2 强化学习入门

本章将介绍传统强化学习的基础，并概览深度强化学习。我们将从强化学习中的基本定义和概念开始，包括智能体、环境、动作、状态、奖励函数、马尔可夫（Markov）过程、马尔可夫奖励过程和马尔可夫决策过程，随后会介绍一个经典强化学习问题——赌博机问题，给读者提供对传统强化学习潜在机理的基本理解。这些概念是系统化表达强化学习任务的基石。马尔可夫奖励过程和价值函数估计的结合产生了在绝大多数强化学习方法中应用的核心结果——贝尔曼（Bellman）方程。最优价值函数和最优策略可以通过求解贝尔曼方程得到，还将介绍三种贝尔曼方程的主要求解方式：动态规划（Dynamic Programming）、蒙特卡罗（Monte-Carlo）方法和时间差分（Temporal Difference）方法。

我们进一步介绍深度强化学习策略优化中对策略和价值的拟合。策略优化的内容将会被分为两大类：基于价值的优化和基于策略的优化。在基于价值的优化中，我们介绍基于梯度的方法，如使用深度神经网络的深度 Q 网络（Deep Q-Networks）；在基于策略的优化中，我们详细介绍确定性策略梯度（Deterministic Policy Gradient）和随机性策略梯度（Stochastic Policy Gradient），并提供充分的数学证明。结合基于价值和基于策略的优化方法产生了著名的 Actor-Critic 结构，这导致诞生了大量高级深度强化学习算法。

2.1 简介

本章介绍强化学习和深度强化学习的基础知识，包括基本概念的定义和解释、强化学习的一些基本理论证明，这些内容是深度强化学习的基础。因此，我们鼓励读者能够掌握本章的内容后再去学习之后的章节。下面，从强化学习的基本概念开始学习。

如图 2.1 所示，**智能体**（Agent）与**环境**（Environment）是强化学习的基本元素。环境是智能

体与之交互的实体。如图 2.2 右边所示，一个环境可以是一个雅达利乒乓球游戏（Pong Game）。智能体控制一个球拍来反弹小球，从而使环境产生变化。智能体的“交互”是通过预先定义好的**动作集合**（Action Set）$\mathcal{A} = \{A_1, A_2...\}$ 来实现的。动作集合描述了所有可能的动作。在这个乒乓球游戏中，动作集合是球拍 {向上移动, 向下移动}。那么强化学习的目的就是教会智能体如何很好地与环境交互，从而在预先定义好的**评价指标**（Evaluation Metric）下获得好的成绩。在乒乓球游戏中，评价指标是玩家获得的分数。若小球穿过了对手的防线，智能体则获得奖励 $r = 1$。相反，若小球穿过了智能体的防线，则智能体获得奖励 $r = -1$。

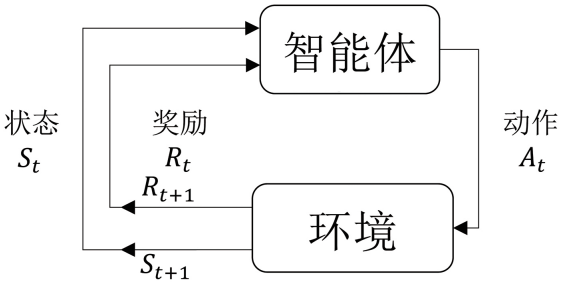


图 2.1　智能体与环境

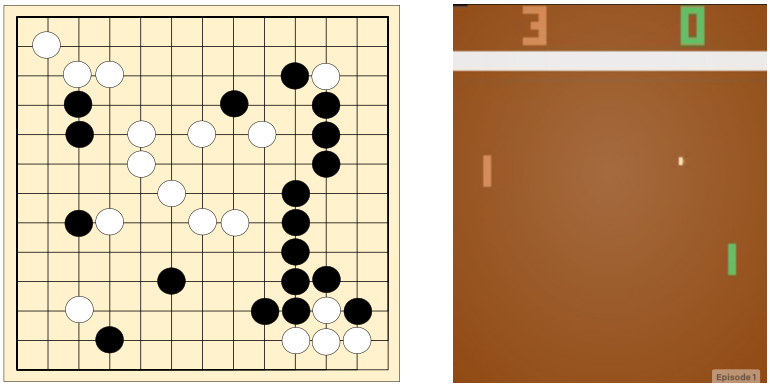


图 2.2　两类游戏环境：围棋（左边）的观测包含了环境状态的所有信息，这个环境是完全可观测的。雅达利乒乓球（右边）的观测如果只有单帧画面，不能包含小球的速度和运动方向，这个环境是部分可观测的

我们现在通过图 2.1 来看看智能体与环境的关系细节。在任意的一个时间步（Time Step）t，智能体首先观测到当前环境的状态 S_t，以及当前对应的奖励值 R_t。基于这些状态和奖励信息，智能体决定如何行动。智能体要执行的动作 A_t 从环境得到新的反馈，获得下一时间步的状态 S_{t+1} 和奖励 R_{t+1}。对环境状态 s（s 是一个与时间步 t 无关的通用状态表示符号）的**观测**（Observation）并不一定能保证包含环境的所有信息。如果观测只包含了环境的局部状态信息

（Partial State Information），这个环境是**部分可观测的**（Partially Observable）。而如果观测包含了环境的全部状态信息（Complete State Information），这个环境是**完全可观测的**（Fully Observable）。在实践中，观测通常是系统真实状态的函数，使得我们有时很难辨别观测是否包含了状态的所有信息。一个更容易理解的方法是从信息角度，一个完全可观测的环境不应从整个环境的潜在状态中遗漏任何信息，而应该可以把所有信息提供给智能体。

为了更好地理解部分可观测环境和完全可观测环境的区别，我们来看两个例子：图 2.2 左边的围棋游戏是一个典型的完全可观测环境的例子，环境的信息是所有的棋子的位置。而在图 2.2 右边的雅达利乒乓球游戏中，如果观测是单帧画面，就是一个部分可观测环境。这是因为小球的速度和运动方向并不能从单帧画面中获得。

在很多强化学习的文献中，在环境对智能体是完全可观测的条件下，动作 a（a 是一个与时间步 t 无关的通用动作表示符号）通常是基于状态 s 表示的智能体动作。而如果环境对智能体是部分可观测的，智能体不能直接获得环境潜在状态（Underlying State）的信息，因此在没有其他处理时，动作是基于观测量（Observation）而不是真正状态 s 的。

为了从环境中给智能体提供反馈，一个**奖励函数**（Reward Function）记为 R，会根据环境状态而在每一个时间步上产生一个**立即奖励**（Immediate Reward）R_t，并将其发送给智能体。在一些情况下，奖励函数只取决于当前的状态，即 $R_t = R(S_t)$。例如，在乒乓球游戏中，如果小球穿过了对手的防线，玩家会立即获得正数的奖励。这个例子中，奖励函数只取决于当前状态，但是在很多情况下，奖励函数不仅取决于当前状态，而且取决于当前的动作，甚至可能是之前的状态和动作。一个简单的例子是：如果我们需要一个智能体记住环境中另一个智能体的一系列连续动作，并重复模仿执行。一个动作的偏差会导致后续状态和动作都难以对齐，那么这个奖励不仅需要考虑另一个智能体和这个智能体运动过程中的状态-动作对（State-Action Pair），而且需要考虑状态-动作对的序列。这时，基于当前状态的奖励函数，或者基于当前状态和动作的函数，都无法对智能体模仿整个连续序列有足够的指导性意义。

在强化学习中，**轨迹**（Trajectory）是一系列的状态、动作和奖励：

$$\tau = (S_0, A_0, R_0, S_1, A_1, R_1, \cdots)$$

用以记录智能体如何和环境交互。轨迹的初始状态 S_0，是从**起始状态分布**（Start-State Distribution）中随机采样而来的，该状态分布记为 ρ_0，从而有 $S_0 \sim \rho_0(\cdot)$。例如，雅达利乒乓球游戏开始的状态总是小球在画面的正中间。而围棋的开始状态则可以是棋子在棋盘上的任意位置。

一个状态到下一个状态的**转移**（Transition）可以分为：要么是**确定性转移过程**（Deterministic Transition Process），要么是 **随机性转移过程**（Stochastic Transition Process）。对于确定性转移过程，下一时刻的状态 S_{t+1} 由一个确定性函数支配：

$$S_{t+1} = f(S_t, A_t), \tag{2.1}$$

其中 S_{t+1} 是唯一的下一个状态。而对于随机性转移过程，下一时刻的状态 S_{t+1} 是用一个概率分布（Probabilistic Distribution）来描述的：

$$S_{t+1} \sim p(S_{t+1}|S_t, A_t) \tag{2.2}$$

而下一时刻的实际状态是从其概率分布中采样得到的。

一个轨迹有时候也称为**片段**（Episode）或者回合，是一个从初始状态（Initial State）到最终状态（Terminal State）的序列。比如，玩一整盘游戏的过程可以看作一个片段，若智能体赢了或者输了这盘游戏，则到达最终状态。在一些时候，一个片段可以是由多局子游戏（Sub-Games）组成的（而不仅仅是一盘游戏）。比如在雅达利乒乓球游戏中，一个片段可以包含多个回合。

我们用两个重要的概念来结束本小节：**探索**（Exploration）与**利用**（Exploitation，有时候也叫守成），以及一个著名的概念：**探索-利用的权衡**（Exploration-Exploitation Trade-off）。**利用**指的是使用当前已知信息来使智能体的表现达到最佳，而智能体的表现通常是用期望奖励（Expected Reward）来评估的。举例来说，一个淘金者发现了一个每天能提供两克黄金的金矿，同时他也知道最大的金矿可以每天提供五克黄金。但是如果他花费时间去找更大的金矿，就需要停下挖掘当前的金矿，这样的话如果找不到更大的金矿，那么在找矿耗费的时间中就没有任何收获。基于这位淘金者的经验，去探索新的金矿会有很大的风险，淘金者于是决定继续挖掘当前的金矿来最大化他的奖励（这个例子中奖励是黄金的数量），他放弃了**探索**而选择了**利用**。淘金者选择的**策略**（Policy）是**贪心**（Greedy）策略，即智能体持续地基于当前已有的信息来执行能够最大化期望奖励的动作，而不去做任何的冒险行为，以免导致更低的期望奖励。

探索是指通过与环境交互来获得更多的信息。回到淘金者的例子中，探索指的是淘金者希望花费一些时间来寻找新的金矿，而如果他找到更大的金矿，那么他每天能获得更多的奖励。但是为了获得更大的**长期回报**（Long-Term Return），**短期回报**（Short-Term Return）可能会被牺牲。淘金者需要面对在探索与利用间抉择的难题，要决定当一个金矿产量为多少时应当进行**利用**而少于多少时应当开始**探索**。上述的例子描述了**探索-利用的权衡**问题，这个问题关乎智能体如何平衡**探索**和**利用**，是强化学习研究非常重要的问题。我们下面进一步通过赌博机问题（Bandit Problem）来讨论它。

2.2　在线预测和在线学习

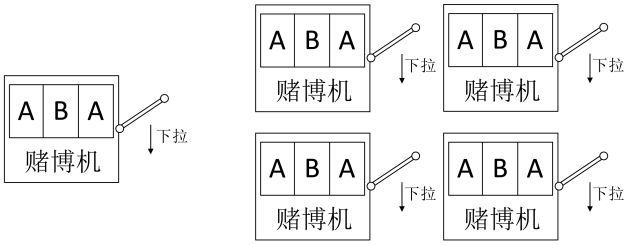
2.2.1　简介

在线预测（Online Prediction）问题是一类智能体需要为未来做出预测的问题。假如你在夏威夷度假一周，需要预测这一周是否会下雨；或者根据一天上午的石油价格涨幅来预测下午石油的价格。在线预测问题需要在线解决。在线学习和传统的统计学习有以下几方面的不同：

- 样本是以一种有序的（Ordered）方式呈现的，而非无序的批（Batch）的方式。

- 我们更多需要考虑最差情况而不是平均情况，因为我们需要保证在学习过程中随时都对事情有所掌控。
- 学习的目标也是不同的，在线学习企图最小化后悔值（Regret），而统计学习需要减少经验风险。我们会稍后对后悔值进行介绍。

如图 2.3 左侧所示，**单臂赌博机**（Single-Armed Bandit）是一种简单的赌博机，智能体通过下拉机械手臂来和这个赌博机进行互动。当这个机器到达头奖的时候，这个智能体就会得到一个奖励。在赌场中，我们常常能看见很多赌博机被摆在一排。一个智能体就可以选择去下拉其中任何一只手臂。奖励值 r 的分布 $P(r|a)$ 以动作 a 为条件，它对于不同的赌博机来说是不同的，但是对某一台赌博机来说是固定的。智能体在一开始是不知道奖励分布的，而只能通过不断的实验和尝试来增进对分布的了解。智能体的目标是将其做出一些选择后所得到的奖励最大化。智能体需要在每个时间步上从众多的赌博机中进行选择，我们把这种游戏称为**多臂赌博机**（Multi-Armed Bandit，MAB），如图 2.3 中右侧所示。MAB 给予了一个智能体有策略地选择拉下哪一根拉杆的自由。



单臂赌博机　　多臂赌博机

图 2.3　单臂赌博机（左）与多臂赌博机（右）

我们尝试通过一般的强化学习方法来解决 MAB 问题。智能体的动作 a 用来选择具体拉哪一根拉杆。在这个动作完成以后，它会得到一个奖励值。在时间步 t 的一个动作 a 的价值定义为

$$q(a) = \mathbb{E}[R_t|A_t = a]$$

我们试图用它来选择动作。如果我们知道了每个动作 a 的真实的动作值 $q(a)$，那么解决这个问题就很简单，只需始终选择对应最大 q 值的动作即可。然而，现实中我们往往要估计 q 值，把它的估计值写为 $Q(a)$，而 $Q(a)$ 值应当尽可能接近 $q(a)$ 的值。

对于展示探索-利用的权衡问题，MAB 可以作为一个很好的例子。当我们已经对一些状态的 q 值进行估计之后，如果一个智能体一直选择有最大 Q 值的动作的话，那么这个智能体就是贪心的（Greedy），因为它一直在利用已经估计过的 q 值。如果一个智能体总是根据最大化 Q 值来选取动作，那么我们认为这样的智能体是有一定探索（Exploration）性的。只做探索或者只对已有估计值进行利用（Exploitation），在大多数情况下都不能很好地改善策略。

一个种单的基于动作价值的（Action-Value Based）方法是，通过将在时间 t 前选择动作 a 所获得的总体奖励除以这个动作被选择的次数来估算 $Q_t(a)$ 的值：

$$Q_t(a) = \frac{\text{在时间 } t \text{ 前选择动作 } a \text{ 的奖励值的总和}}{\text{在时间 } t \text{ 前动作 } a \text{ 被选择的次数}} = \frac{\sum_{i=0}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=0}^{t-1} \mathbb{1}_{A_i=a}}$$

$\mathbb{1}_x$ 的值在 x 为真时为 1，否则为 0。一种贪心的策略可以写成：

$$A_t = \arg\max_a Q_t(a) \tag{2.3}$$

然而，我们也可以把这个贪心策略转化成有一定探索性的策略，即让它以 ϵ 的概率去探索其他动作。我们把这种方法叫作 ϵ-贪心（ϵ-Greedy），因为它在概率为 ϵ 的情况下随机选择一个动作，而在其他情况下，它的动作是贪心的。如果我们有无限的时间步长，那么就可以保证 $Q_t(a)$ 收敛为 $q(a)$。更重要的是，这个简单的基于动作-价值的方法也是一种基于在线学习（Online Learning）的方法。

让我们以多臂赌博机问题为例来具体介绍在线学习。假设我们在每个时间步 t 上观测到了回报 R_t，一个简单的用来找最佳动作的想法是通过 R_t 和 A_t 来更新 q 的估计。之前介绍的用来计算平均值的办法是对所有在时间 t 之前选择 A_t 的奖励值求和，然后除以 A_t 出现的次数。这样更像一个批量学习，因为每一次我们都得对一批数据点进行重新计算。在线学习的方法则利用一个移动的平均值，每次运算都基于之前的估算结果，如 $Q_i(A_t) = Q_i(A_t) - Q_i(A_t)/N; Q_{i+1}(A_t) = Q_i(A_t) + R_t/N$。$Q_i$ 是在 A_t 被选择过 i 次以后的 q 估计值，N 是 A_t 被选择的次数。

2.2.2　随机多臂赌博机

当我们有 $K \geqslant 2$ 只机器手臂时，我们需要在每个时间步上 $t = 1, 2, \cdots, T$ 下拉一只手臂。在任何时间 t，如果我们下拉的手臂是第 i 只，那么相应地也会观察到奖励 R_t^i。

算法 2.3 多臂赌博机学习

初始化 K 只手臂
定义总时长为 T
每一只手臂都有一个对应的 $v_i \in [0, 1]$。每一个奖励都是独立同分布地从 v_i 中采样得到的
for $t = 1, 2, \cdots, T$ **do**
　智能体从 K 只手臂中选择 $A_t = i$
　环境返回奖励值向量 $\boldsymbol{R}_t = (R_t^1, R_t^2, \cdots, R_t^K)$
　智能体观测到 R_t^i
end for

从传统意义上来说，我们会尝试最大化奖励值。但是在随机多臂赌博机（Stochastic Multi-Armed Bandit）里，我们会关注另外一个指标（Metric），即后悔值（Regret）。在 n 步之后的后悔

值被定义为

$$\mathrm{RE}_n = \max_{j=1,2,\cdots,K} \sum_{t=1}^{n} R_t^j - \sum_{t=1}^{n} R_t^i$$

第一项是我们走到 n 步之后，每一次都能获得的最大奖励值之和；第二项是在 n 步中，真实获得的奖励之和。

因为我们的动作和回报带来了随机性，为了选择最好的动作，我们应该尝试最小化后悔值的期望值。我们需要把两种不同的后悔值的期望值区分开来：后悔值和伪后悔值（Pseudo-Regret）的期望值。我们将后悔值的期望值定义为

$$\mathbb{E}[\mathrm{RE}_n] = \mathbb{E}[\max_{j=1,2,\cdots,T} \sum_{t=1}^{n} R_t^j - \sum_{t=1}^{n} R_t^i] \tag{2.4}$$

我们将伪后悔值的期望值定义为

$$\overline{\mathrm{RE}}_n = \max_{j=1,2,\cdots,T} \mathbb{E}\left[\sum_{t=1}^{n} R_t^j - \sum_{t=1}^{n} R_t^i\right] \tag{2.5}$$

以上两种后悔值最主要的区别在于它们最大化和计算期望值的顺序是不一样的。后悔值的期望值会相对更难计算一些，这是因为对于伪后悔值来说，我们只需要优化后悔值的期望值；而对于后悔值的期望值来说，我们则需要每次试验时都找到最优的后悔值再取期望值。而这两个值满足一定关系，即 $\mathbb{E}[\mathrm{RE}_n] \geqslant \overline{\mathrm{RE}}_n$。

定义 μ_i 为 v_i 的平均值，而 v_i 是第 i 只手臂的奖励值，$\mu^* = \max_{i=1,2,\cdots,T} \mu_i$。在一个随机的环境下，我们把公式 (2.5) 改写为

$$\overline{\mathrm{RE}}_n = n\mu^* - \mathbb{E}\left[\sum_{t=1}^{n} R_t^i\right] \tag{2.6}$$

一种最小化伪后悔值的方法是选择最好的那只手臂来下拉，并通过之前介绍的 ϵ-贪心策略来获得样本。一种更先进的方法叫作置信上界（Upper Confidence Bound，UCB）算法。置信上界算法使用霍夫丁引理（Hoeffding's Lemma）来估计置信上界，然后选择那只基于目前估计对应最大奖励平均值的手臂。

我们现在开始介绍置信上界策略。具体关于置信上界算法在随机多臂赌博机中对后悔值的优化可以在 (Bubeck et al., 2012) 中找到。我们现在来具体看一看置信上界基于奖励进行策略优化的过程。尽管在随机 MAB 里，奖励是从一个分布中采样得到的，这个奖励函数分布在时间上是稳定的。以 ϵ-贪心策略为例，ϵ-贪心以一定概率（值为 ϵ）来探索那些非最优动作，但问题是，它认

为所有的非最优动作都是一样的，从而不对这些动作进行任何区别对待。如果我们想尝试每一个动作，则可能需要优先尝试那些没有采用过的或者采用次数更少的动作。置信上界算法通过改写公式 (2.3) 来解决这个问题:

$$A_t = \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right] \tag{2.7}$$

式中，$N_t(a)$ 是动作 a 在到时间 t 前被选择的次数；c 是一个决定还需要进行多少次探索的正实数。如果我们有一个稳定的奖励函数分布，可以通过公式 (2.7) 来选择动作。当 $N_t(a)$ 为零时，认为动作 a 有最大值。为了更好地了解置信上界算法的具体运作方式，平方根项反映了我们对 a 的 q 值估算的不确定性：随着 a 被选择的次数增加，它的不确定性也在减小。同样地，当除 a 外的动作被选择后，不确定性就变大了，因为 $\ln t$ 增大但是 $N_t(a)$ 保持不变。t 的自然对数使得新的时间步的影响越来越小。置信上界算法给出动作 q 值的上限，而 c 表示置信程度。

2.2.3　对抗多臂赌博机

随机 MAB 的回报函数是用随时间不变的概率分布来表示的。但是在现实中，这个条件往往不成立。因此，在奖励函数不再简简单单地由一些稳定概率分布决定，而由一个对抗者（Adversary）决定的情况下，我们需要研究对抗多臂赌博机（Adversarial Multi-Armed Bandit）。在对抗多臂赌博机的情景中，第 i 只手臂在时间 t 上的奖励为 $R_t^i \in [0, 1]$。同时一个玩家在 t 时所拉的手臂会被写作 $I_t \in \{1, 2, \cdots, K\}$。

有人可能会想，万一对抗者干脆把所有的奖励都设为 0 了呢？如果这种情况发生，就没有人可以得到任何的奖励。事实上，就算对抗者可以自由决定奖励的多少，也不会把所有的奖励都设为 0，反之给玩家足够多的奖励作为诱惑，让他们有赢的感觉，但是其实玩了许多轮后，最终还是对抗者获利。

算法 2.4 是对抗多臂赌博机的基本设定。在每一个时间步上，智能体都会选择一只手臂 I_t，而对抗者会决定在这个时刻的奖励值向量 $\boldsymbol{R}_t$。这个智能体有可能只能观测到它所选择的手臂的奖励 $R_t^{I_t}$，也有可能观测到每一个机器的奖励 $\boldsymbol{R}_t(\cdot)$。分两点来完整描述这个问题。第一点是，一个对抗者到底对一个玩家之前的动作选择了解多少。这个很重要，对抗者可能会为了获得更多的利益根据玩家的动作来调整机器。我们将那些不考虑过去玩家历史的对抗者叫作**健忘对抗者**（Oblivious Adversary），而将那些考虑过去历史的对抗者叫作**非健忘对抗者**（Non-Oblivious Adversary）。第二点是一个玩家能够了解到奖励值向量的多少内容。我们将那些玩家知道关于奖励值向量的全部信息的情况叫作**全信息博弈**（Full-Information Game），而将那些玩家只知道一部分回报向量信息的情况叫作**部分信息博弈**（Partial-Information Game）。

健忘对抗者和非健忘对抗者的区别，只对一个非确定性（Non-Deterministic）玩家才显现出来。如果我们有一个确定性玩家，或者一个玩家的策略不变，一个对抗者就很容易让后悔值 $\overline{\mathrm{RE}} \geqslant n/2$，

其中 n 代表这个玩家下拉手臂的次数。所以，全信息非确定性玩家更有研究价值，可以使用 Hedge 算法来解决这个问题。

算法 2.4 对抗多臂赌博机

初始化 K 只机器手臂
for $t = 1, 2, \cdots, T$ **do**
  智能体在 K 只手臂当中选中 I_t
  对抗者选择一个奖励值向量 $\boldsymbol{R}_t = (R_t^1, R_t^2, \cdots, R_t^K) \in [0, 1]^K$
  智能体观察到奖励 $R_t^{I_t}$（根据具体的情况也有可能看到整个奖励值向量）
end for

在算法 2.5 中，我们首先把每只手臂的函数 G 都设为零，然后使用 Softmax 来获得一个新动作的概率密度函数（Probability Density Function）。η 是一个用来控制温度的正值参数。G 函数更新是通过把所有的手臂的新奖励值都加起来，从而使有最高奖励值的手臂有最大的概率被选中的。我们把这个算法叫作 Hedge。Hedge 也是部分信息博弈方法的一个基础。如果我们把一个智能体的观察局限到只有 R_t^i，那么就需要把奖励标量扩展成一个向量，这样它才可以被 Hedge 使用。探索和利用的指数加权算法（Exponential-Weight Algorithm for Exploration and Exploitation，Exp3）即为一个基于 Hedge 来解决不完全信息博弈的算法。它进一步利用了 $p(t)$ 和平均分布（Uniform Distribution）的结合来确保所有的机器都会被选到，达到了平衡探索和利用的目的。文献 (Auer et al., 1995) 中有关于探索和利用的指数加权算法更详尽的介绍。

