Dong Wang

现代机器学习技术导论

2018 年 2 月 6 日

Springer

Contents

Chapter 1
机器学习概述

Chapter 2
线性模型

Chapter 3
神经模型

Chapter 4
深度学习

Chapter 5
核方法

Chapter 6
图模型

Chapter 7
非监督学习

Chapter 8
非参数模型

Chapter 9
遗传学习

Chapter 10
强化学习

本书前面已经讨论了监督学习和无监督学习两种主要学习方法。然而，我们人类的学习通常是另一种样子：我们从生下来开始，并没有人告诉我们应该如何做，而是通过主动和环境打交道，从环境中得到反馈，逐渐学习如何做事。这一学习过程与监督学习和无监督学习具有明鲜区别：我们不需要有明确的学习目标，仅通过行为后果来判断行为的对错；我们需要主动和环境打交道，而不是坐等别人告诉我们经验；我们的行为所产生的后果通常不是即时的，有些行为的价值需要较长时间才能显现。事实上，现实生活中绝大多数学习任务需要用上述方法来解决。拿最简单的练习走路来说，家长通常不会告诉孩子如何抬腿、如何迈步、如何保持平衡这些细节，而是让孩子自己尝试站起来，一点点挪动双腿，鼓励他达到行走的目标。“尝试”和“鼓励”是人类完成绝大多数学习任务的基本方式，也是机器完成学习任务的重要方式。特别是对复杂任务和复杂系统，细节指导变得越来越困难，通过激励和反馈指导学习过程几乎是唯一有效的学习方式。事实上从机器学习发展的早期，研究者们已经开始关注这一方向，并发展出一套丰富的学习理论和学习算法，称为“强化学习”。强化学习被认为是让机器具有人类智能的重要手段，是机器学习蛋糕上漂亮的草莓。本章将介绍强化学习的基本原理和基本方法，并给出一些强化学习的实际例子，特别是和深度学习结合起来之后表现出的强大学习功能。

10.1 强化学习

10.1.1 什么是强化学习

强化学习是一类学习方法的总称，这类学习方法通过和环境进行交互，利用环境给出的反馈信息进行学习。再次拿幼儿学习走路的例子来说：刚开始的时候孩子并不会任何有目的性动作，只会随机活动四肢，家长也不会刻意帮他们抬腿、迈步，也不会向他们解释行走的好处，但会在他们偶尔可以站立、扶着墙挪动的时候给以鼓掌、拥抱等鼓励，让孩子倾向往这方面尝试。同时，当孩子站错了姿势摔倒时家长并不会马上批评他，但他们会感到疼痛，下次就会避免类似错误动作。经过多次尝试，孩子就会渐渐学习应该如何正确站立、迈步、保持平衡，从而在获得更大自由空间的同时得到更多赞扬，最终一点点学会走路。

上述例子包括几个需要特别注意的地方：一、学习过程中的每一个具体动作（如关节关动）并没有明确的正确、错误标签，也没有这两方面的明确样例。整个学习过程几乎没有一个明确的目标，只有达到目标后的某种奖励，或收益；二、学习过程必须通过不断尝试；三、学习信号（奖励）通常是完成某一系列动作之后（如站立）才延迟出现；四、某一个具体动作可能会对后续动作产生一系列影响，如某一关节活动会直接影响后续动作；五、我们不关注某一特定动作的成败，而是所有动作得到的总体效果（如最终学会走路）。这些特点是强化学习任务的典型特征，也是判断是否应该使用强化学习方法的标准。事实上，几乎所有复杂、连续任务都具有类似特点，因此强化学习具有广泛的应用价值。

图 10.1给出了强化学习的结构框架，其中学习结构是机器的所有内在数据结构与算法，可以形象地看作一个机器人。让机器人不停和环境交互，每次交互观察当前环境状态（State），根据这一状态选择某一动作（Action），依此得到奖励或收益（Reward）。通过多次交互，机器人学习在特定环境下选择合适动作的策略（Policy），使得总体收益（Return）最大化。

10.1.2 与其它学习方法的区别

强化学习与监督学习和非监督学习有很大不同。监督学习对每一个数据样本有明确标注，对应到强化学习任务中，这意味着在某一状态下应采取的

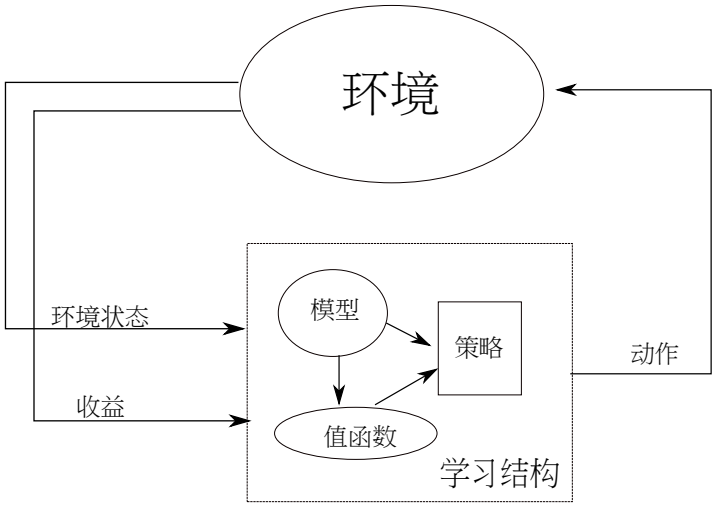

Fig. 10.1 强化学习概念图。学习结构包括模型、值函数、策略等可学习对象和相应学习算法。通过和环境交互，观察环境状态变化，学习结构依当前策略决定采取何种动作。该动作将影响环境，并从环境中得到收益。强化学习通过一系列交互过程，学习得到长期收益的最优策略。

每一个动作有明确标注。这显然不是强化学习的典型场景。非监督学习则是另一极端，不对数据做任何标注，算法通过数据之间的关系发现其分布规律。相比非监督学习，强化学习还是提供了一定的“标注”，即奖励信号。尽管这些标注既不及时，又不直接，还可能充满噪音，但毕竟对学习方向提供了指导。从这一角度看，强化学习可以认为是一种弱标记学习。特别是，这一标记对某一具体动作来说是弱标记，但对于整个学习任务却是非常强的，强到直接标注了任务的成败。因此，强化学习是纯粹的目标驱动，是以成败论英雄的“逐利学习”。

同时，强化学习是一种主动学习方法，通过主动和环境进行交互产生学习样本。因此，如何“主动”才学习的更全面更有效，是强化学习中的一个核心问题，即探索（Exploration）和应用（Exploitation）的权衡：太多无用的尝试会浪费大量资源；过于相信当前的经验又可能错失更好的机会。这一权衡显然不是监督学习或非监督学习考虑的重点。

基于上述鲜明特性，很多学者将强化学习看作是和监督学习、非监督学习并列的另一大类学习方法，如图 10.2 所示。这三类方法在某些方面具有相似性，但又各自具有鲜明特色。

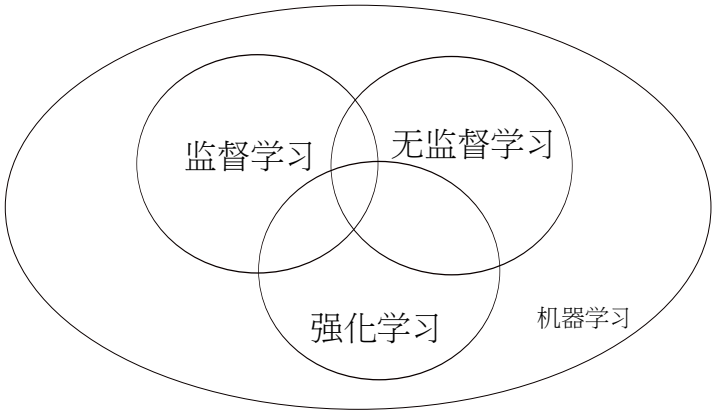

Fig. 10.2 监督学习、无监督学习与强化学习是机器学习领域三种主要学习方法。

另有一些学者认为强化学习是比其它学习方法更高级、更通用的方法，例如，监督学习只是强化学习的一种简化，是一种具有即时的、明确奖励信号的强化学习。图 10.3给出Kober等人对强化学习的归类。他们从学习信号复杂性和交互复杂性两个角度对一些机器学习算法进行归类，发现当学习信号是分类类别等简单的标记，且学习过程中没有与环境的交互时，即得到传统的监督学习方法。保持学习信号的即时性，当学习信号考虑到与任务相关的成本，则可认为是一种成本敏感学习（Cost-sensitive learning）；如果学习信号仅为一些间接的奖励或消耗（如赌博机的收益、机器人的耗电、股市里的手续费等），这一学习任务通常称为上下文相关Bandit（Contextual Bandit）。从另一维度看，当我们考虑学习过程的交互性和时序性，学习任务会进一步复杂化。对于简单数据标注，考虑交互的时序性，即可得到一种简单的模仿学习（Imitation Learning）。例如，我们想让机器学习如何做面条，学习方法是让人做一遍面条，机器照着样子学习，这里学习既要照顾最终做面条的结果，也要照顾做面条的过程，让机器人每一步都不要偏离太远，即可学会做面条的方法。注意，这个任务里做面条的每个步骤都是相对比较确定的，机器人可以明确知道自己离标准过程的差距，因此学习信号很明确，但因为是个时序过程，学习起来比传统监督学习要困难。进一步，如果学习信号进一步复杂化，仅与是否完成任务目标相关而不考虑学习过程中的具体步骤，同时，考虑与环境的交互，即得到强化学习。例如在无人机自主飞行任务中，没有一个标准的飞行过程供机器学习，需要机器在实际飞行过程中和环境交互自主完成，而学习效果的好坏并不取决于每一个飞行动作是否标准，而是取决于一个飞行任务是否顺利完成。这种目标驱动的主动学习方式

即为强化学习。从这一分类方法看，强化学习可以认为是传统监督学习方法在学习信号和交互性上的扩展，前者关注目标驱动，后者强调在学习过程中的主动性和探索性。

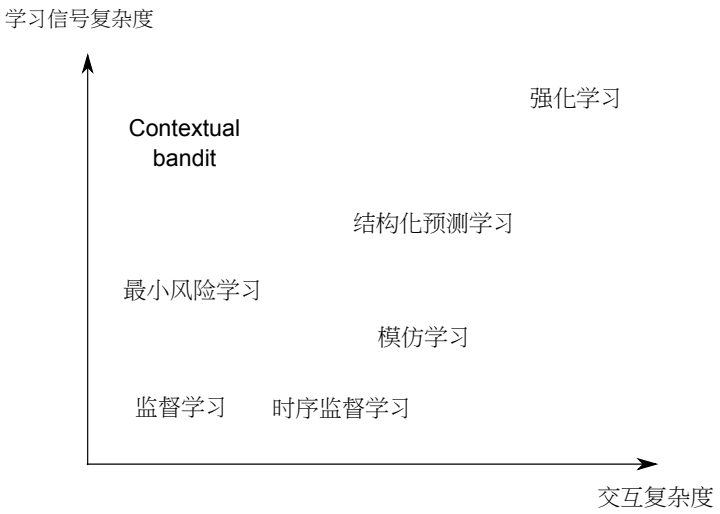

Fig. 10.3 强化学习与其它学习方法的关系。可以认为强化学习是传统监督学习的扩展，处理在结构和时序上更复杂的学习信号。图片参考 [4]。

值得注意的是，“目标驱动”和“主动学习”这两个特点也是前一章所述演化学习的主要特征。在演化学习里，算法随机生成一系列可能的模型，通过计算每个模型的适应函数去劣存优，逐渐学习到优化的模型。与演化学习相比，强化学习更加关注和环境进行交互的细节，利用交互过程的结构化知识指导学习，如我们后面要介绍的马尔可夫属性。在这一过程中，算法的主动性不是简单依赖随机性，而是通过交互过程中的模型细节产生更有效的探索，优化过程也不是只看结果进行取舍，而是依赖结构化知识对现有模型进行更新。因此，强化学习（如果可能应用的话）通常比演化学习效率高的多。

10.1.3 强化学习：真正的人工智能？

很多学者认为强化学习可能是实现真正人工智能的学习方法，可能因为强化学习具有如下优点：其一，强化学习具有利用某种学习信号进行建模的

能力，这使其学习过程具有类似监督学习的方向性和针对性，另一面，强化学习中学习信号的标注成本往往比监督学习要低的多，使其兼具有非监督学习的优势。其二，强化学习不关注过程，仅关注结果，这使其非常适合那些过程复杂，具有很大不确定性的复杂时序任务；其三，强化学习强调通过主动探索发现环境内部的规律，这种主动交互主动学习使机器的表现更有智能性；其四，强化学习通常并不需要对环境的随机性和动态性建模，仅需对不同状态下的行为选择建模，因此非常适用合未知环境。其五，强化学习通常是在线学习，当环境发生变化时可以快速适应到新环境。

