第 5 章 策略梯度

5.1 基于策略的算法

在之前的章节中介绍了 Q 函数和时序差分方法，并学习了基于此而设计的 QLearning、SARSA、DQN 算法，这些算法有一个共同特征，它们都是基于价值的算法，也就是说，它们的基本原理都是计算在一种状态下执行各个动作的价值，决策时选择价值最大的动作执行。

换句话说，这些算法的底层原理都是在计算一张 Q 表，Q 表中记录了各种状态下各个动作的价值，最后根据该 Q 表进行决策。此类算法统一被称为基于价值的算法。

下面介绍一种完全不同的思路，被称为策略迭代，它是基于策略的算法。为了理解该算法为何不同，下面还是使用冰湖游戏环境进行举例说明，冰湖游戏环境如图 1-4 所示。

冰湖游戏环境在前面章节中已经反复出现，相信读者已经对该游戏环境很熟悉，此处不再反复说明该游戏的规则。

回顾基于价值的算法，它们的共同特点是都在计算一张 Q 值表，如表 5-1 所示。

表 5-1 所有状态下所有动作的分数

行	列	上	下	左	右
第 1 行	第 1 列				
第 1 行	第 2 列				
第 1 行	第 3 列				
第 1 行	第 4 列				
第 2 行	第 1 列				
第 2 行	第 2 列				
第 2 行	第 3 列				
第 2 行	第 4 列				
第 3 行	第 1 列				
第 3 行	第 2 列				
第 3 行	第 3 列	一般	高	一般	低

续表

行	列	上	下	左	右
第 3 行	第 4 列				
第 4 行	第 1 列				
第 4 行	第 2 列	一般	一般	低	高
第 4 行	第 3 列	一般	一般	一般	高
第 4 行	第 4 列				

基于价值的算法本质上都是在算这样的一张表，只要该表计算得足够精确，就可以基于该表做出很好的动作决策。

从上面的叙述可以看出，计算 Q 表是为了决策，那么是否只要能做出好的决策，有没有 Q 表不重要呢？答案是肯定的，考虑做出决策的过程如图 5-1 所示。

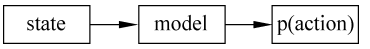

图 5-1　做出决策的过程

使用神经网络模型可以根据游戏环境的状态信息做出决策，只是该决策的质量是否足够好。在基于价值的算法中，决策质量的好坏基本取决于 Q 表的质量，但是从图 5-1 可以看出，此处已经使用了神经网络模型来做出决策。

前面的章节介绍过，神经网络模型几乎可以拟合任意复杂度的数据，只要数据存在潜在的统计规律，就能进行拟合，如果不能拟合，则说明神经网络的体量还不够大。

既然神经网络模型具有如此强大的计算能力，不妨尝试着直接让神经网络模型来做出决策，而放弃计算 Q 表。

回顾强化学习的定义，也就是根据机器人做出的动作给予反馈，要求机器人求反馈的最大化。设想一下在这样的要求下，机器人可能会做出怎样的尝试呢？

也许机器人一开始会给自己想一套策略，例如总是往右，它使用这套策略在环境中活动，而环境也会给它反馈，很显然，总是往右不是一个好的策略，所以反馈值也不太好，如图 5-2 所示。

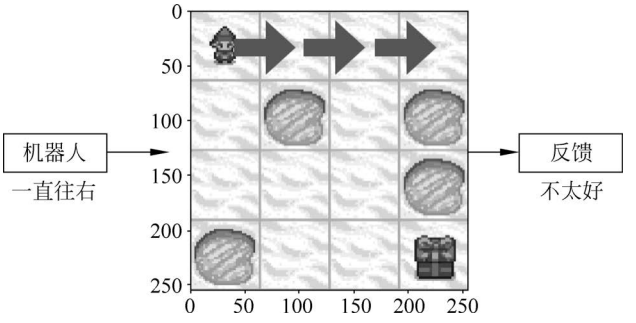

图 5-2　机器人使用策略和环境互动(1)