算法 2.5 针对对抗多臂赌博机的 Hedge 算法

初始化 K 只手臂
$G_i(0)$ for $i = 1, 2, \cdots, K$
for $t = 1, 2, \cdots, T$ **do**
  智能体从 $p(t)$ 分布中选择 $A_t = i_t$，其中

$$p_i(t) = \frac{\exp(\eta G_i(t-1))}{\sum_j^K \exp(\eta G_j(t-1))}$$

  智能体观测到奖励 g_t
  让 $G_i(t) = G(t-1) + g_t^i,\ \forall i \in [1, K]$
end for

2.2.4 上下文赌博机

上下文赌博机（Contextual Bandit）有的时候也被叫作**关联搜索**（Associative Search）任务。我们把关联搜索任务和**非关联搜索**（Non-Associative Search）任务放在一起，以更好地了解它们的意义。我们刚刚所描述的多臂赌博机就是一个非关联搜索任务。当一个任务的奖励函数是稳定的

时候，我们只需要找到那个最好的动作。当一个任务是不稳定的时候，我们就需要把它的变化记录下来，这个是非关联搜索任务的范畴。对于强化学习问题，事情会变得复杂很多。假设有几个多臂赌博机任务，我们需要在每一个时间点来选择其中的一个任务。虽然我们仍然可以估算奖励的期望值，得到的表现未必会达到最优。在这种情况下，我们最好把一些特征和赌博机已经学习到的奖励期望值联系起来。试想一下，如果每一个机器在不同时间都有一个 LED 灯来发出不同颜色的灯光，如果当赌博机亮红灯时总是比亮蓝灯时给出更大的奖励值，那么我们就可以把这些信息和动作选择策略联系起来辅助动作选择，即可以选择那些红灯亮得更多的机器。

上下文赌博机是介于多臂赌博机和完整的强化学习两者之间的问题。它和多臂赌博机有很多类似点，比如它们的动作都只影响**立即奖励**（Immediate Reward）。上下文赌博机也和完整强化学习设置类似，因为两者都需要学习一个策略函数。如果要把一个上下文赌博机变成一个完整的强化学习任务，那么动作将不只是影响立即奖励，也会影响未来的环境状态。

2.3　马尔可夫过程

2.3.1　简介

马尔可夫过程（Markov Process，MP）是一个具备马尔可夫性质（Markov Property）的离散随机过程（Discrete Stochastic Process）。图 2.4 展示了一个马尔可夫过程的例子。每个圆圈表示一个状态，每个边（箭头）表示一个状态转移（State Transition）。这个图模拟了一个人做两种不同的任务（Tasks），以及最后去床上睡觉的这样一个例子。为了更好地理解这个图，我们假设这个人当前的状态是在做“Task1”，他有 0.7 的概率会转到做“Task2”的状态；如果他进一步从“Task2”以 0.6 的概率跳转到“Pass”状态，则这个人就完成了所有任务可以去睡觉了，因为“Pass”到“Bed”的概率是 1。

图 2.5 用**概率图模型**（Probabilistic Graphical Model）来表示马尔可夫过程，后面的章节会经常使用这种表达方式。在概率图模型中，本书统一使用圆形来表达变量，单向箭头来表达两个变量的关系。例如，“$a \to b$”表示的是变量 b 依赖于变量 a。空白圆形中的变量表示一个常规变量，而有阴影圆形的变量表示一个观测变量（Observed Variable）（这在随后的 2.7 节图片中展示），观测变量可以为其他常规变量的推理过程提供了信息。包含一些变量圆圈在内的实体黑色方框表示这些变量是重复的，同样可以在随后的图片中看到。概率图模型可以帮助我们对强化学习中变量关系有更直观的理解，以及在我们对沿着 MP 链的不同变量求导梯度时提供细致的参考。

马尔可夫过程基于**马尔可夫链**（Markov Chain）的假设，下一状态 S_{t+1} 只取决于当前状态 S_t。一个状态跳转到下一状态的概率如下：

$$p(S_{t+1}|S_t) = p(S_{t+1}|S_0, S_1, S_2, \cdots, S_t) \tag{2.8}$$

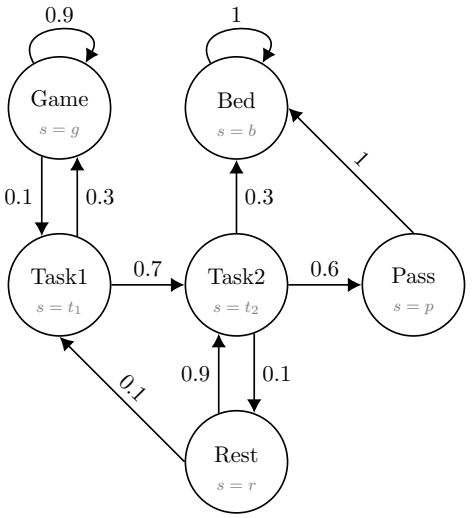


图 2.4 马尔可夫过程例子。s 表示当前状态，箭头上的数值表示从一个状态转移到另一个状态的概率

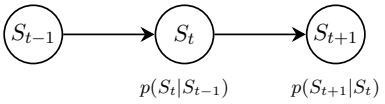


图 2.5 马尔可夫过程的概率图模型：t 表示时间步，$p(S_{t+1}|S_t)$ 表示状态转移概率

这个式子描述了“无记忆的（Memoryless）”的特性，即马尔可夫链的**马尔可夫性质**（Markov Property）。如果 $p(S_{t+2} = s'|S_{t+1} = s) = p(S_{t+1} = s'|S_t = s)$ 对任意时间步 t 和所有可能状态成立，那么它是一个沿时间轴的稳定转移函数（Stationary Transition Function），称为**时间同质性**（Time-Homogeneous），而相应的马尔可夫链为时间同质马尔可夫链（Time-Homogeneous Markov Chain）。

我们也常用 s' 来表示下一个状态，在一个时间同质马尔可夫链中，在时间 t 由状态 s 转移到时间 $t+1$ 的状态 s' 的概率满足：

$$p(s'|s) = p(S_{t+1} = s'|S_t = s) \tag{2.9}$$

时间同质性是对本书中大多数推导的一个基本假设，我们在后续绝大多数情况中默认满足这一假设而不再提及。然而，实践中，时间同质性可能不总是成立的，尤其是对非稳定的（Non-Stationary）环境、多智能体强化学习（Multi-Agent Reinforcement Learning）等，而这些时候会涉及时间不同质（Time-Inhomogeneous）的情况。

给定一个有限的**状态集**（State Set）$\mathcal{S}$，我们有一个**状态转移矩阵**（State Transition Matrix）$\boldsymbol{P}$。

比如，图 2.4 例子中对应的 $\boldsymbol{P}$ 如下所示：

$$\boldsymbol{P} = \begin{array}{c} \begin{matrix} g & t_1 & t_2 & r & p & b \end{matrix} \\ \begin{bmatrix} 0.9 & 0.1 & 0 & 0 & 0 & 0 \\ 0.3 & 0 & 0.7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.1 & 0.6 & 0.3 \\ 0 & 0.1 & 0.9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \begin{matrix} g \\ t_1 \\ t_2 \\ r \\ p \\ b \end{matrix}$$

其中 $P_{i,j}$ 是当前状态 S_i 到下一状态 S_j 的转移概率。例如，图 2.4 中状态 $s = r$ 跳转到状态 $s = t_1$ 有 0.1 的概率，跳转到状态 $s = t_2$ 的概率为 0.9。$\boldsymbol{P}$ 是一个方矩阵，每一行的和为 1。这个转移概率矩阵表示整个转移过程是随机的（Stochastic）。马尔可夫过程可以用一个元组来表示 $< \mathcal{S}, \boldsymbol{P} >$。现实中的很多简单过程可以用这样一个随机过程来近似，而这也正是强化学习方法的基础。数学上来说，下一时刻状态可以从 $\boldsymbol{P}$ 中采样，如下：

$$S_{t+1} \sim \boldsymbol{P}_{S_t,\cdot} \tag{2.10}$$

其中符号 $\sim$ 表示下一个状态 S_{t+1} 是随机地从类别分布（Categorical Distribution）$\boldsymbol{P}_{S_t,\cdot}$ 中采样得到的。

对于状态集合无限大的情况（例如说状态空间是连续的），一个有限的矩阵无法完整地表达这样状态转移的关系。因此可以使用转移函数 $p(s'|s)$，其与有限状态时的转移矩阵有对应关系，如 $p(s'|s) = \boldsymbol{P}_{s,s'}$。

2.3.2　马尔可夫奖励过程

在马尔可夫过程中，虽然智能体可以通过状态转移矩阵 $\boldsymbol{P}_{s,s'} = p(s'|s)$ 来实现与环境的交互，但是马尔可夫过程并不能让环境提供奖励反馈给智能体。为了提供反馈，**马尔可夫奖励过程**（Markov Reward Process，MRP）把马尔可夫过程从 $< \mathcal{S}, \boldsymbol{P} >$ 拓展到 $< \mathcal{S}, \boldsymbol{P}, R, \gamma >$。其中 R 和 γ 分别表示**奖励函数**（Reward Function）和**奖励折扣因子**（Reward Discount Factor）。图 2.6 是一个马尔可夫奖励过程的例子。图 2.7 是马尔可夫奖励过程的图模型，奖励函数取决于当前的状态：

$$R_t = R(S_t) \tag{2.11}$$

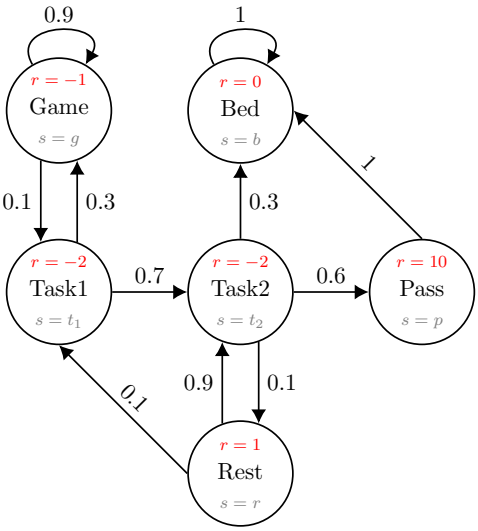


图 2.6 马尔可夫奖励过程的例子：s 表示当前的状态，r 表示每一个状态的立即奖励（Immediate Reward）。箭头边上的数值表示从一个状态到另一个状态的概率

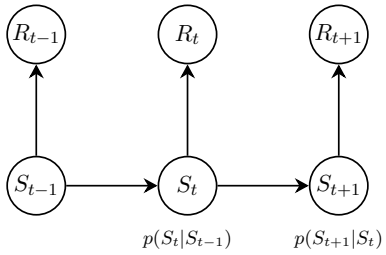


图 2.7 马尔可夫奖励过程的图模型

在这个模型中，奖励仅取决于当前状态，而当前状态是基于之前状态和之前动作产生的结果。为了更好地理解奖励是状态的函数，我们看看如下的例子。如果智能体能通过（Pass）考试，智能体可以获得的立即奖励为 10，休息（Rest）能获得立即奖励为 1，但如果智能体执行任务（Task）会损失值为 2 的立即奖励。给定一个轨迹 τ 上每个时间步的立即奖励 r，**回报**（Return）是一个轨迹的**累积奖励**（Cumulative Reward）。严格来说，**非折扣化的回报**（Undiscounted Return）在一个有 T 时间步长的有限过程中的值如下：

$$G_{t=0:T} = R(\tau) = \sum_{t=0}^{\mathrm{T}} R_t \tag{2.12}$$

其中 R_t 是 t 时刻的立即奖励，T 是最终状态的步数，或者是整个片段的步数，例如，轨迹 (g, t_1, t_2, p, b) 的非折扣化的回报是 $5 = -1 - 2 - 2 + 10$。需要注意的是，一些文献使用 G 来表示

回报，而用 R 来表示立即奖励，但在本书中，我们用 R 来表示奖励函数（Reward Function）。因此，在本书中 $R_t = R(S_t)$ 是时间步 t 时候的立即奖励，而 $R(\tau) = G_{t=0:T}$ 表示长度为 T 的轨迹 $\tau_{0:T}$ 的回报，r 是立即奖励的通用表达。

通常来说，距离更近的时间步比相对较远的时间步会产生更大的影响。这里我们介绍**折扣化回报**（Discounted Return）的概念。折扣化回报是奖励值的加权求和，它对更近的时间步给出更大的权重。定义折扣化回报如下：

$$G_{t=0:T} = R(\tau) = \sum_{t=0}^{\mathrm{T}} \gamma^t R_t. \tag{2.13}$$

其中**奖励折扣因子**（Reward Discount Factor）$\gamma \in [0, 1]$ 被用来实现随着时间步的增加而减小权重值。举例来说，图 2.6 中，当 $\gamma = 0.9$，且轨迹为 (g, t_1, t_2, p, b) 时，折扣回报为 $2.87 = -1 - 2 \times 0.9 - 2 \times 0.9^2 + 10 \times 0.9^3$。如果 $\gamma = 0$，则回报值只与当前的立即奖励有关，智能体会非常“短视”。如果 $\gamma = 1$，就是非折扣化的回报。当处理无限长 MRP 情况时，这个折扣因子会非常关键，因为它能避免回报值随着时间步增大到无穷而增大到无穷，从而使得无限长 MRP 过程是可评估的。

对折扣因子的 γ 另一个理解角度是：为了简便，奖励折扣因子 γ 有时在文献 (Levine, 2018) 中的离散时间有限范围 MRP 的情况下被省去，而这时折扣因子也可以理解为被并入了动态过程中，通过直接修改转移动态函数来使得任何产生转移至一个吸收状态（Absorbing State）的动作都有概率 $1 - \gamma$，而其他标准的转移概率都乘以 γ。

价值函数（Value Function）$V(s)$ 是状态 s 的**期望回报**（Expected Return）。举例来说，如果下一步有两个不同的状态 S_1 和 S_2，基于当前策略评估它们的价值分别为 $V^\pi(S_1)$ 和 $V^\pi(S_2)$。智能体的策略通常是选择价值更高的状态作为下一步。如果智能体的行动基于某种**策略** π，我们把相应的价值函数写为 $V^\pi(s)$：

$$V^\pi(s) = \mathbb{E}_{\tau \sim \pi}[R(\tau) | S_0 = s] \tag{2.14}$$

对于状态 s 而言，它的价值是以它为初始状态下回报的期望，而这个期望是对策略 π 给出的轨迹所求的。一种估计价值 $V(s)$ 的简单方法是**蒙特卡罗法**，给定一个状态 s，我们用状态转移矩阵 $\boldsymbol{P}$ 随机采样大量的轨迹，来求近似期望。以图 2.6 为例，给定 $\gamma = 0.9$ 和 $\boldsymbol{P}$，如何估计 $V^\pi(s = t_2)$？我们可以如下随机采样出 4 个轨迹（注意，实际中采样的轨迹要远大于 4，但这里为了描述方法，我们只采样 4 个轨迹）：

- $s = (t_2, b), R = -2 + 0 \times 0.9 = -2$
- $s = (t_2, p, b), R = -2 + 10 \times 0.9 + 0 \times 0.9^2 = 7$
- $s = (t_2, r, t_2, p, b), R = -2 + 1 \times 0.9 - 2 \times 0.9^2 + 10 \times 0.9^3 + 0 \times 0.9^4 = 4.57$
- $s = (t_2, r, t_1, t_2, b), R = -2 + 1 \times 0.9 - 2 \times 0.9^2 - 2 \times 0.9^3 + 0 \times 0.9^4 = -0.178$

给定这些 $s = t_2$ 为初始状态的轨迹，我们可以计算每个轨迹的回报 R，然后估计出状态 $s = t_2$ 的期望回报 $V(s = t_2) = (-2 + 7 + 4.57 - 0.178)/4 = 2.348$，作为状态 $s = t_2$ 的价值衡量。

图 2.8 用这个方法估算出每个状态的期望回报。给定这些期望回报，一个最简单的智能体策略是每一步都往期望回报更高的状态移动。这样所产生的动作就是最大化期望回报，见图 2.8 的虚线箭头。除了蒙特卡罗方法，还有很多方法可以用来计算 $V(s)$，比如贝尔曼期望方程（Bellman Expectation Equation）、逆矩阵方法（Inverse Matrix Method）等，我们将会在稍后逐一介绍。

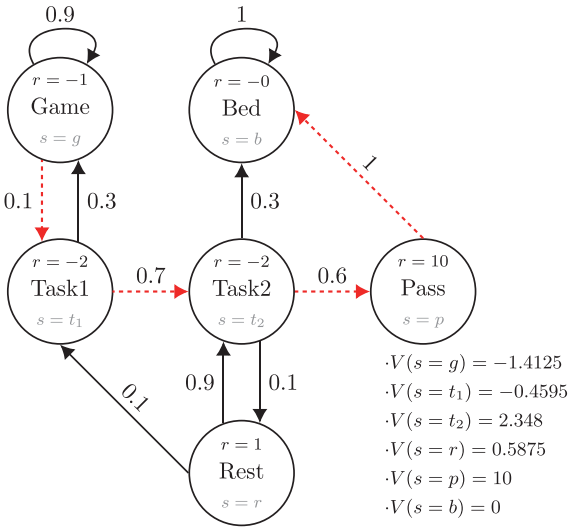


图 2.8 马尔可夫奖励过程和价值估计函数 $V(s)$：每个状态都随机采样 4 个轨迹，用蒙特卡罗方法估算每个状态的价值。虚线箭头表示学习出的简单策略，则智能体往价值更高的状态移动

2.3.3 马尔可夫决策过程

马尔可夫决策过程（Markov Decision Process，MDP）从 20 世纪 50 年代已经开始被广泛地研究，在包括经济、控制理论和机器人等很多领域都有应用。在模拟序列决策过程的问题上，马尔可夫决策过程比马尔可夫过程和马尔可夫奖励过程要好用。如图 2.9 所示，和马尔可夫奖励过程不同的地方在于，马尔可夫奖励过程的立即奖励只取决于状态（奖励值在节点上），而马尔可夫决策过程的立即奖励与状态和动作都有关（奖励值在边上）。同样地，给定一个状态下的一个动作，马尔可夫决策过程的下一个状态不一定是固定唯一的。举例来说，如图 2.10 所示，当智能体在状态 $s = t_2$ 时执行休息（rest）动作后，下一时刻的状态有 0.8 的概率保留在状态 $s = t_2$ 下，有 0.2 的概率变为 $s = t_1$。

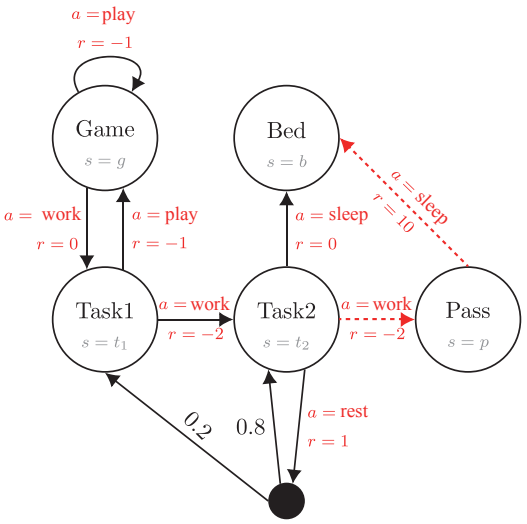


图 2.9　马尔可夫决策过程例子。在马尔可夫奖励过程中，立即奖励只与状态有关。而马尔可夫决策过程的立即奖励与状态和动作都有关

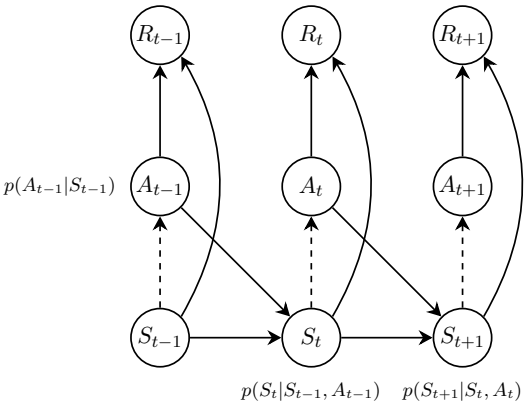


图 2.10　马尔可夫决策过程的图模型：t 表示时间步，$p(A_t|S_t)$ 表示根据当前状态 S_t 选择的动作 A_t 的概率，$p(S_{t+1}|S_t, A_t)$ 是基于当前状态和动作下的状态转移概率。虚线表示智能体的决策过程

之前说过，马尔可夫过程可以看成一个元组 <$\mathcal{S}, \boldsymbol{P}$>，而马尔可夫奖励过程是 <$\mathcal{S}, \boldsymbol{P}, R, \gamma$>，其中状态转移矩阵的元素（Element）值是 $\boldsymbol{P}_{s,s'} = p(s'|s)$。这个表示将有限维（Finite-Dimension）状态转移矩阵拓展成无穷维（Infinite-Dimension）概率函数。这里，马尔可夫决策过程是 <$\mathcal{S}, \mathcal{A}, \boldsymbol{P}, R, \gamma$>，其状态转移矩阵的元素变为

$$p(s'|s, a) = p(S_{t+1} = s'|S_t = s, A_t = a) \tag{2.15}$$

例如图 2.9 中很多状态转移概率为 1，比如 $p(s'=t_2|s=t_1,a=\text{work})=1$；但是也有一些不是，比如 $p(s'|s=t_2,a=\text{rest})=[0.2,0.8]$，它表示的是，如果智能体在状态 $s=t_2$ 下执行动作 $a=\text{rest}$，它有 0.2 的概率会跳到状态 $s'=t_1$，而有 0.8 的概率会保持原来的状态。那些不存在的边代表转移概率为 0，比如 $p(s'=t_2|s=t_1,a=\text{rest})=0$。

$\mathcal{A}$ 表示**有限的动作集合**（Finite Action Set）$\{a_1,a_2,\cdots\}$，则立即奖励变成

$$R_t=R(S_t,A_t) \tag{2.16}$$

一个**策略**（Policy）表示智能体根据它对环境的观测来行动的方式。具体来说，策略是从每一个状态 $s\in\mathcal{S}$ 和动作 $a\in\mathcal{A}$ 到动作概率分布 $\pi(a|s)$ 的映射，这个概率分布是在状态 s 下采取动作 a 的概率，可以写为

$$\pi(a|s)=p(A_t=a|S_t=s),\exists t \tag{2.17}$$

期望回报（Expected Return）是在一个策略下给定所有可能轨迹的回报的期望值，**强化学习的目的就是通过优化策略来使得期望回报最大化**。数学上来说，给定起始状态分布 ρ_0 和策略 π，马尔可夫决策过程中一个 T 步长的轨迹的发生概率是：

$$p(\tau|\pi)=\rho_0(S_0)\prod_{t=0}^{T-1}p(S_{t+1}|S_t,A_t)\pi(A_t|S_t) \tag{2.18}$$

给定奖励函数 R 和所有可能的轨迹 τ，**期望回报** $J(\pi)$ 可以定义为

$$J(\pi)=\int_{\tau}p(\tau|\pi)R(\tau)=\mathbb{E}_{\tau\sim\pi}[R(\tau)] \tag{2.19}$$

其中 p 表示轨迹发生的概率，发生概率越高，则对期望回报计算的权重越大。**强化学习优化问题**（RL Optimization Problem）通过优化方法来提升策略，从而最大化期望回报。**最优策略**（Optimal Policy）π^* 可以表示为

$$\pi^*=\arg\max_{\pi}J(\pi) \tag{2.20}$$

其中 $*$ 符号在本书中表示“最优的”含义。

给定一个策略 π，价值函数 $V(s)$，即给定状态下的期望回报，可以定义为

$$V^{\pi}(s)=\mathbb{E}_{\tau\sim\pi}[R(\tau)|S_0=s]$$

$$=\mathbb{E}_{A_t\sim\pi(\cdot|S_t)}\left[\sum_{t=0}^{\infty}\gamma^t R(S_t,A_t)|S_0=s\right] \tag{2.21}$$

其中 $\tau\sim\pi$ 表示轨迹 τ 是通过策略 π 采样获得的，$A_t\sim\pi(\cdot|S_t)$ 表示动作是在一个状态下从策略中采样得到的（如果策略是有随机性的），下一个状态取决于状态转移矩阵 $\boldsymbol{P}$ 及其状态 s 和动作 a。