10.1.4 强化学习的应用

强化学习在各个领域得到广泛应用，如学习直升机的自主操作，学习打电子游戏，学习象棋、围棋等棋类的玩法，进行投资组合选择，控制电站的发电量，帮助机器象人类一样行走等等。我们通过对几种典型场景的应用来介绍强化学习的能力。

在机器人领域，强化学习可以用来让机器学习如何行走，包括在凹凸不平的地面上保持稳定，绕过障碍物等，抓取物品等。基于现实情况的复杂性，不论是环境状态还是输出动作都是高维的，几乎无法对状态空间进行足够多的采样（如各种不同障碍物），也无法对每个动作给出是否合理的具体标注。更重要的是，即使能够标注，我们学习的目的也不是一个状态下某个动作的合理性，而是作为一个整体的动作序列是否可以有效达到目标。因此，与其关注每个具体状态的具体动作，不如关注动作组合产生的总体效果，这正是强化学习的优势。

类似的，在商业规划任务中，强化学习也得到广泛应用。在投资管理任务中，强化学来可用来决定对每种资产的购买比例；在资源规划任务中，可决定对哪些任务环节优先配置资源（如在呼叫中心系统中决定对哪些客户进行优先服务）；在路径规划任务中，学习采取何种方案进行装运、存储更节约成本。这些问题看似简单，但要达到优化设计并不容易。强化学习目标驱动的特性使得它可以通过不断尝试得到优化的行为策略。

在金融领域中，强化学习可以用来做投资决策，如进行股票配比，资产组合等。这些任务都有一个共同的特点，其策略的有效性都具有长期性和动态性。所谓长期性，是指策略的效果不能即时显现，需要通过一系列市场操作，由此获得收益来判断；所谓动态性，是指市场具有时变性，每天都在变化，必须通过在线方式不断更新操作方式才可能保持有效性。强化学习具有

处理时序动态信号的能力，并以最终盈利为学习目标，因此非常适合金融处理领域。

在媒体领域，强化学习可以通过和用户交互，学习用户的浏览倾向性，进而对不同用户展示不同的内容。例如，可以通过考察加入广告后的点击率，来学习哪些方式是合理策略，哪些是让用户反感、起不到广告效果的策略。有意思的是，仅通过大数据统计往往并不能达到引导用户关注广告的效果，强化学习可以通过和用户交互，设计一套逐渐引起用户注意，进而引导用户进行广告点击的策略。

在医疗领域，强化学习用来对药品测试方案进行设计，如应进行哪些测试项目，应向哪些人群发出测试邀请。强化学习也用来设计病理检查方案和诊疗方案，在这些应用里，强化学习不注重每一个诊疗步骤的具体效果，而是关注方案的最终结果，并通过设计合理的探索方案（如新药使用），从整体上提高医疗知识的积累。

强化学习的另一个应用领域是人机对战游戏。事实上，历史上第一个机器学习程序，Samuel的西洋棋游戏，即是基于强化学习。在该程序中，Samuel提出了后来发展为时序差分算法（Temporal Difference, TD）的强化学习方法，结合启发式搜索，取得了战胜业余选手的棋力。今天，机器学习甚至整个人工智能的热潮，很大一部分也要归因于以AlphaGo为代表的人机对战游戏程序对人类顶尖选手的胜出。在这些胜利背后，强化学习居功至伟。通过强化学习，机器不考虑每一个游戏动作或每一步棋的优劣，只需考虑采取的策略最后能否获胜。有趣的是，这种只重结果不重过程的学习方法让机器拥有更长远的眼光，不再纠缠一时一地的得失，转而关注长远收益和全局收益，最终以更大的概率战胜对手。

综合上面的例子，我们可以看到强化学习最能发挥作用的地方是系统复杂、数据难以标注、时间和空间相关性强的复杂任务。人类在学习这些复杂任务的时候用的是不断尝试，逐渐获得经验，渐次提高处理能力的策略，而这正是强化学习的基本思路。

10.2 强化学习的基本元素

本节对强化学习任务进行形式化定义，后续各节所述具体算法将以这些定义为基础。我们将定义状态、动作和收益三个基本元素，由此定义值函数、最优值函数及其与策略优化之间的关系。最后我们介绍强化学习算法的基本分类。

10.2.1 强化学习三元素

强化学习是“面向目标”的与“环境”“主动交互”学习方法。这一定义概括了强化学习的三个基本元素：代表学习目标的奖励或收益（Reward）$R_t \in R$，代表环境的状态（Status）$S_t \in \mathscr{S}$, 代表交互的动作（Action）$A_t \in \mathscr{A}$，其中$\mathscr{S}$和$\mathscr{A}$分别为状态和动作的集合。注意依问题不同，这两个集合可能是有限的，也可能是无限的，有可能是离散的，也可能是连续的。为描述方便，现阶段暂且假定状态和动作集合都是离散有限的，后续将介绍处理无限连续状态和动作空间的方法。

定义了状态、动作和收益后，我们定义策略π为在某一状态下的动作选择方案，以概率表示为：

$$\pi(a|s) = P[A_t = a|S_t = s],$$

其中S_t和A_t分别表示在t时刻所处的状态和采取的动作。上式意味着该策略是时不变的，即在任何时刻，只要系统处在s状态，其采取的动作符合同一分布$\pi(a|s)$。强化学习的目的是学习某一策略π_*，使得依该策略进行交互获得的长期收益最大化。为此，我们需要定义长期收益（Return），并依此确定策略间的偏序关系。

10.2.2 长期收益

设依某一策略π运行的某一交互过程如下：

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, ...$$

对于任一时刻t,可定义该序列的长期收益（Return）为t时刻后所有交互的收益之和，即：

$$G_t = R_{t+1} + R_{t+1} + ... + R_T, \tag{10.1}$$

其中T为交互完成的时刻。上式称为加合收益（Sum Return）。该长期收益的定义适用于在有限步骤内明确可结束的任务，如下一盘棋，打一局游戏，走一次迷宫等。这种任务我们称为‘多轮任务(Episode Task)’，其中每一次交互

序列称为‘一轮’。在这种任务中，每个交互序列都有一个明确的结束状态，每轮结束之后重新开始，不同轮之间没有相关性。

上述加合收益对没有明确结束时间的任务并不适用，如股票买卖，操作每天进行，但并没有明确结束的时间，只要我们愿意，可以一直操作下去。这类任务称为‘连续任务’，对这种任务，T值是无穷大的，因此式 10.1所定义的加合收益可能是无界的。一种解决办法对未来收益进行折扣，越远的收益折扣越多，即：

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... \tag{10.2}$$

其中γ是折扣因子。该方法称为折扣收益（Discount Return），是最常用的收益计算方法，其中折算因子可以理解为对近期收益的重视，也可理解为远期收益的折现风险。

另一种求连续任务总体收益的方法是求对未来平均收益的极限值，称为平均收益（Average Return），即：

$$G_t = \lim_{T\to\infty} \frac{1}{T}(R_{t+1} + R_{t+2} + ... + R_T), \tag{10.3}$$

该方法在在连续任务的策略梯度优化算法中经常用到。

10.2.3 值函数与策略优化

基于长期收益G_t，可定义一个交互系统处在状态s时的“价值”，$V_\pi(s)$为：

$$V_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s], \tag{10.4}$$

即当系统处在状态s下时，基于策略π进行交互操作，在未来取得的长期收益的期望。类似，可以定义系统处在状态s，并采取动作a所能产生的价值:

$$Q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]. \tag{10.5}$$

显然,状态值函数$V_\pi(s)$与动作值函数$Q_\pi(s,a)$具有如下简单关系：

$$V_\pi(s) = \sum_a \pi(a|s) Q_\pi(s,a).$$

注意这两个值函数都是策略π的函数。强化学习中一个重要任务是给定某个策略π，计算V_π和Q_π，这一任务通常称为“策略评价”(Policy Evaluation)。依公式10.4和公式10.5，这一任务似乎并不复杂，但事实上并非如此。第一个困难是我们只知道每一个动作产生的即时收益R_t，计算长期收益G_t需要与环境进行长期交互；第二个困难是对G_t求期望，这需要对各种可能的交互路径进行尝试。要解决上述两个困难，或是对系统的动态性有明确了解，因而可通过采样方式进行路径模拟，或是与实际系统进行现实交互，从中得到实际知识。不论哪种方法，计算值函数都是不容易的事。我们后面将介绍各种优化计算方法。

强化学习的最终目的是学习最优策略，使得依该策略得到的长期收益最大化。为了定义最优策略，我们首先需要定义策略的偏序关系如下：

$$\pi > \pi' \quad if \quad \forall s \;\; V_\pi(s) \geq V'_\pi(s).$$

上式意味着，若要某一策略π优于另一策略π'，则需要策略π在所有状态上的长期收益期望都大于π'。定义优化值函数如下：

$$V_*(s) = \max_\pi V_\pi(s),$$

和

$$Q_*(s,a) = \max_\pi Q_\pi(s,a).$$

上述两式意味着一个优化值函数需在所有状态上优于其它值函数。结合优化策略的定义，如果我们能发现一个优化策略，则该策略对应的值函数必然是优化值函数。反过来，如果某一策略对应的值函数是最优值函数，则该策略必然是最优策略。问题是，这样一个策略是否存在？幸运的是，在一定条件下（如下节讨论的马尔可夫决策过程），可以证明对一个强化学习任务，这一策略是存在的，并可以通过最优值函数构造出来。

假设已经知道了一个优化的Q函数，可以在每一状态s选择$Q(s,a)$最大的动作a，即：

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \arg\max_{a \in \mathscr{A}} Q_*(s,a) \\ 0 & otherwise \end{cases} \tag{10.6}$$

上述策略称为贪心策略（Greedy Policy），可以证明该策略是优化的。

值得注意的是，在上述策略优化过程中，我们并没有直接对策略进行优化，而是通过引入一个值函数来产生相应策略。在本章后续章节中，我们看

到大多数强化学习算法都是基于值函数的。事实上基于值函数进行策略优化也是强化学习区别于演化学习的主要特点：通过对过程中每个状态进行估值，并通过状态间的关系对估值进行修正（如后面要介绍的TD算法），再基于修正后的估值优化策略，强化学习可以充分利用交互过程中的状态转移关系。相比强化学习，演化学习仅通过策略所产生的结果来判断策略的优劣，这种选择型优化不考虑交互过程中每个状态的估值，效率要低很多。

10.2.4 通用策略迭代

可以证明 [11]，给定一个策略π的值函数$Q_\pi(s,a)$，如下贪心策略将优于π：

$$\pi'(a|s) = \begin{cases} 1 & \text{if } a = \arg\max_{a \in \mathscr{A}} Q_\pi(s,a) \\ 0 & otherwise \end{cases} \tag{10.7}$$

同样，如果知道$V_\pi(s)$，也可以得到类似基于贪心原则的优化策略。与Q函数不同的是，这里我们需要知道环境的动态特征，如在状态间的转移概率$P(s'|s,a)$和动作的即时收益$P(r|s,a)$：

$$\pi'(a|s) = \begin{cases} 1 & \text{if } a = \arg\max_{a \in \mathscr{A}} \pi(a|s)P(r|s,a) + \sum_{s'} P(s'|s,a)V_\pi(s') \\ 0 & otherwise \end{cases} \tag{10.8}$$

需要注意的是，动作值函数和状态值函数优化策略的不同本质上源于状态和动作这两个元素的不同性质。动作是策略的持行行为，因而动作值函数的优化表现为最优动作的选择；而状态是策略持行的结果，因而对状态值函数的优化应考虑到持行过程的细节，如状态承载的动作，状态与环境的交互等等。从某种程度上来说，状态值函数比动作值函数意义更加广泛。

以上两式意味着我们可以通过一个迭代过程对策略进行优化：首先设定一个随机策略，基于该策略确定值函数V或Q，再基于这些函数利用贪心原则对策略进行改进。这种值函数和策略交叉优化并迭代改进的过程称为通用策略迭代算法（General Policy Iteration，GPI）。可以证明，这一过程可实现值函数和策略的最优化，如图 10.4所示。GPI是很多强化学习算法的基本框架。

