迭代这个过程，最终会收敛到一个所求表面的近似值。

蒙特卡洛算法最早用来解决的问题类型与这个问题十分相似。我们可以不进行上述的迭代计算，而是在表面上选一个网格点开始随机行走，每一步等概率地走到相邻的四个网格点之一，直到到达边界。这样到达的边界点的高度的期望值就是起点高度的一个近似（事实上，之前迭代方法计算的也是此值）。因此，某个网格点的高度可以通过从此点开始的多次随机行走的终点的高度的平均值来近似。如果人们只对一个点，或者一小部分点组成的集合比较感兴趣，那么这种蒙特卡洛法比起迭代法更加有效率。 ■

5.2 动作价值的蒙特卡洛估计

如果无法得到环境的模型，那么计算动作的价值（“状态-动作”二元组的价值，也即动作价值函数的值）比起计算状态的价值更加有用一些。在有模型的情况下，单靠状态价值函数就足以确定一个策略：用在 DP 那一章讲过的方法，我们只需要简单地向前看一步，选取特定的动作，使得当前收益与后继状态的状态价值函数之和最大即可。但在没有模型的情况下，仅有状态价值函数是不够的。我们必须通过显式地确定每个动作的价值函数来确定一个策略。所以这里我们主要的目标是使用蒙特卡洛算法确定 q_* 。因此我们首先考虑动作价值函数的策略评估问题。

动作价值函数的策略评估问题的目标就是估计 $q_\pi(s, a)$ ，即在策略 π 下从状态 s 采取动作 a 的期望回报。只需将对状态的访问改为对“状态-动作”二元组的访问，蒙特卡洛算法就可以用几乎和之前完全相同的方式来解决此问题。如果在某一幕中状态 s 被访问并在这个状态中采取了动作 a ，我们就称“状态-动作”二元组 (s, a) 在这一幕中被访问到。每次访问型 MC 算法将所有“状态-动作”二元组得到的回报的平均值作为价值函数的近似；而首次访问型 MC 算法则将每幕第一次在这个状态下采取这个动作得到的回报的平均值作为价值函数的近似。和之前一样，在对每个“状态-动作”二元组的访问次数趋向无穷时，这些方法都会二次收敛到动作价值函数的真实期望值。

这里唯一的复杂之处在于一些“状态-动作”二元组可能永远不会被访问到。如果 π 是一个确定性的策略，那么遵循 π 意味着在每一个状态中只会观测到一个动作的回报。在无法获取回报进行平均的情况下，蒙特卡洛算法将无法根据经验改善动作价值函数的估计。这个问题很严重，因为学习动作价值函数就是为了帮助在每个状态的所有可用的动作之间进行选择。为了比较这些动作，我们需要估计在一个状态中可采取的所有动作的价值函数，而不仅仅是当前更偏好的某个特定动作的价值函数。

如同在第 2 章的 k 胳膊赌博机问题中提及的一样，这是一个如何保持试探的普遍问题。为了实现基于动作价值函数的策略评估，我们必须保证持续的试探。一种方法是将指定的“状态-动作”二元组作为起点开始一幕采样，同时保证所有“状态-动作”二元组都有非零的概率可以被选为起点。这样就保证了在采样的幕个数趋向于无穷的时候，每一个“状态-动作”二元组都会被访问到无数次。我们把这种假设称为试探性出发。

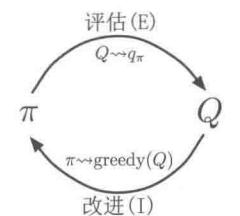
试探性出发假设有时非常有效，但当然也并非总是那么可靠。特别是当直接从真实环境中进行学习时，这个假设就很难满足了。作为替代，另一种常见的确保每一个“状态-动作”二元组被访问到的方法是，只考虑那些在每个状态下所有动作都有非零概率被选中的随机策略。我们将在后文讨论此方法的两个重要的变种。现在，我们先保留试探性出发假设来完成完整的蒙特卡洛控制算法的展示。

练习 5.3 计算 q_π 的蒙特卡洛估计的回溯图是什么样的？ □

5.3 蒙特卡洛控制

现在已经可以考虑如何使用蒙特卡洛估计来解决控制问题了，即如何近似最优的策略。基本的思想是采用在 DP 章节中介绍的广义策略迭代 (GPI)。在 GPI 中，我们同时维护一个近似的策略和近似的价值函数。价值函数会不断迭代使其更加精确地近似对应当前策略的真实价值函数，而当前的策略也会根据当前的价值函数不断调优，右图显示了这一过程。这两个过程在一定程度上会相互影响，因为它们互相为对方确定了一个变化的优化目标，但它们整体会使得策略与价值函数趋向最优解。

首先，我们讨论经典策略迭代算法的蒙特卡洛版本。这种方法从任意的策略 π_0 开始交替进行完整的策略评估和策略改进，最终得到最优的策略和动作价值函数



$$\pi_0 \xrightarrow{E} q_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} q_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \cdots \xrightarrow{I} \pi_* \xrightarrow{E} q_*,$$

这里, \xrightarrow{E} 表示策略评估, 而 \xrightarrow{I} 表示策略改进。策略评估完全按照前一节所述的方法进行。经历了很多幕后, 近似的动作价值函数会渐近地趋向真实动作价值函数。现在假设我们观测到了无限多幕的序列, 并且这些幕保证了试探性出发假设。在这种情况下, 对于任意的 π_k , 蒙特卡洛算法都能精确地计算对应的 q_{π_k} 。

策略改进的方法是在当前价值函数上贪心地选择动作。由于我们有动作价值函数, 所以在贪心的时候完全不需要使用任何的模型信息。对于任意的一个动作价值函数 q , 对应的贪心策略为: 对于任意一个状态 $s \in \mathcal{S}$, 必定选择对应动作价值函数最大的动作

$$\pi(s) \doteq \arg \max_a q(s, a). \quad (5.1)$$

策略改进可以通过将 q_{π_k} 对应的贪心策略作为 π_{k+1} 来进行。这样的 π_k 和 π_{k+1} 满足策略改进定理 (4.2 节), 因为对所有的状态 $s \in \mathcal{S}$,

$$\begin{aligned} q_{\pi_k}(s, \pi_{k+1}(s)) &= q_{\pi_k}(s, \arg \max_a q_{\pi_k}(s, a)) \\ &= \max_a q_{\pi_k}(s, a) \\ &\geq q_{\pi_k}(s, \pi_k(s)) \\ &\geq v_{\pi_k}(s). \end{aligned}$$

我们在之前章节中讨论过, 这个定理保证 π_{k+1} 一定比 π_k 更优, 除非 π_k 已经是最优策略, 这种情况下两者均为最优策略。这个过程反过来保证了整个流程一定收敛到最优的策略和最优的价值函数。这样, 在只能得到若干幕采样序列而不知道环境动态知识时, 蒙特卡洛算法就可以用来寻找最优策略。

我们在上文提出了两个很强的假设来保证蒙特卡洛算法的收敛。一个是试探性出发假设, 另一个是在进行策略评估的时候有无限多幕的样本序列进行试探。为了得到一个实际可用的算法, 我们必须去除这两个假设。第一个假设在本章后面讨论。

首先我们先讨论在进行策略评估时可以观测到无限多幕样本序列这一假设。这个假设比较容易去除。事实上, 在经典 DP 算法, 比如迭代策略评估中也出现了同样的问题, 它的结果也仅仅是渐近地收敛于真实的价值函数。无论是 DP 还是蒙特卡洛算法, 有两个方法可以解决这一问题。一种方法是想方设法在每次策略评估中对 q_{π_k} 做出尽量好的逼近。这就需要做一些假设并定义一些测度, 来分析逼近误差的幅度和出现概率的上下界, 然后采取足够多的步数来保证这些界足够小。这种方法可以保证收敛到令人满意的近似水平。然而在实际使用中, 即使问题规模很小, 这种方法也可能需要有大量的幕序列以用于计算。

第二种避免无限多幕样本序列假设的方法是不再要求在策略改进前就完成策略评估。在每一个评估步骤中，我们让动作价值函数逼近 q_{π_k} ，但我们并不期望它在经过很多步之前非常接近真实的值。在 4.6 节中我们首次介绍 GPI 时就提及了这种思想。这种思想的一种极端实现形式就是价值迭代，即在相邻的两步策略改进中只进行一次策略评估，而不要求多次迭代后的收敛。价值迭代的“就地更新”版本则更加极端：在单个状态中交替进行策略的改进与评估。

对于蒙特卡洛策略迭代，自然可以逐幕交替进行评估与改进。每一幕结束后，使用观测到的回报进行策略评估，然后在该幕序列访问到的每一个状态上进行策略的改进。使用这个思路的一个简单的算法，称作基于试探性出发的蒙特卡洛（蒙特卡洛 ES），下框中给出了这个算法。

蒙特卡洛 ES (试探性出发)，用于估计 $\pi \approx \pi_*$

初始化：

对所有 $s \in \mathcal{S}$, 任意初始化 $\pi(s) \in \mathcal{A}(s)$

对所有 $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$, 任意初始化 $Q(s, a) \in \mathbb{R}$

对所有 $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$, $Returns(s, a) \leftarrow$ 空列表

无限循环 (对每幕)：

选择 $S_0 \in \mathcal{S}$ 和 $A_0 \in \mathcal{A}(S_0)$ 以使得所有“状态-动作”二元组的概率都 > 0

从 S_0, A_0 开始根据 π 生成一幕序列： $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

对幕中的每一步循环， $t = T-1, T-2, \dots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

除非二元组 S_t, A_t 在 $S_0, A_0, S_1, A_1, \dots, S_{t-1}, A_{t-1}$ 中已出现过：

将 G 加入 $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a)$

练习 5.4 该框中的蒙特卡洛 ES 伪代码效率不高。因为对于每一个“状态-动作”二元组，它都需要维护一个列表存放所有的回报值，然后反复地计算它们的平均值。我们其实可以采用更高效的方法来计算，类似 2.4 节中介绍的，我们仅需维护一个平均值和一个统计量（对每一个“状态-动作”二元组），然后增量式地去更新就可以了。描述一下如何修改伪代码来实现这个更高效的算法。 \square

在蒙特卡洛 ES 中，无论之前遵循的是哪一个策略，对于每一个“状态-动作”二元组的所有回报都被累加并平均。可以很容易发现，蒙特卡洛 ES 不会收敛到任何一个次优策

略。因为如果真的收敛到次优策略，其价值函数一定会收敛到该策略对应的价值函数，而在这种情况下还会得到一个更优的策略。只有在策略和价值函数都达到最优的情况下，稳定性才能得到保证。因为动作价值函数的变化随着时间增加而不断变小，所以应该一定能收敛到这个不动点，然而这一点还没有得到严格的证明。我们认为这是强化学习基础理论中最重要的开放问题之一（部分解法可以参见 Sitsiklis, 2002）。

例 5.3 解决二十一点问题 使用蒙特卡洛 ES 可以很直接地解决二十一点问题。由于所有幕都是模拟的游戏，因此可以很容易地使所有可能的起点概率都不为零，确保试探性出发假设成立。在这个例子中，只需要简单地随机等概率选择庄家的扑克牌、玩家手牌的点数，以及确定是否有可用的 A 即可。我们使用之前的例子中提及的策略作为初始策略：即只在 20 或 21 点停牌。初始的动作价值函数可以全部为零。图 5.2 显示了蒙特卡洛 ES 得出的二十一点游戏的最优策略和状态价值函数。这个策略与 Thorp (1966) 发现的基础策略相比，除了其中不存在可用的 A 外，其他完全相同。我们不确定造成这一差异的原因，但是我们相信这是我们描述的二十一点游戏版本的最优策略。

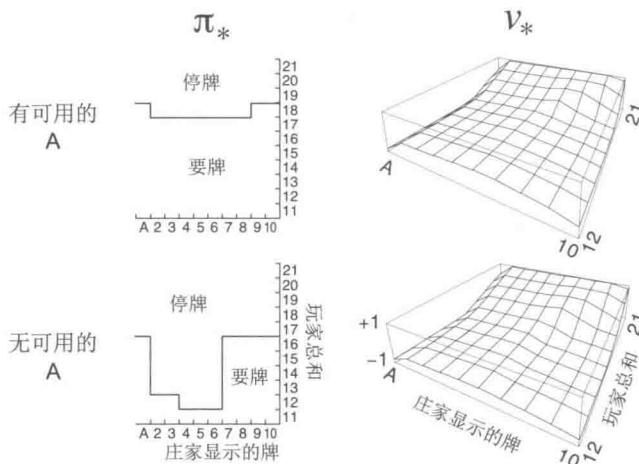


图 5.2 通过蒙特卡洛 ES 得到的二十一点游戏的最优策略和状态价值函数。这里的状态价值函数是由通过蒙特卡洛 ES 得到的动作价值函数计算得到的 ■

5.4 没有试探性出发假设的蒙特卡洛控制

如何避免很难被满足的试探性出发假设呢？唯一的一般性解决方案就是智能体能够持续不断地选择所有可能的动作。有两种方法可以保证这一点，分别被称为同轨策略 (*on-policy*) 方法和离轨策略 (*off-policy*) 方法。在同轨策略中，用于生成采样数据序列

的策略和用于实际决策的待评估和改进的策略是相同的；而在离轨策略中，用于评估或者改进的策略与生成采样数据的策略是不同的，即生成的数据“离开”了待优化的策略所决定的决策序列轨迹。上文提及的蒙特卡洛 ES 算法是同轨策略方法的一个例子。这一节我们将介绍如何设计一个同轨策略的蒙特卡洛控制方法，使其不再依赖于不合实际的试探性出发假设。离轨策略方法将在下一节中介绍。

在同轨策略方法中，策略一般是“软性”的，即对于任意 $s \in \mathcal{S}$ 以及 $a \in \mathcal{A}(s)$ ，都有 $\pi(a|s) > 0$ ，但它们会逐渐地逼近一个确定性的策略。第2章中提到的很多方法都能做到这一点。在这一节中我们介绍的同轨策略方法称为 ε -贪心策略，意思是在绝大多数时候都采取获得最大估计值的动作价值函数所对应的动作，但同时以一个较小的 ε 概率随机选择一个动作。即对于所有的非贪心动作，都有 $\frac{\varepsilon}{|\mathcal{A}(s)|}$ 的概率被选中，剩余的大部分的概率为 $1 - \varepsilon + \frac{\varepsilon}{|\mathcal{A}(s)|}$ ，这是选中贪心动作的概率。这种 ε -贪心策略是 ε -软性策略的一个例子，即对某个 $\varepsilon > 0$ ，所有的状态和动作都有 $\pi(a|s) \geq \frac{\varepsilon}{|\mathcal{A}(s)|}$ 。在所有 ε -软性策略中， ε -贪心策略在某种层面上是最接近贪心策略的。

同轨策略算法的蒙特卡洛控制的总体思想依然是 GPI。如同蒙特卡洛 ES 一样，我们使用首次访问型 MC 算法来估计当前策略的动作价值函数。然而，由于缺乏试探性出发假设，我们不能简单地通过对当前价值函数进行贪心优化来改进策略，否则就无法进一步试探非贪心的动作。幸运的是，GPI 并不要求优化过程中所遵循的策略一定是贪心的，只需要它逐渐逼近贪心策略即可。在我们的同轨策略算法中，我们仅仅改为遵循 ε -贪心策略。对于任意一个 ε -软性策略 π ，根据 q_π 生成的任意一个 ε -贪心策略保证优于或等于 π 。完整的算法在下框中给出。