在马尔可夫决策过程中，给定一个动作，就有**动作价值函数**（Action-Value Function），这个函数依赖于状态和刚刚执行的动作，是基于状态和动作的期望回报。如果一个智能体根据策略 π 来运行，则把动作价值函数写为 $Q^\pi(s,a)$，其定义为

$$\begin{aligned}Q^\pi(s,a)=&\mathbb{E}_{\tau\sim\pi}[R(\tau)|S_0=s,A_0=a]\\=&\mathbb{E}_{A_t\sim\pi(\cdot|S_t)}\left[\sum_{t=0}^{\infty}\gamma^t R(S_t,A_t)|S_0=s,A_0=a\right]\end{aligned} \tag{2.22}$$

我们需要记住的是，$Q^\pi(s,a)$ 是基于策略 π 来估计的，因为对值的估计是策略 π 所决定的轨迹上的期望。也就是说，如果策略 π 改变了，$Q^\pi(s,a)$ 也会相应地跟着改变。因此我们通常称基于一个特定策略估计的价值函数为**在线价值函数**（On-Policy Value Function），来与用最优策略估计的**最优价值函数**（Optimal Value Function）进行区分。

我们可以发现价值函数 $v_\pi(s)$ 和动作价值函数 $q_\pi(s,a)$ 之间有如下关系：

$$q_\pi(s,a)=\mathbb{E}_{\tau\sim\pi}[R(\tau)|S_0=s,A_0=a] \tag{2.23}$$

$$v_\pi(s)=\mathbb{E}_{a\sim\pi}[q_\pi(s,a)] \tag{2.24}$$

有两种简单方法来计算价值函数 $v_\pi(s)$ 和动作价值函数 $q_\pi(s,a)$：第一种方法是**穷举法**（exhaustive method），如公式 (2.18) 所示，首先计算出从一个状态开始的所有可能轨迹的概率，然后用公式 (2.21) 和 (2.22) 来计算出这个状态的 $V^\pi(s)$ 和 $Q^\pi(s,a)$。每个状态都用穷举法来单独计算。然而实际中，可能的轨迹数量是非常大的，甚至是无穷个的。因此除了使用所有可能的轨迹，第二种方法是使用之前介绍的蒙特卡罗方法通过采样大量的轨迹来估计 $V^\pi(s)$。这两种方法都非常简单，但都有各自的缺点。而实际上，估计价值函数的公式可以根据马尔可夫性质做进一步的简化，即下一小节要介绍的贝尔曼方程。

2.3.4 贝尔曼方程和最优性

贝尔曼方程

贝尔曼方程（Bellman Equation），也称为贝尔曼期望方程，用于计算给定策略 π 时价值函数在策略指引下所采轨迹上的期望。我们称之为“在线（On-Policy）”估计方法（注意它与之后的在线策略和离线策略更新区分），因为强化学习中的策略一直是变化的，而价值函数（Value Function）是以当前策略为条件或者用其估计的。

回想状态价值函数或动作价值函数（Action-Value Function）的定义，即 $v_\pi(s)=\mathbb{E}_{\tau\sim\pi}[R(\tau)|S_0=s]$ 和 $q_\pi(s,a)=\mathbb{E}_{\tau\sim\pi}[R(\tau)|S_0=s,A_0=a]$，我们可以利用递归关系得出**在线状态价值函数**（On-Policy State-Value Function）的贝尔曼方程：

$$\begin{aligned}
v_\pi(s)&=\mathbb{E}_{a\sim\pi(\cdot|s),s'\sim p(\cdot|s,a)}[R(\tau_{t:T})|S_t=s]\\
&=\mathbb{E}_{a\sim\pi(\cdot|s),s'\sim p(\cdot|s,a)}[R_t+\gamma R_{t+1}+\gamma^2R_{t+2}+\cdots+\gamma^{\mathrm{T}}R_T|S_t=s]\\
&=\mathbb{E}_{a\sim\pi(\cdot|s),s'\sim p(\cdot|s,a)}[R_t+\gamma(R_{t+1}+\gamma R_{t+2}+\cdots+\gamma^{T-1}R_T)|S_t=s]\\
&=\mathbb{E}_{a\sim\pi(\cdot|s),s'\sim p(\cdot|s,a)}[R_t+\gamma R_{\tau_{t+1:T}}|S_t=s]\\
&=\mathbb{E}_{A_t\sim\pi(\cdot|S_t),S_{t+1}\sim p(\cdot|S_t,A_t)}[R_t+\gamma\mathbb{E}_{a\sim\pi(\cdot|s),s'\sim p(\cdot|s,a)}[R_{\tau_{t+1:T}}]|S_t=s]\\
&=\mathbb{E}_{A_t\sim\pi(\cdot|S_t),S_{t+1}\sim p(\cdot|S_t,A_t)}[R_t+\gamma v_\pi(S_{t+1})|S_t=s]\\
&=\mathbb{E}_{a\sim\pi(\cdot|s),s'\sim p(\cdot|s,a)}[r+\gamma v_\pi(s')]
\end{aligned}\tag{2.25}$$

上式最后一个等式成立，是因为 s,a 是对状态和动作的一般表示，而 S_t,A_t 是状态和动作只在时间步 t 上的表示。在上面的一些公式中，S_t,A_t 有时从一般表示 s,a 分离出来，从而更清楚地展示期望是关于哪些变量求得的。

注意上面的推导过程中，我们展示了基于 MDP 的贝尔曼方程，然而，对 MRP 的贝尔曼方程可以直接通过从中去掉动作来得到：

$$v(s)=\mathbb{E}_{s'\sim p(\cdot|s)}[r+\gamma v(s')]\tag{2.26}$$

除上述外，也有基于**在线动作价值函数**（On-Policy Action-Value Function）的贝尔曼方程：$q_\pi(s,a)=\mathbb{E}_{s'\sim p(\cdot|s,a)}[R(s,a)+\gamma\mathbb{E}_{a'\sim\pi(\cdot|s')}[q_\pi(s',a')]]$，可以通过如下推导得到：

$$\begin{aligned}
q_\pi(s,a)&=\mathbb{E}_{a\sim\pi(\cdot|s),s'\sim p(\cdot|s,a)}[R(\tau_{t:T})|S_t=s,A_t=a]\\
&=\mathbb{E}_{a\sim\pi(\cdot|s),s'\sim p(\cdot|s,a)}[R_t+\gamma R_{t+1}+\gamma^2R_{t+2}+\cdots+\gamma^{\mathrm{T}}R_T|S_t=s,A_t=a]\\
&=\mathbb{E}_{a\sim\pi(\cdot|s),s'\sim p(\cdot|s,a)}[R_t+\gamma(R_{t+1}+\gamma R_{t+2}+\cdots+\gamma^{T-1}R_T)|S_t=s,A_t=a]\\
&=\mathbb{E}_{S_{t+1}\sim p(\cdot|S_t,A_t)}[R_t+\gamma\mathbb{E}_{a\sim\pi(\cdot|s),s'\sim p(\cdot|s,a)}[R_{\tau_{t+1:T}}]|S_t=s]
\end{aligned}$$

$$
\begin{aligned}
&= \mathbb{E}_{S_{t+1}\sim p(\cdot|S_t,A_t)}[R_t + \gamma \mathbb{E}_{A_{t+1}\sim\pi(\cdot|S_{t+1})}[q_\pi(S_{t+1}, A_{t+1})]|S_t = s] \\
&= \mathbb{E}_{s'\sim p(\cdot|s,a)}[R(s,a) + \gamma \mathbb{E}_{a'\sim\pi(\cdot|s')}[q_\pi(s', a')]]
\end{aligned} \tag{2.27}
$$

上面的推导是基于最大长度为 T 的有限 MDP，然而，这些等式对无穷长度 MDP 也成立，只要将 T 用“∞”替代即可。同时，这两个贝尔曼方程也不依赖于策略的具体形式，这意味着它们对随机性策略 $\pi(\cdot|s)$ 和确定性策略 $\pi(s)$ 都有效。这里 $\pi(\cdot|s)$ 的使用是为了简化。而且，在确定性转移过程中，我们有 $p(s'|s,a) = 1$。

贝尔曼方程求解

如果转移函数或转移矩阵是已知的，公式 (2.26) 中对 MRP 的贝尔曼方程可以直接求解，称为**逆矩阵方法**（Inverse Matrix Method）。我们用矢量形式对离散有限状态空间的情况将公式 (2.26) 改写为

$$
\boldsymbol{v} = \boldsymbol{r} + \gamma \boldsymbol{P}\boldsymbol{v} \tag{2.28}
$$

其中 $\boldsymbol{v}$ 和 $\boldsymbol{r}$ 矢量，其单元 $v(s)$ 和 $R(s)$ 是对所有 $s \in \mathcal{S}$ 的，而 $\boldsymbol{P}$ 是转移概率矩阵，其元素 $p(s'|s)$ 对所有 $s, s' \in \mathcal{S}$ 成立。

由 $\boldsymbol{v} = \boldsymbol{r} + \gamma \boldsymbol{P}\boldsymbol{v}$，我们可以直接对它求解：

$$
\boldsymbol{v} = (\boldsymbol{I} - \gamma \boldsymbol{P})^{-1}\boldsymbol{r} \tag{2.29}
$$

求解的复杂度是 $O(n^3)$，其中 n 是状态的数量。因此这种方法对有大量状态的情况难以求解，这意味着它可能对大规模或连续值问题不适用。幸运的是，有一些迭代方法可以在实践中解决大规模的 MRP 问题，比如动态规划（Dynamic Programming）、蒙特卡罗估计（Monte-Carlo Estimation）和时间差分（Temporal Difference）学习法，这些方法将在随后的小节中详细介绍。

最优价值函数

由于在线价值函数是根据策略本身来估计的，即使是在相同的状态和动作集合上，不同的策略也将会带来不同的价值函数。对于所有不同的价值函数，我们定义最优价值函数为

$$
v_*(s) = \max_\pi v_\pi(s), \forall s \in \mathcal{S}, \tag{2.30}
$$

这实际是**最优状态价值函数**（Optimal State-Value Function）。我们也有**最优动作价值函数**（Optimal Action-Value Function）：

$$q_*(s,a) = \max_{\pi} q_\pi(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}, \tag{2.31}$$

它们之间的关系为

$$q_*(s,a) = \mathbb{E}[R_t + \gamma v_*(S_{t+1})|S_t = s, A_t = a], \tag{2.32}$$

上式可以直接通过对式 (2.84) 的最后一个等式最大化并代入式 (2.24) 和 (2.30) 来得到：

$$\begin{aligned} q_*(s,a) &= \mathbb{E}[R(s,a) + \gamma \max_{\pi} \mathbb{E}[q_\pi(s',a')]] \\ &= \mathbb{E}[R(s,a) + \gamma \max_{\pi} v_\pi(s')] \\ &= \mathbb{E}[R_t + \gamma v_*(S_{t+1})|S_t = s, A_t = a]. \end{aligned} \tag{2.33}$$

它们之间的另一种关系为

$$v_*(s) = \max_{a \sim \mathcal{A}} q_*(s,a) \tag{2.34}$$

这可以直接通过最大化式 (2.24) 的两边来得到。

贝尔曼最优方程

在上面小节中，我们介绍了一般在线价值函数的贝尔曼方程，以及最优价值函数的定义。因此我们可以在预定义的最优价值函数上使用贝尔曼方程，这会得到**贝尔曼最优方程**（Bellman Optimality Equation），或称对最优价值函数的贝尔曼方程（Bellman Equation for Optimal Value Functions），推导如下。

对最优状态价值函数的贝尔曼方程为

$$v_*(s) = \max_{a} \mathbb{E}_{s' \sim p(\cdot|s,a)}[R(s,a) + \gamma v_*(s')], \tag{2.35}$$

它可以通过下面推导来得到：

$$\begin{aligned} v_*(s) &= \max_{a} \mathbb{E}_{\pi^*, s' \sim p(\cdot|s,a)}[R(\tau_{t:T})|S_t = s] \\ &= \max_{a} \mathbb{E}_{\pi^*, s' \sim p(\cdot|s,a)}[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots + \gamma^{\mathrm{T}} R_T|S_t = s] \\ &= \max_{a} \mathbb{E}_{\pi^*, s' \sim p(\cdot|s,a)}[R_t + \gamma R_{\tau_{t+1:T}}|S_t = s] \\ &= \max_{a} \mathbb{E}_{s' \sim p(\cdot|s,a)}[R_t + \gamma \max_{a'} \mathbb{E}_{\pi^*, s' \sim p(\cdot|s,a)}[R_{\tau_{t+1:T}}]|S_t = s] \\ &= \max_{a} \mathbb{E}_{s' \sim p(\cdot|s,a)}[R_t + \gamma v_*(S_{t+1})|S_t = s] \end{aligned}$$

$$= \max_a \mathbb{E}_{s' \sim p(\cdot|s,a)}[R(s,a) + \gamma v_*(s')] \tag{2.36}$$

最优动作价值函数的贝尔曼方程为

$$q_*(s,a) = \mathbb{E}_{s' \sim p(\cdot|s,a)}[R(s,a) + \gamma \max_{a'} q_*(s',a')], \tag{2.37}$$

上式可以通过与前面类似的方式得到。读者可以练习完成这个证明。

2.3.5　其他重要概念

确定性和随机性策略

在之前的小节中，策略用概率分布 $\pi(a|s) = p(A_t = a|S_t = s)$ 表示，其中智能体的动作是从分布中采样得到的。一个动作从概率分布中采样的策略称为**随机性策略分布**（Stochastic Policy Distribution），其动作为

$$a \sim \pi(\cdot|s) \tag{2.38}$$

然而，如果我们减少随机性策略分布的方差并将其范围缩窄到极限情况，则将得到一个狄拉克函数（δ 函数）作为其分布，即为一个**确定性策略**（Deterministic Policy）$\pi(s)$。确定性策略 $\pi(s)$ 也意味着给定一个状态，将得到唯一的动作，如下：

$$a = \pi(s) \tag{2.39}$$

注意确定性策略不再是从状态和动作到条件概率分布（Conditional Probability Distribution）的映射，而是一个从状态到动作的直接映射。这点不同将导致随后介绍的策略梯度方法中的一些推导过程的不同。更多关于强化学习中策略类别的细节，尤其是深度强化学习中的参数化策略，将在 2.7.3 节中介绍。

部分可观测马尔可夫决策过程

如前面小节中所述，当强化学习环境中的状态无法由智能体的观测量完全表示的时候，环境是部分可观测的。对于一个马尔可夫决策过程，它被称为部分可观测的马尔可夫决策过程（Partially Observed Markov Decision Process，POMDP），而这构成了一个利用不完整环境状态信息来改进策略的挑战。

2.4　动态规划

20 世纪 50 年代，Richard E. Bellman 首次提出**动态规划**（Dynamic Programming）的概念。随后，动态规划算法被成功地应用到一系列有挑战的场景中。在“动态规划”一词中，“动态”指求

解的问题是序列化的，“规划”指优化策略。动态规划将复杂的动态问题拆解为子问题，提供了一种通用的求解框架。例如，在斐波那契数列中的每一个数字由两个先前的数字相加得到，从 0 和 1 开始。如第 4 个数 F_4 可以写为前两个数 F_3、F_2 之和 $F_4 = F_3 + F_2$。在这个算式中，我们可以进一步将 F_3 拆解为 $F_3 = F_2 + F_1$，从而得到 $F_4 = (F_2 + F_1) + F_2$，于是我们用朴素的子问题 F_1 和 F_2 表示了 F_4。动态规划需要知道求解问题的全部信息，例如，强化学习问题中的奖励机制和状态转移方程，但是在强化学习的场景中，这些信息是很难被获取的。尽管如此，动态规划依旧提供了一种通过在马尔可夫过程中进行交互来学习的基本思路，被大多数强化学习算法所沿用。

可以应用动态规划的问题必须具备两个性质：**最优子结构**（Optimal Substructure）和**重叠子问题**（Overlapping Sub-Problems）。最优子结构是指一个给定问题的最优解可以分解成它的子问题的解。重叠子问题是指子问题的数量是有限的，以及子问题递归地出现，使其可以被存储和重用。有限动作和状态空间的 MDP 满足以上两个性质，贝尔曼方程实现了递归式的分解，价值函数存储了子问题的最优解。因此在本小节中，我们假设状态集和动作集都是有限的，并且有一个环境的理想化模型。

2.4.1 策略迭代

策略迭代（Policy Iteration）的目的在于直接操控策略。从任意策略 π 开始，我们可以通过递归地调用贝尔曼方程来评估策略：

$$v_\pi(s) = \mathbb{E}_\pi[R_t + \gamma v_\pi(S_{t+1}) \,|S_t = s] \tag{2.40}$$

这里的期望是针对基于环境全部知识的所有可能的转移。一个获得更好策略的自然想法是根据 v_π 来贪心地执行动作：

$$\pi'(s) = \text{greedy}(v_\pi) = \arg\max_{a \in \mathcal{A}} q_\pi(s, a). \tag{2.41}$$

这样的提升可以由以下证明：

$$\begin{aligned}
v_\pi(s) &= q_\pi(s, \pi(s)) \\
&\leqslant q_\pi(s, \pi'(s)) \\
&= \mathbb{E}_{\pi'}[R_t + \gamma v_\pi(S_{t+1}) \,|S_t = s] \\
&\leqslant \mathbb{E}_{\pi'}[R_t + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \,|S_t = s] \\
&\leqslant \mathbb{E}_{\pi'}[R_t + \gamma R_{t+1} + \gamma^2 q_\pi(S_{t+2}, \pi'(S_{t+2})) \,|S_t = s] \\
&\leqslant \mathbb{E}_{\pi'}[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots |S_t = s] = v_{\pi'}(s).
\end{aligned} \tag{2.42}$$

接连地使用以上的策略评估和贪心提升，直到 $\pi = \pi'$ 形成策略迭代。一般地，策略迭代的过程可以总结如下：给定任意一个策略 π_t，对于每一次迭代 t 中的每一个状态 s，我们首先评估 $v_{\pi_t}(s)$，然后找到一个更好的策略 π_{t+1}。我们把前一个阶段称为**策略评估**（Policy Evaluation），把后一个阶段称为**策略提升**（Policy Improvement）。此外，我们使用术语**泛化策略迭代**（Generalized Policy Iteration，GPI）来指代一般的策略评估和策略提升交互过程，如图 2.11 所示。

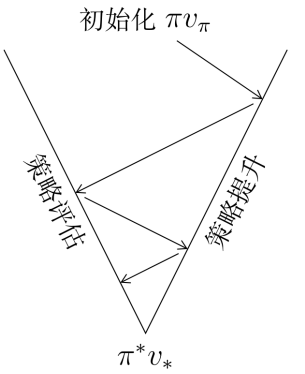


图 2.11　泛化策略迭代

一个基本的问题是，策略迭代的过程是否在最优值 v_* 上收敛。在策略评估的每一次迭代中，对于固定的、确定性的策略 π，价值函数更新可以被**贝尔曼期望回溯算子** $\mathcal{T}^\pi$ 重写为

$$(\mathcal{T}^\pi V)(s) = (\mathcal{R}^\pi + \gamma \mathcal{P}^\pi V)(s) = \sum_{r,s'} (r + \gamma V(s')) P(r, s'|s, \pi(s)). \tag{2.43}$$

那么对于任意的价值函数 V 和 V'，我们对于 $\mathcal{T}^\pi$ 有如下的收缩（Contraction）证明：

$$\begin{aligned}
|\mathcal{T}^\pi V(s) - \mathcal{T}^\pi V'(s)| &= \left| \sum_{r,s'} (r + \gamma V(s')) P(r, s'|s, \pi(s)) - \sum_{r,s'} (r + \gamma V'(s')) P(r, s'|s, \pi(s)) \right| \\
&= \left| \sum_{r,s'} \gamma (V(s') - V'(s')) P(r, s'|s, \pi(s)) \right| \\
&\leqslant \sum_{r,s'} \gamma |V(s') - V'(s')| P(r, s'|s, \pi(s)) \\
&\leqslant \sum_{r,s'} \gamma \|V - V'\|_\infty P(r, s'|s, \pi(s)) \\
&= \gamma \|V - V'\|_\infty,
\end{aligned} \tag{2.44}$$

此处 $\|V - V'\|_\infty$ 是 ∞ 范数。通过收缩映射定理（Contraction Mapping Theorem，即巴拿赫不动点

定理，Banach Fixed-Point Theorem），迭代策略评估会收敛到唯一的固定点 $\mathcal{T}^{\pi}$。由于 $\mathcal{T}^{\pi}v_{\pi} = v_{\pi}$ 是固定点，迭代策略评估会收敛到 v_{π}。需要指出的是，策略提升是单调的，并且在有限 MDP 中的价值函数只对应于有限个数的贪心策略。策略提升会在有限步数后停止，也就是说，策略迭代会收敛到 v_*。

算法 2.6 策略迭代

对于所有的状态初始化 V 和 π
repeat
  //执行策略评估
  repeat
    $\delta \leftarrow 0$
    for $s \in \mathcal{S}$ **do**
      $v \leftarrow V(s)$
      $V(s) \leftarrow \sum_{r,s'}(r + \gamma V(s'))P(r, s'|s, \pi(s))$
      $\delta \leftarrow \max(\delta, |v - V(s)|)$
    end for
  until δ 小于一个正阈值
  //执行策略提升
  stable $\leftarrow$ true
  for $s \in \mathcal{S}$ **do**
    $a \leftarrow \pi(s)$
    $\pi(s) \leftarrow \arg\max_a \sum_{r,s'}(r + \gamma V(s'))P(r, s'|s, a)$
    if $a \neq \pi(s)$ **then**
      stable $\leftarrow$ false
    end if
  end for
until stable = true
return 策略 π

2.4.2 价值迭代

价值迭代（Value Iteration）的理论基础是**最优性原则**（Principle of Optimality）。这个原则告诉我们当且仅当 π 取得了可以到达的任何后续状态上的最优价值时，π 是一个状态上的最优策略。因此如果我们知道子问题 $v_*(s')$ 的解，就可以通过一步完全回溯（One-Step Full Backup）找到任意一个初始状态 s 的解：

$$v_*(s) = \max_{a \in \mathcal{A}} R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a)v_*(s'). \tag{2.45}$$

价值迭代的过程是将上面的更新过程从最终状态开始，一个一个状态接连向前进行。和策略迭代中的收敛证明类似，**贝尔曼最优算子** $\mathcal{T}^*$ 为

$$(\mathcal{T}^*V)(s) = (\max_{a\in\mathcal{A}} \mathcal{R}^a + \gamma\mathcal{P}^a V)(s) = \max_{a\in\mathcal{A}} R(s,a) + \gamma \sum_{s'\in\mathcal{S}} P(s'|s,a)V(s') \tag{2.46}$$

这也是对于任意价值函数 V 和 V' 的收缩映射：

$$\begin{aligned}
&|\mathcal{T}^*V(s) - \mathcal{T}^*V'(s)| \\
&= \left| \max_{a\in\mathcal{A}} \left[R(s,a) + \gamma \sum_{s'\in\mathcal{S}} P(s'|s,a)V(s') \right] - \max_{a\in\mathcal{A}} \left[R(s,a) + \gamma \sum_{s'\in\mathcal{S}} P(s'|s,a)V'(s') \right] \right| \\
&\leqslant \max_{a\in\mathcal{A}} \left| R(s,a) + \gamma \sum_{s'\in\mathcal{S}} P(s'|s,a)V(s') - R(s,a) - \gamma \sum_{s'\in\mathcal{S}} P(s'|s,a)V'(s') \right| \\
&= \max_{a\in\mathcal{A}} \left| \gamma \sum_{s'\in\mathcal{S}} P(s'|s,a)(V(s') - V'(s')) \right| \\
&\leqslant \max_{a\in\mathcal{A}} \gamma \sum_{s'\in\mathcal{S}} P(s'|s,a)|V(s') - V'(s')| \\
&\leqslant \max_{a\in\mathcal{A}} \gamma \sum_{s'\in\mathcal{S}} P(s'|s,a)\|V - V'\|_\infty \\
&= \gamma\|V - V'\|_\infty \max_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} P(s'|s,a) \\
&= \gamma\|V - V'\|_\infty.
\end{aligned} \tag{2.47}$$

由于 v_* 是 $\mathcal{T}^*$ 的一个固定点，价值迭代会收敛到最优值 v_*。需要指出的是，在价值迭代中，只有后续状态的实际价值是已知的。换句话说，价值是不完整的，因此，我们在以上的证明中使用估计价值函数 V，而不是真实价值 v。

何时停止价值迭代算法不是显而易见的。文献 (Williams et al., 1993) 在理论上给出了一个充分的（Sufficient）停止标准：如果两个连续价值函数的最大差异小于 ϵ，那么在任意状态下，贪心策略的价值与最优策略的价值函数的差值不会超过 $\frac{2\epsilon\gamma}{1-\gamma}$。