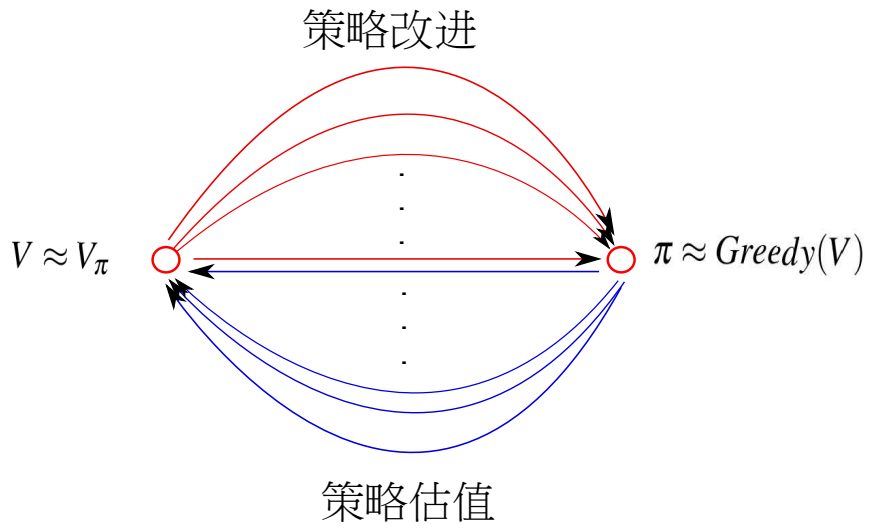

Fig. 10.4 通用策略迭代（GPI）算法。以随机策略π_0开始，在第t次迭代时基于某种策略估值方法求$V^t \approx V_\pi^{t-1}$，再基于贪心策略改进得到$\pi_t \approx Greedy(V^t)$。这一过程保证值函数的提高（$V^{t-1} \leq V^t$）和策略的改进（$\pi^{t-1} \leq \pi^t$），直到二者同时达到最优化。

10.2.5 强化学习算法分类

强化学习是一个庞大的家族，包括众多算法，每一种算法有其特有属性，用于解决不同场景下的特别问题。一般来说，在开始一个强化学习任务时，我们通常要考虑如下问题：

- 交互是一轮一轮进行（如下棋）还是连续进行（飞机自动控制）?
- 状态是可见的（如下棋）还是部分可见的（如打扑克）?
- 状态是离散的（如走迷宫）还是连续的（如股票市场）?
- 动作是离散的（如下棋中的落子）还是连续的（机器人控制中的力矩大小）?
- 动作的影响是即时的（如少量股票操作）还是长时的（如大量股票操作）?
- 环境动态性是已知的（如迷宫）还是未知的（如股市）?
- 收益信号是即时的（股票日操作收益）还是按轮给出的（下棋或打游戏中的胜负）?

对这些问题的回答决定了应选择何种学习方法。我们首先考虑状态和动作空间离散且有限，因而$V(s)$ 和$Q(s,a)$可以用离散形式表示的情况，再将讨论推广到连续状态和连续动作的情况。

即使是简单的离散环境，对各种学习方法有个清晰的分类也不容易。首先我们从宏观上对强化学习任务有个总体认识。如前所述，一个强化学习过程可由通用策略迭代（GPI）算法表示，其中包括对某一特定策略进行估值（Evaluation）和基于当前估值对策略进行改进（Policy Improvement），这两个过程迭代进行直到收敛，即可完成策略优化（Policy Optimization）。策略优化任务也经常被称为控制任务（Control）。由GPI算法可知，如果解决了估值任务，即可通过贪心原则对策略进行改进，策略优化任务也可以在很大程度解决。因此，我们讨论具体学习算法时，一般会重点考虑估值任务。

从学习的知识源角度，强化学习可以基于一个已知模型进行学习，也可以基于实际经验学习，前者一般称为‘规划任务（Planning）’，后者一般称为‘学习任务（Learning）’。例如在走迷宫任务中，当处在迷宫的某一位置时，下一个位置可以到哪里，每一个可能的方向得到多少即时收益是可以确定的，这等同于设计了一个描述环境动态特性的模型，基于该模型可以通过GPI算法得到最优策略，这一过程不需要与实际环境打交道，因此是规划任务。再如关于股票操作任务，事先并没有一个已知模型，只能通过和真实股市打交道，实际操作，从而获得股市的动态特性，进而对操作策略进行改进和优化。这一任务即是学习任务。从整体上看，规划任务基于模型，学习任务基于采样，虽然在具体算法上不论是模型还是采样在规划任务和学习任务中都有广泛应用，但这一原则基本是符合的。值得注意的是，不论是规划还是学习，都需要对策略进行估值和优化。图 10.5 中的(a)表示规划任务，(b)-(e)表示学习任务。

对于学习任务，可学习对象包括三种：环境模型、值函数和策略。注意强化学习的目的是策略优化，因此策略学习是最直接的学习，值函数学习次之，环境模型学习最间接。学习对象越直接，学习过程越简洁明了，但因缺少结构化知识，泛化能力越弱，需要的经验数据也越多。另外，任何间接学习都需要额外模型来生成策略，如当学习值函数时，需要一个策略生成方法产生优化策略，当学习环境模型时，需要一个规划方法由模型生成优化值函数和策略。

对环境模型进行学习的方法称为模型方法（Model-based Method)。注意这里的模型并非规划任务中的指定模型，而是基于某种学习结构，利用和环境交互得到的经验学习得到的。然而，一旦模型学习完成，则生成优化策略

就成了规划任务，相应的估值和优化方法即同样适用。图 10.5中的(b)给出了模型方法的学习框架。

对值函数进行学习的方法一般称为无模型方法(Model-free Method)。这种方法不对环境动态性进行建模，通过经验样本来学习值函数并以基于此得到优化策略。这一方法的基本思路是只要经验数据足够多，即可通过采样来代表环境模型。与模型方法相比，无模型方法可避免建模上的不精确性，但由于缺少结构化知识，在学习中需要更多经验数据。图 10.5中的(c)给出了无模型方法的学习框架。

策略学习（Policy Learning）是最直接的强化学习方法。策略学习中没有状态值的概念，由状态直接映射到行为决策。简单的策略学习可以统计在每个状态上的决策选择并得到经验分布，然而，绝大多数强化学习任务中这样明确的决策标准并不存在，事实上这也是强化学习区别于监督学习的主要方面。一个解决办法是对每个动作给以一定奖励，基于该奖励学习状态到动作的映射函数。因此策略学习多用在基于函数近似的学习方法中。近年来随着深度学习的发展，函数近似方法取得一系列重要进展，相应的策略学习方法也越来越受到重视。图 10.5中的(d)给出了策略学习方法的学习框架。注意策略学习也是无模型的，但在提到无模型方法时，大多数情况下仅指值函数学习。

上述三种学习方法可结合起来，得到混合学习方法。混合学习可选择任两种或三种学习方法结合，平衡不同学习方法的权重，发挥他们各自优势。如后面我们要讲到的Dyna算法，即是结合了模型学习和值函数学习，利用模型方法的可扩展性和无模型方法的学习能力。

在本章后续几节中，我们将介绍三种典型算法：完全基于模型的动态规划（Dynamic Programming, DP）算法，完全基于采样的蒙特卡洛（Monte Carlo, MC）算法，介于两者之间的时序差分（Temporal Difference, TD）算法。

10.3 全回溯与动态规划算法

我们首先考虑基于已知环境模型进行策略估值和优化,即规划任务。所谓环境模型，主要包括两部分：一是环境的动态发展规律，二是环境对动作的反馈规律。由于环境模型已知，理论上优化策略是可计算的。然而，如果模型过于复杂，策略求解带来的计算开销依然难以容忍。为此，人们往往要对环境进行一些简单假设，基于该假设可以推导效率较高的算法。强化学

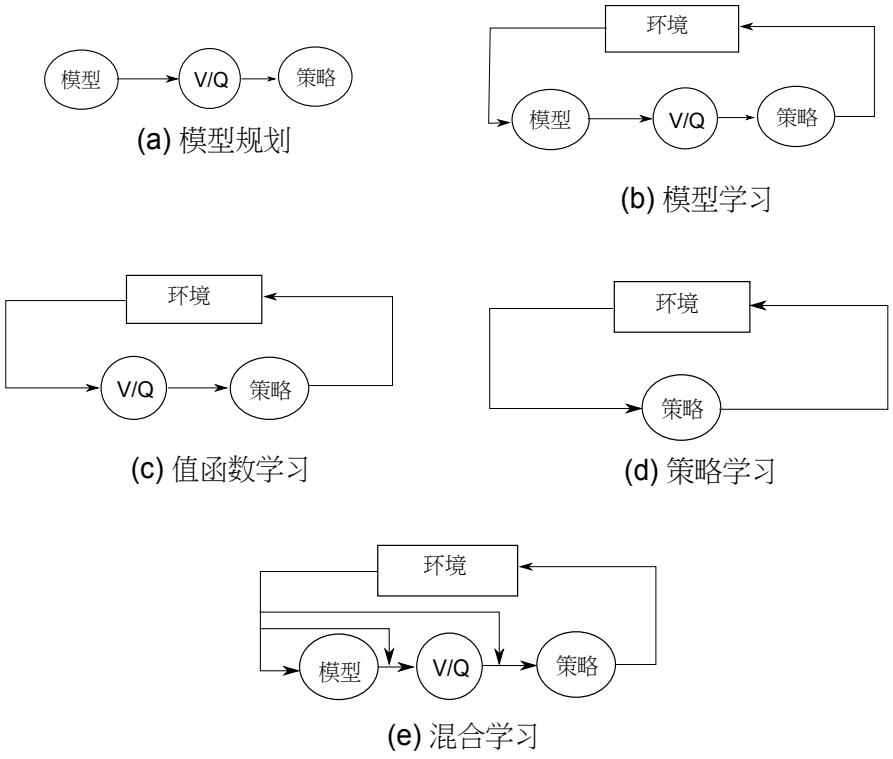

Fig. 10.5 强化学习的典型方法。(a)模型规划：当与环境无交互时，基于某个已知模型进行策略优化。(b)模型学习：与环境交互，学习环境模型，基于模型进行规划；(c)值函数学习：与环境进行交互，学习值函数，基于此优化策略。(d)策略学习：与环境交互，直接学习策略。(e)混合学习：结合多种学习方法，学习模型、值函数、策略中的任意组合。

习中，我们往往假设环境具有马尔可夫性（Markovian），相应的交互过程可表达为一个马尔可夫决策过程（Markov Decision Process, MDP）。基于MDP，系统的未来状态和即时收益只与当前状态相关，这极大简化了对交互过程的数学表达。基于该模型，可以推导出一种基于动态规划的*值函数学习算法*。

10.3.1 马尔可夫决策过程

马尔可夫决策过程（MDP）是马尔可夫过程的扩展，在马尔可夫过程基础上加入动作和动作的即时收益。MDP一般描述状态空间离散且完全可知、动作空间离散的交互过程，经过扩展后也可描述部分可见的交互过程，即

部分可见MDP（Partially Observable MDP, POMDP），或连续动作空间的交互过程，即连续MDP（Continuous MDP）。下面我们定义离散状态、离散动作的MDP。

一个MDP定义为在一个离散状态空间上的交互过程，在交互中每次操作所引起的状态转移概率和获得的即时收益概率仅与当前状态及所选择的动作相关，而与历史交互过程无关。“与历史过程无关”这一特性称为马尔可夫性，可形式化表示为：

$$P(S_{t+1}|S_0,A_0,R_1,S_1,A_1,R_2,...) = P(S_{t+1}|S_t,A_t),$$
$$P(R_{t+1}|S_0,A_0,R_1,S_1,A_1,R_2,...) = P(R_{t+1}|S_t,A_t),$$

其中$S_t \in \mathscr{S}$为t时刻状态，$A_t \in \mathscr{A}$为t时刻所选择的动作，R_{t+1}为第t时刻采取动作A_t后的即时收益。注意马尔可夫性如何简化了交互过程的数学表达。

MDP可以表示成一个有限状态自动机（Finite State Machine, FSM），其中每一个节点表示一个状态，每条边表示状态间转移，边上标记为动作和即时收益。图 10.6 给出一个表示资金理财任务的简单MDP。

*10.3.2 MDP*中的策略及值函数

上节所定义的马尔可夫过程仅定义了环境的动态性，包括状态转移特性$P(S_{t+1}|S_t,A_t)$和反馈特性$P(R_{t+1}|S_t,A_t)$。为了确定一个交互过程，还需定义策略π，即在某一状态下如何选择动作的方案。基于该策略，即可定义该过程的状态值函数$V_\pi(s)$和动作值函数$Q_\pi(s,a)$如下：

$$V_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s],$$

$$Q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s,A_t = a],$$

其中G_t为上节定义的长期收益。我们推导一个适用于MDP的值函数计算方法。注意在MDP中，基于历史无关假设，每个状态的值函数是固定的。为保证重复进入某一状态的值函数相等，一般选择折扣收益作为长期收益。