机器人获得了反馈以后，它会尝试着改变一下策略，看一看反馈是否有上升，如图 5-3 所示。

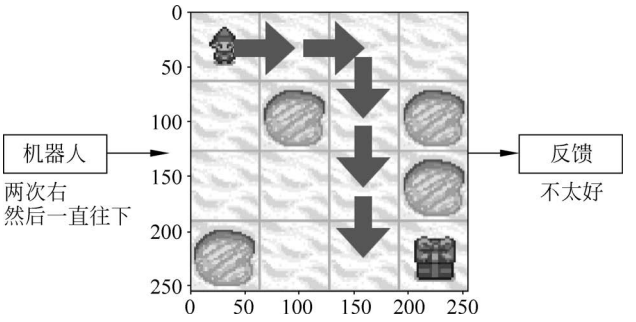

图 5-3　机器人使用策略和环境互动(2)

机器人改变策略后应该说是更靠近成功了，在该策略下，机器人只差一步就能获得礼物了，但此时此刻，环境还是会给予它不太好的反馈。

得到反馈后，也许机器人还会改变一下策略，这样的改变是非常随机的，几乎就是瞎猜的，但总有让它猜中的时候，如图 5-4 所示。

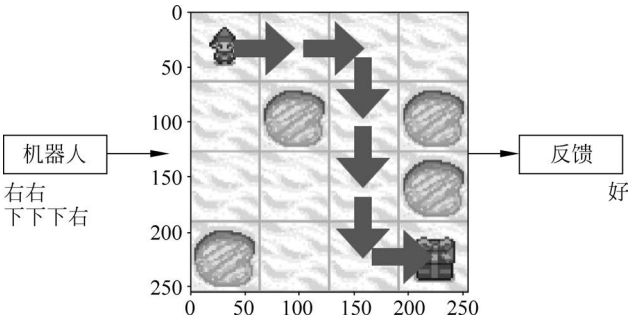

图 5-4　机器人使用策略和环境互动(3)

这时机器人终于取得了突破，环境给予了它很好的反馈，此时机器人获得了一个很好的策略，使用这套策略，它能很好地处理冰湖这个游戏环境。

上面举的例子其实就是基于策略的算法，它的核心思想不再是计算出 Q 表，而是找到一个好的策略，根据该策略做出决策，避免了计算 Q 表。不过其中还有很多细节问题没有解决。

上面举的例子看起来就是在瞎猜，似乎机器人只是凑巧找到了一个好的策略，但其实选择策略时也有很多技巧，并不是完全在瞎猜，其中重要的指导性思想就是策略梯度方法。

和时序差分方法一样，策略梯度方法也是帮助算法更新的方法，下面从感性层面上来认识一下策略梯度方法。

5.2　一个直观的例子

为了更好地理解策略梯度的思想，下面举一个更简单的例子进行说明，如表 5-2 所示。

表 5-2　按钮的中奖概率

	中奖概率
按钮 1	0.2
按钮 2	0.5
按钮 3	0.8

这是 3 个中奖概率已知的按钮，根据概率表，应该永远选择按钮 3，因为这是最优策略，在概率已知的情况下当然会推理出这样的行动策略。

但是现在让我们假设概率是未知的，并且按按钮的次数是有限的，如何寻求最优行动策略呢？

有经验的读者可能已经发现，这是一个典型的求探索利用比例的问题，此类问题有一个典型特征，就是要把有限的行动次数分为两部分，一部分被称为探索，即要尽量地求出环境信息，另一部分被称为利用，即利用上一步已经探索出的环境信息获得尽可能多的奖励。

在游戏开始的初期，可以通过大量的尝试，摸索出各个按钮中奖的概率，进而求出中奖概率最大的按钮，明确了这一点以后，把剩余的尝试次数全部用于按该按钮即可，这样就可以最大化地获得奖励，该算法被称为贪婪算法。