同轨策略的首次访问型 MC 控制算法 (对于 ε -软性策略)，用于估计 $\pi \approx \pi_*$

算法参数：很小的 $\varepsilon > 0$

初始化：

$\pi \leftarrow$ 一个任意的 ε -软性策略

对所有 $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$, 任意初始化 $Q(s, a) \in \mathbb{R}$

对所有 $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$, $Returns(s, a) \leftarrow$ 空列表

无限循环 (对每幕)：

根据 π 生成一幕序列： $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

对幕中的每一步循环， $t = T-1, T-2, \dots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

除非“状态-动作”二元组 S_t, A_t 在 $S_0, A_0, S_1, A_1, \dots, S_{t-1}, A_{t-1}$ 已出

现过：

把 G 加入 $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$A^* \leftarrow \arg\max_a Q(S_t, a)$ (有多个最大值时任意选取)

对所有 $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

根据策略改进定理，对于一个 ε -软性策略 π ，任何一个根据 q_π 生成的 ε -贪心策略都是对其的一个改进。假设 π' 是一个 ε -贪心策略。策略改进定理成立，因为对任意的 $s \in \mathcal{S}$ ，

$$\begin{aligned} q_\pi(s, \pi'(s)) &= \sum_a \pi'(a|s) q_\pi(s, a) \\ &= \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a) + (1 - \varepsilon) \max_a q_\pi(s, a) \\ &\geq \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a) + (1 - \varepsilon) \sum_a \frac{\pi(a|s) - \frac{\varepsilon}{|\mathcal{A}(s)|}}{1 - \varepsilon} q_\pi(s, a) \end{aligned} \quad (5.2)$$

(由于期望累加和是一个用和为 1 的非负权重进行的加权平均值，所以一定小于等于其中的最大值)。

$$\begin{aligned} &= \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a) - \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a) + \sum_a \pi(a|s) q_\pi(s, a) \\ &= v_\pi(s). \end{aligned}$$

所以，根据策略改进定理， $\pi' \geq \pi$ ，即对于任意的 $s \in \mathcal{S}$ ，满足 $v_{\pi'}(s) \geq v_\pi(s)$ 。下面证明该式的等号成立的条件是：当且仅当 π' 和 π 都为最优的 ε -软性策略，即它们都比所有其他的 ε -软性策略更优或相同。

设想一个与原先环境几乎相同的新环境，唯一的区别是我们将策略是 ε -软性这一要求“移入”环境中。新的环境与旧的环境有相同的动作与状态集，并表现出如下的行为。在状态 s 采取动作 a 时，新环境有 $1 - \varepsilon$ 的概率与旧环境的表现完全相同。然后以 ε 的概率重新等概率选择一个动作，然后有和旧环境采取这一新的随机动作一样的表现。在这个新环境中的最优策略的表现和旧环境中最优的 ε -软性策略的表现是相同的。令 \tilde{v}_* 和 \tilde{q}_* 为新环境的最优价值函数，则当且仅当 $v_\pi = \tilde{v}_*$ 时，策略 π 是一个最优的 ε -软性策略。根据 \tilde{v}_* 的定义，我们知道它是下式的唯一解

$$\tilde{v}_*(s) = (1 - \varepsilon) \max_a \tilde{q}_*(s, a) + \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a \tilde{q}_*(s, a)$$

$$\begin{aligned}
&= (1 - \varepsilon) \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma \tilde{v}_*(s')] \\
&\quad + \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a \sum_{s', r} p(s', r | s, a) [r + \gamma \tilde{v}_*(s')].
\end{aligned}$$

当等式成立且 ε -软性策略 π 无法再改进的时候，由公式 (5.2) 我们有

$$\begin{aligned}
v_\pi(s) &= (1 - \varepsilon) \max_a q_\pi(s, a) + \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a) \\
&= (1 - \varepsilon) \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_\pi(s')] \\
&\quad + \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_\pi(s')].
\end{aligned}$$

然而，除了下标为 v_π 而不是 \tilde{v}_* 以外，这个等式和上一个完全相同。由于 \tilde{v}_* 是唯一解，因此一定有 $v_\pi = \tilde{v}_*$ 。

实际上，这里我们已经介绍了如何让策略迭代适用于 ε -软性策略。将贪心策略的原生概念扩展到 ε -软性策略上，除了得到最优的 ε -软性策略外，每一步都能保证策略有改进。尽管这个分析假设了动作价值函数可以被精确计算，然而它与动作价值函数是如何计算的无关。这样我们就得到了和上一节几乎相同的结论。虽然现在我们只能获得 ε -软性策略集合中的最优策略，但我们已经不再需要试探性出发假设了。

5.5 基于重要度采样的离轨策略

所有的学习控制方法都面临一个困境：它们希望学到的动作可以使随后的智能体行为是最优的，但是为了搜索所有的动作（以保证找到最优动作），它们需要采取非最优的行动。如何在遵循试探策略采取行动的同时学习到最优策略呢？在前面小节中提到的同轨策略方法实际上是一种妥协——它并不学习最优策略的动作值，而是学习一个接近最优而且仍能进行试探的策略的动作值。一个更加直接的方法是干脆采用两个策略，一个用来学习并最终成为最优策略，另一个更加有试探性，用来产生智能体的行动样本。用来学习的策略被称为目标策略，用于生成行动样本的策略被称为行动策略。在这种情况下，我们认为学习所用的数据“离开”了待学习的目标策略，因此整个过程被称为离轨策略学习。

在本书的后面章节我们同时考虑同轨策略和离轨策略方法。同轨策略方法通常更简单，更容易想到。离轨策略方法则需要一些额外的概念与记号，而且因为其数据来自一个不同的策略，所以离轨策略方法方差更大，收敛更慢。但另一方面，离轨策略方法更强

大，更通用。也可将同轨策略方法视为一种目标策略与行动策略相同的离轨策略方法的特例。离轨策略方法在实际应用中有许多其他用途。例如，它常用来学习通过传统的非学习型控制器或人类专家生成的数据。离轨策略学习也被视作学习外部世界动态特性的多步预测模型中的关键思想（参见 17.2 节；Sutton, 2009; Sutton et al., 2011）。

在本节中，我们通过讨论预测问题来开始对离轨策略方法的学习，在这个实例中，目标策略和行动策略都是固定的。假设我们希望预测 v_π 或 q_π ，但是我们只有遵循另一个策略 b ($b \neq \pi$) 所得到的若干幕样本。在这种情况下， π 是目标策略， b 是行动策略，两个策略都固定且已知。

为了使用从 b 得到的多幕样本序列去预测 π ，我们要求在 π 下发生的每个动作都至少偶尔能在 b 下发生。换句话说，对任意 $\pi(a|s) > 0$ ，我们要求 $b(a|s) > 0$ 。我们称其为覆盖假设。根据这个假设，在与 π 不同的状态中， b 必须是随机的。另一方面，目标策略 π 则可能是确定性的。实际上，这种情况是控制应用中很有趣的一类问题。在控制过程中，目标策略通常是一个确定性的贪心策略，它由动作价值函数的当前估计值所决定。而当行动策略是随机的且具有试探性时（例如可使用 ϵ -贪心策略），这个策略会成为一个确定性的最优策略。不过在本节中，我们仅仅讨论当 π 不变且已知时的“预测问题”。

几乎所有的离轨策略方法都采用了重要度采样，重要度采样是一种在给定来自其他分布的样本的条件下，估计某种分布的期望值的通用方法。我们将重要度采样应用于离轨策略学习，对回报值根据其轨迹在目标策略与行动策略中出现的相对概率进行加权，这个相对概率也被称为重要度采样比。给定起始状态 S_t ，后续的状态-动作轨迹 $A_t, S_{t+1}, A_{t+1}, \dots, S_T$ 在策略 π 下发生的概率是

$$\begin{aligned} & \Pr\{A_t, S_{t+1}, A_{t+1}, \dots, S_T \mid S_t, A_{t:T-1} \sim \pi\} \\ &= \pi(A_t|S_t)p(S_{t+1}|S_t, A_t)\pi(A_{t+1}|S_{t+1}) \cdots p(S_T|S_{T-1}, A_{T-1}) \\ &= \prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k), \end{aligned}$$

这里， p 是在公式 (3.4) 中定义的状态转移概率函数。因此，在目标策略和行动策略轨迹下的相对概率（重要度采样比）是

$$\rho_{t:T-1} = \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)}{\prod_{k=t}^{T-1} b(A_k|S_k)p(S_{k+1}|S_k, A_k)} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}. \quad (5.3)$$

尽管整体的轨迹概率值与 MDP 的状态转移概率有关，而且 MDP 的转移概率通常是未知的，但是它在分子和分母中是完全相同的，所以可被约分。最终，重要度采样比只与两个策略和样本序列数据相关，而与 MDP 的动态特性（状态转移概率）无关。

回想一下，之前我们希望估计目标策略下的期望回报（价值），但我们只有行动策略中的回报 G_t 。这些从行动策略中得到的回报的期望 $\mathbb{E}[G_t | S_t = s] = v_b(s)$ 是不准确的，所以不能用它们的平均来得到 v_π 。这里就要用到重要度采样了。使用比例系数 $\rho_{t:T-1}$ 可以调整回报使其有正确的期望值

$$\mathbb{E}[\rho_{t:T-1} G_t | S_t = s] = v_\pi(s). \quad (5.4)$$

现在我们已经做好了准备工作，下面我们给出通过观察到的一批遵循策略 b 的多幕采样序列并将其回报进行平均来预测 $v_\pi(s)$ 的蒙特卡洛算法。为了方便起见，我们在这里对时刻进行编号时，即使时刻跨越幕的边界，编号也递增。也就是说，如果这批多幕采样序列中的第一幕在时刻 100 时在某个终止状态结束，下一幕就起始于 $t = 101$ 。这样我们就可以使用唯一的时刻编号来指代特定幕中的特定时刻。特别地，对于每次访问型方法，我们可以定义所有访问过状态 s 的时刻集合为 $\mathcal{T}(s)$ 。而对于首次访问型方法， $\mathcal{T}(s)$ 只包含在幕内首次访问状态 s 的时刻。我们用 $T(t)$ 来表示在时刻 t 后的首次终止，用 G_t 来表示在 t 之后到达 $T(t)$ 时的回报值。那么 $\{G_t\}_{t \in \mathcal{T}(s)}$ 就是状态 s 对应的回报值， $\{\rho_{t:T(t)-1}\}_{t \in \mathcal{T}(s)}$ 是相应的重要度采样比。为了预测 $v_\pi(s)$ ，我们只需要根据重要度采样比来调整回报值并对结果进行平均即可

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{|\mathcal{T}(s)|}. \quad (5.5)$$

通过这样一种简单平均实现的重要度采样被称为普通重要度采样。

另一个重要的方法是加权重要度采样，它采用一种加权平均的方法，其定义为

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1}}, \quad (5.6)$$

如果分母为零，式 (5.6) 的值也定义为零。为了理解两种重要度采样方法的区别，我们讨论它们在首次访问型方法下，获得一个单幕回报之后的估计值。在加权平均的估计中，比例系数 $\rho_{t:T(t)-1}$ 在分子与分母中被约分，所以估计值就等于观测到的回报值，而与重要度采样比无关（假设比例系数非零）。考虑到这个回报值是仅有的观测结果，所以这是一个合理的估计，但它的期望是 $v_b(s)$ 而不是 $v_\pi(s)$ ，在统计学意义上这个估计是有偏的。作为对比，使用简单平均公式 (5.5) 得到的结果在期望上总是 $v_\pi(s)$ （是无偏的），但是其值可能变得很极端。假设比例系数是 10，这意味着，被观测到的决策序列轨迹遵循目标策略的可能性是遵循行动策略的 10 倍。在这种情况下，普通重要度采样的估计值会是观测到的回报值的 10 倍。也就是说，尽管该幕数据序列的轨迹应该可以非常有效地反映目标策略，但其估计值却会离观测到的回报值很远。

在数学上，两种重要度采样算法在首次访问型方法下的差异可以用偏差与方差来表示。普通重要度采样的估计是无偏的，而加权重要度采样的估计是有偏的（偏差值渐近收敛到零）。另一方面，普通重要度采样的方差一般是无界的，因为重要度采样比的方差是无界的。而在加权估计中任何回报的最大权值都是 1。其实如果假设回报值有界，那么即使重要度采样比的方差是无穷的，加权重要度采样估计的方差仍能收敛到零 (Precup、Sutton 和 Dasgupta, 2001)。在实际应用时，人们偏好加权估计，因为它的方差经常很小。尽管如此，我们不会完全放弃普通重要度采样，因为它更容易扩展到我们将在本书第 II 部分提到的基于函数逼近的近似方法。

在 5.7 节中，我们将给出一个使用加权重要度采样进行离轨策略估计的完整的每次访问型 MC 算法。

练习 5.5 考虑只有一个非终止状态和一个单一动作的马尔可夫决策过程，动作设定为：以概率 p 跳回非终止状态，以概率 $1 - p$ 转移到终止状态。令每一时刻所有转移的收益都是 +1，且 $\gamma = 1$ 。假设你观察到一幕序列持续了 10 个时刻，获得了值为 10 的回报。则对这个非终止状态的首次访问型和每次访问型的价值估计分别是什么？□

例 5.4 对二十一点游戏中的状态值的离轨策略估计 我们分别采用普通重要度采样和加权重要度采样，从离轨策略数据中估计单个二十一点游戏状态（见例 5.1）的值。之前的蒙特卡洛控制方法的优点之一是它们无须建立对其他状态的估计就能对单个状态进行估计。在这个例子中，我们评估这样一个状态，庄家露出一张牌 2，玩家的牌的和是 13，且玩家有一张可用的 A（也就是说，玩家有一张 A，一张 2，或者与之等价的情况有三张 A）。通过从这个状态开始等概率选择要牌或停牌（行动策略）可以得到采样数据。目标策略只在和达到 20 或 21 时停牌，这和在例 5.1 中一样。在目标策略中，这个状态的值大概

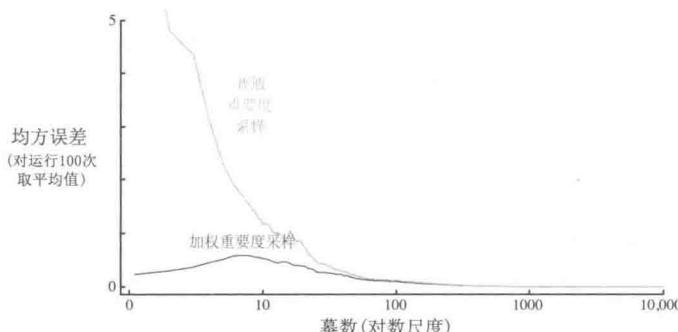


图 5.3 加权重要度采样可以从离轨策略数据中，对二十一点游戏中的状态的价值进行误差更小的估计（见例 5.3）

是 -0.27726 (这是利用目标策略独立生成 1 亿幕数据后对回报进行平均得到的)。两种离轨策略方法在采用随机策略经过 1000 幕离轨策略数据采样后都很好地逼近了这个值。为了验证结果的可靠性，我们独立运行了 100 次，每一次都从估计为零开始学习 10 000 幕。图 5.3 展示了得到的学习曲线——图中我们将每种方法的估计的均方误差视为幕数量的函数，并将运行 100 次的结果进行平均。两种算法的错误率最终都趋近于零，但是加权重要度采样在开始时错误率明显较低，这也是实践中的典型现象。■