以状态值函数为例，可推导出下面的关系：

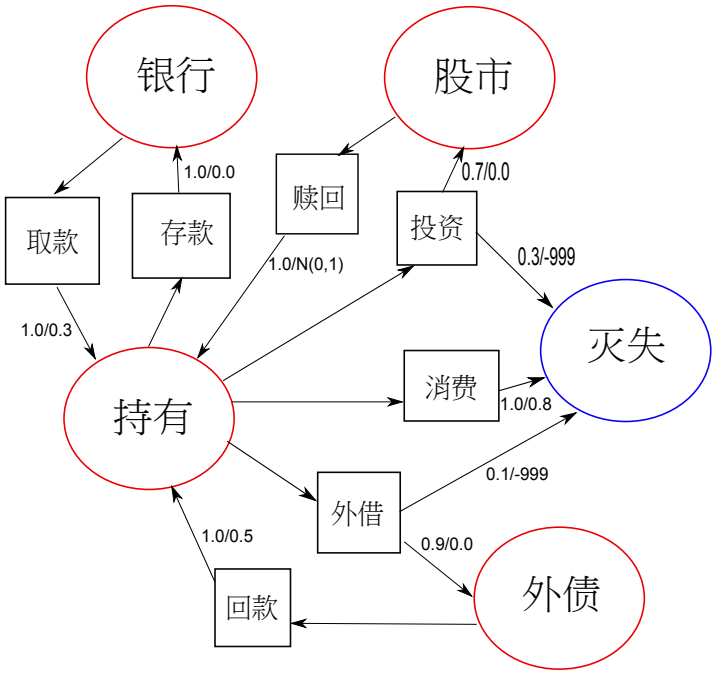

Fig. 10.6 一个表示资金操作的MDP，其中圆圈表示状态，方框表示操作，边上的值表示概率/收益。注意在消费操作中，0.8的收益表示消费获得的满足，投资赎回时获得的收益是一个高斯分布。

$$
\begin{aligned}
V_\pi(s) &= \mathbb{E}_\pi[G_t|S_t=s] \\
&= \mathbb{E}_\pi[R_{t+1}+\gamma R_{t+2}+\ldots|S_t=s] \\
&= \mathbb{E}_\pi[R_{t+1}|S_t=s]+\gamma\mathbb{E}_\pi[R_{t+2}+\gamma R_{t+3}+\ldots|S_t=s] \\
&= \sum_a \pi(s,a)\sum_r rP(r|s,a)+\gamma\sum_a \pi(s,a)\sum_{s'}P(s'|s,a)\mathbb{E}_\pi[R_{t+2}+\gamma R_{t+3}+\ldots|S_{t+1}=s'] \\
&= \sum_a \pi(s,a)\sum_r rP(r|s,a)+\gamma\sum_a \pi(s,a)\sum_{s'}P(s'|s,a)V_\pi(s'). \qquad (10.9)
\end{aligned}
$$

上式意味着对于一个特定的MDP（即状态转移概率和即时收益概率已经确定），对一个已知的策略π，其相应的值函数在不同状态上的取值需满足一定的限制关系。这些限制关系可以表示为以状态$V(s)$为变量的线性方式，如式 10.9所示，称为贝尔曼公式。因为每个状态都有一个独立的限制，加起来的独立线性方程的总数和状态数$|\mathcal{S}|$ 是一样的。所以从理论上讲，基于这些方程可以确定解出每个$V(s)$的值。

相应的，动作值函数也有类似的贝尔曼形式：

$$
\begin{aligned}
Q_\pi(s,a) &= \mathbb{E}_\pi[G_t|S_t=s,A_t=a] \\
&= \mathbb{E}_\pi[R_{t+1}+\gamma R_{t+2}+...|S_t=s,A_t=a] \\
&= \mathbb{E}_\pi[R_{t+1}|S_t=s,A_t=a]+\gamma\mathbb{E}_\pi[R_{t+2}+\gamma R_{t+3}+...|S_t=s,A_t=a] \\
&= \sum_r rP(r|s,a)+\gamma\sum_{s'}P(s'|s,a)\sum_{a'}\pi(s',a')\mathbb{E}_\pi[R_{t+2}+\gamma R_{t+3}+...|S_{t+1}=s',A_{t+1}=a'] \\
&= \sum_r rP(r|s,a)+\gamma\sum_{s'}P(s'|s,a)\sum_{a'}\pi(s',a')Q_\pi(s',a') \qquad (10.10)
\end{aligned}
$$

同状态值函数类似，上述动作值函数的贝尔曼公式也可以列出一个以$Q(s,a)$为变量的线性方程组，其中变量个数与方程数相等，因此通过解线性方程组直接求得$Q(s,a)$的值。

注意的是状态值函数和动作值函数的关系也具有类似贝尔曼的形式，甚至更简洁。首先看状态值函数：

$$
\begin{aligned}
V_\pi(s) &= \mathbb{E}_\pi[G_t|S_t=s] \\
&= \mathbb{E}_\pi\pi(s,a)[R_{t+1}+\gamma R_{t+2}+...|S_t=s,A_t=a] \\
&= \sum_a \pi(s,a)Q(s,a). \qquad (10.11)
\end{aligned}
$$

上式表明$V(s)$可表示成基于s的所有$Q(s,a)$的加权平均，每个动作对应的$Q(s,a)$的权重由相应动作在策略$\pi(s,a)$中的概率决定。动作值函数$Q(s,a)$有类似形式：

$$
\begin{aligned}
Q_\pi(s,a) &= \sum_r rP(r|s,a)+\gamma\sum_{s'}P(s'|s,a)\sum_{a'}\pi(s',a')Q_\pi(s',a') \\
&= \sum_r rP(r|s,a)+\gamma\sum_{s'}P(s'|s,a)V(s'),
\end{aligned}
$$

其中第二步推导应用了式 10.11。

不考虑计算资源的限制，基于上述贝尔曼公式可以对任何固定的MDP和固定策略π计算值函数，完成策略估值任务。同时，基于GPI算法，可以基于贪心原则对策略π进行改进。因此，我们可以从一个随机策略开始，迭代更新值函数和策略，最终得到最优策略。

10.3.3 策略估值：动态规划算法

利用贝尔曼公式，我们可以用解线性方程组的方法确定值函数$V(s)$或$Q(s,a)$。然而在实际应用中，如果状态数较大或可选动作较多，解这一方程组会遇到计算上的困难。一种解决方法是基于贝尔曼公式的递归性质，利用迭代法对值函数进行求解。

以状态值函数为例，依公式10.9，如果我们已知一个状态s的所有子状态s'的值函数$V(s')$，则可直接求得$V(s)$。困难在于精确的$V(s')$ 是未知的，因此求得的$V(s)$是不精确的，尽管如此，通过公式10.9重新计算的$V(s)$也比原来精确。这启发我们用一种迭代求解的方法，先对值函数做个初始估计，再利用公式10.9所示状态之间的递归关系进行逐步求精，求精公式可表示为：

$$V(s) \leftarrow \sum_a \pi(s,a) \sum_r rP(r|s,a) + \gamma \sum_a \pi(s,a) \sum_{s'} P(s'|s,a) V_\pi(s'). \tag{10.9}$$

上式表明计算状态s的时候，基于所有相关状态s'的当前值，并加入当前动作产生的即时收益，事实上是一种动态规划（Dynamic Programming, DP）算法。可以证明这一迭代过程收敛于实际状态值函数$V_\pi(s)$。

上述DP算法具有如下性质：首先，对当前状态s的计算是基于下一个状态s'的信息，因此是一种信息的递归式后向传递，这在强化学习中一般称为回溯（Backup），公式 10.3.3亦称为回溯公式。

类似的，动作值函数$Q(s,a)$也可利用DP算法进行迭代求解，其回溯公式为：

$$Q(s,a) \leftarrow \sum_r rP(r|s,a) + \gamma \sum_{s'} P(s'|s,a) \sum_{a'} \pi(s',a') Q_\pi(s',a').$$

回溯是强化学习的基本概念，回溯方法上的不同是各种学习方法的主要区别。一般用图来形象化表示回溯过程，称为回溯图。在图中，空白圆圈表示状态s，黑色实心圆表示状态-动作对(s,a)，空心方块表示结束状态，连接表示依赖关系。图 10.7 给出基于DP对值函数的回溯图。从图上可知，DP算法的回溯具有如下性质：（1）DP只进行一步回溯，因此也称为浅度回溯（Shallow Backup）。相应的，后面我们要介绍的蒙特卡洛（MC）等方法进行多步回溯，一般称为深度回溯（Deep Backup）；（2）这一回溯是基于当前不精确的$V(s')$对$V(s)$进行估计。这种基于不精确信息进行迭代估计的方法称为Booststraping；（3）回溯中考虑所有可能的后续状态和动作，这种回溯

称为全回溯（Full Backup），相应的，后面要介绍的MC方法和TD方法在回溯时仅考虑采样出的动作和状态，因而称为采样回溯（Sampling Backup）。

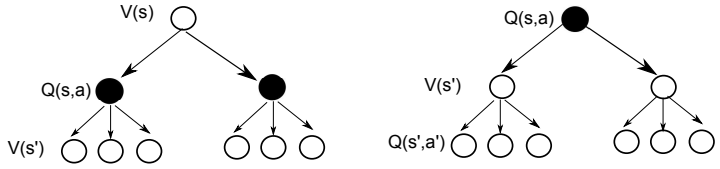

Fig. 10.7 DP算法对（左）状态值函数和（右）动作值函数的回溯图。

10.3.4 策略优化：策略迭代和值迭代

前面提到过，基于通用策略迭代（GPI）框架，通过策略估值和策略改进，最终可以得到在当前MDP下的最优策略。算法 1给出策略迭代过程。

```
Initialize V(s), π(s,a)  ∀s,a;
while True do
    Policy Evaluation:
    while True do
        Δ = 0;
        for s ∈ 𝒮 do
            v = V(s);
            V(s) = Σ_{s',r} P(s',r|s,π(s))[r + γV(s')];
            Δ = max(Δ, |v − V(s)|);
        end
        if Δ < δ then
            Break while;
        end
    end
    Policy Improvement:
    Conv = True;
    for s ∈ 𝒮 do
        Act = π(s);
        π(s) = arg max_a Σ_{s',r} P(s',r|s,a)[r + γV(s')];
        if Act ≠ π(s) then
            Conv = False;
        end
    end
    if Conv=True then
        Break while;
    end
end
```

Algorithm 1: 基于DP的策略迭代算法，其中δ是控制策略估值精度的小量。

注意在上述策略迭代算法中，每一次策略估值包含一个到收敛的值函数求精过程，会消耗大量计算。事实上是当值函数迭代若干次后，策略估值通常就变化不大了。这意味着我们可以对策略估值进行简化，只保留少数几次迭代。依GPI框架，这样简化过的迭代依然会收敛到最优策略。如果我们将策略估值的迭代减少为一轮，则对所有状态更新一次后即进行策略改进，这

一算法称为值迭代算法（Value Iteration）。以状态值函数为例（动作值函数有类似形式），回溯公式为：

$$\begin{aligned} V(s) &\leftarrow \sum_a \pi(s,a) \sum_r rP(r|s,a) + \gamma \sum_a \pi(s,a) \sum_{s'} P(s'|s,a)V(s') \\ &= \max_a \sum_r rP(r|s,a) + \gamma \max_a \sum_{s'} P(s'|s,a)V(s') \\ &= \max_a \sum_{r,s'} P(r,s'|s,a)[r+\gamma V(s')]. \end{aligned} \tag{10.10}$$

其中第二步推导应用了贪心原则。注意在这一算法中，策略提升并没有显示出现，而是隐含在值函数的更新公式中。算法 2给出值迭代算法的伪代码。较一般的策略迭代，值迭代更早利用值函数的更新结果，因此一般效率更高。然而，策略迭代是更通用的学习框架，可通过选择合理的估值和优化方法，满足多种优化任务。