贪婪算法求解过程中的难点是确定探索和利用各自的比例，如果探索的比例太小，则可能并没有找到中奖概率最高的按钮，进而影响利用时所能获得的奖励，而探索太多，虽然能更加明确中奖概率最高的按钮，但是由于探索占用了太多的行动次数，所以会导致利用时的行动次数不足，难以获得大量的奖励，从而导致最终性能不佳。

读者可能会想在这样的一个求解过程中，应该有一个最佳平衡点，以此来平衡探索和利用两部分的占用比例，如果读者这样想了，那其实就已经在用策略梯度的思路来求解该问题了。

接下来梳理一下策略梯度方法是如何求解该问题的。从游戏的开局状态开始，此时策略如表 5-3 所示。

表 5-3　开局阶段的策略

	真实中奖概率	选择概率	猜测中奖概率
按钮 1	0.2	0.33	1.0
按钮 2	0.5	0.33	1.0
按钮 3	0.8	0.33	1.0

因为此时对每个按钮中奖的概率一无所知，所以认为它们中奖的概率都是 1.0，也就是说在开局阶段，对它们的期望都是很高的，在后续的不断尝试中会逐渐降低对它们的期望，

最终收敛到真实中奖概率。

由于对所有按钮的中奖概率未知，所以每个按钮被选择的概率都是相等的，即都是0.33，在后续的不断尝试中会渐渐降低中奖概率低的按钮的被选择概率，从而节省宝贵的行动次数。

经过一些步数的尝试，策略会逐渐调整，渐渐减少探索，增加探索，此时策略如表5-4所示。

表5-4 经过一些步数的尝试后的策略

	真实中奖概率	选择概率	猜测中奖概率
按钮1	0.2	0.1	0.3
按钮2	0.5	0.1	0.6
按钮3	0.8	0.8	0.7

此时的猜测中奖概率已经逐渐靠近真实中奖概率，每个按钮被选择的概率也逐渐接近利用，探索的成分逐渐减少，此时机器人从环境中获取奖励的效率比最开始时大幅度地提升了。

再经过一些步数的尝试，此时策略如表5-5所示。

表5-5 再经过一些步数的尝试后的策略

	真实中奖概率	选择概率	猜测中奖概率
按钮1	0.2	0.001	0.211
按钮2	0.5	0.001	0.511
按钮3	0.8	0.998	0.799

此时的策略基本收敛，行动次数基本是利用，到这里策略基本不再发生改变，机器人的行为方式趋于稳定，从环境获得奖励的效率最大化。

综上所述，在整个算法的进步过程中不计算 Q 表，只是调整不同动作被选择的概率，每次改变一点点，每次进步一点点，最终求得最优的策略，这就是策略迭代方法的思路。

至此可以概述地说，策略迭代的基本思路如下：

(1) 在特定状态下有 N 个可选择的动作，每次以特定的概率选择一个动作执行。

(2) 根据后续得到的反馈，修改各个动作被选择的概率。

(3) 修改的依据是提高得高分的动作的概率，而降低得分低的动作的概率。

(4) 反复迭代多次，最终收敛各个动作被选择的概率，也就求得了最优策略。

以上就是策略迭代算法的基本思路。

5.3 数学表达

经过上面的感性认识，相信读者已经大致了解了策略梯度方法是如何求得最优策略的，下面从数学层面大致介绍策略梯度函数的推导过程。

由于强化学习算法是从数据中学习的，所以首先要有数据才能谈后续的优化算法，一局正常的游戏会产生的数据大致如式(5-1)所示。

$$S_t, A_t, R_t, \quad S_{t+1}, A_{t+1}, R_{t+1}, \quad S_{t+2}, A_{t+2}, R_{t+2}, \cdots \tag{5-1}$$