例 5.5 无穷方差 普通重要度采样的估计的方差通常是无穷的，尤其当缩放过的回报值具有无穷的方差时，其收敛性往往不尽人意，而这种现象在带环的序列轨迹中进行离轨策略学习时很容易发生。图 5.4 显示了一个简单的例子。图中只有一个非终止状态 s 和两种动作：向左和向右。采取向右的动作将确定地向终止状态转移。采取向左的动作则有 0.9 的概率回到 s ，0.1 的概率转移到终止状态，后一种转移的收益是 $+1$ ，其他转移则都是 0。我们讨论总是选择向左的动作的目标策略。所有在这种策略下的采样数据都包含了一些回到 s 的转移，最终转移到终止状态并返回 $+1$ 。所以， s 在目标策略下的价值始终是 1 ($\gamma = 1$)。假设我们正从离轨策略数据中估计这个价值，并且该离轨策略中的行动策略以等概率选择向右或向左的动作。

图 5.4 的下部展示了基于普通重要度采样的首次访问型 MC 算法的 10 次独立运行的结果。即使在数百万幕后，预测值仍然没有收敛到正确的值 1。作为对比，加权重要度采样算法在第一次以向左的动作结束的幕之后就给出了 1 的预测。所有不等于 1 的回报 (即

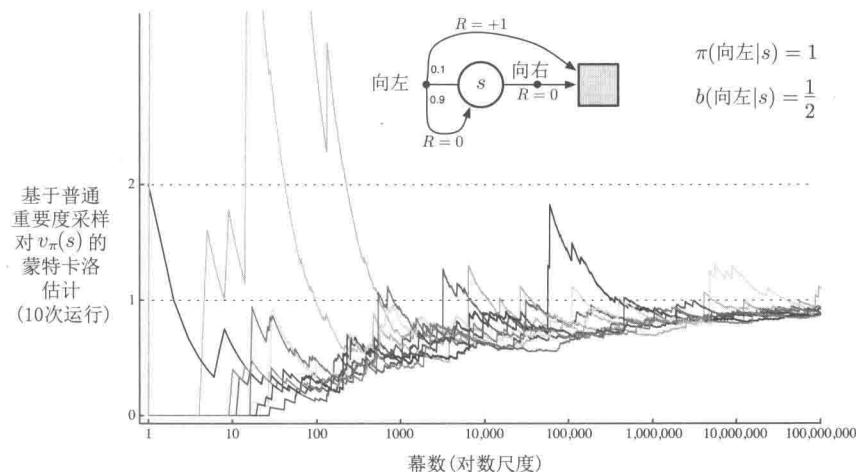


图 5.4 在图中的单状态 MDP 中 (见例 5.5)，普通重要度采样得到的估计值非常不稳定。这里正确的估计值是 1 ($\gamma = 1$)。尽管这确实是采样回报 (使用重要度采样) 的期望值，但这个采样的方差是无限大的，估计值并不会收敛到这个值。这些结果是通过离轨策略的首次访问型 MC 算法得到的

以向右的动作结束时得到的回报) 会与目标策略不符, 因此重要度采样比 $\rho_{t:T(t)-1}$ 的值为零, 对式 (5.6) 中的分子与分母没有任何贡献。加权重要度采样算法只对与目标策略一致的回报进行加权平均, 而所有这些回报值都是 1。

我们可以通过简单计算来确认, 在这个例子中经过重要度采样加权过的回报值的方差是无穷的。随机变量 X 的方差的定义是其相对于均值 \bar{X} 的偏差的期望值, 可以写为

$$\text{Var}[X] \doteq \mathbb{E}[(X - \bar{X})^2] = \mathbb{E}[X^2 - 2X\bar{X} + \bar{X}^2] = \mathbb{E}[X^2] - \bar{X}^2.$$

因此, 如果像在这个例子中这样, 均值是有限的, 那么当且仅当随机变量平方的期望是无穷时方差才为无穷。因此我们需要证明经过重要度采样加权过的回报的平方的期望是无穷

$$\mathbb{E}_b \left[\left(\prod_{t=0}^{T-1} \frac{\pi(A_t|S_t)}{b(A_t|S_t)} G_0 \right)^2 \right].$$

为了计算这个期望, 我们把它根据幕的长度和终止情况进行分类。首先注意到, 任意以向右的动作结束的幕, 其重要度采样比都是零, 因为目标策略永远不会采取这样的动作。这样的采样序列不会对期望产生贡献 (括号中的值等于零), 因此可以忽略。我们现在只需要考虑包含了一定数量 (可能是零) 的向左的动作返回到非终止状态, 然后再接着一个向左的动作转移到终止状态的幕。所有这些幕的回报值都是 1, 所以可以忽略 G_0 项。为了获得平方的期望, 我们只需要考虑每种长度的幕, 将每种长度的幕的出现概率乘以重要度采样比的平方, 再将它们加起来:

$$\begin{aligned} &= \frac{1}{2} \cdot 0.1 \left(\frac{1}{0.5} \right)^2 && \text{(长度为 1 的幕)} \\ &+ \frac{1}{2} \cdot 0.9 \cdot \frac{1}{2} \cdot 0.1 \left(\frac{1}{0.5} \frac{1}{0.5} \right)^2 && \text{(长度为 2 的幕)} \\ &+ \frac{1}{2} \cdot 0.9 \cdot \frac{1}{2} \cdot 0.9 \cdot \frac{1}{2} \cdot 0.1 \left(\frac{1}{0.5} \frac{1}{0.5} \frac{1}{0.5} \right)^2 && \text{(长度为 3 的幕)} \\ &+ \dots \\ &= 0.1 \sum_{k=0}^{\infty} 0.9^k \cdot 2^k \cdot 2 = 0.2 \sum_{k=0}^{\infty} 1.8^k = \infty. \end{aligned}$$

■

练习 5.6 给定用 b 生成的回报, 类比公式 (5.6), 用动作价值函数 $Q(s, a)$ 替代状态价值函数 $V(s)$, 得到的式子是什么? □

练习 5.7 实际在使用普通重要度采样时, 与图 5.3 所示的学习曲线一样, 错误率随着训练一般都是下降的。但是对于加权重要度采样, 错误率会先上升然后下降。为什么会这样? □

练习 5.8 在图 5.4 和例 5.5 的结果中采用了首次访问型 MC 方法。假设在同样的问题中采用了每次访问型 MC 方法。估计器的方差仍会是无穷吗？为什么？ \square

5.6 增量式实现

蒙特卡洛预测方法可以通过拓展第 2 章 (2.4 节) 中介绍的方法逐幕地进行增量式实现。我们在第 2 章中对收益进行平均，在蒙特卡洛方法中我们对回报进行平均。除此之外，在第 2 章中使用的方法可以以完全相同的方式用于同轨策略的蒙特卡洛方法中。对于离轨策略的蒙特卡洛方法，我们需要分别讨论使用普通重要度采样和加权重要度采样的方法。

在普通重要度采样的条件下，回报先乘上重要度采样比 $\rho_{t:T(t)-1}$ (式 5.3) 进行缩放再简单地取平均。对此，我们同样可以使用第 2 章中介绍的增量式方法，只要将第 2 章中的收益替换为缩放后的回报即可。下面需要讨论一下采用加权重要度采样的离轨策略方法。对此我们需要对回报加权平均，而且需要一个略微不同的增量式算法。

假设我们有一个回报序列 G_1, G_2, \dots, G_{n-1} ，它们都从相同的状态开始，且每一个回报都对应一个随机权重 W_i (例如， $W_i = \rho_{t:T(t)-1}$)。我们希望得到如下的估计，并且在获得了一个额外的回报值 G_n 时能保持更新。

$$V_n \doteq \frac{\sum_{k=1}^{n-1} W_k G_k}{\sum_{k=1}^{n-1} W_k}, \quad n \geq 2, \quad (5.7)$$

我们为了能不断跟踪 V_n 的变化，必须为每一个状态维护前 n 个回报对应的权值的累加和 C_n 。 V_n 的更新方法是

$$V_{n+1} \doteq V_n + \frac{W_n}{C_n} [G_n - V_n], \quad n \geq 1, \quad (5.8)$$

以及

$$C_{n+1} \doteq C_n + W_{n+1},$$

这里 $C_0 \doteq 0$ (V_1 是任意的，所以不用特别指定)。下方的框中包含了一个完整的用于蒙特卡洛策略评估的逐幕增量算法。这个算法表面上针对离轨策略的情况，采用加权重要度采样，但是对于同轨策略的情况同样适用，只需要选择同样的目标策略与行动策略即可 (这种情况下 $\pi = b$ ， W 始终为 1)。近似值 Q 收敛于 q_π (对于所有遇到的“状态-动作”二元组)，而动作则根据 b 进行选择，这个 b 可能是一个不同的策略。

练习 5.9 对使用首次访问型蒙特卡洛的策略评估 (5.1 节) 过程进行修改，要求采用 2.4 节中描述的对样本平均的增量式实现。 \square

离轨策略 MC 预测算法 (策略评估), 用于估计 $Q \approx q_\pi$

输入: 一个任意的目标策略 π

初始化, 对所有 $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

任意初始化 $Q(s, a) \in \mathbb{R}$

$C(s, a) \leftarrow 0$

无限循环 (对每幕):

$b \leftarrow$ 任何能包括 π 的策略

根据 b 生成一幕序列: $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

$W \leftarrow 1$

对幕中的每一步循环, $t = T-1, T-2, \dots, 0$, 当 $W \neq 0$ 时:

$G \leftarrow \gamma G + R_{t+1}$

$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$

$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$

$W \leftarrow W \frac{\pi(A_t | S_t)}{b(A_t | S_t)}$

如果 $W = 0$, 则退出内层循环

练习 5.10 从式 (5.7) 推导出加权平均的更新规则 (式 5.8)。推导时遵循非加权规则 (式 2.3) 的推导思路。 \square

5.7 离轨策略蒙特卡洛控制

我们下面讨论本书涉及的第二类学习控制方法——离轨策略方法。之前介绍的同轨策略方法与其的区别在于, 同轨策略方法在使用某个策略进行控制的同时也对那个策略的价值进行评估。在离轨策略方法中, 这两个工作是分开的。用于生成行动数据的策略, 被称为行动策略。行动策略可能与实际上被评估和改善的策略无关, 而被评估和改善的策略称为目标策略。这样分离的好处在于当行动策略能对所有可能的动作继续进行采样时, 目标策略可以是确定的 (例如贪心)。

离轨策略蒙特卡洛控制方法用到了在前两节中提及的一种技术。它们遵循行动策略并对目标策略进行学习和改进。这些方法要求行动策略对目标策略可能做出的所有动作都有非零的概率被选择。为了试探所有的可能性, 我们要求行动策略是软性的 (也就是说它在所有状态下选择所有动作的概率都非零)。

下面的框中展示了一个基于 GPI 和重要度采样的离轨策略蒙特卡洛控制方法, 用于

估计 π_* 和 q_* 。目标策略 $\pi \approx \pi_*$ 是对应 Q 得到的贪心策略，这里 Q 是对 q_π 的估计。行动策略 b 可以是任何策略，但为了保证 π 能收敛到一个最优的策略，对每一个“状态-动作”二元组都需要取得无穷多次回报，这可以通过选择 ε -软性的 b 来保证。即使动作是根据不同的软策略 b 选择的，而且这个策略可能会在幕间甚至幕内发生变化，策略 π 仍能在所有遇到的状态下收敛到最优。

离轨策略 MC 控制算法，用于估计 $\pi \approx \pi_*$

初始化，对所有 $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \in \mathbb{R} \text{ (任意值)}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \operatorname{argmax}_a Q(s, a) \text{ (出现平分的情况下选取方法应保持一致)}$$

无限循环 (对每幕):

$$b \leftarrow \text{任意软性策略}$$

根据 b 生成一幕数据: $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

对幕中的每一时刻循环, $t = T-1, T-2, \dots, 0$:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \operatorname{argmax}_a Q(S_t, a) \text{ (出现平分的情况下选取方法应保持一致)}$$

如果 $A_t \neq \pi(S_t)$ 那么退出内层循环 (处理下一幕数据)

$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

一个潜在的问题是，当幕中某时刻后剩下的所有动作都是贪心的时候，这种方法也只会从幕的尾部进行学习。如果非贪心的行为较为普遍，则学习的速度会很慢，尤其是对于那些在很长的幕中较早出现的状态就更是如此。这可能会极大地降低学习速度。目前，对于这个问题在离轨策略蒙特卡洛方法中的严重程度尚无足够的研究和讨论。但如果真的很严重，处理这个问题最重要的方法可能是时序差分学习，这一算法思想将在第 6 章中介绍。另外，如果 γ 小于 1，则下一节中提到的方法也可能对此有明显的帮助。

练习 5.11 在算法框内的离轨策略 MC 控制算法中，你也许会觉得 W 的更新应该采用重要度采样比 $\frac{\pi(A_t | S_t)}{b(A_t | S_t)}$ ，但它实际上却用了 $\frac{1}{b(A_t | S_t)}$ 。这为什么是正确的? □

练习 5.12 赛道问题 (编程) 你正在像图 5.5 中所示的那样开赛车经过弯道。你想开得尽量快，但又不想因为开得太快而冲出赛道。简化的赛道是网格位置 (也就是图中的单元

格) 的离散集合, 赛车在其中一个格内。速度也同样是离散的, 在每一步中, 赛车可以在水平或垂直方向上同时移动若干格。动作是速度其中一个分量的变化量。在一个时刻中, 任何一个分量的改变量可能为 $+1$ 、 -1 或 0 , 总共就有 9 (3×3) 种动作。两种速度分量都限制在非负且小于 5 , 并且除非赛车在起点, 否则两者不能同时为零。每幕起始于一个随机选择的起始状态, 此时两个速度分量都为零。当赛车经过终点时幕终止。在赛车没有跨过终点线前, 每步的收益都是 -1 。如果赛车碰到了赛道的边缘, 它会被重置到起点线上的一个随机位置, 两个速度分量都被置为零且幕继续。在每步更新赛车的位置前, 先检查赛车预计的路径是否与赛道边界相交。如果它和终点线相交, 则该幕结束; 如果它与其他任意的地方相交, 就认为赛车撞到了赛道边界并把赛车重置回起点线。为了增加任务的挑战性, 在每步中, 不管预期的增量是多少, 两个方向上的速度增量有 0.1 的概率可能同时为 0 。使用蒙特卡洛控制方法来计算这个任务中最优的策略并展示最优策略的一些轨迹 (展示轨迹时不要加随机噪声)。 □

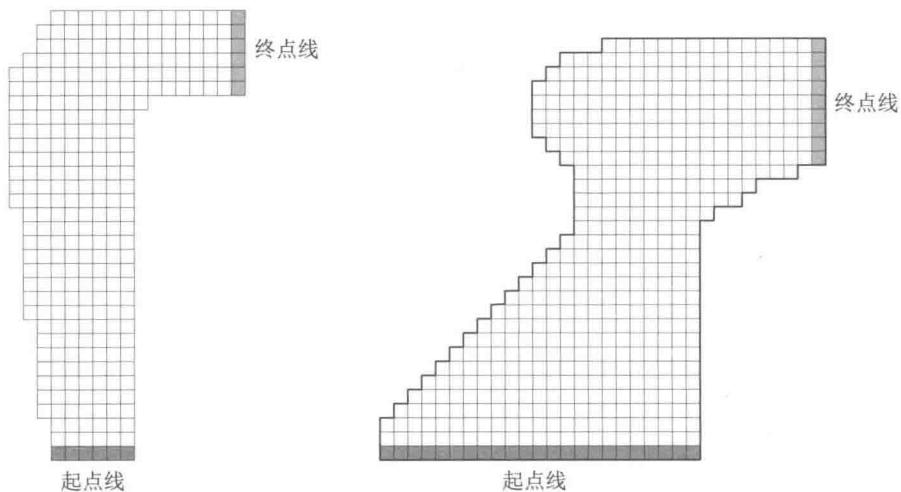


图 5.5 赛道任务中的两个右转弯赛道

5.8 * 折扣敏感的重要度采样

到目前为止, 我们讨论的离轨策略都需要为回报计算重要度采样的权重, 它把回报视为单一整体, 而不考虑回报是每个时刻的折后收益之和这一内部结构。现在我们简要地介绍一些使用这种结构来大幅减少离轨策略估计的方差的前沿研究思想。