2.4.3　其他 DPs：异步 DP、近似 DP 和实时 DP

目前描述的 DP 方法均使用同步回溯（Synchronous Backup），即每个状态的价值基于系统性的扫描（Systematic Sweeps）来回溯。一种有效的变体是异步的更新（Asynchronous Update），而这也是速度和准确率之间的权衡。异步 DP 对于强化学习的设定也是适用的，且如果所有状态被持续选择的话，可以保证收敛。异步 DP 有三种简单的思路:

算法 2.7 价值迭代

为所有状态初始化 V
repeat
  $\delta \leftarrow 0$
  for $s \in \mathcal{S}$ **do**
    $u \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{r,s'} P(r, s'|s, a)(r + \gamma V(s'))$
    $\delta \leftarrow \max(\delta, |u - V(s)|)$
  end for
until δ 小于一个正阈值
输出贪心策略 $\pi(s) = \arg\max_a \sum_{r,s'} P(r, s'|s, a)(r + \gamma V(s'))$

1. 在位更新（In-Place Update）
   同步价值迭代（Synchronous Value Iteration）存储价值函数 $V_{t+1}(\cdot)$ 和 $V_t(\cdot)$ 的两个备份：

$$V_{t+1}(s) \leftarrow \max_{a \in \mathcal{A}} R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V_t(s'). \tag{2.48}$$

   在位价值迭代只存储价值函数的一个备份：

$$V(s) \leftarrow \max_{a \in \mathcal{A}} R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V(s'). \tag{2.49}$$

2. 优先扫描（Prioritized Sweeping）
   在异步 DP 中，另一个需要考虑的事情是更新顺序。给定一个转移 (s, a, s')，优先扫描将它的贝尔曼误差（Bellman Error）的绝对值作为它的大小：

$$\left| V(s) - \max_{a \in \mathcal{A}} (R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V(s')) \right|. \tag{2.50}$$

   它可以通过保持一个优先权队列来有效地实现，该优先权队列在每个回溯后存储和更新每个状态的贝尔曼误差。

3. 实时更新（Real-Time Update）
   在每个时间步 t 之后，不论采用哪个动作，实时更新将只会通过以下方式回溯当前状态 S_t：

$$V(S_t) \leftarrow \max_{a \in \mathcal{A}} R(S_t, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|S_t, a) V(s'). \tag{2.51}$$

   它可以被视为根据智能体的经验来指导选择要更新的状态。

同步 DP 和异步 DP 都在全部状态集上回溯，估计下一个状态的预期回报。从概率的角度来

看，一个有偏差的但有效的选择是使用采样的数据。我们将在下一个小节中深入讨论此问题。

2.5　蒙特卡罗

和动态规划不同的是，蒙特卡罗 (Monte Carlo, MC) 方法不需要知道环境的所有信息。蒙特卡罗方法只需基于过去的经验就可以学习。它也是一种基于样本的（Sampling-Based）方法。蒙特卡罗可以在对环境只有很少的先验知识时从经验中学习来取得很好的效果。“蒙特卡罗”可以用来泛指那些有很大随机性的算法。

当我们在强化学习中使用蒙特卡罗方法的时候，需要对来自不同片段中的每个状态-动作对（State-Action Pair）相应的奖励值取平均。一个例子是，在本章之前内容中介绍的上下文赌博机（Contextual Bandit）问题中，如果在不同的机器上有一个 LED 灯，那么玩家就可以逐渐地学习 LED 灯状态信息和回报之间的联系。我们在这里把一种灯的排列组合作为一种状态，那么其可能的奖励值就作为这个状态的价值。最开始，我们可能无法对状态价值有一个很好的预估，但是当我们做出更多的尝试以后，平均状态价值会向它们的真实值靠近。在这个章节，我们会探索我们怎么更合理地做出估算。假设问题是回合制的（Episodic），因而不论一个玩家做出了哪些的动作，一个回合最后都会终止。

2.5.1　蒙特卡罗预测

首先，我们一起来看给定一个策略 π 如何用蒙特卡罗方法来评估状态价值函数。直观上的一种方式是，通过对具体策略产生的回报取平均值来从经验中评估状态价值函数。更具体地，让函数 $v_\pi(s)$ 作为在策略 π 下的状态价值函数。我们接着收集一组经过状态 s 的回合，并把每一次状态 s 在一个回合里的出现叫作一次对状态 s 的访问。这样一来，我们就有两种估算方式：**首次蒙特卡罗**（First-Visit Monte Carlo）和 **每次蒙特卡罗**（Every-Visit Monte Carlo）。首次蒙特卡罗只考虑每一个回合中第一次到状态 s 的访问，而每次蒙特卡罗就是考虑每次到状态 s 的访问。这两种方式有很多的相似点，但是也有一些理论上的不同。在算法 2.8，我们展示了如何用首次蒙特卡罗来对 $v_\pi(s)$ 估算。把首次蒙特卡罗变成每次蒙特卡罗在操作上很简单，我们只需要把对首次访问检查条件去掉即可。假如我们对状态 s 有无限次访问的话，那最终这两种方式都会收敛到 $v_\pi(s)$。

蒙特卡罗方法可以独立地对不同的状态值进行估算。和动态规划不同的是，蒙特卡罗不使用自举（Bootstrapping），也就是说，它不用其他状态的估算来估算当前的状态值。这个独特的性质可以让我们直接通过采样的回报来对状态值进行估算，从而有更小的偏差但会有更大的方差。

当我们有了环境的模型以后，状态价值函数就会很有用处了，因为我们就可以通过比较对一个状态的不同动作的价值平均值来选择在任意状态下的最好动作，就和在动态规划里一样。当模型未知时，我们需要把状态-动作价值估算出来。每一个状态-动作值需要被分别估计。现在，我们的学习目标就变成了 $q_\pi(s,a)$，即在状态 s 下根据策略 π 采取动作 a 时的预期回报。这在本质上与对状态价值函数的估计基本一致，而我们现在只是取状态 s 在动作 a 上的平均值而已。不过

有时，可能会有一些状态从来都没有被访问过，所以就没有回报。为了选择最优的策略，我们必须要探索所有的状态。一个简单的方法是直接选择那些没有可能被选择的状态-动作对来作为初始状态。这样一来，就可以保证在足够的回合数过后，所有的状态-动作对都是可以被访问的。我们把这样的一个假设叫作叫作探索开始（Exploring Starts）。

算法 2.8 首次蒙特卡罗预测

输入：初始化策略 π
初始化所有状态的 $V(s)$
初始化一列回报：Returns(s) 对所有状态
repeat
  通过 π: $S_0, A_0, R_0, S_1, \cdots, S_{T-1}, A_{T-1}, R_t$ 生成一个回合
  $G \leftarrow 0$
  $t \leftarrow T-1$
  for $t >= 0$ **do**
    $G \leftarrow \gamma G + R_{t+1}$
    if $S_0, S_1, \cdots, S_{t-1}$ 没有 S_t **then**
      Returns(S_t).append(G)
      $V(S_t) \leftarrow$ mean(Returns(S_t))
    end if
    $t \leftarrow t-1$
  end for
until 收敛

2.5.2 蒙特卡罗控制

现在我们可以把泛化策略迭代运用到蒙特卡罗中去，来看看它是怎么用来控制的。泛化策略迭代有两个部分：策略评估（Policy Evaluation）和策略提升（Policy Improvement）。策略评估的过程与之前小节中介绍的动态规划是一样的，所以我们主要来介绍策略提升。我们会对状态-动作值使用贪心策略，在这种情况下不需要使用环境模型。贪心策略会一直选择在一个状态下有最高价值的动作：

$$\pi(s) = \arg\max_a q(s,a) \tag{2.52}$$

对于每一次策略提升，我们都需要根据 q_{π_t} 来构造 π_{t+1}。这里展示策略提升是怎么实现的：

$$\begin{aligned} q_{\pi_t}(s, \pi_{t+1}(s)) &= q_{\pi_t}(s, \arg\max_a q_{\pi_t}(s,a)) \\ &= \max_a q_{\pi_t}(s,a) \\ &\geqslant q_{\pi_t}(s, \pi_t(s)) \end{aligned}$$

$$\geqslant v_{\pi_t}(s) \tag{2.53}$$

上面的式子证明了 π_{t+1} 不会比 π_t 差，而我们会在迭代策略提升后最终找到最优策略。这也意味着，我们可以对环境没有太多了解而只有采样得到的回合才使用蒙特卡罗。这里我们需要解决两个假设。第一个假设是探索开始，第二个是假设有无穷多个回合。我们先跳过第一个假设，从第二个假设开始。简化这个假设的一种简单方法是，通过直接在单个状态的评估和改进之间交替变更，来避免策略评估所需的无限多的片段（Episodes）。

2.5.3　增量蒙特卡罗

从算法 2.8 和算法 2.9 中可以看出，我们需要对观察到的回报序列求平均值，并且将状态价值和状态-动作价值的估计分开。其实我们还有一种更加高效的计算办法，它能让我们把回报序列省去，从而简化均值计算步骤。这样一来，我们就需要一个回合一个回合地更新。我们让 $Q(S_t, A_t)$ 作为它已经被选中 $t-1$ 次以后的状态-动作价值的估计，从而将其改写为

$$Q(S_t, A_t) = \frac{G_1 + G_2 + \cdots + G_{t-1}}{t-1} \tag{2.54}$$

算法 2.9 蒙特卡罗探索开始

初始化所有状态的 $\pi(s)$
对于所有的状态-动作对，初始化 $Q(s,a)$ 和 Returns(s,a)
repeat
　随机选择 S_0 和 A_0，直到所有状态-动作对的概率为非零
　根据 π: $S_0, A_0, R_0, S_1, \cdots, S_{T-1}, A_{T-1}, R_t$ 来生成 S_0, A_0
　$G \leftarrow 0$
　$t \leftarrow T-1$
　for $t >= 0$ **do**
　　$G \leftarrow \gamma G + R_{t+1}$
　　if $S_0, A_0, S_1, A_1 \cdots, S_{t-1}, A_{t-1}$ 没有 S_t, A_t **then**
　　Returns(S_t, A_t).append(G)
　　$Q(S_t, A_t) \leftarrow$ mean(Returns(S_t, A_t))
　　$\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$
　　end if
　　$t \leftarrow t-1$
　end for
until 收敛

对该式的一个简单实现是将所有的回报 G 值都记录下来，然后将它的和值除以它的访问次数。然而，我们同样也可以通过以下的公式来计算这个值：

$$
\begin{aligned}
Q_{t+1} &= \frac{1}{t}\sum_{i=1}^{t} G_i \\
&= \frac{1}{t}\left(G_t + \sum_{i=1}^{t-1} G_i\right) \\
&= \frac{1}{t}\left(G_t + (t-1)\frac{1}{t-1}\sum_{i=1}^{t-1} G_i\right) \\
&= \frac{1}{t}(G_t + (t-1)Q_t) \\
&= Q_t + \frac{1}{t}(G_t - Q_t)
\end{aligned} \tag{2.55}
$$

这个形式可以让我们在计算回报的时候更加容易操作。它的通用形式是：

$$\text{新估计值} \leftarrow \text{旧估计值} + \text{步伐大小} \cdot (\text{目标值} - \text{旧估计值}) \tag{2.56}$$

“步伐大小”是我们用来控制更新速度的一个参数。

2.6 时间差分学习

时间差分（Temporal Difference，TD）是强化学习中的另一个核心方法，它结合了动态规划和蒙特卡罗方法的思想。与动态规划相似，时间差分在估算的过程中使用了自举（Bootstrapping），但是和蒙特卡罗一样，它不需要在学习过程中了解环境的全部信息。在这章中，我们首先介绍如何将时间差分用于策略评估，然后详细阐释时间差分、蒙特卡罗和动态规划方法的异同点。最后，我们会介绍 Sarsa 和 Q-Learning 算法，这是一个在经典强化学习中很有用的算法。

2.6.1 时间差分预测

从这个方法的名字可以看出，时间差分利用差异值进行学习，即目标值和估计值在不同时间步上的差异。它使用自举法的原因是它需要从观察到的回报和对下个状态的估值中来构造它的目标。具体来说，最基本的时间差分使用以下的更新方式：

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{2.57}$$

这个方法也被叫作 TD(0), 或者是单步 TD。也可以通过将目标值改为在 N 步未来中的折扣回报和 N 步过后的估计状态价值（Estimated State Value）来实现 N 步 TD。如果我们观察得足够仔细，蒙特卡罗在更新时的目标值为 G_t，这个值只有在一个回合过后才能得知。但是对于 TD

来说，这个目标值是 $R_{t+1}+\gamma V(S_{t+1})$，而它可以在每一步都算出。在算法 2.10 中，我们展示了 TD(0) 是如何用来做策略评估的。

算法 2.10 TD(0) 对状态值的估算

输入策略 π
初始化 $V(s)$ 和步长 $\alpha \in (0,1]$
for 每一个回合 **do**
　初始化 S_0
　for 每一个在现有回合的 S_t **do**
　　$A_t \leftarrow \pi(S_t)$
　　$R_{t+1}, S_{t+1} \leftarrow \text{Env}(S_t, A_t)$
　　$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$
　end for
end for

这里分析一下动态规划、蒙特卡罗和时间差分方法的异同点。它们都是在现代强化学习中的核心算法，而且经常是被结合起来使用的。它们都可以被用于策略评估和策略提升，它们之间区别却是深度强化学习效果不同的主要来源之一。

这三种方法都涉及泛化策略迭代（GPI），它们主要区别在于策略评估的过程，其中最明显的区别是，动态规划和时间差分都使用了自举法，而蒙特卡罗没有。动态规划需要整个环境模型的所有信息，但是蒙特卡罗和时间差分不需要。进一步地，我们来看一下它们的学习目标的区别。

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] \tag{2.58}$$

$$= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \tag{2.59}$$

$$= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] \tag{2.60}$$

公式 (2.58) 是蒙特卡罗方法的状态价值估计方式，公式 (2.60) 是动态规划的。它们都不是真正的状态值而是估计值。时间差分则把蒙特卡罗的采样过程和动态规划的自举法结合了起来。现在我们就简单解释实践中时间差分可以比动态规划或者蒙特卡罗更有效的原因。

首先，时间差分不需要一个模型而动态规划需要。将时间差分与蒙特卡罗做比较，时间差分使用的是在线学习，这也就意味着它每一步都可以学习，但是蒙特卡罗却只能在一个回合结束以后再学习，这样回合很长时会比较难以处理。当然，也存在一些连续性的问题无法用片段式的形式来表示一个回合。另外，时间差分在实践中往往收敛得更快，因为它的学习是来自状态转移的信息而不需要具体动作信息，而蒙特卡罗往往需要动作信息。在理想情况下，两种方法最终都会渐进收敛到 $v_\pi(s)$。

这里我们介绍时间差分和蒙特卡罗方法中的**偏差和方差的权衡**（Bias and Variance Trade-off）。我们知道在监督学习的设置下，较大的偏差往往意味着这个模型欠拟合（Underfitting），而较大的方差伴随较低的偏差往往意味着一个模型过拟合（Overfitting）。一个拟合器（Estimator）的偏差

是估计值和真正值间的差异。我们对状态价值进行估计时，偏差可以被定义为 $\mathbb{E}[V(S_t)] - V(S_t)$。拟合器的方差描述了这个拟合有多大的噪声。同样对于状态价值估计，方差定义为 $\mathbb{E}[(\mathbb{E}[V(S_t)] - V(S_t))^2]$。在预测时，不管它是状态价值估计，还是状态-动作价值估计，时间差分和蒙特卡罗的更新都有如下形式：

$$V(S_t) \leftarrow V(S_t) + \alpha[\text{TargetValue} - V(S_t)]$$

实质上，我们对不同回合进行了加权平均计算。时间差分法和蒙特卡罗法在处理目标值时分别采用不同的方式。蒙特卡罗法直接估算到一个回合结束累计的回报。这也正是状态值的定义，它是没有偏差的。而时间差分法会有一定的偏差，因为它的目标值是由自举法估计得到的，如 $R_{t+1} + \gamma v_\pi(S_{t+1})$。另一方面，蒙特卡罗法，所以在不同回合中积累到最后的回报会有较大的方差由于不同回合的经过和结果都不同。时间差分法通过关注局部估计的目标值来解决这个问题，只依赖当前的奖励和下一个状态或动作价值的估计。自然地，时间差分法方差更小。

我们可以在动态规划和蒙特卡罗之间找到一个中间方法来更有效地解决问题，即 TD (λ)。在此之前，我们需要先介绍**资格迹**（Eligibility Trace）和 **λ-回报**（λ-Return）概念。

简单来说，资格迹可以给我们带来一些计算优势。为了更好地了解其优势，我们需要介绍半梯度（Semi-Gradient）方法，然后再来看如何使用资格迹。关于策略梯度方法在 2.7 节中有介绍，而这里我们简单地使用一些策略梯度方法中的概念来方便解释资格迹。假如说我们的状态价值函数不是表格（Tabular）形式而是一种函数形式，这个函数由矢量 $\boldsymbol{w} \in \mathbb{R}^n$ 参数化。比如 $\boldsymbol{w}$ 可以是一个神经网络的权重。为了得到 $V(s, \boldsymbol{w}) \approx v_\pi(s)$，我们使用随机梯度更新来减小估计值和真正的状态价值的平方损失（Quadratic Loss）。权重向量的更新规则就可以写为

$$\begin{aligned}\boldsymbol{w}_{t+1} &= \boldsymbol{w}_t - \frac{1}{2}\alpha\nabla_{\boldsymbol{w}_t}[v_\pi(S_t) - V(S_t, \boldsymbol{w}_t)]^2 \\ &= \boldsymbol{w}_t + \alpha[v_\pi(S_t) - V(S_t, \boldsymbol{w}_t)]\nabla_{\boldsymbol{w}_t} V(S_t, \boldsymbol{w}_t)\end{aligned} \tag{2.61}$$

其中 α 为一个正值的步长变量。

资格迹是一个向量：$\boldsymbol{z}_t \in \mathbb{R}^n$，在学习的过程中，每当 $\boldsymbol{w}_t$ 的一个部分被用于估计，则它在 $\boldsymbol{z}_t$ 里的那个相对应的部分需要随之增加，而在增加以后它又会慢慢递减。如果轨迹上的资格值回落到零之前，有一定的 TD 误差，就进行学习。首先把所有资格值都初始化为 0，然后使用价值函数的梯度来增加资格迹，而资格值递减的速度是 $\gamma\lambda$。资格迹的更新满足如下公式：

$$\boldsymbol{z}_{-1} = 0 \tag{2.62}$$

$$\boldsymbol{z}_t = \gamma\lambda\boldsymbol{z}_{t-1} + \nabla_{\boldsymbol{w}_t} V(S_t, \boldsymbol{w}_t) \tag{2.63}$$

如算法 2.11 所示，TD(λ) 使用资格迹来更新其价值函数估计。易见，当 $\lambda = 1$ 时，TD(λ) 变为蒙

特卡罗法；而当 $\lambda = 0$ 时，它就变成了一个单步 TD（One-Step TD）法。因此，资格迹可以看作是把时间差分法和蒙特卡罗法相结合的一个方法。

算法 2.11 状态值半梯度 TD(λ)

输入策略 π
初始化一个可求导的状态值函数 v、步长 α 和状态值函数权重 $\boldsymbol{w}$
for 对每一个回合 **do**
　初始化 S_0
　$\boldsymbol{z} \leftarrow 0$
　for 每一个本回合的步骤 S_t **do**
　　使用 π 来选择 A_t
　　$R_{t+1}, S_{t+1} \leftarrow \text{Env}(S_t, A_t)$
　　$\boldsymbol{z} \leftarrow \gamma\lambda\boldsymbol{z} + \nabla V(S_t, \boldsymbol{w}_t)$
　　$\delta \leftarrow R_{t+1} + \gamma V(S_{t+1}, \boldsymbol{w}_t) - V(S_t, \boldsymbol{w}_t)$
　　$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha\delta\boldsymbol{z}$
　end for
end for

λ-回报是之后 n 步中的估计回报值。λ-回报是 n 个已经折扣化的回报和一个在最后一步状态下的估计值相加得到的。我们可以把它写作：

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n v(S_{t+n}, \boldsymbol{w}_{t+n-1}) \tag{2.64}$$

t 是一个不为零的标量，它小于或等于 $T - n$。我们可以使用加权平均回报来估算，只要它们的权重满足和为 1。TD(λ) 在其更新中使用了加权平均（$\lambda \in [0, 1]$）：

$$G_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n} \tag{2.65}$$

直观地讲，这就意味着下一步的回报将有最大的权重 $1 - \lambda$，下两步回报的权重是 $(1 - \lambda)\lambda$。每一步权重递减的速率是 λ。为了有更清晰的理解，我们让结束状态发生于时间 T，从而上面的公式可以改写成

$$G_t^\lambda = (1 - \lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{T-t-1} G_t \tag{2.66}$$

TD 误差 δ_t 可以被定义为

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}, \boldsymbol{w}_t) - V(S_t, \boldsymbol{w}_t) \tag{2.67}$$

这个更新规则是基于 TD 误差和迹的比重的。算法 2.11 里有其细节。

2.6.2 Sarsa：在线策略 TD 控制

对于 TD 控制，我们使用的方法和预测任务一样，唯一的不同是，我们需要将从状态到状态的转移变为状态-动作对的交替。这样的更新规则就可以被写为

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \tag{2.68}$$

当 S_t 是终止状态（Terminal State）的时候，下一个状态-动作对的 Q 值就会变成 0。我们用首字母缩写 Saras 来表示这个算法，因为我们有这样的一个行为过程：首先在一个状态（S）下，选择了一个动作（A），同时也观察到了回报（R），然后我们就到了另外一个状态（S）下，需要选择一个新的动作（A）。这样的过程让我们可以做一个简单的更新步骤。对于每一个转移，状态价值都得到更新，更新后的状态价值会影响决定动作的策略，即**在线策略**法。在线策略法一般用来描述这样一类算法，它们的更新策略和行动策略（Behavior Policy）同样。而离线策略法往往是不同的。Q-Learning 就是离线策略算法的一个例子。我们会在之后的章节中提到。Q-Learning 在更新 Q 函数时假设了一种完全贪心的方法，而它在选择其动作时实际上用的是另外一种类似于 ϵ-贪心（ϵ-Greedy）的方法。现在我们在算法 2.12 中列出 Sarsa 的细节。在每一个状态-动作对都会被访问无数次的假设下，会有最优策略和状态动作价值的收敛性保证。

算法 2.12 Sarsa （在线策略 TD 控制）

对所有的状态-动作对初始化 $Q(s, a)$
for 每一个回合 **do**
  初始化 S_0
  用一个基于 Q 的策略来选择 A_0
  for 每一个在当前回合的 S_t **do**
    用一个基于 Q 的策略从 S_t 选择 A_t
    $R_{t+1}, S_{t+1} \leftarrow \text{Env}(S_t, A_t)$
    从 S_{t+1} 中用一个基于 Q 的策略来选择 A_{t+1}
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$
  end for
end for

上面展示的方法只有一步的时间范围，这就意味着它的估算只需要考虑下一步的状态-动作价值。我们把它叫作单步 Sarsa 或者 Sarsa(0)。我们可以简单地使用自举法把未来的步骤也都容纳到目标值中从而减少它的偏差。从图 2.12 的回溯树展示中，我们可以看见 Sarsa 很多不同的变体。从最简单的一步 Sarsa 到无限步 Sarsa，也就是蒙特卡罗方法的另外一个形态。为了把这样的

一个变化融入原来的方法，我们需要把折扣回报写为

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}) \tag{2.69}$$

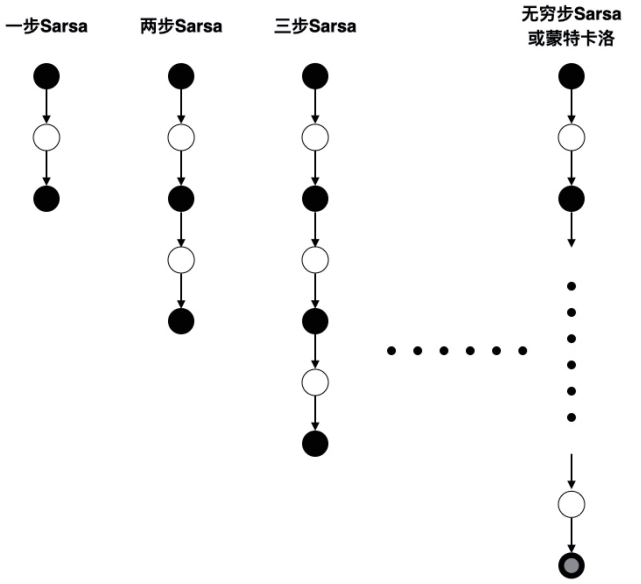


图 2.12　对于 n 步 Sarsa 方法的回溯树。每一个黑色的圆圈都代表了一个状态，每一个白色的圆圈都代表了一个动作。在这个无穷多步的 Sarsa 里，最后一个状态就是它的终止状态

n 步 Sarsa 已经在算法 2.13 中有所描述了。和单步版本最大的不同是，它需要回到过去的时间来做更新，而单步的版本只需要一边向前进行一边更新即可。

现在讨论 Sarsa 算法在有限的动作空间里的收敛理论。我们首先需要以下的几个条件。

定义 2.1 一个学习策略被定义为：在无限的探索中的极限贪婪（Greedy in the Limit with Infinite Exploration, GLIE）。如果它能够满足以下两个性质：1. 如果一个状态被无限次访问，那么在该状态下的每个可能的动作都应当被无限次选择，即 $\lim_{k\to\infty} N_k(s,a) = \infty, \forall a, \text{if} \lim_{k\to\infty} N_k(s) = \infty$。