Output: $\pi(s,a)$
Initialize $V(s) \quad \forall s$;
Value Iteration:
while *True* **do**
    $\Delta = 0$;
    for $s \in \mathscr{S}$ **do**
        $v = V(s)$;
        $V(s) = \max_a \sum_{s',r} P(s',r|s,a)[r+\gamma V(s')]$;
        $\Delta = \max(\Delta, |v - V(s)|)$;
    end
    if $\Delta < \delta$ **then**
        Break while;
    end
end
Policy Generation:
$$\pi(s,a) = \begin{cases} 1 & if\, a = \arg\max_a \sum_{s',r} P(s',r|s,a)[r+\gamma V(s')] \\ 0 & otherwise \end{cases}$$

Algorithm 2: 基于DP的值迭代算法，其中δ是控制策略估值精度的小量。

策略迭代和值迭代既可以是同步的，也可以是异步的。同步迭代算法用一个临时空间保留所有已更新过的状态，待所有状态更新之后，再用临时空

间保存的状态值统一更新$V(s)$或$Q(s,a)$。如果状态数非常大，则完全更新一次需要很长时间，在所有状态更新之前无法利用本轮迭代中已经更新过的状态。异步更新算法允许即时更新，即一个状态的值更新后，这一更新立时生效，并可在本轮迭代中用于更新其它状态。

异步更新与状态的选择顺序相关，可以顺序选择，也可以随机选择。一种比较高效的选择方法是依据贝尔曼公式上的误差，误差越大的状态其前趋状态被选择的概率越大，这一方法称为Prioritised Sweeping。如果是实际系统，还可依实际交互中的状态顺序进行更新，一般称为实时动态规划（Real-Time DP）。

10.4 蒙特卡洛方法

10.4.1 学习任务与采样方法

前一节讨论的动态规划（DP）方法是一种全回溯值函数计算方法。这一方法之所以可进行全回溯，是因为环境动态模型已知。因此，DP方法只能用在规划任务中，且环境模型为MDP。现实中的大多数问题是学习任务，在这种任务中环境动态性未知，因此优化策略必须通过和环境交互来逐渐学习。

前面已经说过，学习可在模型、值函数、策略三个层面进行，定义如下：

- 模型学习：通过与环境交互学习一个MDP，再依前节所述的动态规划方法来得到优化策略；
- 值函数学习：不建立环境模型，通过交互直接学习值函数$V(s)$或$Q(s,a)$，依此得到优化策略。
- 策略学习：直接学习策略$\pi(s,a)$，一般基于函数近似方法。

与环境交互的学习任务必须通过采样来完成，即通过和环境主动交互，生成多个交互过程，通过这些过程对模型、值函数或策略进行学习。如果采样量足够大，覆盖足够全面，得到的样本分布即可模拟环境的动态性。因此，采样可以认为是一种对复杂环境进行模拟的方法，在强化学习任务中具有重要意义。围绕采样过程衍生出一系列问题，特别是探索性和实用性的权衡（Exploration-Exploitation Trade-Off）。

值得注意的是，采样方法是一种通用方法，不仅在环境模型未知的学习任务中是必须的，在模型已知的规划任务中也经常用到，特别是当模型比较

复杂时，基于DP求解需要考虑所有状态，学习效率低。利用采样方法可以将观注点集中在重要状态上，因而可提高算法效率。这种效率提高的代价是当采样出现偏差时，有可能会偏离最优策略。

本节介绍的蒙特卡洛方法（Monte Carlo, MC）即是一种简单的值函数采样学习方法，该方法不对环境做任何模型假设，单纯依靠采样来学习。值得说明的是，采样法在模型学习、策略学习、优化搜索等亦有广泛应用，本节我们重点关注基于MC的值函数学习，同时简单讨论基于采样的决策搜索，称为MC搜索。

10.4.2 蒙特卡洛策略估值

我们首先考虑基于MC的策略估值任务（不考虑策略更新），即给定策略π，求值函数$V_\pi(s)$ (动作值函数$Q_\pi(s,a)$的估计将在稍后讨论)。回到状态值函数定义：

$$V_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + ..\gamma^{T-t-1}R_T|S_t = s],$$

其中$\mathbb{E}_\pi(\cdot)$对所有基于策略π生成的交互过程求期望，包括所有以s为起始状态的序列。上节我们基于DP对$V_\pi(s)$进行迭代求解，事实上更简单的方法是依概率采样所有可能的交互序列，并依上式求所有路径长期收益的均值即可得到$V_\pi(s)$。这一方法称为蒙特卡洛（MC）方法。图 10.8 给出MC和DP的回溯图。从图中可以看到，MC和DP的回溯过程主要有两点区别：一是DP是依模型的全回溯，而MC是依采样的单路径回溯;二是MC是从起始状态到结束状态的全路径回溯，而DP是一步回溯，一步之后的路径收益由下一状态的值函数$V(S_{t+1})$得到，即Bootstraping方法。

相比DP方法，MC方法不依赖一个环境模型，因此可以学习未知环境（如果环境模型已知，也可以基于环境模型进行采样，利用得到的样本进行MC学习）。同时，MC方法不对环境做任何假设，因此适合学习非MDP的交互过程。另外，MC方法对状态的估计是独立的，这和DP方法以及后面要介绍的TD方法明显不同，后者基于Bootstraping，因此不同状态的估值是互相影响的。这一特性在状态空间特别庞大的问题上特别有价值，因为我们可以在最有价值的状态点上生成更多采样，而不必考虑其它状态的估计精确度。

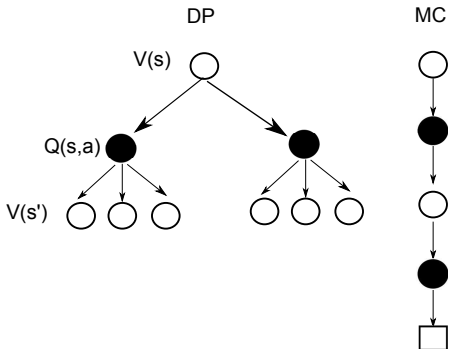

Fig. 10.8 DP和MC状态值函数估值过程回溯图。DP是一步全回溯，是Bootstrapping方法；MC是深度采样回溯，非Bootstrapping方法。

MC方法也有一定局限性。首先，MC方法基于一个完整的交互过程，因此只适用于在有限时间内明确结束的多轮任务。其次，MC方法一般具有较强的不稳定性，因为一个采样过程需要很多随机步骤，累加起来增加了不确定性。三是状态独立估计虽然可以避免关注某些不重要的状态，但状态之间无法共享学习，导致效率下降。下节要介绍的时序差分（TD）方法将模型和采样结合起来，可部分解决不确定性的问题，且可以处理连续任务。

虽然MC是全路径回溯，对状态值函数进行回溯时一般采用增量法，即在一个交互过程中的每个时刻t都对$V(S_t)$进行一定量更新。注意$V(S_t)$是所有收益的平均值，因此回溯公式可写成如下形式：

$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t)),$$

其中$N(S_t)$为到目前为止，包括所有交互过程中状态s的访问次数。由上式可见，$V(S_t)$ 的增量部分是实际收益G_t与当前对该收益估值（即当前$V(S_t)$）之间的偏差。该偏差随着累积访问次数$N(S_t)$的增加越来越小，对$V(S_t)$的调整贡献相应减小，最终趋近于零，得到的$V(S_t)$ 即是G_t的期望。增量MC可以及时得到环境动态特性的部分信息，这对于提高采样效率，加快策略优化至关重要。同时，增量MC可以及时反映环境变化，对于时变系统的学习很有价值。注意在上述公式中，如果一个交互过程的采样对某一状态s重复访问，则每次访问得到的收益都会被用来更新$V(s)$。这种回溯方式称为‘多次访问MC’。如果对一个交互过程只考虑某一状态在第一次出现时的收益，称为‘首次访问MC’。

10.4.3 蒙特卡洛策略优化

基于通用策略迭代（GPI）框架可实现基于MC的策略优化，其中策略估值基于MC，策略改进基于贪心原则。如果环境模型已知，策略改进可以基于值函数$V_\pi(s)$，如果环境模型未知，则只能基于动作值函数$Q_\pi(s,a)$。在绝大多数实际任务中，环境模型是未知的，因此对$Q_\pi(s,a)$的采样估计显得尤为重要。

基于MC对动作值函数进行估值需要考虑采样覆盖度问题。为了估计$Q_\pi(s,a)$，需要对所有(s,a)组合进行采样。基于GPI框架，当对策略进行估值的时候需要基于该策略进行采样，生成交互序列。而策略优化任务中的策略都是基于贪心原则的确定性策略，即对任何一个状态只有某一个动作可选，其结果是$Q_\pi(s,a)$中只有一个动作是可以被覆盖的，其余动作的概率为0，这意味着动作值函数无法得到有效估值，策略也就无法改进。

解决这一问题的一个简单方法是采样时考虑所有可能的状态-动作对(s,a)作为起始状态。这一方法可保证覆盖性，但可能遇到无法自由选择起始状态的情况。另一个可能的方法是与环境交互过程中引入随机性，不仅选择$Q_\pi(s,a)$最大的动作，也允许其它动作以某一概率ε方式出现，这称为ε-贪心策略，形式化表示如下：

$$\pi(a|s)=\begin{cases}1-\varepsilon+\frac{\varepsilon}{|\mathscr{A}(f)|} & \text{if } a=\arg\max_{a\in\mathscr{A}} Q_\pi(s,a)\\ \frac{\varepsilon}{|\mathscr{A}(f)|} & otherwise\end{cases} \tag{10.8}$$

其中$\mathscr{A}(s)$为在状态s下可能选择的动作集合。可以证明，基于ε-贪心策略的GPI算法将收敛到ε随机条件下的最优策略π。值得注意的是，ε越小，ε-贪心策略越接近于贪心策略，这意味着如果我们在优化过程中逐渐减小ε，将越来越趋近到全局最优策略。上述收敛性以值函数$Q_\pi(s,a)$的充分估计为前提，因此在实际应用中需要大量采样以保证每个状态-动作对被充分访问。

另一种增加覆盖度的方法是将待优化的策略（称为目标策略, Target Policy）和生成动作的策略（称为动作策略, Action Policy）分开，利用较随机的动作策略与环境交互，基于生成的交互过程对目标策略进行优化。这种方法称为Off-Policy方法。

Off-Policy 这一方法可基于Importance Sampling实现。设两个概率分布π和μ，某一函数$f(x)$基于这两个分布的期望有如下关系：

$$\mathbb{E}_\pi f(x)=\mathbb{E}_\mu \frac{\pi(x)}{\mu(x)} f(x). \tag{10.8}$$

上式表明基于分布π的期望可由基于另一分布μ的期望得到，只需对原函数乘以一个重要性因子$\frac{\pi(x)}{\mu(x)}$。如果用采样均值代替期望，我们可以用基于某一分布μ的采样来估计基于另一分布π的期望。这种方法称为Importance Sampling。

在基于Importance-sampling的MC方法中，设π为目标策略，μ为动作策略。记策略μ生成的一个交互过程如下:

$$\lambda = S_t, A_t, R_{t+1}, S_{t+1}, ..., S_T.$$

注意该过程由t时刻开始，以结束状态S_T完成，是某一个完整交互过程的子序列。依π和μ计算这一交互过程的概率，分别为：

$$P_\pi(\lambda) = \prod_{k=t}^{T} \pi(A_k|S_k) p(S_{k+1}|S_k, A_k),$$

$$P_\mu(\lambda) = \prod_{k=t}^{T} \mu(A_k|S_k) p(S_{k+1}|S_k, A_k).$$

其中$p(S_{k+1}|S_k, A_k)$为系统模型决定的状态转移概率。对策略π的动作值函数可写为：

$$\begin{aligned} Q_\pi(s,a) &= \mathbb{E}_\pi G_t \ \ s.t. \ \ S_t = s, A_t = a && (10.9)\\ &= \mathbb{E}_\mu \frac{P_\pi(\lambda)}{P_\mu(\lambda)} G_t \ \ s.t. \ \ S_t = s, A_t = a && (10.10)\\ &= \mathbb{E}_\mu \rho_t^T(\lambda) G_t \ \ s.t. \ \ S_t = s, A_t = a && (10.11) \end{aligned}$$

其中第二步推导利用了Importance-sampling的公式 10.4.3，第三步推导中的ρ_t^T定义了基于π和μ的交互路径概率比，如下：

$$\begin{aligned} \rho_t^T(\lambda) &= \frac{\prod_{k=t}^{T} \pi(A_k|S_k) p(S_{k+1}|S_k, A_k)}{\prod_{k=t}^{T} \mu(A_k|S_k) p(S_{k+1}|S_k, A_k)} && (10.12)\\ &= \prod_{k=t}^{T} \frac{\pi(A_k|S_k)}{\mu(A_k|S_k)}. && (10.13) \end{aligned}$$