举例来说, 考虑一种幕很长且 γ 显著小于 1 的情况。具体来说, 假设幕持续 100 步并

且 $\gamma = 0$ 。那么 0 时刻的回报就会是 $G_0 = R_1$, 但它的重要度采样比却会是 100 个因子之积, 即 $\frac{\pi(A_0|S_0)}{b(A_0|S_0)} \frac{\pi(A_1|S_1)}{b(A_1|S_1)} \dots \frac{\pi(A_{99}|S_{99})}{b(A_{99}|S_{99})}$ 。在普通重要度采样中会用整个乘积对回报进行缩放, 但实际上只需要按第一个因子来衡量, 即 $\frac{\pi(A_0|S_0)}{b(A_0|S_0)}$ 。另外 99 个因子: $\frac{\pi(A_1|S_1)}{b(A_1|S_1)} \dots \frac{\pi(A_{99}|S_{99})}{b(A_{99}|S_{99})}$ 都是无关的, 因为在得到首次收益之后, 整幕回报就已经决定了。后面的这些因子与回报相独立且期望值为 1, 不会改变预期的更新, 但它们会极大地增加其方差。在某些情况下, 它们甚至可以使得方差无穷大。下面我们讨论一种思路来避免这种与期望更新无关的巨大方差。

这个思路的本质是把折扣看作幕终止的概率, 或者说, 部分终止的程度。对于任何 $\gamma \in [0, 1]$, 我们可以把回报 G_0 看作在一步内, 以 $1 - \gamma$ 的程度部分终止, 产生的回报仅仅是首次收益 R_1 , 然后再在两步后以 $(1 - \gamma)\gamma$ 的程度部分终止, 产生 $R_1 + R_2$ 的回报, 依此类推。后者的部分终止程度对应于第二步的终止程度 $1 - \gamma$, 以及在第一步尚未终止的 γ 。因此, 第三步的终止程度是 $(1 - \gamma)\gamma^2$, 其中 γ^2 对应的是在前两步都不终止。这里的部分回报被称为平价部分回报

$$\bar{G}_{t:h} \doteq R_{t+1} + R_{t+2} + \dots + R_h, \quad 0 \leq t < h \leq T,$$

其中“平价”表示没有折扣, “部分”表示这些回报不会一直延续到终止, 而在 h 处停止, h 被称为视界 (T 是幕终止的时间)。传统的全回报 G_t 可被看作上述平价部分回报的总和

$$\begin{aligned} G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots + \gamma^{T-t-1} R_T \\ &= (1 - \gamma)R_{t+1} \\ &\quad + (1 - \gamma)\gamma(R_{t+2} + R_{t+3}) \\ &\quad + (1 - \gamma)\gamma^2(R_{t+1} + R_{t+2} + R_{t+3}) \\ &\quad \vdots \\ &\quad + (1 - \gamma)\gamma^{T-t-2}(R_{t+1} + R_{t+2} + \dots + R_{T-1}) \\ &\quad + \gamma^{T-t-1}(R_{t+1} + R_{t+2} + \dots + R_T) \\ &= (1 - \gamma) \sum_{h=t+1}^{T-1} \gamma^{h-t-1} \bar{G}_{t:h} + \gamma^{T-t-1} \bar{G}_{t:T}. \end{aligned}$$

现在我们需要使用一种类似的截断的重要度采样比来缩放平价部分回报。由于 $\bar{G}_{t:h}$ 只涉及到视界 h 为止的收益, 因此我们只需要使用到 h 为止的概率值。我们定义了如下普通重要度采样估计器, 它是式 (5.5) 的推广

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \left((1 - \gamma) \sum_{h=t+1}^{T(t)-1} \gamma^{h-t-1} \rho_{t:h-1} \bar{G}_{t:h} + \gamma^{T(t)-t-1} \rho_{t:T(t)-1} \bar{G}_{t:T(t)} \right)}{|\mathcal{T}(s)|}, \quad (5.9)$$

以及如下的加权重要度采样估计器，它是公式 (5.6) 的推广

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \left((1 - \gamma) \sum_{h=t+1}^{T(t)-1} \gamma^{h-t-1} \rho_{t:h-1} \bar{G}_{t:h} + \gamma^{T(t)-t-1} \rho_{t:T(t)-1} \bar{G}_{t:T(t)} \right)}{\sum_{t \in \mathcal{T}(s)} \left((1 - \gamma) \sum_{h=t+1}^{T(t)-1} \gamma^{h-t-1} \rho_{t:h-1} + \gamma^{T(t)-t-1} \rho_{t:T(t)-1} \right)}. \quad (5.10)$$

我们将这两个估计器称为折扣敏感的重要度采样估计。它们都考虑了折扣率，但当 $\gamma = 1$ 时没有任何影响 (和 5.5 节中离轨策略的估计器一样)。

5.9 * 每次决策型重要度采样

还有一种方法也可以在离轨策略的重要度采样中考虑回报是收益之和的内部结构，这种方法即使在没有折扣 (即 $\gamma = 1$) 时，也可以减小方差。在离轨策略估计器 (式 5.5) 和估计器 (式 5.6) 中，分子中求和计算中的每一项本身也是一个求和式

$$\begin{aligned} \rho_{t:T-1} G_t &= \rho_{t:T-1} (R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-t-1} R_T) \\ &= \rho_{t:T-1} R_{t+1} + \gamma \rho_{t:T-1} R_{t+2} + \cdots + \gamma^{T-t-1} \rho_{t:T-1} R_T. \end{aligned} \quad (5.11)$$

离轨策略的估计器依赖于这些项的期望值，那么我们看看是否可以用更简单的方式来表达这个值。注意，式 (5.11) 中的每个子项是一个随机收益和一个随机重要度采样比的乘积。例如，用式 (5.3) 可以将第一个子项写为

$$q \rho_{t:T-1} R_{t+1} = \frac{\pi(A_t|S_t)}{b(A_t|S_t)} \frac{\pi(A_{t+1}|S_{t+1})}{b(A_{t+1}|S_{t+1})} \frac{\pi(A_{t+2}|S_{t+2})}{b(A_{t+2}|S_{t+2})} \cdots \frac{\pi(A_{T-1}|S_{T-1})}{b(A_{T-1}|S_{T-1})} R_{t+1}. \quad (5.12)$$

现在可以注意到，在所有这些因子中，只有第一个和最后一个 (收益) 是相关的；所有其他比率均为期望值为 1 的独立随机变量

$$\mathbb{E}\left[\frac{\pi(A_k|S_k)}{b(A_k|S_k)}\right] \doteq \sum_a b(a|S_k) \frac{\pi(a|S_k)}{b(a|S_k)} = \sum_a \pi(a|S_k) = 1. \quad (5.13)$$

因此，由于独立随机变量乘积的期望是变量期望值的乘积，所以我们就可以把除了第一项以外的所有比率移出期望，只剩下

$$\mathbb{E}[\rho_{t:T-1} R_{t+1}] = \mathbb{E}[\rho_{t:t} R_{t+1}]. \quad (5.14)$$

如果我们对式 (5.11) 的第 k 项重复这样的分析，就会得到

$$\mathbb{E}[\rho_{t:T-1} R_{t+k}] = \mathbb{E}[\rho_{t:t+k-1} R_{t+k}].$$

这样一来，原来式 (5.11) 的期望就可以写成

$$\mathbb{E}[\rho_{t:T-1} G_t] = \mathbb{E}\left[\tilde{G}_t\right],$$

其中

$$\tilde{G}_t = \rho_{t:t} R_{t+1} + \gamma \rho_{t:t+1} R_{t+2} + \gamma^2 \rho_{t:t+2} R_{t+3} + \cdots + \gamma^{T-t-1} \rho_{t:T-1} R_T.$$

我们可以把这种思想称为每次决策型重要度采样。借助这个思想，对于普通重要度采样估计器，我们就找到了一种替代方法来进行计算：使用 \tilde{G}_t 来计算与公式 (5.5) 相同的无偏期望，以减小估计器的方差

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \tilde{G}_t}{|\mathcal{T}(s)|}, \quad (5.15)$$

对于加权重要度采样，是否也有一个每次决策型的版本？我们还不知道。到目前为止，我们已知的所有为其提出的估计器都不具备统计意义上的一致性（即数据无限时，它们也不会收敛到真实值）。

* 练习 5.13 给出从公式 (5.12) 推导公式 (5.14) 的详细步骤。 \square

* 练习 5.14 使用截断加权平均估计器 (式 5.10) 的思想修改离轨策略的蒙特卡洛控制算法（见 5.7 节）。注意，首先需要把这个式子转换为动作价值函数。 \square

5.10 本章小结

本章介绍了从经验中学习价值函数和最优策略的蒙特卡洛方法，这些“经验”的表现形式是多幕采样数据。相比于 DP 方法，这样做至少有三个优点。首先，它们不需要描述环境动态特性的模型，而可以直接通过与环境的交互来学习最优的决策行为。其次，它们可以使用数据仿真或采样模型。在非常多的应用中，虽然很难构建 DP 方法所需要的显式状态概率转移模型，但是通过仿真采样得到多幕序列数据却是很简单的。第三，蒙特卡洛方法可以很简单和高效地聚焦于状态的一个小的子集，它可以只评估关注的区域而不评估其余的状态（第 8 章会进一步讨论这一点）。

后文还会提到蒙特卡洛方法的第四个优点，蒙特卡洛方法在马尔可夫性不成立时性能损失较小。这是因为它不用后继状态的估值来更新当前的估值。换句话说，这是因为它不需要自举。

在设计蒙特卡洛控制方法时，我们遵循了第 4 章介绍的广义策略迭代 (GPI) 的总体架构。GPI 包含了策略评估和策略改进的交互过程。蒙特卡洛方法提供了另一种策略评

估的方法。蒙特卡洛方法不需要建立一个模型计算每一个状态的价值，而只需简单地将从这个状态开始得到的多个回报平均即可。因为一个状态的价值就是回报的期望，因此这个平均值就是状态的一个很好的近似。在控制方法中，我们特别感兴趣的是去近似动作价值函数，因为在没有环境转移模型的情况下，它也可以用于改进策略。蒙特卡洛方法逐幕地交替进行策略评估和策略改进，并可以逐幕地增量式实现。

保证足够多的试探是蒙特卡洛控制方法中的一个重要问题。贪心地选择在当前时刻的最优动作是不够的，因为这样其他的动作就不会得到任何回报，并且可能永远不会学到实际上可能更好的其他动作。一种避免这个问题的方法是随机选择每幕开始的“状态-动作”二元组，使其能够覆盖所有的可能性。这种试探性出发方法有时可以在具备仿真采样数据的应用中使用，但不太可能应用在具有真实经验的学习中。在同轨策略方法中，智能体一直保持试探并尝试寻找也能继续保持试探的最优策略。在离轨策略方法中，虽然智能体也在试探，但它实际学习的可能是与它试探时使用的策略无关的另一个确定性最优策略。

离轨策略预测指的是在学习目标策略的价值函数时，使用的是另一个不同的行动策略所生成的数据。这种学习方法基于某种形式的重要度采样，即用在两种策略下观察到的动作的概率的比值对回报进行加权，从而把行动策略下的期望值转化为目标策略下的期望值。普通重要度采样将加权后的回报按照采样幕的总数直接平均，而加权重要度采样则进行加权平均。普通重要度采样得到的是无偏估计，但具有更大甚至可能是无限的方差。在实践中更受青睐的是加权重要度采样，它的方差总是有限的。尽管离轨策略蒙特卡洛方法的概念很简单，但它在预测和控制问题中的应用问题还尚未解决，还正在研究中。

本章中讨论的蒙特卡洛方法与第 4 章中讨论的 DP 方法的不同之处在于以下两个方面。第一，蒙特卡洛方法是用样本经验计算的，因此可以无需环境的概率转移模型，直接学习。第二，蒙特卡洛方法不自举。也就是说，它们不通过其他价值的估计来更新自己的价值估计。这两个差异并不是绑定在一起的，而是可以分开的。在第 6 章中，我们考虑像蒙特卡洛方法一样从经验中学习，但也像 DP 方法一样自举的方法。

参考文献和历史评注

“蒙特卡洛”一词可以追溯到 20 世纪 40 年代，当时 Los Alamos 的物理学家设计了一些靠运气取胜的游戏，用来学习并帮助了解一些有关原子弹的复杂物理现象。从这一角度介绍蒙特卡洛方法的教科书有 Kalos 和 Whitlock, 1986; Rubinstein, 1981。

5.1-2 Singh 和 Sutton (1996) 区分了每次访问型和首次访问型这两种 MC 方法并证明了这些方法与强化学习存在关联。二十一点的例子基于 Widrow、Gupta 和 Maitra (1973) 使用的一个

例子。肥皂泡的例子是经典的狄利克雷问题，其蒙特卡洛解决方案是由 Kakutani (1945；参见 Hersh 和 Griego, 1969; Doyle 和 Snell, 1984) 首次提出的。

Barto 和 Duff (1994) 在经典蒙特卡洛算法的背景下讨论了用于求解线性方程组的策略评估。他们用 Curtiss (1954) 的分析指出了蒙特卡洛策略评估在大尺度问题中的计算优势。

- 5.3-4 蒙特卡洛 ES 是在本书 1998 版中提出的。这个工作首次将基于策略迭代的蒙特卡洛估计和控制方法明确联系在了一起。Michie 和 Chambers (1968) 早期使用蒙特卡洛方法在强化学习的背景下估计动作价值。在杆平衡 (见 3.3 节的例 3.4) 中，他们使用平均幕长度来评估每个状态中每个可能动作的价值 (期望的平衡“生命周期”)，然后他们使用这些评估来控制动作的选择。他们的方法在精神上与使用每次访问型 MC 估计的蒙特卡洛 ES 相似。Narendra 和 Wheeler (1986) 研究了有限遍历马尔可夫链的蒙特卡洛方法，此方法使用连续访问相同状态时累积的回报作为调整自动学习机动作概率的收益。
- 5.5 高效的离轨策略学习被认为是在多个领域出现的一个重要挑战。例如，这与概率图 (贝叶斯) 模型中的“干预”和“反设事实”的概念密切相关 (例如，Pearl, 1995; Balke 和 Pearl, 1994)。使用重要度采样的离轨策略有着悠久的历史，但仍未被充分理解。Rubinstein (1981)、Hesterberg (1988)、Shelton (2001) 和 Liu (2001) 等人讨论了加权重要度采样，有时其也被称为归一化重要度采样 (例如，Koller 和 Friedman, 2009)。
- 注意，离轨策略方法里的目标策略有时在文献中被称为“估计”策略，本书的第 1 版就采用了这样的术语。
- 5.7 赛道那道练习题改编自 Barto、Bradtko、Singh (1995) 和 Gardner (1973)。
- 5.8 我们对折扣敏感的重要度采样的研究是基于 Sutton、Mahmood、Precup 和 van Hasselt (2014) 的分析。到目前为止，Mahmood (2017; Mahmood、van Hasselt 和 Sutton, 2014) 已经基本完成了这个工作。
- 5.9 Precup、Sutton 和 Singh (2000) 提出了“每次决策型重要度采样”。他们也将离轨策略与时序差分学习、资格迹和近似方法相结合，提出了我们在后面章节中会讨论的各类精细问题。