2. 策略根据学习到的 Q 函数在 $t \to \infty$ 的极限下收敛到一个贪婪策略，即 $\lim_{k\to\infty} \pi_k(s,a) = \mathbb{1}(a == \arg\max_{a' \in \mathcal{A}} Q_k(s,a'))$，其中“==”是一个比较算子，当 $\mathbb{1}(a == b)$ 的括号内为真时，它的值为 1，否则为 0。

GLIE 是学习策略收敛的一个条件，对于任何收敛到最优价值函数且估计值都有界（Bounded）的强化学习算法来说，它都成立。举例来说，我们可以通过 ϵ 贪心方法来推导出一个 GLIE 的策略，如下：

引理 2.1 如果 ϵ 以 $\epsilon_k = \frac{1}{k}$ 的形式随 k 增大而渐趋于零，那么 ϵ-贪心是 GLIE。

算法 2.13 n 步 Sarsa

对所有的状态-动作对初始化 $Q(s,a)$
初始化步长 $\alpha \in (0,1]$
决定一个固定的策略 π 或者使用 ϵ-贪心策略
for 每一个回合 **do**
  初始化 S_0
  使用 $\pi(S_0,A)$ 来选择 A_0
  $T \leftarrow$ INTMAX （一个回合的长度）
  $\gamma \leftarrow 0$
  for $t \leftarrow 0,1,2,\cdots$ until $\gamma - T - 1$ **do**
    if $t < T$ **then**
      $R_{t+1}, S_{t+1} \leftarrow \text{Env}(S_t, A_t)$
      if S_{t+1} 是终止状态 **then**
        $T \leftarrow t+1$
      else
        使用 $\pi(S_t,A)$ 来选择 A_{t+1}
      end if
    end if
    $\tau \leftarrow t-n+1$ （更新的时间点。这是 n 步 Sarsa，只需更新 $n+1$ 前的一步，持续下去直到所有状态都被更新。）
    if $\tau \geqslant 0$ **then**
      $G \leftarrow \sum_{i=\tau+1}^{\min(r+n,T)} \gamma^{i-\gamma-1} R_i$
      if $\gamma + n < T$ **then**
        $G \leftarrow G + \gamma^n Q(S_{t+n}, A_{\gamma+n})$
      end if
      $Q(S_\gamma, A_\gamma) \leftarrow Q(S_\gamma, A_\gamma) + \alpha[G - Q(S_\gamma, A_\gamma)]$
    end if
  end for
end for

因而就有了 Sarsa 算法的收敛定理。

定理 2.1 *对于一个有限状态-动作的 MDP 和一个 GLIE 学习策略，其动作价值函数 Q 在时间步 t 上由 Sarsa（单步的）估计为 Q_t，那么如果以下两个条件得到满足，Q_t 会收敛到 Q^* 并且学习策略 π_t，也会收敛到最优策略 π^*：*

1. *Q 的值被存储在一个查找表（Lookup Table）里；*

2. *在时间 t 与状态-动作对 (s,a) 相关的学习速率（Learning Rate）$\alpha_t(s,a)$ 满足 $0 \leqslant \alpha_t(s,a) \leqslant 1$，$\sum_t \alpha_t(s,a) = \infty$，$\sum_t \alpha_t^2(s,a) < \infty$，并且 $\alpha_t(s,a) = 0$ 除非 $(s,a) = (S_t, A_t)$；*

3. *方差 $\text{Var}[R(s,a)] < \infty$。*

符合第二个条件对学习速率的要求的一个典型数列是 $\alpha_t(S_t,A_t) = \frac{1}{t}$。我们在这里对上面定理的证明不做介绍，有兴趣的读者可以查看文献 (Singh et al., 2000)。

2.6.3　Q-Learning：离线策略 TD 控制

Q-Learning 是一种离线策略方法，与 Saras 很类似，在深度学习应用中有很重要的作用，如深度 Q 网络（Deep Q-Networks）。如公式 (2.70) 所示，Q-Learning 和 Sarsa 主要的区别是，它的目标值现在不再依赖于所使用的策略，而只依赖于状态-动作价值函数。

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{2.70}$$

在算法 2.14 中，我们展示了如何用 Q-Learning 控制 TD。将 Q-Learning 变成 Sarsa 算法也很容易，可以先基于状态和回报选择动作，然后在更新步中将目标值改为估计的下一步动作价值。上面展示的是单步 Q-Learning，我们也可以把 Q-Learning 变成 n 步的版本。具体做法是将公式 (2.70) 里的目标值加入未来的折扣后的回报。

算法 2.14 Q-Learning （离线策略 TD 控制）

初始化所有的状态-动作对的 $Q(s, a)$ 及步长 $\alpha \in (0, 1]$
for 每一个回合 **do**
　初始化 S_0
　for 每一个在当前回合的 S_t **do**
　　使用基于 Q 的策略来选择 A_t
　　$R_{t+1}, S_{t+1} \leftarrow \text{Env}(S_t, A_t)$
　　$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$
　end for
end for

Q-Learning 的收敛性条件和 Sarsa 算法的很类似。除了对策略有的 GLIE 条件，Q-Learning 中 Q 函数的收敛还对学习速率和有界奖励值要求，这里不再复述，具体的证明可以在文献 (Szepesvári, 1998; Watkins et al., 1992) 中找到。

2.7　策略优化

2.7.1　简介

在强化学习中，智能体的最终目标是改进它的策略来获得更好的奖励。在优化范畴下的策略改进叫策略优化（图 2.13）。对深度强化学习而言，策略和价值函数通常由深度神经网络中的变量来参数化，因此可以使用基于梯度的优化方法。举例来说，图 2.14 展示了使用参数化策略的 MDP 的概率图模型（Graphical Model），其中策略由变量 θ 参数化，在离散时间范围 $t = 0, \cdots, N-1$ 内。奖励函数表示为 $R_t = R(S_t, A_t)$，而动作表示为 $A_t \sim \pi(\cdot|S_t; \theta)$。图模型中变量的依赖关系可以帮助我们理解 MDP 估计中的潜在关系，而且可以有助于我们在依赖关系图中对最终目标求

导而优化变量有帮助，因此我们将在本章展示所有的图模型来帮助理解推导过程，尤其对那些可微分的过程。近来，文献 (Levine, 2018) 和文献 (Fu et al., 2018) 提出了一种“推断式控制（Control as Inference）”的方法，这个方法在 MDP 的图模型上添加了额外的表示最优性（Optimality）的变量，从而将概率推断或变分推断（Variational Inference）的框架融合到有相同目标的最大熵强化学习（Maximum Entropy Reinforcement Learning）中。这个方法使得推断类工具（Inference Tools）可以应用到强化学习的策略优化过程中。但是关于这些方法的具体细节超出了本书范围。

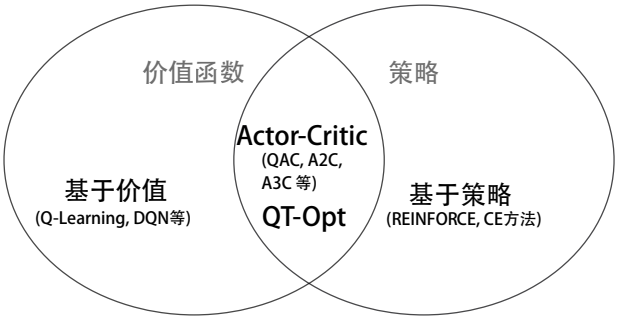


图 2.13 强化学习中策略优化概览

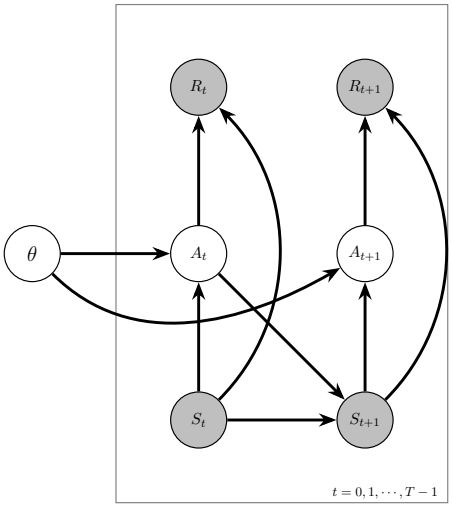


图 2.14 使用参数化策略的 MDP 概率图模型

除了一些线性方法，使用深度神经网络对价值函数参数化是一种实现**价值函数拟合**（Value Function Approximation）的方式，而这是现代深度强化学习领域中最普遍的方式，而在多数实际情况中，我们无法获得真实的价值函数。图 2.15 展示了使用参数化策略 π_θ 和参数化价值函数 $V_w^\pi(S_t)$ 的 MDP 概率图模型，它们的参数化过程分别使用了参数 θ 和 w。图 2.16 展示了使用参

数化策略 π_θ 和参数化 Q 值函数 $Q_w^\pi(S_t, A_t)$ 的 MDP 概率图模型。一般通过在强化学习术语中被称为**策略梯度**（Policy Gradient）的方法改进参数化策略。然而，也有一些非基于梯度的方法（Non-Gradient-Based Methods）可以优化不那么复杂的参数化策略，比如交叉熵（Cross-Entropy，CE）方法等。

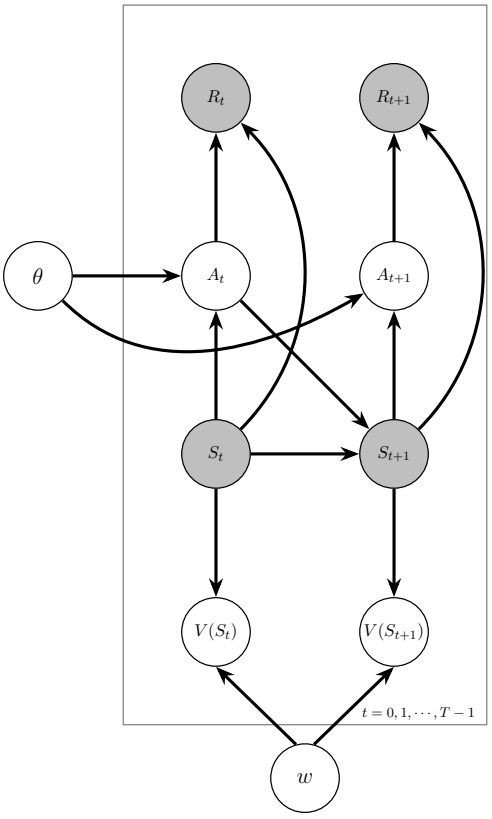


图 2.15　使用参数化策略和参数化价值函数的 MDP 概率图模型

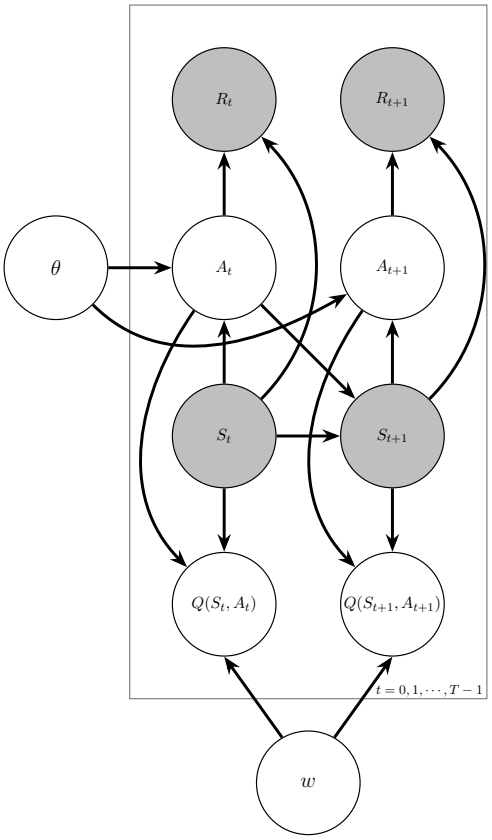


图 2.16　使用参数化策略和参数化 Q 值函数的 MDP 概率图模型

如图 2.13 所示，策略优化算法往往分为两大类：（1）**基于价值的优化**（Value-Based Optimization）方法，如 Q-Learning、DQN 等，通过优化动作价值函数（Action-Value Function）来获得对动作选择的偏好；（2）**基于策略的优化**（Policy-Based Optimization）方法，如 REINFORCE、交叉熵算法等，通过根据采样的奖励值来直接优化策略。这两类的结合被人们 (Kalashnikov et al., 2018; Peters et al., 2008; Sutton et al., 2000) 发现是一种更加有效的方式，而这构成了一种在无模型（Model-Free）强化学习中应用最广的结构，称为 **Actor-Critic**。Actor-Critic 方法通过对价值函数的优化来引导策略改进。在这类结合型算法中的典型包括 Actor-Critic 类的方法和以其为基础的

其他算法，后续有关于这些算法的详细介绍。

回顾强化学习梗概

在线价值函数（On-Policy Value Function），$v_\pi(s)$，给出以状态 s 为起始并在后续过程始终遵循策略 π 的期望回报（Expected Return）：

$$v_\pi(s) = \mathbb{E}_{\tau\sim\pi}[R(\tau)|S_0 = s] \tag{2.71}$$

强化学习的优化问题可以被表述为

$$\pi_* = \arg\max_\pi J(\pi) \tag{2.72}$$

最优价值函数（Optimal Value Function），$v^*(s)$，给出以状态 s 为起始并在后续过程始终遵循环境中最优策略的期望回报：

$$v_*(s) = \max_\pi v_\pi(s) \tag{2.73}$$

$$v_*(s) = \max_\pi \mathbb{E}_{\tau\sim\pi}[R(\tau)|S_0 = s] \tag{2.74}$$

在线动作价值函数（On-Policy Action-Value Function），$q_\pi(s,a)$，给出以状态 s 为起始并采取任意动作 a（有可能不来自策略），而随后始终遵循策略 π 的期望回报：

$$q_\pi(s,a) = \mathbb{E}_{\tau\sim\pi}[R(\tau)|S_0 = s, A_0 = a] \tag{2.75}$$

最优动作价值函数（Optimal Action-Value Function），$q_*(s,a)$，给出以状态 s 为起始并采取任意动作 a，而随后始终遵循环境中最优策略的期望回报：

$$q_*(s,a) = \max_\pi q_\pi(s,a) \tag{2.76}$$

$$q_*(s,a) = \max_\pi \mathbb{E}_{\tau\sim\pi}[R(\tau)|S_0 = s, A_0 = a] \tag{2.77}$$

价值函数（Value Function）和**动作价值函数**（Action-Value Function）的关系：

$$v_\pi(s) = \mathbb{E}_{a\sim\pi}[q_\pi(s,a)] \tag{2.78}$$

$$v_*(s) = \max_a q_*(s,a) \tag{2.79}$$

最优动作：

$$a_*(s) = \arg\max_a q_*(s,a) \tag{2.80}$$

贝尔曼方程:
对状态价值和动作价值的贝尔曼方程分别为：

$$v_\pi(s) = \mathbb{E}_{a\sim\pi(\cdot|s),s'\sim p(\cdot|s,a)}[R(s,a) + \gamma v_\pi(s')] \tag{2.81}$$

$$q_\pi(s,a) = \mathbb{E}_{s'\sim p(\cdot|s,a)}[R(s,a) + \gamma \mathbb{E}_{a'\sim\pi(\cdot|s')}[q_\pi(s',a')]] \tag{2.82}$$

贝尔曼最优方程：
对状态价值和动作价值的贝尔曼最优方程分别为：

$$v_*(s) = \max_a \mathbb{E}_{s'\sim p(\cdot|s,a)}[R(s,a) + \gamma v_*(s')] \tag{2.83}$$

$$q_*(s,a) = \mathbb{E}_{s'\sim p(\cdot|s,a)}[R(s,a) + \gamma \max_{a'} q_*(s',a')] \tag{2.84}$$

2.7.2　基于价值的优化

基于价值的优化（Value-Based Optimization）方法经常需要在（1）基于当前策略的价值函数估计和（2）基于所估计的价值函数进行策略优化这两个过程之间交替。然而，估计一个复杂的价值函数并不容易，如图 2.17 所示。

从之前小节中我们可以看到，Q-Learning 可以被用来解决强化学习中一些简单的任务。然而，现实世界或者即使准现实世界中的应用也都可能有更大和更复杂的状态动作空间，而且实际应用中很多动作是连续的。比如，在围棋游戏中有约 10^{170} 个状态。在这些情况下，Q-Learning 中的传统查找表（Lookup Table）方法因为每个状态需要有一条记录（Entry）而每个状态-动作对也需要一条 $Q(s,a)$ 记录而使其可扩展性（Scalability）有待提升。实践中，这个表中的值需要一个一个地更新。所以基于表格（Tabular-Based）的 Q-Learning 对内存和计算资源的需求可能是巨大的。此外，在实践中，状态表征（State Representations）通常也需要人为指定成相匹配的数据结构。

价值函数拟合

为了将基于价值的强化学习应用到相对大规模的任务上，函数拟合器（Function Approximators）可用来应对上述限制条件（图 2.18）。图 2.18 总结了不同类型的价值函数拟合器。

- **线性方法**（Linear Methods）：拟合函数是权重 $\boldsymbol{\theta}$ 和特征实数向量 $\boldsymbol{\phi}(s) = (\phi_1(s), \phi_2(s)), \cdots, \phi_n(s))^{\mathrm{T}}$ 的线性组合，其中 s 是状态。拟合函数表示为 $v(s,\boldsymbol{\theta}) = \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{\phi}(s)$。TD($\lambda$) 方法因使用线性函数拟合器而被证明在一定条件下可以收敛 (Tsitsiklis et al., 1997)。尽管线性方法的

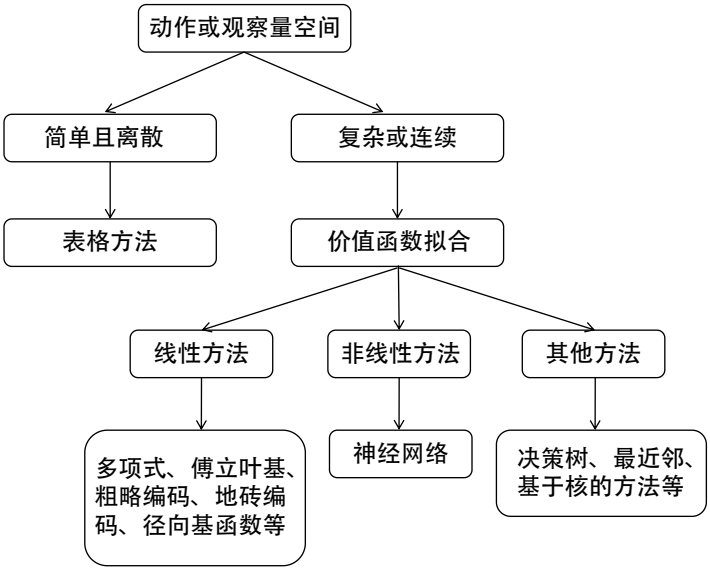


图 2.17 求解价值函数的方法概览

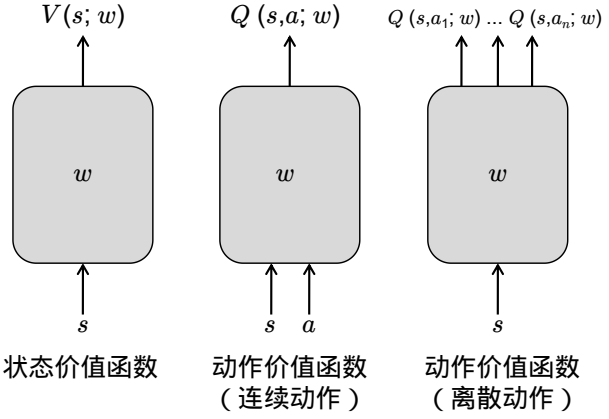


图 2.18 不同的价值函数拟合方式。内含参数 w 的灰色方框是函数拟合器

收敛性保证很诱人，但实际上在使用该方法时特征选取或特征表示 $\boldsymbol{\phi}(s)$ 有一定难度。如下是线性方法中构建特征的不同方式：

- **多项式**（Polynomials）：基本的多项式族（Polynomial Families）可以用作函数拟合的特征矢量（Feature Vectors）。假设每一个状态 $\boldsymbol{s} = (S_1, S_2, \cdots, S_d)^{\mathrm{T}}$ 是一个 d 维向量，那么我们有一个 d 维的多项式基（Polynomial Basis）$\phi_i(\boldsymbol{s}) = \prod_{j=1}^{d} S_j^{c_{i,j}}$，其中每个 $c_{i,j}$ 是集合 $\{0, 1, \cdots, N\}$ 中的一个整数。这构成秩（Order）为 N 的多项式基和 $(N+1)^d$ 个不同的函数。

  - **傅立叶基**（Fourier Basis）：傅立叶变换（Fourier Transformation）经常用于表示在时间域或频率域的序列信号。有 $N+1$ 个函数的一维秩为 N 的傅立叶余弦（Cosine）基为 $\phi_i(s) = \cos(i\pi s)$，其中 $s \in [0,1]$ 且 $i = 0, 1, \cdots, N$。
  - **粗略编码**（Coarse Coding）：状态空间可以从高维缩减到低维，例如用一个区域覆盖决定过程（Determination Process）来进行二值化表示（Binary Representation），这被称为粗略编码。
  - **瓦式编码**（Tile Coding）：在粗略编码中，瓦式编码对于多维连续空间是一种高效的特征表示方式。瓦式编码中特征的感知域（Receptive Field）被指定成输入空间的不同分割（Partitions）。每一个分割称为一个瓦面（Tilling），而分割中的每一个元素称为一个瓦片（Tile）。许多有着重叠感知域的瓦面往往被结合使用，以得到实际的特征矢量。
  - **径向基函数**（Radial Basis Functions，RBF）：径向基函数自然地泛化了粗略编码，粗略编码是二值化的，而径向基函数可用于 $[0,1]$ 内的连续值特征。典型的 RBF 是以高斯函数（Gaussian）的形式 $\phi_i(s) = \exp(-\frac{\|s-c_i\|^2}{2\sigma_i^2})$，其中 s 是状态，c_i 是特征的原型（Prototypical）或核心状态（Center State），而 σ_i 是特征宽度（Feature Width）。

- **非线性方法**（Non-Linear Methods）：
  - **人工神经网络**（Artificial Neural Networks）：不同于以上的函数拟合方法，人工神经网络被广泛用作非线性函数拟合器，它被证明在一定条件下有普遍的拟合能力（Universal Approximation Ability）(Leshno et al., 1993)。基于深度学习技术，人工神经网络构成了现代基于函数拟合的深度强化学习方法的主体。一个典型的例子是 DQN 算法，使用人工神经网络来对 Q 值进行拟合。

- **其他方法**：
  - **决策树**（Decision Trees）：决策树 (Pyeatt et al., 2001) 可以用来表示状态空间，通过使用决策节点（Decision Nodes）对其分割。这构成了一种重要的状态特征表示方法。
  - **最近邻**（Nearest Neighbor）方法：它测量了当前状态和内存中之前状态的差异，并用内存中最接近状态的值来近似当前状态的值。

使用价值函数拟合的好处不仅包括可以扩展到大规模任务，以及便于在连续状态空间中进行从所见状态到未见过状态的泛化，而且可以减少或缓解人为设计特征来表示状态的需要。对于无模型方法，拟合器的参数 w 可以用蒙特卡罗（Monte-Carlo，MC）或时间差分（Temporal Difference，TD）学习来更新，可以对批量样本进行参数更新而非像基于表格的方法一样逐个更新。这使得处理大规模问题时有较高的计算效率。对基于模型的方法，参数可以用动态规划（Dynamic Programming，DP）来更新。关于 MC、TD 和 DP 的细节在之前已经有所介绍。