可见该比值只与这两个策略的动作分布函数有关，与状态转移概率无关，意味着该方法不受环境条件的限制。另外注意为使ρ_t^T有意义，在分子不为零的时候分母不能为零，这意味着对目标策略π允许的动作，动作策略μ 必须在

该动作上有概率分布。事实上，引入μ的主要目的是解决对π进行估值时的状态-动作覆盖度问题，所以μ的随机性一般都要大于π，覆盖π的状态-动作空间只是基本要求。

实际系统中，对动作值函数的更新一般采用增量方式，写成回溯公式如下：

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \frac{\rho_t^T}{C(s)}(G_t - Q(S_t,A_t)) \tag{10.14}$$

其中$C(s)$是状态s的加权访问量，如下：

$$C(s) = \sum_{\lambda} \rho_t^T(\lambda) \quad s.t.\ S_t(\lambda) = s$$

其中$S_t(\lambda)$ 给出交互过程λ的初始状态。

上述Off-policy估值方法可应用在GPI算法中实现对策略π的优化。具体过程是：在每一次迭代中，基于某一已知策略μ进行采样，依公式 10.11估计当前策略π的值函数，再依贪心原则对策略π进行改进。注意，在公式10.13中，$\pi(A_k|S_k)$的取值是贪心的，即：

$$\pi(A_k|S_k) = \begin{cases} 1 & \text{if } A_k = \arg\max_{a \in \mathscr{A}} Q_\pi(S_k,a) \\ 0 & otherwise \end{cases} \tag{10.14}$$

Output: $\pi(s,a)$

; Initialize $Q(s,a);\pi(s,a)=Greedy(Q);C(s,a)=0;$

for *Each Iteration* **do**

  Sample $S_0,A_0,R_1,...,R_t,S_T$ by μ;

  W = 1;

  G = 0;

  for *t:=T-1 to 0* **do**

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

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

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

    $$\pi(S_t,a)=\begin{cases}1 & a=\arg\max_a Q(S_t,a)\\ 0 & otherwise\end{cases}$$

    if $\pi(S_t,a)\neq 1$ **then**

      Break;

    end

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

  end

end

Algorithm 3: 基于MC的Off-Policy策略优化算法。

实际算法实现时，为提高对每一个采样过程的利用率，可采取从采样的结束状态开始往前回溯，对采样路径上的所有状态依次更新，直到出现使$\pi(A_k|S_k)$为零的A_k选择。算法 3给出增量Off-policy策略优化过程。一个值得思考的问题是：由算法 3可见，只有满足π的路径才会在更新$Q(s,a)$时产生贡献，那么选择另一个策略μ还有什么意义吗？事实上，选择更具探索性的策略μ可以产生更多有价值的路径。假设起始状态固定，则基于π的所有采样产生的路径都是一样的，但基于μ可以产生其它子路径，这些路径依π是有效的（因此在算法中也会参与更新值函数），但如果不引入随机化将永远无法被访问到。因此，引入μ的意义在于提高状态访问的覆盖性，从而得到更好的值函数估计，用以改进策略。然而，只能选择符合π的路径导致很大一部分依μ生成的路径实际上没有有效利用，导致学习效率下降。后面要介绍的时序差分（TD）算法可以很大程度上克服这一问题。

从某种意义上讲，ε-贪心算法也可以认为是一种Off-Policy方法，因为生成动作的策略（ε-贪心策略）和要优化的目标策略（贪心策略）是不同的策略。与标准Off-Policy不同的是，ε-贪心算法在计算值函数时并没有依ρ_t^T对收

益进行加权，因此只能收敛到最优ε-贪心策略而非真正的全局最优策略。不论如何，这两种方法在思路上是类似的，即通过在动作选择上增加随机性来提高MC采样的状态-动作覆盖性，以得到更精确的值函数估计。

10.4.4 蒙特卡洛搜索

前们我们介绍了基于MC的值函数学习，事实上该方法和启发式搜索有很密切的关系。启发式搜索是人工智能领域的基本方法。考虑一个路径优化任务，当任务执行到某一状态s时（如最短路径搜索中的某一个结点），为决定下一步走向，对所有可能的后续位置s'，通过穷举式搜索直到终点，即可得到最优的s'。显然，这种穷举搜索对计算开销很大，在复杂问题上无法应用。启发式搜索对每个状态人为定义一个启发函数$h(s)$表示从状态s到终点的最小路径，当搜索时到达状态s时，由s经由s'到达终点的路径长度即可用下式表示：

$$f(s,s',\tilde{s}) = g(s,s',\tilde{s}) + h(\tilde{s})$$

其中$g(s,s',\tilde{s})$表示由s经过s'到达$\tilde{s}$的最短路径。选择$f(s,s',\tilde{s})$最小s'即得到下一步的优化走向。注意$h(s)$是一个人为指定的最短路径估计值，因此依上式计算得到的f是实际最小路径长度的近似值。一般想况下，$\tilde{s}$越接近终点，$h(\tilde{s})$的估计越精确，f的准确度越高。这意味着在启发式搜索中，搜索深度越深，得到全局最优的可能性越大。

如果我们用基于MC的强化学习方法来解决上述最短路径问题，用状态值函数$V(s)$代表从状态s到终点的最短路径，基于MC对交互过程进行采样，当以状态s为起始状态的某条采样路径比$V(s)$小时，则更新$V(s)$为该路径的长度。经过多次采样，$V(s)$即可接近最优估计。注意这里同样存在采样的覆盖性问题，可用前面所述的ε-贪心策略或Off-Policy 方法解决。得到最优值函数$V(s)$之后，可采用贪心策略得到最优路径。

比较MC方法和启发式搜索，可以看到前者通过采样估计启发函数，后者通过人为指定。在复杂任务中，通过学习得到启发函数显然更有优势，但计算量更大。另一方面，MC方法中在路径决策中所采用的贪心策略相当于以$V(s)$为启发函数的一步启发式搜索。

理论上，当$V(s)$的估计足够优化时，MC的一步搜索已经最优策略，因此不需对更深的路径进行搜索。但是，当$V(s)$的估计不准时，贪心策略不能保证最优。这类似在启发式搜索中，启发函数$h(s)$不精确时，贪心策略有可

能得不到最优路径。为解决这一问题，可以利用启发式搜索中增加搜索深度的办法，基于采样搜索当前状态下可能的路径。搜索的深度越深，$V(\tilde{s})$估计的准确度越高，得到优化路径的可能性越大。这种搜索称为蒙特卡洛搜索（MC Search）。

在强化学习方法中，对值函数的估计和对路径的深度搜索都称为MC方法，但前者是MC值函数学习，后者是MC路径搜索。MC搜索也可认为是只回溯到当前状态结点的MC值函数学习，但这种学习是临时性的，一旦决定了如何动作，则该学习得到的临时值函数$\tilde{V}(s')$,其中s'为当前结点的所有子结点,将被清空并在下一步搜索时重新学习。

在系统实现上，学习和搜索亦可完全分开，如利用时序差分（TD）法对值函数进行学习，再基于MC进行路径搜索。AlphaGo即采用了这一方法，在模型训练阶段基于TD估计值函数，在对奕阶段基于MC构造搜索树（称为蒙特卡洛树，MCT），选择最有价值的落子方式。

10.5 时序差分法

上节讨论的蒙特卡洛方法原则上可学习任何复杂环境，但其采样需要一个完整的交互过程，导致学习效率较低，不确定性较大。本节介绍的时序差分（TD）方法是一种部分采样方法，既保留了采样方法的灵活性，也利用了模型知识，具有高效、稳定的特点，是强化学习中应用最广泛的方法。TD方法起源于如下基本思路：如果不同时刻对某个状态的估值存在差异，则该差异可作为学习信号对该状态和相关状态进行重估。从这个意义上说，TD是基于采样的Bootstrapping方法，因此可认为是DP（Bootstrapping的）和MC（采样的）两种方法的结合。事实上，TD具有更深刻的认知学和生理学基础：最新认知学研究表明，动物和人类的很多学习方式都是基于TD的，神经学研究也表明，人类大脑里处理信息的方式很大程度上是TD模型。本节首先介绍一步TD 算法，之后将其扩展到N-step TD算法和TD(λ)算法，这些算法和MC更为相似。注意的是，本节我们只关注基于TD的值函数学习，对于模型学习和策略学习，即使应用了TD的某些思路，也不是我们要讨论的TD算法。

10.5.1 基于*TD*的策略估值

首先考虑策略估值任务，且只考虑状态值函数$V(s)$（动作值函数$Q(s,a)$将在优化任务中讨论）。回到状态值函数的定义，对策略π的估值基于如下基础公式：

$$V_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s], \tag{10.14}$$

其中$\mathbb{E}_\pi$对以策略π进行交互的所有可能状态序列求期望，G_t为以S_t为起始状态的某一交互序列的长期收益。只考虑折扣收益，对于DP算法，G_t计算如下：

$$G_t = R_{t+1} + \gamma V_\pi(S_{t+1}).$$

上式意味着对一个状态的更新利用了其他状态的知识，因此是Bootstrapping的。这种状态间的协同学习可极大提高学习效率。DP算法的缺点是需要依赖已知的环境模型来计算估值公式 10.5.1中的期望，因此无法在实际交互任务中应用。

MC方法基于采样来计算公式 10.5.1中的期望，因此不受环境模型的制约，可在实际交互任务中应用。MC的长期收益G_t的计算如下：

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...\gamma^{T-1} R_T,$$

其中T是一轮采样到达结束状态的时间。可见，MC采样必须是一个完整交互过程，因此只能应用在多轮任务中，对连续任务无法应用；另外，MC对每个状态是独立学习的，互相没有借鉴，学习效率不高。

TD方法基于公式 10.5.1对策略进行估值，但采用类似MC的采样方法来计算期望。从DP算法角度看，TD方法用采样取代了环境模型，由全回溯变成采样回溯；从MC角度看，TD 方法用一步采样代替全过程采样，并基于Bootstrapping方法估计当前状态的值函数。这一区别可从图 10.9所示的回溯图上清楚看到。

形式上，TD的长期收益G_t计算和DP的计算方法一致，都是Bootstrapping方法：

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

与DP不同的是，在DP中，期望$\mathbb{E}_\pi$由环境模型直接计算得到，而TD基于采样实现，采样的函数分布为策略$\pi(s,a)$。基于采样方法，TD可用于实际交互任

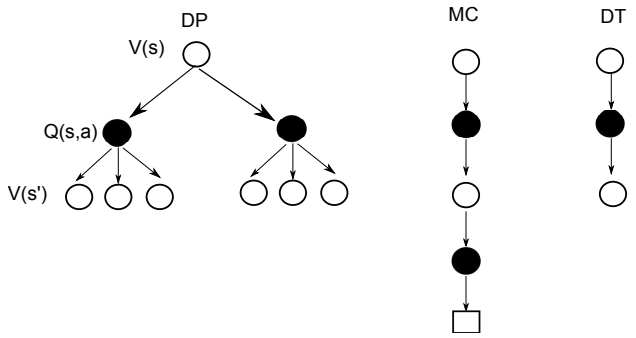

Fig. 10.9 DP, MC和TD三种策略估值算法的回溯图。DP算法是一步全回溯，MC是深度采样回溯，TD是一步采样回溯。DP和TD是Bootstrapping方法。

务，且可进行在线学习。程序实现时，对每个采样$(S_t, A_t, R_{t+1}, S_{t+1})$的回溯可采用增量方式，回溯公式如下：

$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V_\pi(S_{t+1}) - V(S_t)), \tag{10.14}$$

其中α为学习率，$R_{t+1} + \gamma V_\pi(S_{t+1})$为基于当前采样对$S_t$的估计，$R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$称为TD误差。上式意味着凡是当前采样和以往经验出现偏差的时候，该信息将从S_{t+1}向S_t回溯，从而引起S_t值函数的改变。注意上式和MC的增量实现公式很类似，只不过在MC中对$V_\pi(S_t)$的估计为G_t，因而MC误差为$G_t - V_\pi(S_t)$。

算法 4给出一个多轮任务中的TD实现过程。理论证明，对任意一个固定的策略π，当α 足够小时，TD算法将收敛到π的值函数。

Input: π
Output: $V(s)$
Initialize $V(s)$;
for *Each Episode* **do**
  Init S;
  while *S not terminal* **do**
    Sample A by π for S;
    Take A, observe S', R;
    $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$;
    S = S';
  end
end