6

时序差分学习

在强化学习所有的思想中，时序差分 (TD) 学习无疑是最核心、最新颖的思想。时序差分学习结合了蒙特卡洛方法和动态规划方法的思想。与前面提到的蒙特卡洛方法一致，时序差分方法也可以直接从与环境互动的经验中学习策略，而不需要构建关于环境动态特性的模型。与动态规划相一致的是，时序差分方法无须等待交互的最终结果（它使用了自举思想），而可以基于已得到的其他状态的估计值来更新当前状态的价值函数。时序差分、动态规划和蒙特卡洛这三个方法之间的关系是强化学习领域经常出现的话题，本章正式开始对这一话题进行探讨。我们将会看到这三种方法能够相互交融并能以多种方式结合在一起。在第 7 章中，我们将会介绍 n -步算法，这个算法将时序差分方法和蒙特卡洛方法联系在了一起。而在第 12 章中，我们将介绍的 TD (λ) 算法则能无缝地统一时序差分方法和蒙特卡洛方法。

和往常一样，我们首先关注策略评估（或称预测）问题，即如何对于一个给定的策略 π 估计它的价值函数 v_π 。对于控制问题（找到最优的策略），DP、TD 和蒙特卡洛方法都使用了广义策略迭代 (GPI) 的某个变种。这些方法之间的主要区别在于它们解决预测问题的不同方式。

6.1 时序差分预测

TD 和蒙特卡洛方法都利用经验来解决预测问题。给定策略 π 的一些经验，以及这些经验中的非终止状态 S_t ，这两种方法都会更新它们对于 v_π 的估计 V 。大致来说，蒙特卡洛方法需要一直等到一次访问后的回报知道之后，再用这个回报作为 $V(S_t)$ 的目标进行估计。一个适用于非平稳环境的简单的每次访问型蒙特卡洛方法可以表示成

$$V(S_t) \leftarrow V(S_t) + \alpha [G_t - V(S_t)], \quad (6.1)$$

在这里, G_t 是时刻 t 真实的回报, α 是常量步长参数 (对比公式 (2.4))。我们把这个方法称作常量 αMC 。蒙特卡洛方法必须等到一幕的末尾才能确定对 $V(S_t)$ 的增量 (因为只有这时 G_t 才是已知的), 而 TD 方法只需要等到下一个时刻即可。在 $t+1$ 时刻, TD 方法立刻就能构造出目标, 并使用观察到的收益 R_{t+1} 和估计值 $V(S_{t+1})$ 来进行一次有效更新。最简单的 TD 方法在状态转移到 S_{t+1} 并收到 R_{t+1} 的收益时会立刻做如下更新

$$V(S_t) \leftarrow V(S_t) + \alpha [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \quad (6.2)$$

实际上, 蒙特卡洛更新的目标是 G_t , 而 TD 更新的目标是 $R_{t+1} + \gamma V(S_{t+1})$ 。这种 TD 方法被称为 $TD(0)$, 或单步 TD。它实际上是第 12 章将会谈到的 $TD(\lambda)$ 和第 12 章将提到的多步 TD 方法的一种特例。下框中的算法完整地描述了 $TD(0)$ 的过程。

表格型 $TD(0)$ 算法, 用于估计 v_π

输入: 待评估的策略 π

算法参数: 步长 $\alpha \in (0, 1]$

对于所有 $s \in \mathcal{S}^+$, 任意初始化 $V(s)$, 其中 $V(\text{终止状态}) = 0$

对每幕循环:

 初始化 S

 对幕中的每一步循环:

$A \leftarrow$ 策略 π 在状态 S 下做出的决策动作

 执行动作 A , 观察到 R, S'

$V(S) \leftarrow V(S) + \alpha [R + \gamma V(S') - V(S)]$

$S \leftarrow S'$

 直到 S 为终止状态

由于 $TD(0)$ 的更新在某种程度上基于已存在的估计, 类似于 DP, 我们也称它为一种自举法。我们在第 3 章中得知

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] \quad (6.3)$$

$$= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \quad \text{来自式 (3.9)}$$

$$= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]. \quad (6.4)$$

大致来说, 蒙特卡洛方法把对 (6.3) 式的估计值作为目标, 而 DP 方法则把对 (6.4) 式的估计值作为目标。蒙特卡洛的目标之所以是一个“估计值”, 是因为公式 (6.3) 中的期望值是未知的, 我们用样本回报来代替实际的期望回报。DP 的目标之所以是一个“估计值”则不是因为期望值的原因, 其会假设由环境模型完整地提供期望值, 真正的原因是因为真

实的 $v_\pi(S_{t+1})$ 是未知的，因此要使用当前的估计值 $V(S_{t+1})$ 来替代。TD 的目标也是一个“估计值”，理由有两个：它采样得到对式(6.4)的期望值，并且使用当前的估计值 V 来代替真实值 v_π 。因此，TD 算法结合了蒙特卡洛采样方法和 DP 自举法。我们之后会看到，只要大胆想象并小心实现，TD 方法可以很好地结合蒙特卡洛方法和 DP 方法的优势。

右侧的图是表格型 TD(0) 的回溯图。回溯图顶部状态节点的价值的估计值是根据它到一个直接后继状态节点的单次样本转移来更新的。我们将 TD 和蒙特卡洛更新称为 采样更新，因为它们都会通过采样得到一个后继状态（或“状态-动作”二元组），使用后继状态的价值和沿途得到的收益来计算回溯值，然后相应地改变原始状态（或“状态-动作”二元组）价值的估计值。采样更新和 DP 方法使用的期望更新不同，它的计算基于采样得到的单个后继节点的样本数据，而不是所有可能后继节点的完整分布。



最后，注意在 TD(0) 的更新中，括号里的数值是一种误差，它衡量的是 S_t 的估计值与更好的估计 $R_{t+1} + \gamma V(S_{t+1})$ 之间的差异。这个数值，如下面公式所示，被称为 TD 误差，其在强化学习中会以各种形式出现

$$\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t). \quad (6.5)$$

注意，每个时刻的 TD 误差是当前时刻估计的误差。由于 TD 误差取决于下一个状态和下一个收益，所以要到一个时刻步长之后才可获得。也就是说， $V(S_t)$ 中的误差 δ_t 在 $t+1$ 时刻才可获得。还要注意，如果价值函数数组 V 在一幕内没有改变（例如，在蒙特卡洛方法中就是如此），则蒙特卡洛误差可写为 TD 误差之和

$$\begin{aligned} G_t - V(S_t) &= R_{t+1} + \gamma G_{t+1} - V(S_t) + \gamma V(S_{t+1}) - \gamma V(S_{t+1}) && \text{来自式 (3.9)} \\ &= \delta_t + \gamma(G_{t+1} - V(S_{t+1})) \\ &= \delta_t + \gamma\delta_{t+1} + \gamma^2(G_{t+2} - V(S_{t+2})) \\ &= \delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2} + \cdots + \gamma^{T-t-1}\delta_{T-1} + \gamma^{T-t}(G_T - V(S_T)) \\ &= \delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2} + \cdots + \gamma^{T-t-1}\delta_{T-1} + \gamma^{T-t}(0 - 0) \\ &= \sum_{k=t}^{T-1} \gamma^{k-t}\delta_k. \end{aligned} \quad (6.6)$$

如果 V 在该幕中变化了（例如 TD(0) 的情况， V 不断被更新），那么这个等式就不准确，但如果时刻步长较小，则等式仍能近似成立。这个等式的泛化在 TD 学习的理论和算法中很重要。

练习 6.1 如果 V 在幕中发生变化, 那么式 (6.6) 只能近似成立; 等式两侧差在哪里? 令 V_t 表示在 t 时刻在 TD 误差公式 (6.5) 和 TD 更新公式 (6.2) 中使用的状态价值的数据组。重新进行上面的推导, 推出为了让等式右侧仍等于左侧的蒙特卡洛误差, 需要在 TD 误差之和上加上的额外项。□

例 6.1 开车回家 当你每天从工作地点开车回家时, 你会估计一下路上要花多久时间。当你离开办公室时, 你会注意离开的时间、今天是星期几、当日天气, 以及任何其他可能相关的因素。这个星期五, 你在晚上六点整离开办公室, 估计回家需要花费 30 分钟。当你到达你的车旁时, 时间是 6:05, 这时却开始下雨了。在雨中开车通常比较慢, 所以你重新估计, 觉得回家还需要 35 分钟, 即总共需要 40 分钟。15 分钟后, 你很快开完了高速路段, 下高速后你将总时间的估计值减少到 35 分钟。不幸的是, 这时你被堵在一辆缓慢的卡车后面, 且道路太窄不能超车。最终你不得不跟着卡车, 直到 6:40 才开到你居住的街道。3 分钟后你终于到家。在这个场景下, 状态、时间和时长预测的序列如下:

状态	消耗的时间	估计	估计
	(分钟)	剩余时间	总时间
周五晚上六点离开办公室	0	30	30
到车旁, 开始下雨	5	35	40
下高速	20	15	35
在路上堵在卡车后面	30	10	40
开到居住的街道	40	3	43
到家	43	0	43

在这个例子中, 收益是每一段行程消耗的时间¹。过程不加折扣 ($\gamma = 1$), 因此每个状态的回报就是从这个状态开始直到回家实际经过的总时间。每个状态的价值是剩余时间的期望值。第二列数字给出了遇到的每个状态的价值的当前估计值。

如图 6.1 (左) 所示, 一种描述蒙特卡洛方法的步骤的简单办法是在时间轴上画出行车总耗时的预测值 (最后一行数据)。箭头表示的是常量 α MC 方法 (式 6.1) 推荐的对预测值的改变 ($\alpha = 1$)。这个值正是每个状态的价值的估计值 (预估的剩余时间) 与实际回报 (真实的剩余时间) 之差。例如, 当你下高速时, 你估计还要 15 分钟就能到家, 但实际上需要 23 分钟。此时就可以用公式 (6.1) 确定离开高速后的剩余时间估计的增量。这时的误差 $G_t - V(S_t)$ 是 8 分钟。假设步长参数 α 是 $1/2$, 那么根据这一经验, 离开高速后的预估剩余时间会增加 4 分钟。在当前这个例子中, 这个改变可能过大了, 因为堵在卡车

¹ 如果我们把这个问题看作一个求最短行程时间的控制问题, 那么我们当然会使用负的消耗时间作为收益。但在这里我们只关心预测问题 (策略评估问题), 因此我们尽量使问题简单化, 使用正的收益。

后面只是偶然运气不好。无论如何，这种更新只能离线进行，即只有到家以后才能进行更新，因为只有到这时你才知道实际的回报是多少。

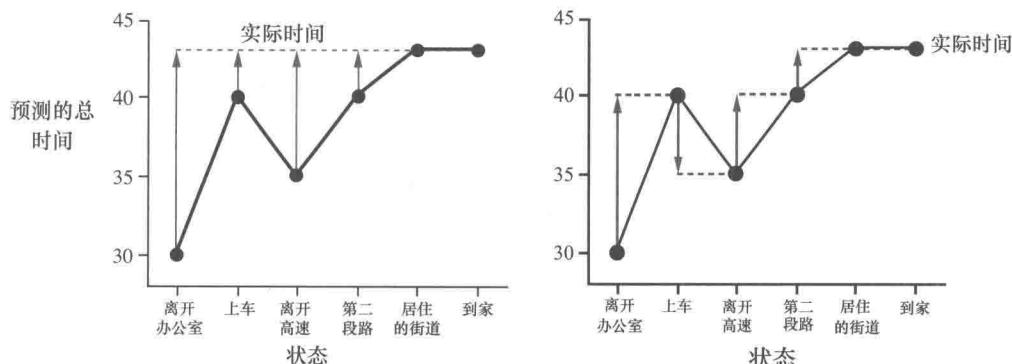


图 6.1 在开车回家这个例子中，蒙特卡洛方法（左）和 TD 方法（右）分别建议的改变

是否真的需要等到知晓最终结果后才能开始学习呢？假设在另一天，你在离开办公室时再次估计需要 30 分钟才能到家，但是之后你陷入了一场严重的交通堵塞，离开办公室 25 分钟后仍然堵在高速上寸步难行。这时你估计还要花 25 分钟才能到家，总共就要 50 分钟。在等待的过程中，你已经发现最初 30 分钟的估计过于乐观了。在这里你是否只有到家后才能增加对初始状态的估计值呢？如果使用蒙特卡洛方法，答案是肯定的，因为我们还不知道真正的回报。

但是根据 TD 方法，我们可以立刻进行学习，将初始估计从 30 分钟增加到 50 分钟。事实上，每一个估计都会跟着其直接后继的估计一起更新。回到我们第一天开车回家那个例子，图 6.1（右图）显示了根据 TD 规则（式 6.2）推荐的总时间预测值的变化（这些改变基于 $\alpha = 1$ 情况下的更新规则）。每个误差都与预测值在时序上的变化，即预测中的时序差分，成正比。

除了让你在堵车时有事可做之外，直接根据当前预测值学习而不是等到最终知道实际回报之后再学习还有几个计算方面的优点。我们在 6.2 节中简要地讨论其中的几个优点。

练习 6.2 这个练习是为了帮助你更好地从直觉上理解为什么 TD 方法通常比蒙特卡洛方法更高效。首先仔细回顾和思考一下 TD 方法和蒙特卡洛方法是如何应用在开车回家这个例子中的。你能想到一个使用 TD 更新会在平均意义上比使用蒙特卡洛更新更好的场景吗？请举出一个你觉得 TD 更新会更好的例子，例子中需要包括对过去经验的描述以及当前的状态。提示：假设你有很多次从办公室驾车回家的经历，某天你搬到一个新的办公

楼和一个新的停车场 (但仍从同一个入口上高速)。现在你开始学习从新办公楼回家的时间预测。在这种情况下，你能否知道为什么 TD 更新至少在一开始时可能会好得多？同样的事情是不是可能在原来的任务中发生过呢？

□

6.2 时序差分预测方法的优势

在 TD 方法中，某个估计值的更新需要部分地用到其他的估计值。它是从一个猜测中学习另一个猜测，即它能够自举。这是好事吗？和蒙特卡洛方法以及 DP 方法相比，TD 方法有什么优势？本书不便于展开讨论这个问题。我们在本节中只进行一个简单介绍。

相比 DP 方法，TD 方法一个显而易见的优势在于它不需要一个环境模型，即描述收益和下一状态联合概率分布的模型。

另一个很显著的优势是，相比蒙特卡洛方法，它自然地运用了一种在线的、完全递增的方法来实现。蒙特卡洛方法必须等到一幕的结束，因为只有那时我们才能知道确切的回报值，而 TD 方法只需等到下一时刻即可。在非常多的情况下，这是在选择方法时的一个关键的考量点。在一些应用场景中，幕非常长，所以把学习推迟到整幕结束之后就太晚了。在另一些应用场景中可能是持续性任务，无法划分出“幕”的概念。最后，我们在上一章中已经提到过，有一些蒙特卡洛方法必须对那些采用实验性动作的幕进行打折或者干脆忽略掉，这可能会大大减慢学习速度。而 TD 方法则不太容易受到这些问题的影响，因为它们从每次状态转移中学习，与采取什么后续动作无关。

但 TD 方法有扎实的理论支撑吗？不必等待实际结果，直接从后一个猜测中学习前一个猜测当然是很方便，但我们是否仍然可以保证结果收敛到正确的值？幸运的是，答案是肯定的。对于任何固定的策略 π ， $TD(0)$ 都已经被证明能够收敛到 v_π 。如果步长参数是一个足够小的常数，那么它的均值能收敛到 v_π 。如果步长参数根据随机近似条件 (2.7) 逐渐变小，则它能以概率 1 收敛。大多数收敛性证明仅适用于在公式 (6.2) 之前讨论的基于表格的算法，但有些也适用于使用广义线性函数近似的情况，那些情况我们将在第 9 章中进一步讨论。