可能的函数拟合器包括特征的线性组合、神经网络、决策树和最近邻方法等。神经网络因其很好的可扩展性和对多样函数的综合能力而成为深度强化学习方法中最实用的拟合方法。神经网络是一个可微分方法，因而可以基于梯度进行优化，这提供了在凸（Convex）函数情况下收敛到最优的保证。然而，实践中，它可能需要极大量的数据来训练，而且可能造成其他困难。

将深度学习问题扩展到强化学习带来了额外的挑战，包括非独立同分布（Not Independently and Identically Distributed）的数据。绝大多数监督学习方法建立在这样一个假设之上，即训练数据是从一个稳定的独立同分布 (Schmidhuber, 2015) 中采样得到的。然而，强化学习中的训练数据通常包括高度相关的样本，它们是在智能体和环境交互中顺序得到的，而这违反了监督学习中的独立性条件。更糟的是，强化学习中的训练数据分布通常是不稳定的，因为价值函数经常根据当前策略来估计，或者至少受当前策略对状态的访问频率影响，而策略是随训练一直在更新的。智能体通过对在状态空间探索不同部分来学习。所有这些情况违反了样本数据来自同分布的条件。

在强化学习中使用价值函数拟合对表征方式也有一些实际要求，而如果没有适当地考虑到这些实际要求，将可能导致发散的情况的发生 (Achiam et al., 2019)。具体来说，不稳定性和发散带来的危险在以下三个条件同时发生时就会产生：（1）在一个转移分布（Distribution of Transitions）上训练，而这个分布不满足由一个过程自然产生且这个过程的期望值被估计（比如在离线学习中）的条件；（2）可扩展的函数拟合，比如，线性半梯度（Semi-Gradient）；（3）自举（Bootstrapping），比如 DP 和 TD 学习。这三个主要属性只有在它们被结合时会导致学习的发散，而这被称为**死亡三件套**（the Deadly Triad）(Van Hasselt et al., 2018)。在使用函数拟合的方式不足够公正的情况下，基于价值的方法使用函数拟合时可能会有过估计或欠估计（Over-/Under-Estimation）的问题。举例来说，原始 DQN 有 Q 值过估计（Over-Estimation）的问题 (Van Hasselt et al., 2016)，这在实践中会导致略差的学习表现，而 Double/Dueling DQN 技术被提出来缓解这个问题。总体来说，使用策略梯度的基于策略的方法相比基于价值的方法有更好的收敛性保证。

基于梯度的价值函数拟合

考虑参数化的价值函数 $V^\pi(s) = V^\pi(s;w)$ 或 $Q^\pi(s,a) = Q^\pi(s,a;w)$，我们可以基于不同的估计方法得到相应的更新规则。优化目标被设置为估计函数 $V^\pi(s;w)$（或 $Q^\pi(s,a;w)$) 和真实价值函数 $v_\pi(s)$（或 $q_\pi(s,a)$）间的均方误差（Mean-Squared Error，MSE）：

$$J(w) = \mathbb{E}_\pi[(V^\pi(s;w) - v_\pi(s))^2] \tag{2.85}$$

或

$$J(w) = \mathbb{E}_\pi[(Q^\pi(s,a;w) - q_\pi(s,a))^2] \tag{2.86}$$

因此，用随机梯度下降（Stochastic Gradient Descent）法所得到的梯度为

$$\Delta w = \alpha(V^\pi(s;w) - v_\pi(s))\nabla_w V^\pi(s;w) \tag{2.87}$$

或

$$\Delta w = \alpha(Q^\pi(s,a;w) - q_\pi(s,a))\nabla_w Q^\pi(s,a;w) \tag{2.88}$$

其中梯度对批中的每一个样本进行计算，而权重以一种随机的方式进行更新。上述等式中的目标价值函数 v_π 或 q_π 通常是被估计的，有时使用一个目标网络（DQN 中）或一个最大化算子（Q-Learning 中）等。我们在这里展示价值函数的一些基本估计方式。

对 **MC** 估计，目标值是用采样的回报 G_t 估计的。因此，价值函数参数的更新梯度为

$$\Delta w_t = \alpha(V^\pi(S_t;w_t) - G_t)\nabla_{w_t} V^\pi(S_t;w_t) \tag{2.89}$$

或

$$\Delta w_t = \alpha(Q^\pi(S_t,A_t;w_t) - G_{t+1})\nabla_{w_t} Q^\pi(S_t,A_t;w_t) \tag{2.90}$$

对 **TD(0)**，根据式 (2.84) 表示的贝尔曼最优方程，目标值是时间差分的目标函数 $R_t+\gamma V_\pi(S_{t+1};w_t)$，因此：

$$\Delta w_t = \alpha(V^\pi(S_t;w_t) - (R_t + \gamma V_\pi(S_{t+1};w_t)))\nabla_{w_t} V^\pi(S_t;w_t) \tag{2.91}$$

或

$$\Delta w_t = \alpha(Q^\pi(S_t,A_t;w_t) - (R_{t+1} + \gamma Q_\pi(S_{t+1},A_{t+1};w_t))\nabla_{w_t} Q^\pi(S_t,A_t;w_t)) \tag{2.92}$$

对 **TD(λ)**，目标值是 λ-回报即 G_t^λ，因此更新规则是

$$\Delta w_t = \alpha(V^\pi(S_t;w_t) - G_t^\lambda)\nabla_{w_t} V^\pi(S_t;w_t) \tag{2.93}$$

或

$$\Delta w_t = \alpha(Q^\pi(S_t,A_t;w_t) - G_t^\lambda)\nabla_{w_t} Q^\pi(S_t,A_t;w_t) \tag{2.94}$$

不同的估计方式对偏差和方差有不同的侧重，这在之前的小节中已经有所介绍，比如 MC 和 TD 估计方法等。

例子：深度 Q 网络

深度 Q 网络（DQN）是基于价值优化的典型例子之一。它使用一个深度神经网络来对 Q-Learning 中的 Q 值函数进行拟合，并维护一个经验回放缓存（Experience Replay Buffer）来存储智能体-环境交互中的转移样本。DQN 也使用了一个目标网络 Q^{T}，而它由原网络 Q 的参数副本来参数化，并且以一种延迟更新的方式，来稳定学习过程，也即缓解深度学习中非独立同分布数据的问题。它使用如式 (2.88) 中的 MSE 损失，以及用贪心的拟合函数 $r + \gamma \max_{a'} Q^{\mathrm{T}}(s', a')$ 替代真实价值函数 q_π。

经验回放缓存为学习提供了稳定性，因为从缓存中采样到的随机批量样本可以缓解非独立同分布的数据问题。这使得策略更新成为一种离线的（Off-Policy）方式，由于当前策略和缓存中来自先前策略的样本间的差异。

2.7.3 基于策略的优化

在开始介绍基于策略的优化（Policy-Based Optimization）之前，我们首先介绍在强化学习中常见的一些策略。如之前小节中所介绍，强化学习中的策略可以被分为确定性（Deterministic）和随机性（Stochastic）策略。在深度强化学习中，我们使用神经网络来表示这两类策略，称为**参数化策略**（Parameterized Policies）。具体来说，这里的参数化指抽象的策略用神经网络（包括单层感知机）参数来，而非其他参量来表示。使用神经网络参数 θ，确定性和随机性策略可以分别写作 $A_t = \mu_\theta(S_t)$ 和 $A_t \sim \pi_\theta(\cdot|S_t)$。

在深度强化学习领域，有一些常见的具体分布用来表示随机性策略中的动作分布：伯努利分布（Bernoulli Distribution），类别分布（Categorical Distribution）和对角高斯分布（Diagonal Gaussian Distribution）。伯努利和类别分布可以用于离散动作空间，如二值的（Binary）或多类别的（Multi-Category），而对角高斯分布可以用于连续动作空间。

一个以 θ 为参数的单变量 $x \in \{0, 1\}$ 的伯努利分布为 $P(s; \theta) = \theta^x (1-\theta)^{(1-x)}$。因而它可以被用于表示二值化的动作，可以是单维，也可以是多维（对一个矢量中含多个变量的情况应用），它可以用作**二值化动作策略**（Binary-Action Policy）。

类别型策略（Categorical Policy）使用类别分布作为它的输出，因而可以用于离散且有限的动作空间，它将策略视为一个分类器（Classifier），以状态为条件（Conditioned on A State）而输出在有限动作空间中每个动作的概率，比如 $\pi(a|s) = \boldsymbol{P}[A_t = a|S_t = s]$。所有概率和为 1，因此，当将类别型策略参数化时，最后输出层（Output Layer）常用 Softmax 激活函数。这里我们具体使用 $\boldsymbol{P}[\cdot|\cdot]$ 矩阵表示有限动作空间的情况，来替代概率函数 $p(\cdot|\cdot)$。智能体可以根据类别分布采样选择一个动作。实践中，这种情况下的动作通常可以编码为一个独热编码矢量（One-Hot Vector）$\boldsymbol{a}_i = (0, 0, \cdots, 1, \cdots, 0)$，这个矢量跟动作空间有相同的维度，从而 $\boldsymbol{a}_i \odot \boldsymbol{p}(\cdot|s)$ 给出 $p(\boldsymbol{a}_i|s)$，其中 $\odot$ 是逐个元素的乘积（Element-Wise Product）算子，而 $\boldsymbol{p}(\cdot|s)$ 是给定状态 s 时的矩阵中的一个矢量（行或列，依状态动作顺序而定），而这通常也是归一化后类别型策略的输出层。**耿贝尔-Softmax**

函数技巧（Gumbel-Softmax Trick）可以在实践中参数化类别型策略后用来保持类别分布采样过程的可微性。在没有使用其他技巧的情况下，有采样过程或像 $\arg\max$ 类操作的随机性节点往往是不可微的（Non-Differentiable），从而在对参数化策略使用基于梯度的优化（在随后小节中介绍）时可能是有问题的。

耿贝尔-Softmax 函数技巧（Gumbel-Softmax Trick）：首先，耿贝尔-最大化技巧（Gumbel-Max Trick）允许我们从类别分布 π 中采样

$$\boldsymbol{z} = \text{one\_hot}[\arg\max_i(z_i + \log \pi_i)] \tag{2.95}$$

其中“one_hot”是一个将标量转换成独热编码矢量的操作。然而，如上所述，$\arg\max$ 操作通常是不可微的。因此，在耿贝尔-Softmax 函数技巧中，一个 Softmax 操作被用来对耿贝尔-最大化技巧中的 $\arg\max$ 进行连续性近似：

$$a_i = \frac{\exp((\log \pi_i + g_i)/\tau)}{\sum_j \exp((\log \pi_j + g_j)/\tau}, \forall i = 0, \cdots, k \tag{2.96}$$

其中 k 是欲求变量 $\boldsymbol{a}$（强化学习策略的动作选择）的维度，而 g_i 是采样自耿贝尔分布（Gumbel Distribution）的耿贝尔（Gumbel）变量。耿贝尔 $(0, 1)$ 分布可以用逆变换（Inverse Transform）采样实现，通过采样均匀分布 $u \sim \text{Uniform}(0, 1)$ 并计算 $g = -\log(-\log(u))$ 得到。

对角高斯策略（Diagonal Gaussian Policy）输出一个对角高斯分布的均值和方差用于连续动作空间。一个普通的多变量高斯分布包括一个均值矢量 $\boldsymbol{\mu}$ 和一个协方差（Covariance）矩阵 $\boldsymbol{\Sigma}$，而对角高斯分布是其特殊情况，即协方差矩阵只有对角元非零，因此我们可以用一个矢量 $\boldsymbol{\sigma}$ 来表示它。当使用对角高斯分布来表示概率性动作时，它移除了不同动作维度间的协相关性。一个策略被参数化时，如下所示的**再参数化**（Reparametrization）技巧（与 Kingma et al. (2014) 提出的变分自动编码器中类似）可以被用来从均值和方差矢量表示的高斯分布中采样，同时保持操作的可微性。

再参数化技巧：从对角高斯分布中采样动作 $a \sim \mathcal{N}(\boldsymbol{\mu}_\theta, \boldsymbol{\sigma}_\theta)$，该分布的均值和方差矢量为 $\boldsymbol{\mu}_\theta$ 和 $\boldsymbol{\sigma}_\theta$（参数化的），而这可以通过从正态分布中采样一个隐藏矢量 $\boldsymbol{z} \sim \mathcal{N}(0, \boldsymbol{I})$ 来得到动作：

$$a = \boldsymbol{\mu}_\theta + \boldsymbol{\sigma}_\theta \odot \boldsymbol{z} \tag{2.97}$$

其中 $\odot$ 是两个相同形状矢量的逐个元素乘积。

深度强化学习中的常用策略如图 2.19 所示，便于读者理解。

基于策略的优化（Policy-Based Optimization）方法在强化学习情景下直接优化智能体的策略而不估计或学习动作价值函数。采样得到的奖励值通常用于改进动作选择的优化过程，而优化过程可以使用基于梯度或无梯度（Gradient-Free）的方法。其中，基于梯度的方法通常采用策略梯

度（Policy Gradient），它在某种程度上代表了连续动作强化学习最受欢迎的一类算法，受益于对高维情况的可扩展性。典型的基于梯度优化方法包括 REINFORCE 等。无梯度方法对策略搜索中相对简单的情况通常有更快的学习过程，无须有复杂计算的求导过程。典型的无梯度类方法包括交叉熵（Cross-Entropy，CE）方法等。

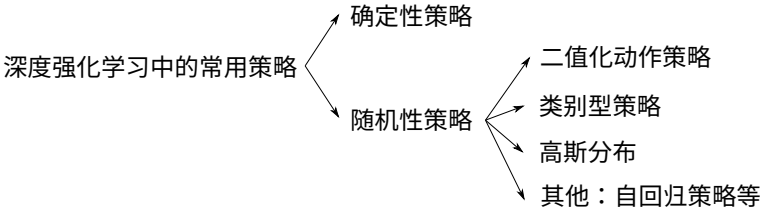


图 2.19 深度强化学习中的不同策略类型

回想我们在强化学习中智能体的目标是从期望或估计的角度去最大化从一个状态开始的累计折扣奖励（Cumulative Discounted Reward），可以将其表示为

$$J(\pi) = \mathbb{E}_{\tau\sim\pi}[R(\tau)] \tag{2.98}$$

其中 $R(\tau) = \sum_{t=0}^{\mathrm{T}} \gamma^t R_t$ 是有限步（适用于多数情形）的折扣期望奖励，而 τ 是采样的轨迹。

基于策略的优化方法将根据以上目标函数 $J(\pi)$ 通过基于梯度的或无梯度的方法，来优化策略 π。我们将首先介绍基于梯度的方法，并给出一个 REINFORCE 法的例子，随后介绍无梯度的算法和 CE 方法的例子。

基于梯度的优化

基于梯度的优化方法是使用在期望回报（总的奖励）上的梯度估计来进行梯度下降（或上升），以改进策略，而这个期望回报是从采样轨迹中得到的。这里我们把关于策略参数的梯度叫作**策略梯度**（Policy Gradient），具体表达式如下：

$$\Delta\theta = \alpha\nabla_\theta J(\pi_\theta) \tag{2.99}$$

其中 θ 表示策略参数，而 α 是学习率。基于策略参数的梯度计算方法叫作策略梯度法。文献 (Sutton et al., 2000) 和文献 (Silver et al., 2014) 提出的**策略梯度定理**（Policy Gradient Theorem）及其证明将在下面介绍。

注：式 (2.99) 中参数 θ 的表示方法实际上是不合适的，根据本书默认的格式，它应当是 $\boldsymbol{\theta}$ 从而表示矢量。然而，这里我们使用基本的 θ 格式作为一种可以在使用模型参数时替代的 $\boldsymbol{\theta}$ 的方式，而这种简单的写法也在文献中常见。一种考虑这种写法合理性的方式是：参数的梯度可以对

每个参数分别得到，而每个参数均可单独表示为 θ，只要方程对所有参数相同，它就可以用 θ 来表示所有参数。本书的其余章节将遵循以上声明。

定理 2.2 策略梯度定理

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\mathrm{T}} \nabla_\theta(\log\pi_\theta(A_t|S_t))Q^{\pi_\theta}(S_t,A_t)\right] \tag{2.100}$$

$$= \mathbb{E}_{S_t\sim\rho^\pi, A_t\sim\pi_\theta}[\nabla_\theta(\log\pi_\theta(A_t|S_t))Q^{\pi_\theta}(S_t,A_t)] \tag{2.101}$$

其中第二项需定义折扣状态分布（Discounted State Distribution）$\rho^\pi(s') := \int_{\mathcal{S}}\sum_{t=0}^{\mathrm{T}}\gamma^{t-1}\ \rho_0(s)\ p(s'|s,t,\pi)\mathrm{d}s$，而 $p(s'|s,t,\pi)$ 是在策略 π 下第 t 个时间步从 s 到 s' 的转移概率（Transition Probability），参见文献 (Silver et al., 2014)。

策略梯度定理对随机性策略和确定性策略都适用。它起初由 Sutton 等人 (Sutton et al., 2000) 为随机性策略而提出，后被 Silver 等人 (Silver et al., 2014) 扩展到确定性策略。对确定性的情况，尽管确定性策略梯度（Deterministic Policy Gradient，DPG）定理（后续介绍）与上述策略梯度定理看起来不同，实际上可以证明确定性策略梯度只是随机性策略梯度（Stochastic Policy Gradient，SPG）的一种特殊（极限）情况。若用一个确定性策略 $\mu_\theta : \mathcal{S}\to\mathcal{A}$ 和一个方差参数 σ 来参数化随机性策略 $\pi_{\mu_\theta,\sigma}$，则有 $\sigma = 0$ 时随机性策略等价于确定性策略，即 $\pi_{\mu_\theta,0}\equiv\mu$。

(1) 随机性策略梯度

首先我们对随机性策略证明策略梯度定理，因而被称为随机性策略梯度方法。为了简便，在本小节中，我们假设有限 MDP 下的片段式（Episodic）设定，每个轨迹长度固定为 $T+1$。考虑一个参数化的随机性策略 $\pi_\theta(a|s)$，对以 $\rho_0(S_0)$ 为初始状态分布的 MDP 过程，有轨迹的概率为 $p(\tau|\pi) = \rho_0(S_0)\prod_{t=0}^{\mathrm{T}} p(S_{t+1}|S_t,A_t)\pi(A_t|S_t)$，因而可以得到基于参数化策略 π_θ 的轨迹概率的对数（Logarithm）为

$$\log p(\tau|\theta) = \log\rho_0(S_0) + \sum_{t=0}^{\mathrm{T}}\Big(\log p(S_{t+1}|S_t,A_t) + \log\pi_\theta(A_t|S_t)\Big). \tag{2.102}$$

我们也需要**对数-导数技巧**（Log-Derivative Trick）：$\nabla_\theta p(\tau|\theta) = p(\tau|\theta)\nabla_\theta\log p(\tau|\theta)$ 得到轨迹概率对数（Log-Probability）的导数为

$$\nabla_\theta\log p(\tau|\theta) = \nabla_\theta\log\rho_0(S_0) + \sum_{t=0}^{\mathrm{T}}\Big(\nabla_\theta\log p(S_{t+1}|S_t,A_t) + \nabla_\theta\log\pi_\theta(A_t|S_t)\Big) \tag{2.103}$$

$$= \sum_{t=0}^{\mathrm{T}}\nabla_\theta\log\pi_\theta(A_t|S_t). \tag{2.104}$$

其中包含 $\rho_0(S_0)$ 和 $p(S_{t+1}|S_t, A_t)$ 的项被移除，因为它们不依赖于参数 θ，尽管是未知的。

回想之前介绍过，学习目标是最大化期望累计奖励（Expected Cumulative Reward）：

$$J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[R(\tau)\right] = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\mathrm{T}} R_t\right] = \sum_{t=0}^{\mathrm{T}} \mathbb{E}_{\tau\sim\pi_\theta}\left[R_t\right], \tag{2.105}$$

其中 $\tau = (S_0, A_0, R_0, \cdots, S_T, A_T, R_T, S_{T+1})$ 且 $R(\tau) = \sum_{t=0}^{\mathrm{T}} R_t$。我们可以直接在策略参数 θ 上进行梯度上升来逐渐改进策略 π_θ 的表现。

注意 R_t 只依赖 τ_t，其中 $\tau_t = (S_0, A_0, R_0, \cdots, S_t, A_t, R_t, S_{t+1})$。

$$\begin{aligned}
\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}\left[R_t\right] &= \nabla_\theta \int_{\tau_t} R_t p(\tau_t|\theta)\mathrm{d}\tau_t && \text{展开期望} && (2.106)\\
&= \int_{\tau_t} R_t \nabla_\theta p(\tau_t|\theta)\mathrm{d}\tau_t && \text{对换梯度和积分} && (2.107)\\
&= \int_{\tau_t} R_t p(\tau_t|\theta)\nabla_\theta \log p(\tau_t|\theta)\mathrm{d}\tau_t && \text{对数-导数技巧} && (2.108)\\
&= \mathbb{E}_{\tau\sim\pi_\theta}\left[R_t \nabla_\theta \log p(\tau_t|\theta)\right] && \text{回归期望形式} && (2.109)
\end{aligned}$$

上面第三个等式是根据之前介绍的对数-导数技巧得到的。

将上面式子代入到 $J(\pi_\theta)$，

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\mathrm{T}} R_t \nabla_\theta \log p(\tau_t|\theta)\right].$$

现在我们需要计算 $\nabla_\theta \log p_\theta(\tau_t)$，其中 $p_\theta(\tau_t)$ 依赖于策略 π_θ 和模型 $p(R_t, S_{t+1}|S_t, A_t)$ 的真实值，而该模型对智能体是不可用的。幸运的是，为了使用策略梯度方法，我们只需要 $\log p_\theta(\tau_t)$ 的梯度而不是它本身的值，而这可以简单地用 $\tau_t = \tau_{0:t}$ 替换式 (2.104) 中的 $\tau = \tau_{0:T}$ 而得到下式：

$$\nabla_\theta \log p(\tau_t|\theta) = \sum_{t'=0}^{t} \nabla_\theta \log \pi_\theta(A_{t'}|S_{t'}). \tag{2.110}$$

从而

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\mathrm{T}} R_t \nabla_\theta \sum_{t'=0}^{t} \log \pi_\theta(A_{t'}|\, S_{t'})\right]$$

$$= \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t'=0}^{\mathrm{T}} \nabla_\theta \log \pi_\theta(A_{t'}|\, S_{t'}) \sum_{t=t'}^{\mathrm{T}} R_t\right]. \tag{2.111}$$

这里最后一个等式是加法重排（Rearranging the Summation）。

注意，我们在以上推导过程中使用了加法和期望之间的置换，以及期望和加法与求导之间的置换（都是合理的），如下：

$$\nabla_\theta J(\pi_\theta) = \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}\left[R(\tau)\right] = \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\mathrm{T}} R_t\right] = \sum_{t=0}^{\mathrm{T}} \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}\left[R_t\right] \tag{2.112}$$

其最终在式 (2.106) 中对长度为 $t+1$ 的部分轨迹 τ_t 进行积分。然而，也有其他方式来对整个轨迹的累计奖励取期望：

$$\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta} R(\tau) && && (2.113)\\
&= \nabla_\theta \int_\tau p(\tau|\theta) R(\tau) && \text{展开期望} && (2.114)\\
&= \int_\tau \nabla_\theta p(\tau|\theta) R(\tau) && \text{对换梯度和积分} && (2.115)\\
&= \int_\tau p(\tau|\theta) \nabla_\theta \log p(\tau|\theta) R(\tau) && \text{对数-导数技巧} && (2.116)\\
&= \mathbb{E}_{\tau\sim\pi_\theta}[\nabla_\theta \log p(\tau|\theta) R(\tau)] && \text{回归期望形式} && (2.117)\\
\Rightarrow \nabla_\theta J(\pi_\theta) &= \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\mathrm{T}} \nabla_\theta \log \pi_\theta(A_t|S_t) R(\tau)\right] && && (2.118)\\
&= \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\mathrm{T}} \nabla_\theta \log \pi_\theta(A_t|S_t) \sum_{t'=0}^{\mathrm{T}} R_{t'}\right] && && (2.119)
\end{aligned}$$