Algorithm 4: 多轮任务中的TD策略估值算法。

TD和MC都是基于采样的估计方法，但TD只进行一步采样，对余下步骤用Bootstrapping方法进行估计。在策略估值任务中，这两种方法都可收敛到目标策略的值函数，实际中TD收敛速度一般更快。另外，TD依赖MDP假设，在有限采样数据条件下，可扩展性更强，而MC 没有这一假设，在非MDP过程中可能会表现更好。

10.5.2 基于*TD*的策略优化

前面我们讨论了基于TD的策略估值方法，结合通用策略迭代（GPI）框架，可完成策略优化。与MC方法类似，TD方法不依赖环境模型，因此需要基于动作值函数$Q(s,a)$进行策略估计和策略优化。

考虑一个交互过程如下：

$$S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2} \ldots.$$

取每个(S_t, A_t)作为一个Q函数的采样值，可得到如下回溯公式：

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

上式中，$R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$是依当前经验对$Q(S_t, A_t)$估计，这一估计与依既往经验的估计$Q(S_t, A_t)$不同时，即产生TD误差信号，依此对$Q(S_t, A_t)$进行回

溯。这一回溯公式中包括如下五个参数：$S_t, A_t, R_t, S_{t+1}, A_{t+1}$,因此称为Sarsa算法。

如果保持策略π不变，Sarsa可用于对该策略进行估值。如果希望对策略进行优化，则需基于$Q(s,a)$应用贪心原则进行策略改进。这里我们面临MC中同样的状态空间覆盖度问题。和MC的解决方法类似，可基于当前$Q(s,a)$的ε-贪心策略作为动作策略生成交互过程，再基于Sarsa进行回溯更新。当ε逐渐减小时，可实现策略优化。这一过程中，动作策略和目标策略是一致的（都是基于当前$Q(s,a)$的ε-贪心策略），因此Sarsa是On-Policy方法。也可以设计Off-Policy的Sarsa算法，只需选择其它动作策略，并基于Importance Sampling公式进行权重修正即可。

Sarsa的一种改进是在回溯时考虑当前采样状态S_{t+1}下所有可能动作的$Q(s,a)$，基于其期望（而非采样动作A_{t+1}对应的$Q(S_{t+1},A_{t+1})$）来计算TD误差，回溯公式如下：

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1},a) - Q(S_t,A_t)],$$

其中$\pi(s,a)$在策略估值任务中是待估值策略，在策略优化任务中是基于当前$Q(s,a)$的ε-贪心策略。这一方法称为Expected-Sarsa算法。基于$Q(S_t,a)$的期望事实上减少了对动作采样的依赖，对TD误差的估计也更合理。和Sarsa一样，Expected-Sarsa也是On-Polilcy方法，并可拓展到Off-Policy方法。注意Expected Sarsa只对动作全回溯，而DP中对状态和动作都进行全回溯。这是因为状态转移规律由环境决定，而Expected Sarsa中环境未知，因此无法对状态求期望；但可能采取的动作由策略决定，因此可求其期望。

对策略优化任务，如果我们用ε-贪心策略作为动作策略，但将目标策略改为贪心策略，则得到一种广泛应用的TD优化方法，称为Q-learning，其回溯公式如下：

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

上式中的最大化操作事实上是基于当前$Q(s,a)$的贪心策略。同Sarsa一样，当动作策略的ε逐渐减小时，Q-learning收敛到最优策略。注意在Q-learning中，动作策略和目标策略不一致，因此是一种Off-Policy方法，但当ε逐渐减小时，动作策略趋近目标策略，因此可收敛到最优。最后要提醒的是，Q-learning是一种策略优化方法，不能用作策略估值，这是因为回溯公式中的最大化操作

已经决定了目标策略。这和Sarsa及Expected Sarsa既可用于策略估值，也可用于策略优化不同。

图 10.10给出Sarsa、Expected Sarsa和Q-learning的回溯图。可以看到三者（在策略优化任务中）的不同：Sarsa考虑采样动作，Expected Sarsa考虑所有可能的动作，而Q-learning考虑最大收益动作。

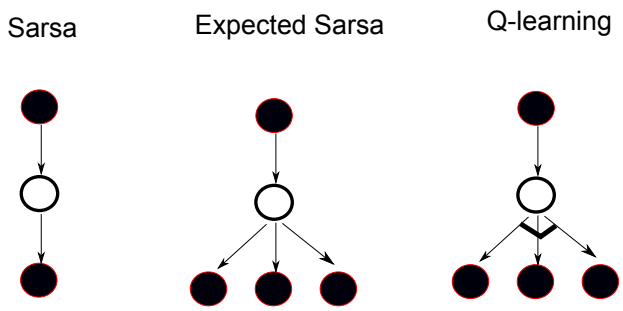

Fig. 10.10 几种典型TD策略优化算法回溯图。Sarsa（左）基于当前采样路径回溯；Expected Sarsa（中）在回溯时考虑当前采样结点后续所有可能的动作；Q-learning（右）考虑当前采样结点后续的最大收益动作（用一个小圆弧表示最大化操作）。

10.5.3 N-step TD 与TD(λ)

在蒙特卡洛搜索一节中我们讨论过，由于MC对策略估值是不精确的，在实际策略执行过程中如果能往目标端多搜索几步，则可能得到更好的动作选择，因为越接近目标，状态估值越准确。

基于类似思路，如果我们让TD往前多看几步，则对当前收益的估计会更准确，对状态的回溯也更合理。本节讨论基于多步采样的TD方法。为简明起见，我们只考虑策略估值算法，策略优化算法可由相应估值算法结合GPI框架及上节所介绍的若干方法得到。

设进行n步采样，则得到长期收益的估计为：

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

注意最后一项是n步之后对应状态的值函数估计。如果$n=1$，则该收益简化为传统TD估计；如果n取无穷大直至结束状态，则该收益等同于MC收益。

基于N步收益的TD算法称为N-step TD，其值函数回溯公式如下：

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t^n - V(S_t))$$

注意$n=1$时，N-step TD 简化为传统TD方法，$n=\infty$时，简化为传统MC方法。可见，传统TD和MC都可以看作N-step TD的特例，是采样为一步和无穷步（直到结束状态）的特殊情况。

在N-step TD 中考虑不同的n在折扣收益上的差异，令n越大带来的折扣收益越小，则可得到一个折扣总收益，定义为G_t^λ：

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^n$$

上式λ是折扣因子。注意λ与折扣收益中的γ不同，λ是对G_t^n做折扣，γ是对即时收益R_t做折扣。基于折扣总收益的TD算法称为TD(λ)算法，其状态值函数的回溯公式为：

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t^\lambda - V(S_t)).$$

注意当λ为0，且n为1时，TD(λ)简化为传统TD算法，因此传统TD算法亦被称为TD(0)。

上述对状态值函数进行估值的方法同样可用于对动作值函数$Q(s,a)$估值, 类似TD(0)中的Sarsa和Expected-Sarsa。N-step Sarsa依照采样进行单路径回溯;对于Expected-Sarsa，可在采样结束时求$Q(s,a)$对所有动作a的期望并沿采样路径回溯，也可在每个采样点求该期望并沿采样路径回溯。前一种方法称为N-step Expected Sarsa，后者称为Tree Sarsa。这两种回溯也可以混合起来，在某些采样点上采用Sarsa方式，某些采样点上采用Expected-Sarsa方式。如果引入一个随机变量σ来控制每一步回溯方式的选择，则每次回溯是随机的，这种方法称为Q(σ)。很明显，$Q(\sigma)$是Sarsa 和Expected Sarsa的扩展。图10.11给出这几种估值方法的回溯图。

基于动作值函数$Q(s,a)$，结合ε-贪心策略，可以实现策略优化。和TD(0)中类似，既可以选择On-policy方式，也可选择Off-policy方式。限于篇幅，这里不再赘述，有兴趣的读者可参考 [11]。

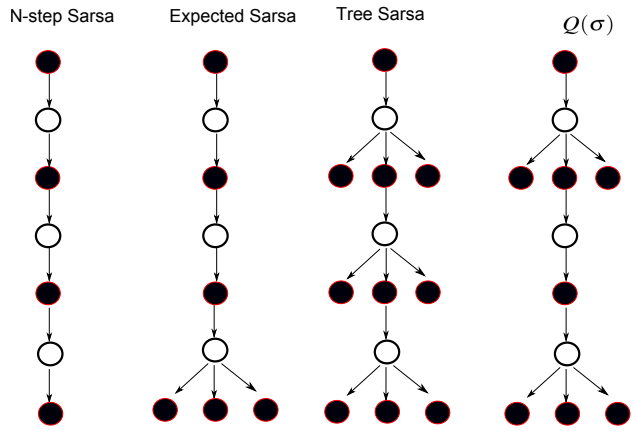

Fig. 10.11 几种多步采样动作值函数估值方法回溯图。N-step Sarsa沿采样路径回溯；Expected Sarsa 在回溯时考虑最后一次采样后的所有可能动作；Tree Sarsa在每个采样点都考虑所有可能动作；$Q(\sigma)$在每个采样点随机选择回溯方式。

10.5.4 三种强化学习方法总结

到目前为止，我们已经讨论了三种主要的强化学习方法，DP, MC和TD。这三种方法基于的假设不同，特性不同，应用的场景不同。图 10.12从回溯的深度和回溯宽度两方面比较了这三种学习方法。从回溯深度上看，DP和TD都是一步回溯，N-step TD和TD(λ)是多步回溯，MC及穷举搜索是全路径回溯。从回溯宽度上，MC和TD、N-step TD、TD(λ)这三种方法基于采样回溯，DP、穷举搜索在都要考虑所有可能后续状态，因此属于全回溯。值得一提的是，Q(σ)在回溯宽度和深度上都具有相当的灵活性，是一个更通用的学习方法。需要注意的是，上述差异主要表现在策略估值任务中。对策略优化任务，这些方法只需和GPI框架结合，在策略改进方法上并没有太大差异。

从应用场景上看，MC和TD这两种方法不依赖环境模型，因此一般称为无模型方法，常用在与实际环境交互的在线学习任务中。然而，这两种无模型方法依然可以用在模型已知的规划任务中。与DP等基于模型的方法不同的是，MC和TD并不会直接利用模型信息，而是基于该模型进行采样，进而估计值函数和优化策略。所以MC 和TD虽然被称为无模型方法，但其更本质特点是其采样方法，而非应用场景中是否有模型。比较MC和TD，这两种方法除了采样深度的差异，更根本的区别在于：TD依赖MDP假设，因此有较强的结构化先验知识，其优化趋向最大似然估计（Maximum Likelihood）；MC不

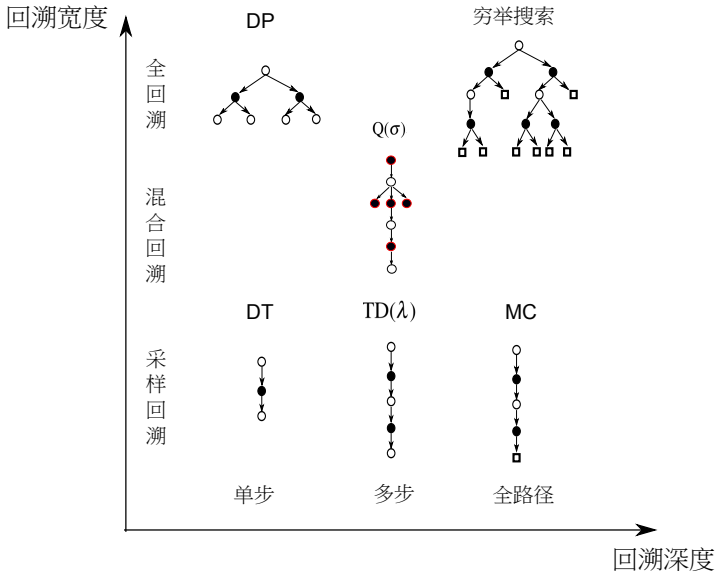

Fig. 10.12 几种深度学习方法在回溯方式上的比较。TD, TD(λ)和MC都基于采样，回溯过程只考虑采样路径；DP和穷举在回溯时考虑所有后续状态或动作，因此称为全回溯。Q(σ) 回溯路径可自由选择，因此最为灵活。从另一个角度，DP和TD是一步回溯，基于Bootstrapping 进行状态估值，MC和穷举搜索都是深度回溯，不基于Bootstraping。TD(λ)和$Q(\sigma)$ 可自由选择采样深度，因此更灵活，适用范围更广。

对环境做任何假设，其优化趋向在训练集上的最小均方误差估计（Minimum Mean Square Error）。因此，环境越近似MDP，TD的假设越趋向合理，TD方法也越有优势。