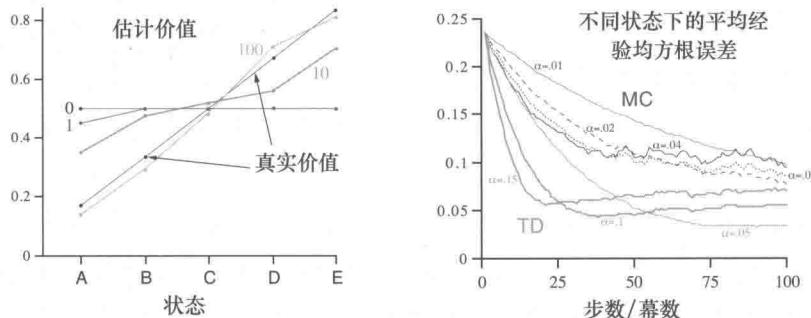
如果 TD 和蒙特卡洛方法都渐近地收敛于正确的预测，那么下一个问题自然是：“哪个收敛更快？”换句话说，哪种方法学得更快？哪种方法能更有效地利用有限的数据？目前，这还都是开放性的问题，没有人能够在数学上证明某种方法比另一种方法更快地收敛。事实上，我们甚至不清楚如何来最恰当地形式化表述这个问题！不过在实践中，如例 6.2 所示，TD 方法在随机任务上通常比常量 α MC 方法收敛得更快。

例 6.2 随机游走

在这个例子中，我们通过实验比较 TD(0) 和常量 α MC 在下图所示的马尔可夫收益过程中的预测能力：



马尔可夫收益过程 (Markov reward process, MRP) 是不包含动作的马尔可夫决策过程。在只关注预测问题时，因为并不需要区分到底是环境导致的变化还是智能体引起的变化，所以经常使用 MRP。在这个 MRP 中，所有阶段都从中心状态 C 开始，在每个时刻以相同的概率向左或向右移动一个状态。幕终止于最左侧或最右侧。终止于最右侧时，会有 +1 的收益；除此之外的收益均为零。例如，一个典型的幕可能包含以下“状态-收益”序列：C, 0, B, 0, C, 0, D, 0, E, 1。由于这个任务没有折扣，所以每个状态的真实价值是从这个状态开始并终止于最右侧的概率。因此，中心状态的真实价值为 $v_\pi(C) = 0.5$ 。状态 A ~ E 的真实价值分别为： $\frac{1}{6}, \frac{2}{6}, \frac{3}{6}, \frac{4}{6}$ 和 $\frac{5}{6}$ 。



上方左侧的图显示了在经历了不同数量的幕采样序列之后，运行一次 TD(0) 所得到的价值估计值。在 100 幕后，估计值就已经非常接近真实值了。由于使用了常数步长参数 (在这个例子中 $\alpha = 0.1$)，所以估计值会反映较近的若干幕的结果，其不规律地上下波动。上方右侧的图给出了对于多个不同的 α 取值，两种方法的学习曲线。图中显示的性能衡量指标是学到的价值函数和真实价值函数的均方根 (RMS) 误差。图中显示的误差是在 5 个状态上的平均误差，并在 100 次运行中取平均的结果。在所有情况下，对于所有 s ，近似价值函数都被初始化为中间值 $V(s) = 0.5$ 。在这个任务中，TD 方法一直比 MC 方法要好。

练习 6.3 在随机游走例子的左图中，第一幕只导致 $V(A)$ 发生了变化。由此，你觉得第一幕发生了什么？为什么只有这一个状态的估计值改变了？它的值到底改变了多少？ □

练习 6.4 在随机游走例子的右图中, 具体结果与步长参数 α 的值是相关的。如果 α 从一个更宽广的值域中取值, 会影响哪种算法更好的结论吗? 是否存在另一个固定的 α , 使得两种算法都能表现出比图中更好的性能? 为什么? \square

练习 6.5 在随机游走例子的右图中, TD 方法的均方根误差先下降后上升, 尤其在 α 较大时更明显。这是由什么导致的? 你认为这种情况总是会发生, 还是与近似价值函数如何初始化相关? \square

练习 6.6 在例 6.2 中的随机游走任务中, 状态 $A \sim E$ 的真实价值分别是: $\frac{1}{6}, \frac{2}{6}, \frac{3}{6}, \frac{4}{6}$ 和 $\frac{5}{6}$ 。描述至少两种不同的计算这些值的方法。猜猜实际中我们使用的是哪种方法? 为什么? \square

6.3 TD(0) 的最优化

假设只有有限的经验, 比如 10 幕数据或 100 个时间步。在这种情况下, 使用增量学习方法的一般方式是反复地呈现这些经验, 直到方法最后收敛到一个答案为止。给定近似价值函数 V , 在访问非终止状态的每个时刻 t , 使用式 (6.1) 或式 (6.2) 计算相应的增量, 但是价值函数仅根据所有增量的和改变一次。然后, 利用新的值函数再次处理所有可用的经验, 产生新的总增量, 依此类推, 直到价值函数收敛。我们称这种方法为批量更新, 因为只有在处理了整批的训练数据后才进行更新。

在批量更新下, 只要选择足够小的步长参数 α , TD(0) 就能确定地收敛到与 α 无关的唯一结果。常数 α MC 方法在相同条件下也能确定地收敛, 但是会收敛到不同的结果。理解这两种不同的结果有助于我们理解这两种方法之间的差异。在正常情况下, 这些方法不会一下子就直接得到它们各自的批量更新最终结果, 但它们都是在朝那个最终方向做出某种程度的更新。在针对所有可能的任务, 一般性地讨论上述两种方法的区别之前, 我们首先来看几个具体例子。

例 6.3 批量更新的随机游走 在随机游走问题 (例 6.2) 这个例子中, 批量更新版本的 TD(0) 和常数 α MC 是这样做的: 每经过新的一幕序列之后, 之前所有幕的数据就被视为一个批次。算法 TD(0) 或常数 α MC 不断地使用这些批次来进行逐次更新。这里 α 要设置得足够小以使价值函数能够收敛。最后将所得的价值函数与 v_π 进行比较, 绘制在 5 个状态下的平均均方根误差 (以整个实验的 100 次独立重复为基础) 的学习曲线, 如图 6.2 所示。注意批量 TD 方法始终优于批量蒙特卡洛方法。

在批量训练下, 常数 α MC 的收敛值 $V(s)$ 是经过状态 s 后得到的实际回报的样本平均值。可以认为它们是最优估计, 因为它们最小化了与训练集中实际回报的均方根误差。

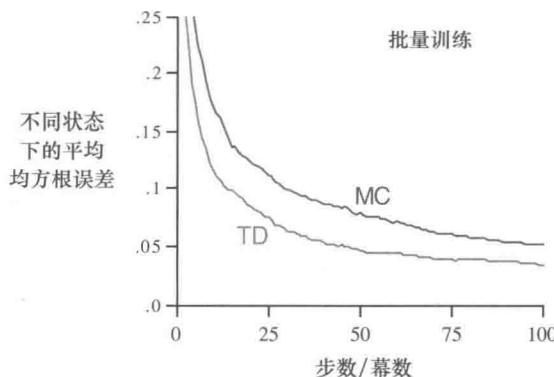


图 6.2 在随机游走任务中，在批量训练下的 $\text{TD}(0)$ 和常量 α MC 的性能

但让人惊讶的是，如图 6.2 所示，批量 TD 方法居然能在均方根误差上比它表现得更好。批量 TD 怎么会比这种最佳方法表现得更好呢？答案是这样的：蒙特卡洛方法只是从某些有限的方面来说最优，而 TD 方法的最优性则与预测回报这个任务更为相关。■

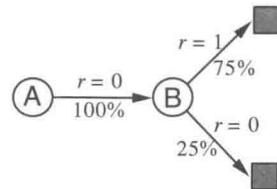
例 6.4 你就是预测者 现在想象你是一个未知的马尔可夫收益过程中对于回报的预测者。假设你观察到以下 8 幕数据：

A 0	B 0	B 1
B 1		B 1
B 1		B 1
B 1		B 0

上面内容的意思是：第一幕从状态 A 开始，转移至状态 B，得到收益 0，然后终结于状态 B，最终收益为 0。其他 7 幕数据甚至更短，从状态 B 开始就立刻终结了。给定这批数据，你认为最佳预测是什么？最优估计值 $V(A)$ 和 $V(B)$ 是多少？每个人应该都会同意 $V(B)$ 的最优估计值是 $\frac{3}{4}$ ，因为在状态 B 终结了 8 次，其中 6 次终结获得了 1 的回报，其他两次回报为 0。

但是在给定这些数据时， $V(A)$ 的最优估计值是什么？下面有两个合理的答案。第一个答案基于如下观察：过程在状态 A 时，会以 100% 的概率立刻转移到状态 B (得到收益为 0)；并且我们已经得到了 B 的价值为 $\frac{3}{4}$ ，所以 A 的价值也一定是 $\frac{3}{4}$ 。这个答案的一种解读是认为它首先将其建模成一个如右图所示的马尔可夫过程，再根据模型计算出正确的预测值，而这个模型给出的就是 $V(A) = \frac{3}{4}$ 。这也是使用批量 $\text{TD}(0)$ 方法会给出的答案。

另一种合理的答案是简单地观察到状态 A 只出现了一次并且对应的回报为 0，因此



就估计 $V(A) = 0$ 。这是批量蒙特卡洛方法会得到的答案。注意这也是使得训练数据上平方误差最小的答案。实际上，这个答案在这批数据上误差为零。尽管如此，我们仍认为前一个答案 $\frac{3}{4}$ 是更好的。即使蒙特卡洛答案在已知数据上表现得更好，但如果是马尔可夫过程，我们会预测前一种使用批量 TD(0) 方法得到的答案会在未来的数据上产生更小的误差。 ■

例 6.4 体现了通过批量 TD(0) 和批量蒙特卡洛方法计算得到的估计值之间的差别。批量蒙特卡洛方法总是找出最小化训练集上均方误差的估计，而批量 TD(0) 总是找出完全符合马尔可夫过程模型的最大似然估计参数。通常，一个参数的最大似然估计是使得生成训练数据的概率最大的参数值。在这个例子中，马尔可夫过程模型参数的最大似然估计可以很直观地从观察到的多幕序列中得到。从 i 到 j 的转移概率估计值，就是观察数据集中从 i 出发转移到 j 的次数占从 i 出发的所有转移次数的比例。而相应的期望收益则是在这些转移中观察到的收益的平均值。我们可以据此来估计价值函数，并且如果模型是正确的，则我们的估计也就完全正确。这种估计被称为确定性等价估计，因为它等价于假设潜在过程参数的估计是确定性的而不是近似的。批量 TD(0) 通常收敛到的就是确定性等价估计。

这一点也有助于解释为什么 TD 方法比蒙特卡洛方法更快地收敛。在以批量的形式学习的时候，TD(0) 比蒙特卡洛方法更快是因为它计算的是真正的确定性等价估计。这就解释了在随机游走任务 (见图 6.2) 的批量学习结果中 TD(0) 为什么显示出优势。与确定性等价估计的关系也可以在一定程度上解释非批量 TD(0) 的速度优势 (如例 6.2 的右图)。尽管非批量 TD(0) 并不能达到确定性等价估计或最小平方误差估计，但它大致朝着这些方向在更新，因此它可能比常数 α MC 更快。目前对于在线 TD 和蒙特卡洛方法的效率的比较还没有明确的结论。

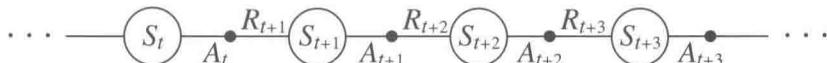
最后，值得注意的是，虽然确定性等价估计从某种角度上来说是一个最优答案，但直接计算它几乎是不可能的。如果 $n = |\mathcal{S}|$ 是状态数，那么仅仅建立过程的最大似然估计就可能需要 n^2 的内存，如果按传统方法，计算相应的价值函数则需要 n^3 数量级的步骤。相比之下，TD 方法则可以使用不超过 n 的内存，并且通过在训练集上反复计算来逼近同样的答案，这一优势确实是惊人的。对于状态空间巨大的任务，TD 方法可能是唯一可行的逼近确定性等价解的方法。

* 练习 6.7 设计一个离轨策略版的 TD(0) 更新算法，使其可以用于任意的目标策略 π ，且公式包含行动策略 b 。注意：在每个时刻 t 要使用重要度采样比 $\rho_{t:t}$ (式 5.3)。 □

6.4 Sarsa：同轨策略下的时序差分控制

现在，我们使用时序差分方法来解决控制问题。按照惯例，我们仍遵循广义策略迭代 (GPI) 的模式，只不过这次我们在评估和预测的部分使用时序差分方法。和使用蒙特卡洛方法时一样，我们同样需要在试探和开发之间做出权衡，方法又同样能被划分为两类：同轨策略和离轨策略。在本节中，我们先展示一种同轨策略下的时序差分 (on-policy TD) 控制方法。

第一步先要学习的是动作价值函数而不是状态价值函数。特别对于同轨策略方法，我们必须对所有状态 s 以及动作 a ，估计出在当前的行动策略下所有对应的 $q_\pi(s, a)$ 。这个估计可以使用与之前学习 v_π 时完全相同的方法。回想一下，所谓“一幕数据”，是指一个状态和动作交替出现的序列：



在 6.3 节中，我们讨论了状态之间的转移并学习了状态的价值。现在我们讨论“状态-动作”二元组之间的转移并学习“状态-动作”二元组的价值。在数学形式上这两者是相同的：它们都是带有收益过程的马尔可夫链。确保状态值在 TD(0) 下收敛的定理同样也适用于对应的关于动作值的算法上

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]. \quad (6.7)$$

每当从非终止状态的 S_t 出现一次转移之后，就进行上面的一次更新。如果 S_{t+1} 是终止状态，那么 $Q(S_{t+1}, A_{t+1})$ 则定义为 0。这个更新规则用到了描述这个事件的五元组 $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$ 中的所有元素。我们根据这个五元组把这个算法命名为 Sarsa。Sarsa 的回溯图如右图所示。

基于 Sarsa 预测方法设计一个同轨策略下的控制算法是很简单和直接的。和所有其他的同轨策略方法一样，我们持续地为行动策略 π 估计其动作价值函数 q_π ，同时以 q_π 为基础，朝着贪心优化的方向改变 π 。下面框中给出了 Sarsa 控制算法的一般形式。

Sarsa 算法的收敛性取决于策略对于 Q 的依赖程度。例如，我们可以采用 ε -贪心或者 ε -软性策略。只要所有“状态-动作”二元组都被无限多次地访问到，并且贪心策略在极限情况下能够收敛（这个收敛过程可以通过令 $\varepsilon = 1/t$ 来实现），Sarsa 就能以 1 的概率收敛到最优的策略和动作价值函数。



Sarsa (同轨策略下的 TD 控制) 算法, 用于估计 $Q \approx q_*$

算法参数: 步长 $\alpha \in (0, 1]$, 很小的 $\varepsilon, \varepsilon > 0$

对所有 $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, 任意初始化 $Q(s, a)$, 其中 $Q(\text{终止状态}, \cdot) = 0$

对每幕循环:

初始化 S

使用从 Q 得到的策略 (例如 ε -贪心), 在 S 处选择 A

对幕中的每一步循环:

执行动作 A , 观察到 R, S'

使用从 Q 得到的策略 (例如 ε -贪心), 在 S' 处选择 A'