仔细的读者可能注意到式 (2.119) 的第二个结果与式 (2.111) 的第一个结果有一些差别。具体来说，累计奖励的时间范围是不同的。第一个结果只使用了动作 A_t 之后的累计未来奖励 $\sum_{t=t'}^{\mathrm{T}} R_t$ 来评估动作，而第二个结果使用整个轨迹上的累计奖励 $\sum_{t=0}^{\mathrm{T}} R_t$ 来评估该轨迹上的每个动作 A_t，包括选择那个动作之前的奖励。直觉上，一个动作不应该用这个动作执行以前的奖励值来对其进行估计，而这也得到数学上的证明，即这个动作之前的奖励对最终期望梯度只有零影响。因此可以在推导策略梯度的过程中直接丢掉那些过去的奖励值来得到式 (2.111)，而这被称为“将得到的奖励（Reward-to-Go）”策略梯度。这里我们不给出两种策略梯度公式等价性的严格证明，感兴趣的读者可以参考相关资料。这里的两种导出方式也可以作为两个结果等价性的论证。

上述公式中的 ∇ 称“nabla”，它是一个物理和数学领域有着三重意义的算子（梯度、散度、

和旋度），依据它做操作的对象而定。而在计算机领域，这个“nabla”算子 ∇ 通常用作偏微分（Partial Derivative），其对紧跟的对象中显式（Explicitly）包含的变量进行求导，而这个变量写在算子脚标的位置。由于上式中的 $R(\tau)$ 不显式包含 θ，因此 ∇_θ 不作用于 $R(\tau)$，尽管 τ 可以隐式（Implicitly）依赖于 θ（根据 MDP 的图模型）。我们也注意到式 (2.119) 的期望可以用采样均值来估计。如果我们收集一个轨迹的集合 $\mathcal{D} = \{\tau_i\}_{i=1,\cdots,N}$，而其中的轨迹是通过使智能体以策略 π_θ 在环境中做出动作来得到的，那么策略梯度可以用以下方式估计：

$$\hat{g} = \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{\mathrm{T}} \nabla_\theta \log \pi_\theta(A_t|S_t) R(\tau), \tag{2.120}$$

EGLP（Expected Grad-Log-Prob）引理在策略梯度优化中经常用到，所以我们在这里介绍它。

引理 2.2 EGLP 引理：*假设 p_θ 是随机变量 x 的一个参数化的概率分布，那么有*

$$\mathbb{E}_{x \sim p_\theta}[\nabla_\theta \log P_\theta(x)] = 0. \tag{2.121}$$

证明: 由于所有概率分布都是归一化的：

$$\int_x p_\theta(x) = 1. \tag{2.122}$$

对上面归一化条件两边取梯度：

$$\nabla_\theta \int_x p_\theta(x) = \nabla_\theta 1 = 0. \tag{2.123}$$

使用对数-导数技巧得到：

$$\begin{aligned} 0 &= \nabla_\theta \int_x p_\theta(x) && (2.124)\\ &= \int_x \nabla_\theta p_\theta(x) && (2.125)\\ &= \int_x p_\theta(x) \nabla_\theta \log p_\theta(x) && (2.126)\\ \therefore 0 &= \mathbb{E}_{x \sim p_\theta}[\nabla_\theta \log p_\theta(x)]. && (2.127) \end{aligned}$$

从 EGLP 引理我们可以直接得出：

$$\mathbb{E}_{A_t \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(A_t|S_t) b(S_t)] = 0. \tag{2.128}$$

其中 $b(S_t)$ 称为基准（Baseline），而它是独立于用于求期望值的未来轨迹的。基准可以是任何一

个只依赖当前状态的函数，而不影响优化公式中的总期望值。

上面公式中的优化目标最终为

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\mathrm{T}} \nabla_\theta \log \pi_\theta(A_t|S_t) R(\tau)\right] \tag{2.129}$$

我们也可以更改整个轨迹的奖励 $R(\tau)$ 为在 t 时间步后将得到的奖励 G_t：

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\mathrm{T}} \nabla_\theta \log \pi_\theta(A_t|S_t) G_t\right] \tag{2.130}$$

通过以上 EGLP 引理，期望回报可以被推广为

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\mathrm{T}} \nabla_\theta \log \pi_\theta(A_t|S_t) \Phi_t\right] \tag{2.131}$$

其中 $\Phi_t = \sum_{t'=t}^{\mathrm{T}}(R(S_{t'}, a_{t'}, S_{t'+1}) - b(S_t))$。

为了便于实际使用，Φ_t 可以变成以下形式：

$$\Phi_t = Q^{\pi_\theta}(S_t, A_t) \tag{2.132}$$

或

$$\Phi_t = A^{\pi_\theta}(S_t, A_t) = Q^{\pi_\theta}(S_t, A_t) - V^{\pi_\theta}(S_t) \tag{2.133}$$

而它们都可以证明等价于期望内的原始形式，只是在实际中有不同的方差。这些证明需要重复期望规则（the Law of Iterated Expectations）：对两个随机变量（离散或连续）有 $\mathbb{E}[X] = \mathbb{E}[\mathbb{E}[X|Y]]$。而这个式子很容易证明。剩余的证明如下：

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\mathrm{T}} \nabla_\theta \log \pi_\theta(A_t|S_t) R(\tau)\right] \tag{2.134}$$

$$= \sum_{t=0}^{\mathrm{T}} \mathbb{E}_{\tau\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(A_t|S_t) R(\tau)] \tag{2.135}$$

$$= \sum_{t=0}^{\mathrm{T}} \mathbb{E}_{\tau_{:t}\sim\pi_\theta}[\mathbb{E}_{\tau_{t:}\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(A_t|S_t) R(\tau)|\tau_{:t}]] \tag{2.136}$$

$$= \sum_{t=0}^{\mathrm{T}} \mathbb{E}_{\tau_{:t}\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(A_t|S_t)\mathbb{E}_{\tau_{t:}\sim\pi_\theta}[R(\tau)|\tau_{:t}]] \tag{2.137}$$

$$= \sum_{t=0}^{\mathrm{T}} \mathbb{E}_{\tau_{:t}\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(A_t|S_t)\mathbb{E}_{\tau_{t:}\sim\pi_\theta}[R(\tau)|S_t, A_t]] \tag{2.138}$$

$$= \sum_{t=0}^{\mathrm{T}} \mathbb{E}_{\tau_{:t}\sim\pi_\theta}[\nabla_\theta(\log \pi_\theta(A_t|S_t))Q^{\pi_\theta}(S_t, A_t)] \tag{2.139}$$

其中 $\mathbb{E}_\tau[\cdot] = \mathbb{E}_{\tau_{:t}}[\mathbb{E}_{\tau_{t:}}[\cdot|\tau_{:t}]]$ 且 $\tau_{:t} = (S_0, A_0, \cdots, S_t, A_t)$ 和 $Q^{\pi_\theta}(S_t, A_t) = \mathbb{E}_{\tau_{t:}\sim\pi_\theta}[R(\tau)|\ S_t, A_t]$。

所以，文献中有常见的形式：

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\mathrm{T}} \nabla_\theta(\log \pi_\theta(A_t|S_t))Q^{\pi_\theta}(S_t, A_t)\right] \tag{2.140}$$

或

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\mathrm{T}} \nabla_\theta(\log \pi_\theta(A_t|S_t))A^{\pi_\theta}(S_t, A_t)\right] \tag{2.141}$$

换句话说，它等价于改变优化目标为 $J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi}[Q^{\pi_\theta}(S_t, A_t)]$ 或 $J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi}[A^{\pi_\theta}(S_t, A_t)]$ 来替换原始形式 $\mathbb{E}_{\tau\sim\pi}[R(\tau)]$。对于优化策略来说，实践中常用 $A^{\pi_\theta}(S_t, A_t)$ 来估计 TD-误差（TD Error）。

根据是否使用环境模型，强化学习算法可以被分为无模型（Model-Free）和基于模型的（Model-Based）两类。对于无模型强化学习，单纯基于梯度的优化算法可以追溯至 REINFORCE 算法，或称策略梯度算法。而基于模型的强化学习算法一类，也有一些基于策略的算法，比如使用贯穿时间的反向传播（Backpropagation Through Time，BPTT）来根据一个片段内的奖励去更新策略。

例子：REINFORCE 算法

REINFORCE 是一个使用式 (2.131) 的随机性策略梯度方法的算法，其中 $\Phi_t = Q^\pi(S_t, A_t)$，而在 REINFORCE 中，它通常用轨迹上采样的奖励值 $G_t = \sum_{t'=t}^{\infty} R_{t'}$（或折扣版本 $G_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} R_{t'}$）来估计。更新策略的梯度为

$$g = \mathbb{E}\left[\sum_{t=0}^{\infty}\sum_{t'=t}^{\infty} R_{t'}\nabla_\theta \log \pi_\theta(A_t|S_t)\right] \tag{2.142}$$

(2) 确定性策略梯度

以上介绍的属于随机性策略梯度（Stochastic Policy Gradient, SPG），它用于优化随机性策略 $\pi(a|s)$，即用一个基于当前状态的概率分布来表示动作的情况。与随机性策略相对的是确定性策略，其中 $a=\pi(s)$ 是一个确定性动作而非概率分布。我们可以用类似于 SPG 的方法得到 DPG，且它在数值上（作为一种极限情况）遵循策略梯度定理，尽管有不同的显式表示。

注：在本小节后续部分，我们使用 $\mu(s)$ 代替之前定义的 $\pi(s)$ 来表示确定性策略，从而消除它与随机性策略 $\pi(a|s)$ 间的歧义。

对于 DGP 的更严格和广泛的定义，我们参考由文献 (Silver et al., 2014) 提出的确定性策略梯度定理，即式 (2.151)。在此之前，我们将逐步介绍确定性策略梯度定理并证明它，先用一种在线策略的方式而后用离线策略的方式，同时我们也将详细讨论 DPG 和 SPG 间的关系。

首先，我们定义确定性策略的表现目标，与随机性策略梯度求解过程中的期望折扣奖励采用同样的定义：

$$J(\mu)=\mathbb{E}_{S_t\sim\rho^\mu,A_t=\mu(S_t)}\left[\sum_{t=1}^{\infty}\gamma^{t-1}R(S_t,A_t)\right] \tag{2.143}$$

$$=\int_{\mathcal{S}}\int_{\mathcal{S}}\sum_{t=1}^{\infty}\gamma^{t-1}\rho_0(s)p(s'|s,t,\mu)R(s',\mu(s'))\mathrm{d}s\mathrm{d}s' \tag{2.144}$$

$$=\int_{\mathcal{S}}\rho^\mu(s)R(s,\mu(s))\mathrm{d}s \tag{2.145}$$

其中 $p(s'|s,t,\mu)=p(S_{t+1}|S_t,A_t)p^\mu(A_t|S_t)$，第一个概率是转移概率，而第二个是动作选择概率。由于它是确定性策略，我们有 $p^\mu(A_t|S_t)=1$，因而 $p(s'|s,t,\mu)=p(S_{t+1}|S_t,\mu(S_t))$。此外，上式中的状态分布是 $\rho^\mu(s'):=\int_{\mathcal{S}}\sum_{t=1}^{\infty}\gamma^{t-1}\rho_0(s)p(s'|s,t,\mu)\mathrm{d}s$。

由于式子 $V^\mu(s)=\mathbb{E}\left[\sum_{t=1}^{\infty}\gamma^{t-1}R(S_t,A_t)|S_1=s;\mu\right]=\int_{\mathcal{S}}\sum_{t=1}^{\infty}\gamma^{t-1}p(s'|s,t,\mu)R\left(s',\mu(s')\right]\mathrm{d}s'$ 在除使用确定性策略这一点外遵循与随机性策略梯度中相同的定义，我们可以得出

$$J(\mu)=\int_{\mathcal{S}}\rho_0(s)V^\mu(s)\mathrm{d}s \tag{2.146}$$

$$=\int_{\mathcal{S}}\int_{\mathcal{S}}\sum_{t=1}^{\infty}\gamma^{t-1}\rho_0(s)p(s'|s,t,\mu)R(s',\mu(s'))\mathrm{d}s\mathrm{d}s' \tag{2.147}$$

这与上面直接用折扣奖励的形式得到的表示式是等价的。这里的关系也对随机性策略梯度适用，只是将确定性策略 $\mu(s)$ 替换成随机性策略 $\pi(a|s)$ 即可。对于确定性策略，我们有 $V^\mu(s)=Q^\mu(s,\mu(s))$，因为状态价值对于随机性策略是关于动作分布的期望，而对于确定性策略没有动作分布而只有单个动作值。因此我们也有对于确定性策略的如下表示：

$$J(\mu) = \int_{\mathcal{S}} \rho_0(s) V^{\mu}(s) \mathrm{d}s \tag{2.148}$$

$$= \int_{\mathcal{S}} \rho_0(s) Q^{\mu}(s, \mu(s)) \mathrm{d}s \tag{2.149}$$

关于表现目标的不同形式和几个条件将被用来证明 DPG 定理。我们在这里列出这些条件如下而不给出详细的推导过程，相关内容请参考文献 (Silver et al., 2014)。

- **C.1 连续导数的存在性：** $p(s'|s,a), \nabla_a p(s'|s,a), \mu_\theta(s), \nabla_\theta \mu_\theta(s), R(s,a), \nabla_a R(s,a), \rho_0(s)$ 对所有参数和变量 s, a, s' 和 x 连续。

- **C.2 有界性条件：** 存在 a, b 和 L 使得 $\sup_s \rho_0(s) < b, \sup_{a,s,s'} p(s'|s,a) < b, \sup_{a,s} R(s,a) < b, \sup_{a,s,s'} \|\nabla_a p(s'|s,a)\| < L, \sup_{a,s} \|\nabla_a R(s,a)\| < L$。

定理 2.3 确定性策略梯度定理： 假设 MDP 满足条件 C.1，即连续的 $\nabla_\theta \mu_\theta(s), \nabla_a Q^{\mu}(s,a)$ 和确定性策略梯度的存在性，那么

$$\nabla_\theta J(\mu_\theta) = \int_{\mathcal{S}} \rho^{\mu}(s) \nabla_\theta \mu_\theta(s) \nabla_a Q^{\mu}(s,a)|_{a=\mu_\theta(s)} \mathrm{d}s \tag{2.150}$$

$$= \mathbb{E}_{s\sim\rho^{\mu}}[\nabla_\theta \mu_\theta(s) \nabla_a Q^{\mu}(s,a)|_{a=\mu_\theta(s)}] \tag{2.151}$$

证明: 确定性策略梯度定理的证明基本遵循与文献 (Sutton et al., 2000) 的标准随机性策略梯度定理一样的步骤。首先，为了方便在后续证明中交换导数和积分，以及积分的顺序，我们需要使用两个引理，它们是微积分里的基本数学公式，如下：

引理 2.3 莱布尼茨积分法则（Leibniz Integral Rule）：$f(x,t)$ 是一个使得 $f(x,t)$ 及其偏导数 $f'_x(x,t)$ 在 (x,t)-平面的部分区域上对 t 和 x 连续的函数，包括 $a(x) \leqslant t \leqslant b(x), x_0 \leqslant x \leqslant x_1$。同时假设函数 $a(x)$ 和 $b(x)$ 都是连续的且在 $x_0 \leqslant x \leqslant x_1$ 上有连续导数。那么，对于 $x_0 \leqslant x \leqslant x_1$，

$$\frac{\mathrm{d}}{\mathrm{d}x} \int_{a(x)}^{b(x)} f(x,t) \mathrm{d}t = f(x, b(x)) \cdot \frac{\mathrm{d}}{\mathrm{d}x} b(x) - f(x, a(x)) \cdot \frac{\mathrm{d}}{\mathrm{d}x} a(x) + \int_{a(x)}^{b(x)} \frac{\partial}{\partial x} f(x,t) \mathrm{d}t \tag{2.152}$$

引理 2.4 富比尼定理（Fubini's Theorem）：假设 $\mathcal{X}$ 和 $\mathcal{Y}$ 是 σ-有限测度空间（σ-Finite Measure Space），并且假设 $\mathcal{X} \times \mathcal{Y}$ 由积测度（Product Measure）给出（由于 $\mathcal{X}$ 和 $\mathcal{Y}$ 是 σ-有限的，这个测度是唯一的）。富比尼定理声明：如果 f 是 $\mathcal{X} \times \mathcal{Y}$ 可积的，那么 f 是一个可测函数（Measurable Function）且有

$$\int_{\mathcal{X}\times\mathcal{Y}} |f(x,y)| \mathrm{d}(x,y) < \infty \tag{2.153}$$

那么

$$\int_{\mathcal{X}}(\int_{\mathcal{Y}} f(x,y)\mathrm{d}y)\mathrm{d}x = \int_{\mathcal{Y}}(\int_{\mathcal{X}} f(x,y)\mathrm{d}x)\mathrm{d}y = \int_{\mathcal{X}\times\mathcal{Y}} f(x,y)\mathrm{d}(x,y) \tag{2.154}$$

为了满足以上两个引理，我们需要 C.1 所提供的充分条件作为莱布尼茨积分法则的要求，即 $V^{\mu_\theta}(s)$ 和 $\nabla_\theta V^{\mu_\theta}(s)$ 是 θ 和 s 的连续函数。我们也遵循状态空间 $\mathcal{S}$ 紧致性（Compactness）的假设，如富比尼定理所要求，即对于任何 θ，$\|\nabla_\theta V^{\mu_\theta}(s)\|$，$\|\nabla_a Q^{\mu_\theta}(s,a)\,|_{a=\mu_\theta(s)}\|$ 和 $\|\nabla_\theta \mu_\theta(s)\|$ 是 s 的有界（Bounded）函数，而这在 C.2 中提供。有了以上条件，我们可以得到以下推导：

$$\begin{aligned}
\nabla_\theta V^{\mu_\theta}(s) &= \nabla_\theta Q^{\mu_\theta}(s,\mu_\theta(s)) \\
&= \nabla_\theta (R(s,\mu_\theta(s)) + \int_{\mathcal{S}} \gamma p(s'|s,\mu_\theta(s)) V^{\mu_\theta}(s')\mathrm{d}s') \\
&= \nabla_\theta \mu_\theta(s) \nabla_a R(s,a)|_{a=\mu_\theta(s)} + \nabla_\theta \int_{\mathcal{S}} \gamma p(s'|s,\mu_\theta(s)) V^{\mu_\theta}(s')\mathrm{d}s' \\
&= \nabla_\theta \mu_\theta(s) \nabla_a R(s,a)|_{a=\mu_\theta(s)} + \int_{\mathcal{S}} \gamma (p(s'|s,\mu_\theta(s)) \nabla_\theta V^{\mu_\theta}(s') \\
&\quad + \nabla_\theta \mu_\theta(s) \nabla_a p(s'|s,a) V^{\mu_\theta}(s'))\mathrm{d}s' \\
&= \nabla_\theta \mu_\theta(s) \nabla_a (R(s,a) + \int_{\mathcal{S}} \gamma p(s'|s,a) V^{\mu_\theta}(s')\mathrm{d}s')|_{a=\mu_\theta(s)}) \\
&\quad + \int_{\mathcal{S}} \gamma p(s'|s,\mu_\theta(s)) \nabla_\theta V^{\mu_\theta}(s')\mathrm{d}s' \\
&= \nabla_\theta \mu_\theta(s) \nabla_a Q^{\mu_\theta}(s,a)|_{a=\mu_\theta(s)} + \int_{\mathcal{S}} \gamma p(s'|s,\mu_\theta(s)) \nabla_\theta V^{\mu_\theta}(s')\mathrm{d}s'
\end{aligned} \tag{2.155}$$

在上面的推导中，莱布尼茨积分法则被用于交换求导和积分的顺序，这要求满足 $p(s'|s,a)$，$\mu_\theta(s)$，$V^{\mu_\theta}(s)$ 和它们的导数对 θ 的连续性条件。现在我们用 $\nabla_\theta V^{\mu_\theta}(s)$ 对以上公式进行迭代，得到：

$$\begin{aligned}
\nabla_\theta V^{\mu_\theta}(s) &= \nabla_\theta \mu_\theta(s) \nabla_a Q^{\mu_\theta}(s,a)|_{a=\mu_\theta(s)} \\
&\quad + \int_{\mathcal{S}} \gamma p(s'|s,\mu_\theta(s)) \nabla_\theta \mu_\theta(s') \nabla_a Q^{\mu_\theta}(s',a)|_{a=\mu_\theta(s')}\mathrm{d}s' \\
&\quad + \int_{\mathcal{S}} \gamma p(s'|s,\mu_\theta(s)) \int_{\mathcal{S}} \gamma p(s''|s',\mu_\theta(s')) \nabla_\theta V^{\mu_\theta}(s'')\mathrm{d}s''\mathrm{d}s' \\
&= \nabla_\theta \mu_\theta(s) \nabla_a Q^{\mu_\theta}(s,a)|_{a=\mu_\theta(s)} \\
&\quad + \int_{\mathcal{S}} \gamma p(s \to s', 1, \mu_\theta(s)) \nabla_\theta \mu_\theta(s') \nabla_a Q^{\mu_\theta}(s',a)|_{a=\mu_\theta(s')}\mathrm{d}s' \\
&\quad + \int_{\mathcal{S}} \gamma^2 p(s \to s', 2, \mu_\theta(s)) \nabla_\theta \mu_\theta(s') \nabla_a Q^{\mu_\theta}(s',a)|_{a=\mu_\theta(s')}\mathrm{d}s' \\
&\quad + \cdots
\end{aligned}$$

$$= \int_{\mathcal{S}} \sum_{t=0}^{\infty} \gamma^t p(s \to s', t, \mu_\theta(s)) \nabla_\theta \mu_\theta(s') \nabla_a Q^{\mu_\theta}(s', a)|_{a=\mu_\theta(s')} \mathrm{d}s' \tag{2.156}$$

其中，我们使用富比尼定理来交换积分顺序，而这要求 $\|\nabla_\theta V^{\mu_\theta}(s)\|$ 的有界性条件。上述积分中包含一种特殊情况，对 $s'=s$ 有 $p(s \to s', 0, \mu_\theta(s)) = 1$ 而对其他 s' 为 0。现在我们对修改过的性能目标即期望价值函数进行求导：

$$\begin{aligned}
\nabla_\theta J(\mu_\theta) &= \nabla_\theta \int_{\mathcal{S}} \rho_0(s) V^{\mu_\theta}(s) \mathrm{d}s \\
&= \int_{\mathcal{S}} \rho_0(s) \nabla_\theta V^{\mu_\theta}(s) \mathrm{d}s \\
&= \int_{\mathcal{S}} \int_{\mathcal{S}} \sum_{t=0}^{\infty} \gamma^t \rho_0(s) p(s \to s', t, \mu_\theta(s)) \nabla_\theta \mu_\theta(s') \nabla_a Q^{\mu_\theta}(s', a)|_{a=\mu_\theta(s')} \mathrm{d}s' \mathrm{d}s \\
&= \int_{\mathcal{S}} \rho^{\mu_\theta}(s) \nabla_\theta \mu_\theta(s) \nabla_a Q^{\mu_\theta}(s, a)|_{a=\mu_\theta(s)} \mathrm{d}s
\end{aligned} \tag{2.157}$$

其中我们使用莱布尼茨积分法则来交换求导和积分顺序，需要满足 $\rho_0(s)$ 和 $V^{\mu_\theta}(s)$ 及其导数对 θ 连续的条件，同样由富比尼定理交换积分顺序，需要满足被积函数（Integrand）的有界性条件。证毕。

离线策略确定性策略梯度

除了上面在线策略版本的确定性策略梯度（DPG）推导，我们也可以用离线策略的方式来导出 DPG，使用上面的 DPG 定理和 γ-折扣状态分布 $\rho^\mu(s') := \int_{\mathcal{S}} \sum_{t=1}^{\infty} \gamma^{t-1} p(s) p\,(s'|s, t, \mu) \mathrm{d}s$。离线策略确定性策略梯度用行为策略（Behaviour Policy，即使用经验回放池时的先前策略）的样本来估计当前策略，而这个策略可能跟当前策略不同。在离线策略的设定下，由一个独特的行为策略 $\beta(s) \neq \mu_\theta(s)$ 所采集轨迹来对梯度进行估计，相应的状态分布为 $\rho^\beta(s)$，这不依赖于策略参数 θ。在离线策略情况下，性能目标被修改为目标策略的价值函数在行为策略的状态分布上的平均 $J_\beta(\mu_\theta) = \int_{\mathcal{S}} \rho^\beta(s) V^\mu(s) \mathrm{d}s = \int_{\mathcal{S}} \rho^\beta(s) Q^\mu(s, \mu_\theta(s)) \mathrm{d}s$，而原始的目标遵循式 (2.149)，即 $J(\mu_\theta) = \int_{\mathcal{S}} \rho_0(s) V^\mu(s) \mathrm{d}s$。注意，这里是我们在导出离线策略确定性策略梯度中进行的第一个近似，即 $J(\mu_\theta) \approx J_\beta(u_\theta)$，而我们将在后面有另外一个近似。我们可以直接对修改过的目标取微分如下：

$$\begin{aligned}
\nabla_\theta J_\beta(\mu_\theta) &= \int_{\mathcal{S}} \rho^\beta(s) (\nabla_\theta \mu_\theta(s) \nabla_a Q^{\mu_\theta}(s, a) + \nabla_\theta Q^{\mu_\theta}(s, a))|_{a=\mu(s)} \mathrm{d}s \\
&\approx \int_{\mathcal{S}} \rho^\beta(s) \nabla_\theta \mu_\theta(s) \nabla_a Q^{\mu_\theta}(s, a) \mathrm{d}s \\
&= \mathbb{E}_{s \sim \rho^\beta} [\nabla_\theta \mu_\theta(s) \nabla_a Q^{\mu_\theta}(s, a)|_{a=\mu(s)}]
\end{aligned} \tag{2.158}$$