根据策略梯度的思想，要求最大化后续的回报，但是此时此刻的时间是 t，对后续回报部分的最大化不能简单地进行求和，因为每个时刻的回报都是估计的，还没有发生，并且每个回报的估计程度都不同，越遥远的回报估计的成分越高，对目前动作的选择影响应该越小。为了体现这一点，所以将后续回报的折扣求和写为 U 函数，如式(5-2)所示。

$$U_t = R_t + \text{gamma} \cdot R_{t+1} + \text{gamma}^2 \cdot R_{t+2} + \cdots + \text{gamma}^{n-t} \cdot R_n \tag{5-2}$$

大体上来看，U 函数是对每个回报进行加权求和，每步的回报都有一个权重，并且是越远的越小，因为越远的回报对目前的影响越小，最关心的是目前的回报，下一步的回报权重就会减轻，越往后减轻得越厉害。

接下来看 Q 函数，Q 函数是 U 函数的期望，Q 函数的定义如式(5-3)所示。

$$Q(s_t, a_t) = E[U_t \mid S_t = s_t, A_t = a_t] \tag{5-3}$$

从 Q 函数的定义来看，它计算的是在特定状态下执行特定动作，可以得到的后续折扣回报的和的期望，所以 Q 函数对选择动作具有非常强的指导意义。

虽然在前面的部分谈到在策略梯度算法中不需要计算 Q 表，但在策略梯度的推导中依然要用到 Q 函数，因为 Q 函数太重要了，只要是涉及动作选择的都绕不开它，但是在最终策略梯度算法的优化过程中会有办法替代 Q 函数的计算，此处暂时按下不表，在后续会进行说明。

定义 V 函数，V 函数表明了在一定的动作选择策略下，计算 Q 函数的期望，V 函数的定义如式(5-4)所示。

$$V_{\text{pi}}(s_t) = E_{A_t \sim \text{pi}(*|s_t, \text{theta})}[Q_{\text{pi}}(s_t, A_t)] \tag{5-4}$$

从式(5-4)可以看出，V 函数是 Q 函数的期望，这里的期望是和动作无关的，即求的是某种状态的价值，在一个好的状态无论做什么动作都不会太差，相反，在一个差的状态，无论做什么动作都不会太好，因此 V 函数评估的是某种状态的价值。

定义 J 函数，J 函数是对 V 函数的期望，J 函数的定义如式(5-5)所示。

$$J(\text{theta}) = E_S[V_{\text{pi}}(S)] \tag{5-5}$$

式(5-5)中的 theta 指的是策略的参数，通过调整 theta 进而影响策略，因此 J 函数计算的是某个策略的价值，即以某种特定的策略、方式去玩游戏，期望可以得到的分数有多高，一个好的策略总是能得高分，而差的策略总是得低分，所以说 J 函数评估的是策略的价值，或者说 J 函数评估的是一套动作策略是好还是不好。

根据生活中的一般经验，同样的游戏状态交给高手处理，总是比普通玩家要更好一些，因为高手有比普通玩家更优的策略参数，高手比普通玩家优秀是和游戏状态无关的，任何状态都是高手要更好一些。根据该指导思想，求 J 函数的最大化参数就是高手的策略参数了，也就找到了最优的行动策略，如式(5-6)所示。

$$\max_{\text{theta}} \rightarrow J(\text{theta}) \tag{5-6}$$

从式(5-6)可以看出,这要求一个函数的极值,而求一个函数的极值一般使用梯度上升方法,如式(5-7)所示。

$$\text{theta}_{\text{new}} = \text{theta}_{\text{now}} + \text{beta} \cdot \nabla_{\text{theta}} J(\text{theta}_{\text{now}}) \tag{5-7}$$