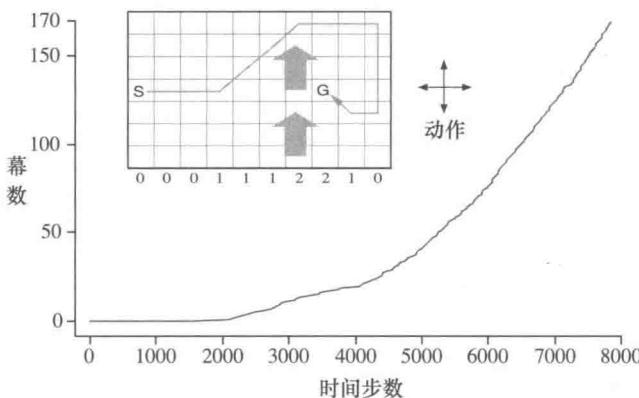
$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$$

$$S \leftarrow S'; A \leftarrow A';$$

直到 S 是终止状态

练习 6.8 证明公式 (6.6) 的“状态-动作”二元组版本也是成立的, “状态-动作”二元组版本的 TD 误差是 $\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$ 。注意, 这里我们同样假设价值函数在不同时刻不发生改变。 \square

例 6.5 有风的网格世界 下方展示的是一个带有起始状态和目标状态的网格世界。相比标准的网格世界, 它有一点不同: 在网格中存在穿过中间部分的向上的侧风。这里智能体的动作包括 4 种: 向上、向下、向右、向左。不过在中间区域, 智能体执行完动作所到达的状态的位置会被“风”向上吹一点。



每一列的风力是不同的, 风力的大小用向上吹的格数表示, 写在了每列的下方。举个例子, 如果你在目标状态右边的格子, 那么向左这个动作会把你带到目标的格子上方的一格。这是一个不带折扣的分幕式任务。在这个任务中, 在到达目标之前, 每步会收到恒定

为 -1 的收益。

上图显示了在这个任务中使用 ε -贪心的 Sarsa 的任务的结果 (横轴是逐幕累积起来的智能体所走的总步数, 纵轴则是智能体成功到达目标状态的总次数, 也即完成任务的总幕数), 其中 $\varepsilon = 0.1$, $\alpha = 0.5$, 对于所有的 s, a 进行同样的初始化 $Q(s, a) = 0$ 。通过图中逐渐变大的斜率可以看出, 随着时间的推移, 目标达成得越来越快。在 8000 时间步的时候, 贪心策略已经达到最优很久了 (图中显示了它的一个轨迹); 持续的 ε -贪心试探导致每幕的平均长度 (即任务的平均完成时间) 保持在 17 步左右, 比最小值 15 多了两步。需要注意, 不能简单地将蒙特卡洛方法用在此任务中, 因为不是所有策略都保证能终止。如果我们发现某个策略使智能体停留在同一个状态内, 那么下一幕任务就永远不会开始。诸如 Sarsa 这样一步一步的学习方法则没有这个问题, 因为它在当前幕的运行过程中很快就能学到当前的策略, 而它不够好, 然后就会切换到其他的策略。 ■

练习 6.9 在有风的网格世界中向 8 个方向移动 (编程) 重新解决有风的网格世界问题。这次假设可以向包括对角线在内的 8 个方向移动, 而不是原来的 4 个方向。利用更多的动作, 你得到的结果比原来能好多少? 如果还有第 9 个动作, 即除了风造成的移动之外不做任何移动, 你能进一步获得更好的结果吗? □

练习 6.10 随机强度的风 (编程) 重新解决在有风的网格世界中向 8 个方向移动的问题。假设有风且风的强度是随机的, 即在每列的平均值上下可以有 1 的变化。也就是说, 有三分之一的时间你和在之前问题中一样, 精确地按照每列下标给出的数值移动, 但另有三分之一的时间你会朝上多移动一格, 还有三分之一的时间你会朝下多移动一格。举个例子, 如果你在目标状态右边的一格, 你向左移动, 那么在三分之一的时间你会移动到目标上方的一格, 在另一个三分之一的时间你会移动到目标上方的两格, 在剩下的三分之一的时间你会正好移动到目标。 □

6.5 Q 学习：离轨策略下的时序差分控制

离轨策略下的时序差分控制算法的提出是强化学习早期的一个重要突破。这一算法被称为 Q 学习 (Watkins, 1989), 其定义为

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]. \quad (6.8)$$

在这里, 待学习的动作价值函数 Q 采用了对最优动作价值函数 q_* 的直接近似作为学习目标, 而与用于生成智能体决策序列轨迹的行动策略是什么无关 (作为对比, Sarsa 的

学习目标中使用的是待学习的动作价值函数本身,由于它的计算需要知道下一时刻的动作 A_{t+1} ,因此与生成数据的行动策略是相关的)。这大大简化了算法的分析,也很早就给出了收敛性证明。当然,正在遵循的行动策略仍会产生影响,它可以决定哪些“状态-动作”二元组会被访问和更新。然而,只需要所有的“状态-动作”二元组可以持续更新,整个学习过程就能够正确地收敛。我们在第5章中已经看到,这是在一般情况下,任何方法要保证能找到最优的智能体行为的最低要求。基于这种假设以及步长参数序列的某个常用的随机近似条件,可以证明 Q 能以1的概率收敛到 q_* 。 Q 学习算法的流程如下面框中所示。

Q 学习(离轨策略下的时序差分控制)算法,用于预测 $\pi \approx \pi_*$

算法参数:步长 $\alpha \in (0, 1]$,很小的 $\varepsilon, \varepsilon > 0$

对所有 $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$,任意初始化 $Q(s, a)$,其中 $Q(\text{终止状态}, \cdot) = 0$

对每幕:

初始化 S

对幕中的每一步循环:

使用从 Q 得到的策略(例如 ε -贪心),在 S 处选择 A

执行 A ,观察到 R, S'

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

$S \leftarrow S'$

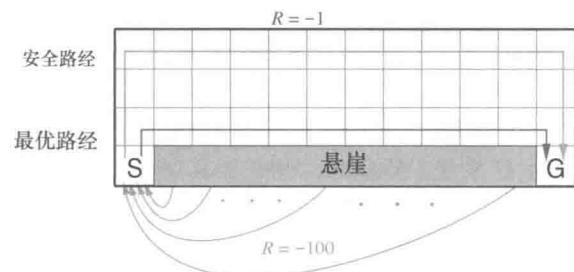
直到 S 是终止状态

Q 学习的回溯图长什么样呢?规则(式6.8)更新的是一个“状态-动作”二元组,因此顶部节点,也即更新过程的根节点,必须是一个代表动作的小实心圆点。更新也都来自于动作节点,即在下一个状态的所有可能的动作中找到价值最大的那个。因此回溯图底部的节点就是所有这些可能的动作节点。最后,你应该还记得,我们使用一条弧线跨过所有“下一步的动作节点”,来表示取这些节点中的最大值(如图3.4的右图所示)。那么现在你能猜出这张图是什么样吗?如果你有

想法,那么在看图6.4的答案前一定先做出自己的猜测。

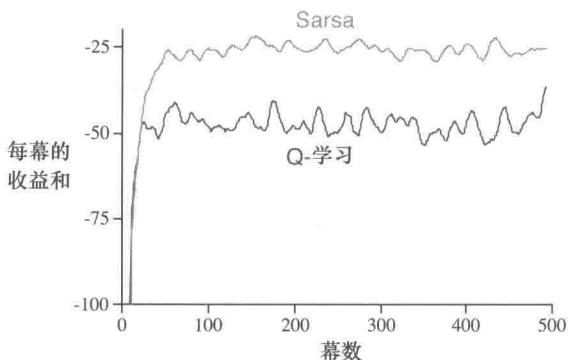
例6.6 在悬崖边上行走 这里我们通过一个网格世界的例子来比较Sarsa和 Q 学习两个算法,比较的重点在于同轨策略(Sarsa)和离轨策略(Q 学

习)两类方法间的区别。考虑右图所示的网格世界。这是一个标准的不含折扣的分幕式任务。它包含起点和目标状态,可以执行上下左右这些常见的动作。除了掉下悬崖之外,其



他的转移得到的收益都是 -1 。而掉入悬崖将会得到 -100 的收益，同时会立即把智能体送回起点。

右图显示的是在使用 ϵ -贪心方法（这里 $\epsilon = 0.1$ ）来选择动作时，Sarsa 和 Q 学习方法的表现。训练一小段时间后，Q 学习学到了最优策略，即沿着悬崖边上走的策略。不幸的是，由于动作是通过 ϵ -贪心的方式来选择的，因此在执行这个策略时，智能体会偶尔掉入悬崖。与之对比，Sarsa 则考虑了动作被选取的方式，学到了一条通



过网格的上半部分的路径，这条路径虽然更长但更安全。虽然 Q 学习实际上学到了最优策略的价值，其在线性能却比学到迂回策略的 Sarsa 更差。当然，如果 ϵ 逐步减小，那么两种方法都会渐近地收敛到最优策略。 ■

练习 6.11 为什么 Q 学习被认为是一种离轨策略控制方法？ □

练习 6.12 假设使用贪心的方式选择动作，那么此时 Q 学习和 Sarsa 是完全相同的算法吗？它们会做出完全相同动作选择，进行完全相同的权重更新吗？ □

6.6 期望 Sarsa

考虑一种与 Q 学习十分类似但把对于下一个“状态-动作”二元组取最大值这一步换成取期望的学习算法。即考虑一种算法，它使用如下更新规则

$$\begin{aligned} Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \mathbb{E}[Q(S_{t+1}, A_{t+1}) | S_{t+1}] - Q(S_t, A_t) \right] \\ &\leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \right], \end{aligned} \quad (6.9)$$

但它又遵循 Q 学习的模式。给定下一个状态 S_{t+1} ，这个算法确定地向期望意义上的 Sarsa 算法所决定的方向上移动。因此这个算法被叫作期望 Sarsa。图 6.4 的右侧是它的回溯图。

期望 Sarsa 在计算上比 Sarsa 更加复杂。但作为回报，它消除了因为随机选择 A_{t+1} 而产生的方差。在相同数量的经验下，我们可能会预想它的表现能略好于 Sarsa，期望 Sarsa 也确实表现更好。图 6.3 显示了在悬崖边上行走这个任务中，期望 Sarsa 与 Sarsa、Q 学习相比较的汇总结果。期望 Sarsa 保持了 Sarsa 优于 Q 学习的显著优势。此外，在步长

参数 α 大量不同的取值下，期望 Sarsa 都显著地优于 Sarsa。在悬崖边上行走这个例子中，状态的转移都是确定的，所有的随机性都来自于策略。在这种情况下，期望 Sarsa 可以放心地设定 $\alpha=1$ ，而不需要担心长期稳态性能的损失。与之相比，Sarsa 仅仅在 α 的值较小时能够有良好的长期表现，而启动期瞬态表现就很糟糕了。不管是在这个例子还是在其他例子中，期望 Sarsa 相比 Sarsa 在实验中都表现出了明显的优势。

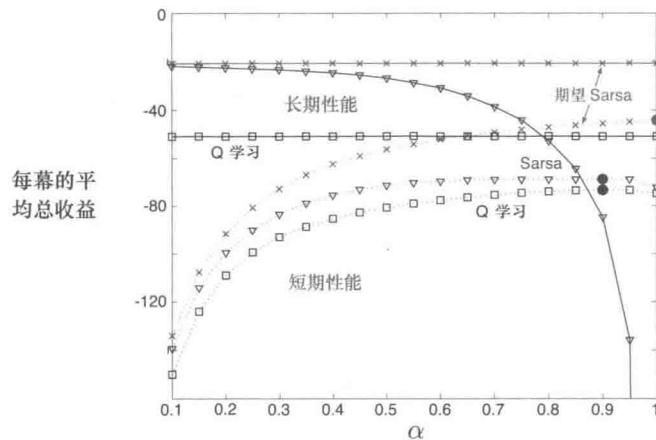


图 6.3 在悬崖边上行走这个任务中，TD 控制方法的启动期瞬态性能和长期稳态性能与步长参数 α 之间的关系函数。这里所有算法都用的是 $\epsilon = 0.1$ 的 ϵ -贪心策略。每次实验中，稳态性能是 100 000 幕数据上的平均值，而瞬态性能则是最初 100 幕数据上的平均值。图中启动期瞬态曲线和长期稳态曲线中的数据则分别是 50 000 次实验运行和 10 次实验运行的平均值。图中实心圆表示的是每种方法最佳的瞬态性能。本图改编自 van Seijen et al. (2009)



图 6.4 Q 学习和期望 Sarsa 的回溯图

在悬崖边上行走这个例子的实验结果中，期望 Sarsa 被用作一种同轨策略的算法。但在一般情况下，它可以采用与目标策略 π 不同的策略来生成行为。在这种情况下期望 Sarsa 就成了离轨策略的算法。举个例子，假设目标策略 π 是一个贪心策略，而行动策略却更注重于试探，那么此时期望 Sarsa 与 Q 学习完全相同。从这个角度来看，期望 Sarsa 推广了 Q 学习，可以将 Q 学习视作期望 Sarsa 的一种特例，同时期望 Sarsa 比起 Sarsa 也稳定地提升了性能。除了增加少许的计算量之外，期望 Sarsa 应该完全优于这

两种更知名的时序差分控制算法。

6.7 最大化偏差与双学习

我们目前讨论的所有的控制算法在构建目标策略时都包含了最大化的操作。例如在 Q 学习算法中，目标策略就是根据所有动作价值的最大值来选取动作的贪心策略的；在 Sarsa 算法中，通常按照 ε -贪心算法选取目标策略，其中也包含了最大化操作。在这些算法中，在估计值的基础上进行最大化也可以被看作隐式地对最大值进行估计，而这就会产生一个显著的正偏差。我们举个例子来对其原因进行说明。假设在状态 s 下可以选择多个动作 a ，这些动作在该状态下的真实价值 $q(s, a)$ 全为零，但是它们的估计值 $Q(s, a)$ 是不确定的，可能有些大于零，有些小于零。真实值的最大值是零，但估计值的最大值是正数，因此就产生了正偏差。我们将其称作最大化偏差。

例 6.7 最大化偏差的例子 最大化偏差会损害 TD 控制算法的性能。这里我们以图 6.5 中的那个简单的 MDP 为例说明。这个 MDP 有两个非终止节点 A 和 B。每幕都从 A 开始并选择向左或向右的动作。选择向右这个动作会立刻转移到终止状态并得到值为 0 的收益和回报。选择向左的动作则会使状态转移到 B，得到的收益也为 0。而在 B 这个状态下就有很多种可能的动作，每种动作被选择后都会立刻终止并得到一个从均值为 -0.1 、方差为 1.0 的正态分布中采样得到的收益。因此，任何一个以向左开始的轨迹的期望回报

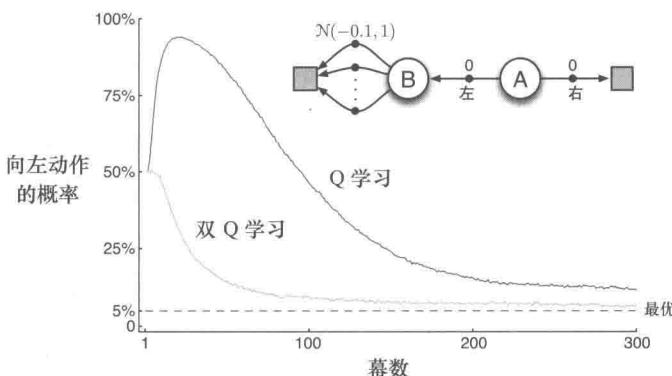


图 6.5 Q 学习和双 Q 学习在一个简单的分幕式 MDP (展示在了图内) 中的对比。Q 学习在开始阶段学到的行为是：执行向左的概率会远比执行向右来得高，并且向左的概率会一直显著地高于 5%。这个 5% 是使用 $\varepsilon = 0.1$ 的 ε -贪心策略选择动作而引起的最低的向左运动的概率。与之相比，双 Q 学习实质上并没有受到最大化偏差的影响。这些数据都是 10 000 次运行的平均值，动作价值都被初始化为零。在用 ε -贪心策略选择动作时，如果有多个最大值，那么会随机选择一个。