上面式子中的约等于（Approximately Equivalent）符号“$\approx$”表示了在线策略 DPG 和离线策略 DPG

的不同。上式中的依赖关系需要小心处理。因为 $\rho^{\beta}(s)$ 是独立于 θ 的，关于 θ 的导数可以进入积分中，并且在 $\rho^{\beta}(s)$ 上没有导数。$Q^{\mu_\theta}(s,\mu_\theta(a))$ 实际上以两种方式依赖于 θ（其表达式中有两个 μ_θ）:（1）它依赖于确定性策略 μ_θ 基于当前状态 s 所决定的动作 a，而（2）对 Q 值的在线策略估计也依赖于策略 μ_θ 来在未来状态下选择的动作，如在 $Q^{\mu_\theta}(s,a)=R(s,a)+\int_{\mathcal{S}}\gamma p(s'|s,a)V^{\mu_\theta}(s')\mathrm{d}s'$ 中所示，所以这个求导需要分别进行。然而，第一个式中的第二项 $\nabla_\theta Q^{\mu_\theta}(s,a)|_{a=\mu(s)}$ 在近似中由于对其估计的困难而被丢掉了，这在离线策略梯度中有类似的相应操作 (Degris et al., 2012)[1][2]。

随机性策略梯度和确定性策略梯度的关系

如式 (2.140) 所示，随机性策略梯度与前文策略梯度定理中公式有相同的形式，而式 (2.151) 中的确定性策略梯度看起来却有不一致的形式。然而，可以证明对于相当广泛的随机策略，DPG 是一个 SPG 的特殊（极限）情况。在这种情况下，DPG 也在一定条件下满足策略梯度定理。为了实现这一点，我们通过一个确定性策略 $\mu_\theta:\mathcal{S}\to\mathcal{A}$ 和一个方差参数 σ 来参数化随机性策略 $\pi_{\mu_\theta,\sigma}$，从而对 $\sigma=0$ 有随机性策略等价于确定性策略，即 $\pi_{\mu_\theta,0}\equiv\mu$。为了定义 SPG 和 DPG 之间的关系，有一个额外的条件需要满足，这是一个定义常规 Delta-近似（Regular Delta-Approximation）的复合条件。

- **C.3 常规 Delta-近似**：由 σ 参数化的函数 v_σ 被称为一个 $\mathcal{R}\subseteq\mathcal{A}$ 上的常规 Delta-近似，如果满足条件:（1）对于 $a'\in\mathcal{R}$ 和适当平滑的 f，v_σ 收敛到一个 Delta 分布 $\lim_{\sigma\downarrow 0}\int_{\mathcal{A}}v_\sigma(a',a)f(a)\mathrm{d}a=f(a')$;（2）$v_\sigma(a',\cdot)$ 在紧致而有利普希茨（Lipschitz）边界的 $\mathcal{C}'_a\subseteq\mathcal{A}$ 上得到支撑，而在边界上消失（Vanish）并且在 $\mathcal{C}_{a'}$ 上连续可微;（3）梯度 $\nabla_{a'}v_\sigma(a',a)$ 总是存在;（4）转移不变性：对任何 $a\in\mathcal{A},a'\in\mathcal{R},a+\delta\in\mathcal{A},a'+\delta\in\mathcal{A}$，有 $v(a',a)=v(a'+\delta,a+\delta)$。

定理 2.4　确定性策略梯度作为随机性策略梯度的极限：*考虑一个随机性策略 $\pi_{\mu_\theta,\sigma}$ 使得 $\pi_{\mu_\theta,\sigma}(a|s)=v_\sigma(\mu_\theta(s),a)$，其中 σ 是一个控制方差的参数且 $v_\sigma(\mu_\theta(s),a)$ 满足 C.3，又有 MDP 满足 C.1 和 C.2，那么有，*

$$\lim_{\sigma\downarrow 0}\nabla_\theta J(\pi_{\mu_\theta,\sigma})=\nabla_\theta J(\mu_\theta) \tag{2.159}$$

这表示 DPG 的梯度（等号右边）是标准 SPG（等号左边）的极限情况。

以上关系的证明超出了本书的范畴，我们在这里不做讨论。细节参考原文 (Silver et al., 2014)。

确定性策略梯度应用和变体

一种最著名的 DPG 算法是深度确定性策略梯度（Deep Deterministic Policy Gradient，DDPG），它是 DPG 的一个深度学习变体。DDPG 结合了 DQN 和 Actor-Critic 算法来使用确定性策略梯度并通过一种深度学习的方式更新策略。行动者（Actor）和批判者（Critic）各自有一个目标网络（Target Network）来便于高样本效率（Sample-Efficient）地学习，但是众所周知，这个算法可能

[1]关于这个操作的细节和相关论断可以参考原文。

[2]论文 *SILVER D, LEVER G, HEESS N, et al. 2014. Deterministic policy gradient algorithms[C].* 中式（15）在近似操作后的 Q 项上丢掉了 ∇_a，这里我们对其勘误。

使用起来有一定挑战性，由于它在实践中往往很脆弱而对超参数敏感 (Duan et al., 2016)。关于 DDPG 算法的细节和实现在后续章节有详细介绍。

从以上可以看到，策略梯度可以用至少两种方式估计：SPG 和 DPG，依赖于具体策略类型。实际上，它们使用了两种不同的估计器，用变分推断（Variational Inference，VI）的术语来说，SPG 是得分函数（Score Function）估计器，而 DPG 是路径导数（Pathwise Derivative）估计器。

再参数化技巧使得来自价值函数的策略梯度可以用于随机性策略，这被称为**随机价值梯度**（Stochastic Value Gradients，SVG）(Heess et al., 2015)。在 SVG 算法中，一个 λ 值通常用于 SVG(λ)，以表明贝尔曼递归被展开了多少步。举例来说，SVG(0) 和 SVG(1) 表示贝尔曼递归分别被展开 0 和 1 步，而 SVG(∞) 表示贝尔曼递归被沿着有限范围的整个片段轨迹展开。SVG(0) 是一个无模型方法，它的动作价值是用当前策略估计的，因此价值梯度被反向传播到策略中；而 SVG(1) 是一个基于模型的方法，它使用一个学得的转移模型来估计下一个状态的值，如论文 (Heess et al., 2015) 中所述。

一个非常简单但有用的再参数化技巧（Reparameterization Trick）的例子是将一个条件高斯概率密度 $p(y|x) = \mathcal{N}(\mu(x), \sigma^2(x))$ 写作函数 $y(x) = \mu(x) + \sigma(x)\epsilon, \epsilon \sim \mathcal{N}(0, 1)$。因而我们可以按程序生成样本，先采样 ϵ 再以一种确定性的方式得到 y，这使得对随机性策略的采样过程进行梯度追踪。实际上根据同样的过程也可以得到从动作价值函数到策略间的反向传播梯度。为了像 DPG 那样通过价值函数来得到随机性策略的梯度，SVG 使用了这个再参数化技巧，并且对随机噪声取了额外的期望值。柔性 Actor-Critic（Soft Actor-Critic, SAC）和原始 SVG (Heess et al., 2015) 算法都遵循这个程序，从而可以使用随机性策略进行连续控制。

比如，在 SAC 中，随机性策略被一个均值和一个方差，以及一个从正态分布（Normal Distribution）中采样的噪声项再参数化。SAC 中的优化目标有一个额外的熵相关项：

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t (R(S_t, A_t, S_{t+1}) + \alpha H(\pi(\cdot|S_t)))\right] \tag{2.160}$$

因此，价值函数和 Q 值函数间的关系变为

$$V^\pi(s) = \mathbb{E}_{a\sim\pi}[Q^\pi(s,a)] + \alpha H(\pi(\cdot|s)) \tag{2.161}$$

$$= \mathbb{E}_{a\sim\pi}[Q^\pi(s,a) - \alpha \log \pi(a|s)] \tag{2.162}$$

SAC 中使用的策略是一个 Tanh 归一化高斯分布，这与传统设置不同。SAC 中的动作表示可以使用如下再参数化技巧：

$$a_\theta(s, \epsilon) = \tanh(\mu_\theta(s) + \sigma_\theta(s) \cdot \epsilon), \epsilon \sim \mathcal{N}(0, I) \tag{2.163}$$

由于 SAC 中策略的随机性，策略梯度可以在最大化期望价值函数时使用再参数化技巧得到，即：

$$\max_{\theta} \mathbb{E}_{a\sim\pi_\theta}[Q^{\pi_\theta}(s,a) - \alpha \log \pi_\theta(a|s)] \tag{2.164}$$

$$= \max_{\theta} \mathbb{E}_{\epsilon\sim\mathcal{N}}[Q^{\pi_\theta}(s,a(s,\epsilon)) - \alpha \log \pi_\theta(a(s,\epsilon)|s)] \tag{2.165}$$

因而，梯度可以经过 Q 网络到策略网络，与 DPG 类似，即：

$$\nabla_\theta \frac{1}{|\mathcal{B}|} \sum_{S_t\in\mathcal{B}} (Q^{\pi_\theta}(S_t,a(S_t,\epsilon)) - \alpha \log \pi_\theta(a(S_t,\epsilon)|S_t)) \tag{2.166}$$

其使用一个采样批 $\mathcal{B}$ 来更新策略，而 $a(S_t,\epsilon)$ 通过再参数化技巧来从随机性策略中采样。在这种情况下，再参数化技巧使得随机性策略能够以一种类似于 DPG 的方式来更新，而所得到的 SVG 是介于 DPG 和 SPG 之间的方法。DPG 也可以被看作 SVG(0) 的一种确定性极限（Deterministic Limit）。

无梯度优化

除了基于梯度（Gradient-Based）的优化方法来实现基于策略（Policy-Based）的学习，也有非基于梯度（Non-Gradient-Based）方法，也称无梯度（Gradient-Gree）优化方法，包括交叉熵（Cross-Entropy，CE）方法、协方差矩阵自适应（Covariance Matrix Adaptation，CMA）(Hansen et al., 1996)、爬山法（Hill Climbing），Simplex / Amoeba / Nelder-Mead 算法 (Nelder et al., 1965) 等。

例子：交叉熵方法

除了对策略使用基于梯度的优化，CE 方法作为一种非基于梯度的方法，在强化学习中也常用于快速的策略搜索。在 CE 方法中，策略是迭代更新的，对参数化策略 π_θ 的参数 θ 的优化目标为

$$\theta^* = \arg\max S(\theta) \tag{2.167}$$

其中 $S(\theta)$ 是整体目标函数，对于这里的情况，它可以是折扣期望回报（Discounted Expected Return）。

CE 方法中的策略可以被参数化为一个多变量线性独立高斯分布（Multi-Variate Linear Independent Gaussian Distribution），参数矢量在迭代步 t 时的分布为 $\boldsymbol{\theta}_t \sim N(\boldsymbol{\mu}_t, \boldsymbol{\sigma}_t^2)$。在采了 n 个样本矢量 $\boldsymbol{\theta}_1,\cdots,\boldsymbol{\theta}_n$ 并评估了它们的值 $S(\boldsymbol{\theta}_1),\cdots,S(\boldsymbol{\theta}_n)$ 后，我们对这些值排序并选取最好的 $\lfloor \rho\cdot n \rfloor$ 个样本，其中 $0<\rho<1$ 是选择比率（Selection Ratio）。所选取的样本的指标记为 $I\in\{1,2,\cdots,n\}$，分布的均值可以用以下式子更新：

$$\boldsymbol{\mu}_{t+1} =: \frac{\sum_{i\in I}\boldsymbol{\theta}_i}{|I|} \tag{2.168}$$

而方差的更新为

$$\boldsymbol{\sigma}_{t+1}^2 := \frac{\sum_{i\in I}(\boldsymbol{\theta}_i - \mu_{t+1})^{\mathrm{T}}(\boldsymbol{\theta}_i - \boldsymbol{\mu}_{t+1})}{|I|} \tag{2.169}$$

交叉熵方法是一个有效且普遍的优化算法。然而，此前研究表明 CE 对强化学习问题的适用性严重局限于一个现象，即分布会过快集中到一个点上。所以，它在强化学习的应用中虽然速度快，但是也有其他限制，因为它经常收敛到次优策略。一个可以预防较早收敛的标准技术是引入噪声。常用的方法包括在迭代过程中对高斯分布添加一个常数或一个自适应值到标准差上，比如：

$$\boldsymbol{\sigma}_{t+1}^2 := \frac{\sum_{i\in I}(\boldsymbol{\theta}_i - \boldsymbol{\mu}_{t+1})^{\mathrm{T}}(\boldsymbol{\theta}_i - \boldsymbol{\mu}_{t+1})}{|I|} + Z_{t+1} \tag{2.170}$$

如在 Szita et al. (2006) 的工作中，有 $Z_t = \max(5 - \frac{t}{10}, 0)$。

2.7.4 结合基于策略和基于价值的方法

根据以上的初版策略梯度（Vanilla Policy Gradient）方法，一些简单的强化学习任务可以被解决。然而，如果我们选择使用蒙特卡罗或 TD(λ) 估计，那么产生的更新经常会有较大的方差。我们可以使用一个如基于价值的优化中的批判者（Critic）来估计动作价值函数。从而，如果我们使用参数化的价值函数近似方法，将会有两套参数：行动者（Actor）参数和批判者参数。这实际上形成了一个非常重要的算法结构，叫作 Actor-Critic （AC），典型的算法包括 Q 值 Actor-Critic、深度确定性策略梯度（DDPG）等。

回想之前小节中介绍的策略梯度理论，性能目标 J 关于策略参数 θ 的导数为

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\sum_{t=0}^{\mathrm{T}} \nabla_\theta \log \pi_\theta(A_t|S_t) Q^\pi(S_t, A_t) \tag{2.171}$$

其中 $Q^\pi(S_t, A_t)$ 是真实动作价值函数，而最简单的估计 $Q^\pi(S_t, A_t)$ 的方式是使用采样得到的累计奖励 $G_t = \sum_{t=0}^{\infty}\gamma^{t-1}R(S_t, A_t)$。在 AC 中，我们使用一个批判者来估计动作价值函数：$Q^w(S_t, A_t) \approx Q^\pi(S_t, A_t)$。因此 AC 中策略的更新规则为

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\sum_{t=0}^{\mathrm{T}} \nabla_\theta \log \pi_\theta(A_t|S_t) Q^w(S_t, A_t) \tag{2.172}$$

其中 w 为价值函数拟合中批判者的参数。批判者可以用一个恰当的策略评估算法来估计，比如时间差分（Temporal Difference，TD）学习，像式 (2.92) 中对 TD(0) 估计的 $\Delta w = \alpha(Q^\pi(S_t, A_t; w) - R_{t+1} + \gamma v_\pi(S_{t+1}, w))\nabla_w Q^\pi(S_t, A_t; w)$。

尽管 AC 结构可以帮助减小策略更新中的方差，它也会引入偏差和潜在的不稳定（Poten-

tial Instability）因素，因为它将真实的动作价值函数替换为一个估计的，而这需要**兼容函数近似**（Compatible Function Approximation）条件来保证无偏差估计，如文献 (Sutton et al., 2000) 所提出的。

兼容函数近似

兼容函数近似条件对 SPG 和 DPG 都适用。我们将对它们分别展示。这里的“兼容”指近似动作价值函数 $Q^w(s,a)$ 与相应策略之间是兼容的。

对于 SPG：具体来说，兼容函数近似提出了两个条件来保证使用近似动作价值函数 $Q^\pi(s,a)$ 时的无偏差估计（Unbiased Estimation）：（1）$Q^w(s,a)=\nabla_\theta \log \pi_\theta(a|s)^{\mathrm{T}} w$ 和（2）参数 w 被选择为能够最小化均方误差（Mean-Squared Error，MSE）$\mathrm{MSE}(w)=\mathbb{E}_{s\sim\rho^\pi,a\sim\pi_\theta}[(Q^w(s,a)-Q^\pi(s,a))^2]$ 的。更直观地，条件（1）是说兼容函数拟合器对随机策略的“特征”是线性的，该“特征”为 $\nabla_\theta \log \pi_\theta(a|s)$，而条件（2）要求参数 w 是从这些特征估计 $Q^\pi(s,a)$ 这个线性回归（Linear Regression）问题的解。实际上，条件（2）经常被放宽以支持策略评估算法，这些算法可以用时间差分学习来更高效地估计价值函数。

如果以上两个条件都被满足，那么 AC 整体算法等价于没有使用批判者做近似，如 REINFORCE 算法中那样。这可以通过使得条件（2）中的 MSE 为 0 并计算梯度，然后将条件（1）代入来证明：

$$\begin{aligned}
\nabla_w \mathrm{MSE}(w) &= \mathbb{E}[2(Q^w(s,a)-Q^\pi(s,a))\nabla_w Q^w(s,a)] \\
&= \mathbb{E}[2(Q^w(s,a)-Q^\pi(s,a))\nabla_\theta \log \pi_\theta(a|s)] \\
&= 0 \\
&\Rightarrow \mathbb{E}[Q^w(s,a)\nabla_\theta \log \pi_\theta(a|s)] = \mathbb{E}[Q^\pi(s,a)\nabla_\theta \log \pi_\theta(a|s)]
\end{aligned} \tag{2.173}$$

对于 DPG：兼容函数近似中的两个条件应按照确定性策略 $\mu_\theta(s)$ 做相应修改：（1）$\nabla_a Q^w(s,a)|_{a=\mu_\theta(s)} = \nabla_\theta \mu_\theta(s)^{\mathrm{T}} w$ 而（2）w 最小化均方误差，$\mathrm{MSE}(\theta,w)=\mathbb{E}[\epsilon(s;\theta,w)^{\mathrm{T}}\epsilon\ (s;\theta,w)]$，其中 $\epsilon(s;\theta,w)=\nabla_a Q^w(s,a)|_{a=\mu_\theta(s)} - \nabla_a Q^w(s,a)|_{a=\mu_\theta(s)}$。同样可以证明这些条件能够保证无偏差估计，通过将拟合过程所做近似转化成一个无批判者的情况：

$$\nabla_w \mathrm{MSE}(\theta,w) = 0 \tag{2.174}$$

$$\Rightarrow \mathbb{E}[\nabla_\theta \mu_\theta(s)\epsilon(s;\theta,w)] = 0 \tag{2.175}$$

$$\Rightarrow \mathbb{E}[\nabla_\theta \mu_\theta(s)\nabla_a Q^w(s,a)|_{a=\mu_\theta(s)}] = \mathbb{E}[\nabla_\theta \mu_\theta(s)\nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}] \tag{2.176}$$

它对在线策略 $\mathbb{E}_{s\sim\rho^\mu}[\cdot]$ 和离线策略 $\mathbb{E}_{s\sim\rho^\beta}[\cdot]$ 的情况都适用。

其他方法

如果我们在式 (2.171) 中用优势函数（Advantage Function）替换动作价值函数 $Q^{\pi}(s,a)$（由于减掉基准值不影响梯度）：

$$A^{\pi_\theta}(s,a) = Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s) \tag{2.177}$$

那么我们实际可以得到一个更先进的算法叫作优势 Actor-Critic（Advantage Actor-Critic，A2C），它可以使用 TD 误差来估计优势函数。这对前面提出的理论和推导不产生影响，但会改变梯度估计的方差。

近来，人们提出了无行动者（actor-free）方法，比如 QT-Opt 算法 (Kalashnikov et al., 2018) 和 Q2-Opt 算法 (Bodnar et al., 2019)。这些方法也结合了基于策略和基于价值的优化，具体是无梯度的 CE 方法和 DQN。它们使用动作价值拟合（Action Value Approximation）来学习 $Q^{\pi_\theta}(s,a)$，而不是使用采样得到的折扣回报作为高斯分布中采样动作的估计，这被证明对现实中机器人学习更高效和有用，尤其是当有示范数据的时候。

参考文献

ACHIAM J, KNIGHT E, ABBEEL P, 2019. Towards characterizing divergence in deep q-learning[J]. arXiv preprint arXiv:1903.08894.

AUER P, CESA-BIANCHI N, FREUND Y, et al., 1995. Gambling in a rigged casino: The adversarial multi-armed bandit problem[C]//Proceedings of IEEE 36th Annual Foundations of Computer Science. IEEE: 322-331.

BODNAR C, LI A, HAUSMAN K, et al., 2019. Quantile QT-Opt for risk-aware vision-based robotic grasping[J]. arXiv preprint arXiv:1910.02787.

BUBECK S, CESA-BIANCHI N, et al., 2012. Regret analysis of stochastic and nonstochastic multi-armed bandit problems[J]. Foundations and Trends® in Machine Learning, 5(1): 1-122.

DEGRIS T, WHITE M, SUTTON R S, 2012. Linear off-policy actor-critic[C]//In International Conference on Machine Learning. Citeseer.

DUAN Y, CHEN X, HOUTHOOFT R, et al., 2016. Benchmarking deep reinforcement learning for continuous control[C]//International Conference on Machine Learning. 1329-1338.

FU J, SINGH A, GHOSH D, et al., 2018. Variational inverse control with events: A general framework for data-driven reward definition[C]//Advances in Neural Information Processing Systems. 8538-8547.

HANSEN N, OSTERMEIER A, 1996. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation[C]//Proceedings of IEEE international conference on evolutionary computation. IEEE: 312-317.

HEESS N, WAYNE G, SILVER D, et al., 2015. Learning continuous control policies by stochastic value gradients[C]//Advances in Neural Information Processing Systems. 2944-2952.

KALASHNIKOV D, IRPAN A, PASTOR P, et al., 2018. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation[J]. arXiv preprint arXiv:1806.10293.

KINGMA D P, WELLING M, 2014. Auto-encoding variational bayes[C]//Proceedings of the International Conference on Learning Representations (ICLR).

LESHNO M, LIN V Y, PINKUS A, et al., 1993. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function[J]. Neural networks, 6(6): 861-867.

LEVINE S, 2018. Reinforcement learning and control as probabilistic inference: Tutorial and review[J]. arXiv preprint arXiv:1805.00909.

NELDER J A, MEAD R, 1965. A simplex method for function minimization[J]. The computer journal, 7(4): 308-313.

PETERS J, SCHAAL S, 2008. Natural actor-critic[J]. Neurocomputing, 71(7-9): 1180-1190.

PYEATT L D, HOWE A E, et al., 2001. Decision tree function approximation in reinforcement learning[C]//Proceedings of the third international symposium on adaptive systems: evolutionary computation and probabilistic graphical models: volume 2. Cuba: 70-77.

SCHMIDHUBER J, 2015. Deep learning in neural networks: An overview[J]. Neural networks, 61: 85-117.

SILVER D, LEVER G, HEESS N, et al., 2014. Deterministic policy gradient algorithms[C].

SINGH S, JAAKKOLA T, LITTMAN M L, et al., 2000. Convergence results for single-step on-policy reinforcement-learning algorithms[J]. Machine learning, 38(3): 287-308.

SUTTON R S, MCALLESTER D A, SINGH S P, et al., 2000. Policy gradient methods for reinforcement learning with function approximation[C]//Advances in Neural Information Processing Systems. 1057-1063.

SZEPESVÁRI C, 1998. The asymptotic convergence-rate of q-learning[C]//Advances in Neural Information Processing Systems. 1064-1070.

SZITA I, LÖRINCZ A, 2006. Learning tetris using the noisy cross-entropy method[J]. Neural computation, 18(12): 2936-2941.

TSITSIKLIS J N, ROY B V, 1997. An analysis of temporal-difference learning with function approximation[R]. IEEE Transactions on Automatic Control.

VAN HASSELT H, GUEZ A, SILVER D, 2016. Deep reinforcement learning with double Q-learning[C]// Thirtieth AAAI conference on artificial intelligence.

VAN HASSELT H, DORON Y, STRUB F, et al., 2018. Deep reinforcement learning and the deadly triad[J]. arXiv preprint arXiv:1812.02648.

WATKINS C J, DAYAN P, 1992. Q-learning[J]. Machine learning, 8(3-4): 279-292.

WILLIAMS R J, BAIRD III L C, 1993. Analysis of some incremental variants of policy iteration: First steps toward understanding actor-critic learning systems[R]. Tech. rep. NU-CCS-93-11, Northeastern University, College of Computer Science.