式(5-7)是一个典型的梯度上升的过程,只要反复迭代该式就可以求得原函数的极值。

式(5-7)计算的难点在于对 J 函数的求导,这里跳过复杂的求导过程,直接给出 J 函数的导函数,如式(5-8)所示。

$$\nabla_{\text{theta}} J(\text{theta}) = E_S[E_{A\sim\text{pi}(*|S;\text{theta})}[Q_{\text{pi}}(S,A) \cdot \nabla_{\text{theta}} \ln\text{pi}(A|S;\text{theta})]] \tag{5-8}$$

式(5-8)即 J 函数的导函数,这个式子看起来十分复杂,但其实最复杂的,恐怕还是包括了 Q 函数。记得本书在前面介绍过想要精确计算 Q 函数是困难的,甚至是不可能的,所以这里的 Q 函数必须想办法替代,否则该函数将无法计算。

Q 函数的定义如式(5-3),可见 Q 函数是 U 函数的期望,U 函数的定义如式(5-2),对 U 函数的计算过程稍做变形,得到式(5-9)。

$$U_t = \sum_{k=t}^{n} \text{gamma}^{k-t} \cdot R_k \tag{5-9}$$

式(5-9)显然是一个可以计算的函数,既然 Q 函数是 U 函数的期望,就可以直接使用 U 函数的计算结果作为 Q 函数的无偏估计,这样的替代被称为蒙特卡洛采样法,替换后的 J 函数的导函数如式(5-10)所示。

$$\nabla_{\text{theta}} J(\text{theta}) = E_S\left[E_{A\sim\text{pi}(*|S;\text{theta})}\left[\sum_{k=t}^{n} \text{gamma}^{k-t} \cdot R_k \cdot \nabla_{\text{theta}} \ln\text{pi}(A|S;\text{theta})\right]\right] \tag{5-10}$$

至此,可以写出 theta 的更新过程,如式(5-11)所示。

$$\text{theta}_{\text{new}} = \text{theta}_{\text{now}} + \text{beta} \cdot \sum_{t=1}^{n} \text{gamma}^{t-1} \cdot U_t \cdot \nabla_{\text{theta}} \ln\text{pi}(A|S;\text{theta}) \tag{5-11}$$

式(5-11)中的 U_t 即式(5-9)。

以上就是策略梯度方法中更新函数的数学推导,其中的重点是对 J 函数的求导过程,最后得到式(5-10)的导函数,不过该函数的写法比较复杂,一般会写成式(5-12),这样会更加清晰一些,容易理解,也容易翻译为计算机代码。

$$\nabla_{\text{theta}} J(\text{theta}) = E\left[\sum_{i=0}^{T}\left(\sum_{j=i}^{T} \text{gamma}^{j=i} r_j\right) \nabla_{\text{theta}} \text{logpi}_{\text{theta}}(a_i|s_i)\right] \tag{5-12}$$

虽然式(5-12)和式(5-10)表达的是同样的内容,但是式(5-12)看起来更加清晰直观,所以后续将会以式(5-12)为蓝本翻译为计算机代码,应用于策略梯度算法中。

式(5-12)在策略梯度算法中的应用很多,该公式十分重要,读者务必熟悉该函数。

5.4 小结

本章向读者介绍了在强化学习中非常重要的策略迭代的思想,所谓策略即机器人处理环境下各种状态的方法,好的策略可以得到高分,坏的策略常常得到低分,寻找一个好的策

略，也就是策略迭代算法要完成的工作。

为了帮助读者理解策略迭代的思路，举了按按钮的例子向读者说明策略迭代的一般过程，这是一个感性的认识，读者需大致留下策略迭代过程的印象，在后续章节中会反复使用该思想。

最后从数学的角度推导了策略迭代方法的优化过程，事实上策略迭代方法是一个梯度上升的过程，求的是时刻 t 后续折扣回报的最大化，最后得出了梯度上升的导函数，该导函数十分重要，在后续章节中会反复使用，读者务必熟悉该函数，如果忘记了该函数，则可以回到本章来复习。