均为 -0.1 , 在 A 这个状态中根本不该选择向左。尽管如此, 我们的控制方法都会偏好向左, 因为最大化偏差会让 B 呈现出正的价值。图 6.5 就显示了使用 ε -贪心策略来选择动作的 Q 学习算法会在开始阶段非常明显地偏好向左这个动作。即使在算法收敛到稳态时, 它选择向左这个动作的概率也比在这里的参数设置条件 ($\varepsilon = 0.1$, $\alpha = 0.1$ 以及 $\gamma = 1$) 下的最优值高了大约 5%。 ■

有没有算法能够避免最大化偏差呢? 让我们先考虑一个赌博机的例子。在这个例子中, 我们对每个动作的价值做一个带噪声的估计, 这是通过对该动作产生的所有收益进行简单平均得到的。正如前文所述, 如果我们将估计值中的最大值视为对真实价值的最大值的估计, 那么就会产生正的最大化偏差。对于这个问题, 有一种看法是, 其根源在于确定价值最大的动作和估计它的价值这两个过程采用了同样的样本 (多幕序列)。假如我们将这些样本划分为两个集合, 并用它们学习两个独立的对真实价值 $q(a), \forall a \in A$ 的估计 $Q_1(a)$ 和 $Q_2(a)$, 那么我们接下来就可以使用其中一个估计, 比如 $Q_1(a)$, 来确定最大的动作 $A^* = \operatorname{argmax}_a Q_1(a)$, 再用另一个 Q_2 来计算其价值的估计 $Q_2(A^*) = Q_2(\operatorname{argmax}_a Q_1(a))$ 。由于 $\mathbb{E}[Q_2(A^*)] = q(A^*)$, 因此这个估计是无偏的。我们也可以交换两个估计 $Q_1(a)$ 和 $Q_2(a)$ 的角色再执行一遍上面这个过程, 那么就又可以得到另一个无偏的估计 $Q_1(\operatorname{argmax}_a Q_2(a))$ 。这就是双学习的思想。注意, 在这里虽然我们一共学习了两个估计值, 但是对每个样本集合只更新一个估计值。双学习需要两倍的内存, 但每步无需额外的计算量。

双学习的思想很自然地就可以推广到那些为完备 MDP 设计的算法中。例如双学习版的 Q 学习就叫双 Q 学习。双 Q 学习把所有的时刻一分为二。假设用投硬币的方式进行划分, 那么当硬币正面朝上时, 进行如下的更新

$$\begin{aligned} Q_1(S_t, A_t) &\leftarrow Q_1(S_t, A_t) + \\ &\alpha \left[R_{t+1} + \gamma Q_2\left(S_{t+1}, \operatorname{argmax}_a Q_1(S_{t+1}, a)\right) - Q_1(S_t, A_t) \right]. \end{aligned} \quad (6.10)$$

而如果硬币反面朝上, 那么就交换 Q_1 和 Q_2 的角色进行同样的更新, 这样就能更新 Q_2 。这两个近似价值函数的地位是完全相同的。两种动作价值的估计值都可以在行动策略中使用。例如使用 ε -贪心策略的双 Q 学习算法可以使用两个动作价值估计值的平均值 (或和)。下面的框中展示了双 Q 学习的完整算法流程。这也是得到图 6.5 中的结果我们采用的算法。在那个例子中, 双学习看上去消除了最大化偏差所带来的性能损失。当然双学习也可以应用到 Sarsa 和期望 Sarsa 中去。

* 练习 6.13 使用 ε -贪心目标策略的双期望 Sarsa 的更新步骤是怎样的? □

双 Q 学习，用于估计 $Q_1 \approx Q_2 \approx q_*$

算法参数：步长 $\alpha \in (0, 1]$ ，很小的 $\varepsilon, \varepsilon > 0$

对所有 $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$ ，初始化 $Q_1(s, a)$ 和 $Q_2(s, a)$ ，其中 $Q(\text{终止状态}, \cdot) = 0$

对每幕循环：

初始化 S

对幕中的每一步循环：

基于 $Q_1 + Q_2$ ，使用 ε -贪心策略在 S 中选择 A

执行动作 A ，观察到 R, S'

以 0.5 的概率执行：

$$Q_1(S, A) \leftarrow Q_1(S, A) + \alpha \left(R + \gamma Q_2(S', \arg\max_a Q_1(S', a)) - Q_1(S, A) \right)$$

或者执行：

$$Q_2(S, A) \leftarrow Q_2(S, A) + \alpha \left(R + \gamma Q_1(S', \arg\max_a Q_2(S', a)) - Q_2(S, A) \right)$$

$S \leftarrow S'$

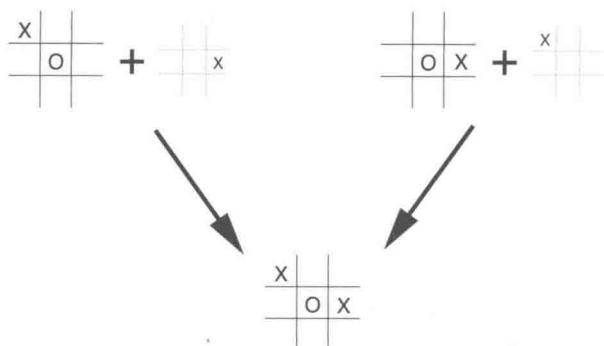
直到 S 是终止状态

6.8 游戏、后位状态和其他特殊例子

在本书中，我们试图为大量不同的任务找到一个统一的解决方案，但当然也总会有一些任务用特定的方法来解决会更好。例如，我们使用的通用方法需要学习一个动作价值函数，但在第 1 章中，我们展示了一种用来学习玩井字棋的时序差分方法，而它学到的更像是一个状态价值函数。仔细分析这个例子，我们会发现，从中学到的函数既不是动作价值函数也不是通常使用的状态价值函数。传统的状态价值函数估计的是在某个状态中，当智能体可以选择执行一个动作时，这个状态的价值，但是在井字棋中使用状态价值函数评估的是在智能体下完一步棋之后的棋盘局面。我们称之为后位状态 (*afterstates*)，并将它们的价值函数称之为是后位状态价值函数。当我们知道环境动态变化中初始的一部分信息，但是不知道完整的环境动态变化信息时，后位状态这个概念是很有用的。比如在游戏中，我们通常都能知道在我们采取一个动作后立刻会造成什么结果。在国际象棋中，对于每种可能的下法，我们都能知道下完之后会形成什么样的局面，但我们不知道对手将会采取什么动作。后位状态价值函数就是利用了这些先验知识的更加高效的学习方法。

通过井字棋的例子，我们可以清晰地看到使用后位状态来设计算法更为高效的原因。传统的动作价值函数将当前局面和出棋动作映射到价值的估计值。然而如右图所示，很多“局面-下法”二元组都会产生相同的局面。在这样的情形下，“局面-下法”二元组是不同

的，但是会产生相同的“后位状态”，因此它们的价值也必须是相同的。传统的价值函数会分别评估这两个“局面-下法”二元组，但是后位状态价值函数会将这两种情况看作是一样的。任何关于上图左边的“局面-下法”二元组的学习，都会立即迁移到右边去。



除了应用在游戏中外，后位状态也能应用在其他任务中。例如，在排队任务中，有分配服务器、拒绝用户和丢弃信息这三种动作。事实上在这里，这些动作就是根据它们完全能确定的即时效果而定义的。

我们无法在这里列出所有的特殊任务，以及可以对应使用的特殊学习算法。但本书中提到的这些原理应该是广泛适用的。例如，后位状态方法仍然符合广义策略迭代的框架，在算法中，策略和(后位状态)价值函数以类似的方式产生相互作用。在很多种情况下，为了持续进行试探，我们也会面临在同轨策略和离轨策略方法之间的选择。

练习 6.14 描述如何使用后位状态来对杰克租车问题(例 4.2)进行建模。为什么在这个任务中使用后位状态能够加速收敛? □

6.9 本章小结

我们在这一章中介绍了一类新的学习方法，即时序差分(TD)学习，并且展示了如何将其应用于强化学习问题中。按照惯例，我们将整个问题划分为预测问题与控制问题。在预测问题中，TD 方法可以用来代替蒙特卡洛方法。对于这两种方法，为了将它们扩展到控制问题中，我们都需要借用在动态规划章节中提到的广义策略迭代(GPI)的思想，即近似的策略和价值函数需要相互作用，以使得它们都向自己的最优值的方向优化。

GPI 由两个过程组成，其中一个驱使价值函数去准确地预测当前策略的回报，这就是所谓的“预测问题”。而另一个过程则驱使策略根据当前的价值函数来进行局部改善(例如可以使用 ϵ -贪心策略)，这就是所谓的“控制问题”。当第一个过程需要使用经验时，如何维持足够的试探就成为一个难题。在解决这个难题时，根据使用的是同轨策略方法还是离轨策略方法，我们可以将 TD 控制方法分为两类。Sarsa 是一种同轨策略的方法，而 Q 学习则是一种离轨策略的方法。我们在这里介绍的期望 Sarsa 也是一种离轨策略的方法。

其实将 TD 方法扩展到解决控制问题还可以用第三种方法。这种在本章中并未介绍的方法叫作“行动器-评判器”方法，第 13 章中会进行介绍。

本章中介绍的这些方法是现在最为常用的强化学习方法。它们的流行也许是因为这些方法惊人地简洁：可以在线地应用它们，而且用几个简单的等式就可以描述，用小型程序就可以实现。在接下来的几章中，我们会扩展这些算法，它们会变得稍微复杂一些，但同时也会明显地更为强大。所有的这些新算法都会保留在这里介绍的一些特性：它们可以在线地用相对较少的计算量处理经验，它们也由 TD 误差驱动。在这一章中介绍的 TD 方法的一个特例更准确的叫法应该是单步、表格型、无模型的 TD 方法。在接下来的两章中，我们会将其扩展到 n 步的形式（与蒙特卡洛方法相联系），再扩展到包含一个环境的模型（与规划和动态规划算法相联系）。之后，在本书的第 II 部分，我们会把它们扩展到使用函数近似而不是表格的方法（与深度学习和人工神经网络相联系）。

最后，在本章中我们完全是在强化学习问题的背景下讨论 TD 方法，但 TD 方法事实上是更为一般性的方法。它们是用来学习如何在动态系统中做出长期预测的一般方法。比如 TD 方法也许能用于预测金融数据、寿命、选举结果、天气规律、动物行为、发电厂需求或顾客的购买行为。只有在脱离了强化学习框架，把 TD 方法当作一种单纯的预测方法去看待的时候，研究者们才深刻地理解了它的许多理论性质。即便是这样，TD 学习方法仍有许多的潜在应用尚未被深入地研究。

参考文献和历史评注

正如我们在第 1 章的概述中提到的，时序差分学习的理念源于动物学习心理学和人工智能，其中最重要的工作是 Samuel (1959) 和 Klopff (1972)。Samuel 的工作会在 16.2 节中作为一个案例研究进行介绍。与时序差分学习相关的工作还有 Holland (1975, 1976) 早期关于价值预测一致性的观点。这些观点影响了本书的作者之一 (Barto)。Holland 在 1970—1975 年间在密歇根大学教课，而他当时是密歇根大学的一名研究生。Holland 的想法促使了一些与时序差分相关的系统出现，其中包括 Booker 的工作 (1982) 和 Holland 的“救火队算法”(1986)，“救火队算法”这一工作和 Sarsa 有关，下面会讨论。

6.1-2 本节中所用的大部分具体的材料来自 Sutton (1988)，包括 TD(0) 算法、随机游走的例子以及“时序差分学习”这个术语。这里提到的它与动态规划和蒙特卡洛方法相联系的特点是受到 Watkins (1989)、Werbos (1987) 以及其他人的影响。回溯图是在本书的第 1 版中首次引入的。

基于 Watkins 和 Dayan 的工作 (1992)，Sutton (1988) 证明了表格型 TD(0) 的均值是收敛的，Dayan (1992) 证明了它以概率 1 收敛。Jaakkola、Jordan 和 Singh (1994) 以及 Tsitsiklis (1994) 使用随机逼近理论扩展和加强了这些结论。其他的扩展和推广将在后面的

章节介绍。

- 6.3 在批量训练下最优的 TD 算法是由 Sutton (1988) 提出的。这个结果的提出受到了 Barnard (1993) 的启发，他对 TD 算法的推导是两个步骤的结合，一个步骤是马尔可夫链模型的增量式学习，另一个步骤则是模型的预测。确定性等价这个术语来自于自适应控制的文献 (例如，Goodwin 和 Sin, 1984)。
- 6.4 Sarsa 算法是由 Rummery 和 Niranjan(1994) 提出的。他们探索了它与神经网络的结合，并将其称为“改进的联结式 Q 学习”。“Sarsa”这个名字是由 Sutton (1996) 提出的。单步表格型 Sarsa (即本章中处理的形式) 的收敛性是由 Singh、Jaakkola、Littman 和 Szepesvári (2000) 证明的。Tom Kalt 建议我们加入“有风的网格世界”的例子。
Holland (1986) 的“救火队算法”的思想演变成了与 Sarsa 密切相关的一个算法。“救火队算法”的最初思路涉及了相互触发的规则链。它的重点在于从当前的规则将信度传回触发它的规则。随着时间的推移，“救火队算法”会将信度传回到所有的前序规则，而不仅仅是触发它的规则，从这一点上来看它越来越像时序差分学习。现代形式的“救火队算法”做了很多自然的简化，已经和单步 Sarsa 几乎完全相同，详见 Wilson (1994)。
- 6.5 Q 学习是由 Watkins (1989) 提出的，但他只提出了收敛性证明的一个梗概。Watkins 和 Dayan (1992) 将这一收敛性证明严格化。Jaakkola、Jordan 和 Singh (1994) 以及 Tsitsiklis (1994) 也提出了更多关于其收敛性的结果。
- 6.6 期望 Sarsa 是由 George John (1994) 提出的，他称这种算法为“ \bar{Q} 学习”，并强调了作为一种离轨策略算法，它相对 Q 学习的优势。在本书第 1 版的练习题中引入期望 Sarsa 时，我们还不知道 John 的工作。van Seijen、van Hasselt、Whiteson 和 Weiring (2009) 在发表他们的工作时，也不知道 John 的工作。van Seijen 等人提出了期望 Sarsa 的收敛性，以及它比普通的 Sarsa 和 Q 学习表现更好的条件。图 6.3 改编自他们的结果。van Seijen 将“期望 Sarsa”仅仅定义为一种同轨策略方法 (就像我们在第 1 版中做的那样)，而我们则用这个名称指代更一般的目标策略和行动策略不同的算法。van Hasselt (2011) 首次注意到了期望 Sarsa 作为一般的离轨策略算法的形式，他称其为“广义 Q 学习”。
- 6.7 van Hasselt (2010, 2011) 提出并深入研究了最大化偏差和双学习。图 6.5 中 MDP 的例子是由 (van Hasselt, 2011) 中的图 4.1 改编的。
- 6.8 “后位状态”的概念和“决策后位状态”(Van Roy、Bertsekas、Lee 和 Tsitsiklis, 1997; Powell, 2011) 是相同的。