另一方面，虽然在学习任务中常用MC和TD，但这并不意味着DP在学习任务中没有价值。事实上，强化学习经常通过和实际环境交互学习一个MDP模型，再基于该MDP应用DP、MC、TD等各种方法进行策略估计与优化。这种基于模型学习的方法将在下节讨论。模型学习不论在理论还是实践上都具有重要价值，只有建立起一个内在模型，才能对环境有更深刻的理解，得到的策略才有更强的泛化能力。模型学习事实上是人类摆脱直觉，上升到理性抽象的重要方式。

图 10.13对三种学习方法的应用场景进行了总结。可见，强化学习的任务场景纷繁复杂，需要基于任务目标、学习信号、复杂性等多种因素选择最合适的学习方法。

表 10.1进一步总结了DP、MC和TD这三种典型算法的特点和主要应用领域。注意表中我们用‘蒙特卡洛’仅指值函数学习中的MC方法。在模型学习和

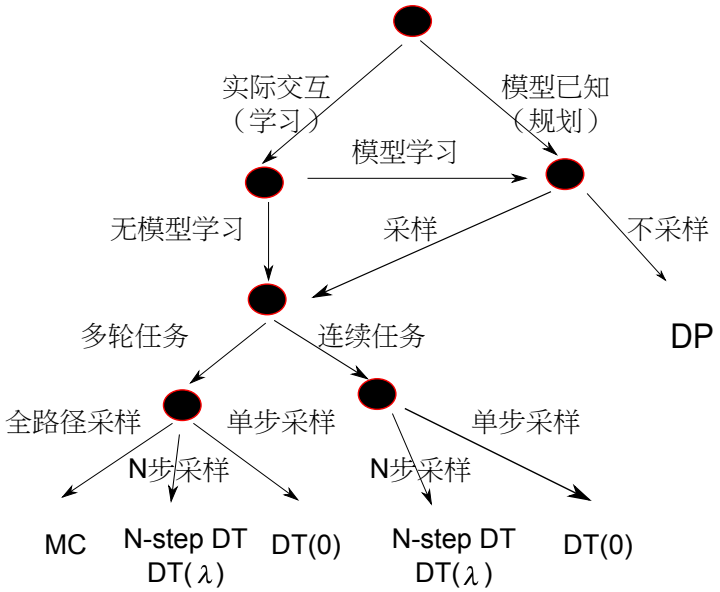

Fig. 10.13 DT、MC、DP三种学习方法的应用场景。

	DP	MC	TD
规划任务	是	是	是
学习任务	是	是	是
多轮学习	是	是	是
连续学习	是	否	是
马尔可夫假设	是	否	是
采样	否	是	是
回溯深度	一步	全路径	一步或多步
回溯宽度	全回溯	采样回溯	采样回溯
Bootstrapping	是	否	是

Table 10.1 三种典型强化学习算法的特点。

策略学习中，经常用到采样方法，这些方法也经常被称为蒙特卡洛方法；广意的蒙特卡洛方法（如Truncated MC）也经常用在连续任务中。事实上，在多步采样中，MC和TD的区别已经相对模糊了。

10.6 模型学习

我们提到在强化学习任务中，可在模型、值函数和策略三个层次进行学习。前面几节介绍的MC和TD方法等主要针对值函数进行学习。本节我们讨论模型学习，并介绍一种模型和值函数混合学习算法，即Dyna。

10.6.1 值函数学习与模型学习

直观上说，值函数学习省去了对环境进行建模的麻烦，不仅简洁优美，而且避免了环境模型偏差造成的先天劣势。例如在DP中，如果模型和实际环境不符合，则再好的学习方法也无法得到好的策略。理论上，如果采样足够丰富，无模型方法（特别是对环境没有任何假设的MC方法）总可以逼近最优策略。这就如同一个孩子经过足够多的锻炼，总可以学会走路、说话等绝大多数生存技能，虽然他/她并没有意识到任何走路、说话模型的存在。既然如此，为什么还要学习模型呢？

这是因为“经验足够丰富”只是理想情况。在绝大多数任务中只允许少量的尝试，依靠这些有限经验很难学习到一个可靠的值函数。例如在股市投资任务中，确实可以通过尝试积累投资技巧，但考虑到资金风险，操作成本，不可能允许我们大量尝试。在这种情况下，少数几次尝试很可能产生对某支股票价值认识上的偏差，导致后续投资的大量损失。值函数学习之所以会面临这一问题，是因为这种学习方法并没有引入较强的结构化信息，因此缺少先验知识，容易过拟合。

模型学习可在一定程度上解决这一问题。这是因为模型存在一定的结构化假设，学习过程仅是对参数的估计，因此可基于少量数据实现较好的学习。如果模型假设与实际环境相符，模型学习一般会比值函数学习高效的多。一旦模型学习完成，则可基于DP、MC、TD等任何一种方法对策略进行估值和优化。

模型学习是人类学习的典型方式。例如在股票操作时，人们一般不会死记硬背哪支股票过去的价格（相当于值函数），而是在操作过程中学习在不同市场环境下的收益和市场走向，前者相当于收益模型$P(r|s,a)$，后者相当于状态转移模型$p(s'|s,a)$。通过建立这些模型，人们在操作股票时才更有方向性，遇到新股票时才能做出更有价值的决策。如果把值函数学习比作直觉记

忆，模型学习更像是认知学习，对人类来说，可能是我们摆脱直觉，上升到理性思维的重要方式。

10.6.2 模型学习方法

模型学习首先定义一个概率模型，再通过与环境交互来学习模型参数。得到模型以后，一般基于该模型进行采样来对值函数进行优化，最后得到优化策略。该过程如图 10.14所示。

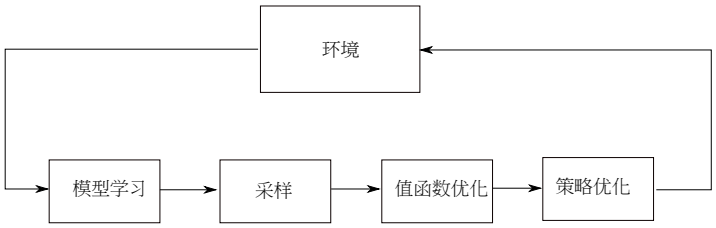

Fig. 10.14 基于模型学习的强化学习方法。通过和环境交互学习一个环境模型，基于该模型进行采样来优化值函数和策略。

如果状态和动作都是有限离散的，一般采用MDP模型。对一个MDP，其参数包括状态转移概率$P(s'|s,a)$和收益概率$P(r|s,a)$。通过和环境进行交互可收集这些参数的统计量，并通过这些统计量对参数进行估计。算法 5给出在连续任务中进行模型在线学习的流程，我们已经假设收益为高斯分布。

Output: $P(s'|s,a), P(r|s,a)$

$T(s,a,s') = 0; R(s,a) = 0; V(s,a) = 0; N(s,a) = 0 \quad \forall s,s',a;$

Initialize s;

for *Each Step* **do**

  Sample $a \sim \pi(s,a)$;

  Sample s', R by Acting a;

  $T(s,a,s') \leftarrow T(s,a,s') + 1;$

  $R(s,a) \leftarrow V(s,a) + R; V(s,a) \leftarrow V(s,a) + R;$

  $N(s,a) \leftarrow N(s,a) + 1;$

  $P(s'|s,a) = \frac{T(s,a,s')}{\sum_{s'} T(s,a,s')};$

  $P(r|s,a) = N(r;\mu,\sigma); \mu = R(s,a)/N(s,a); \sigma^2 = V(s,a)/N(s,a) - \mu^2;$

  $s \leftarrow s';$

end

Algorithm 5: 连续任务中的MDP学习算法。假设收益为高斯分布。

10.6.3 Dyna:混合学习方法

模型学习具有较强的泛化能力，但当环境较复杂，模型假设与之相差较大时，得到的模型会和实际情况产生较大偏差，导致策略无法优化。值函数学习没有模型假设，因此特别适合复杂环境，特别是当数据量较大时，往往具有更好效果。混合学习将这两种学习方法的优势结合起来，既考虑到泛化能力，也可适应更复杂的环境。

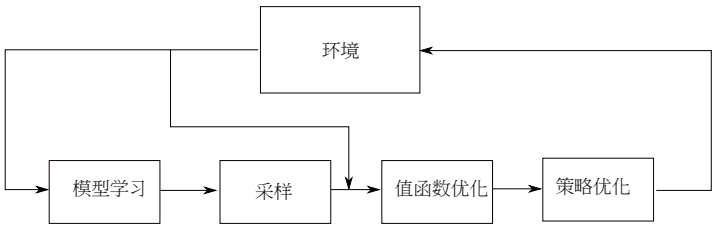

Fig. 10.15 混合学习方法。学习一个环境模型，值函数优化基于从真实环境和模型两方面得到的采样数据。

Dyna-Q即是一种混合学习方法。在该算法中，模型和值函数同时进行在线学习，在对值函数进行学习时，首先基于真实环境采样对Q函数进行回溯，再基于模型的若干次采样对Q函数做进一步学习。算法 6 给出连续任务中的Dyna-Q的算法流程，其中$P(s',r|s,a)$为待学习的环境模型。注意该算法中的回溯公式基于TD方法中的Q-learning，因此叫Dyna-Q。

Output: $Q(s,a)$
Initialize $Q(s,a)$, $P(s',r|s,a)$
for *Each step* **do**
  s ← Current State
  a ← ε-Greedy(Q,S)
  Act a, Observe s', obtain R
  $Q(s,a) \leftarrow Q(s,a) + \alpha[R + \gamma \max_a Q(s',a) - Q(s,a)]$
  Update $P(s',r|s,a)$ using (S,A,R,S')
  for *t:=0* **to** N **do**
    s ← random from $\mathscr{S}$
    a ← ε-Greedy(Q,s)
    Sample s',R from $P(s',r|s,a)$
    $Q(s,a) \leftarrow Q(s,a) + \alpha[R + \gamma \max_a Q(s',a) - Q(s,a)]$
  end
end

Algorithm 6: 连续任务中的Dyna-Q学习算法。其中对模型$P(s',r|s,a)$的更新可由算法 5实现。

10.7 函数近似

前面我们主要讨论了基于离散状态、离散动作的强化学习方法，在这些方法里，环境模型采用表格来记录状态和动作上概率分布。这种离散模型有助于我们理解强化学习的基本概念和基本算法，也适合某些小型问题。然而，在实际问题中，离散方法存在很大局限性，包括：

- 状态空间和动作空间很大时，离散表示不仅需要大量内存空间，在学习过程中也会遇到严重的数据稀缺问题。例如在围棋对弈中，状态空间是10的172 次方，中国象棋、国际象棋分别是10 的48 次方、46 次方。如此庞

大的状态空间，连存状态转移概率表都不可能，更别说对其进行有效学习了。

- 离散方法的可扩展性差。离散状态和动作是都是类型变量（Categorzied Variable），变量取值之间不具有任务距离甚至顺序关系，导致对不同状态和动作建模时不具有相关性，无法在学习时互相促进，对新出现的状态和动作无法处理。
- 现实任务中多数没有明显的离散状态和动作。如在股票投资、无人机飞行、机械臂抓取等众多机器学习任务中，计算机得到的只是一系列连续的观测值，而非明确的离散状态，其动作方式也是一系列控制信号（如投资变动、机械扭矩等），并非离散操作。

上述这些局限性使得标准离散模型方法在实际应用中很少用到。函数近似是解决这些局限性的常用方法，也是强化学习能在实际任务中取得巨大成功的重要原因。

10.7.1 值函数近似

我们考虑强化学习的根本目的，是通过和环境交互直接学习一个优化策略$\pi(s,a)$，或间接学习一个状态值函数$V(s)$或动作值函数$Q(s,a)$，再基于某种决策方法得到优化策略$\pi(s,a)$。不论是$\pi(s,a)$，$V(s)$，还是$Q(s,a)$，本质上都是一个以(s,a)为变量的函数。本节考虑值函数近似，对策略近似将在后面讨论。

离散模型可以认为是一种‘精确函数’，即对每个状态和动作组合都有一个明确取值。这种‘精确性’带来了前述的各种问题，事实上反而更不精确：例如，大量状态-动作空间没有得到充分训练，只能取缺省值；对连续输入和输出任务，需要先将这些输入转化成离散状态，将输出转化成离散动作，导致很大误差；某一训练数据只对应一个状态-动作空间，无法提高对其它状态和动作的学习质量，即使它们非常相似。函数近似是将离散精确的值函数或策略函数用一个基于参数$\boldsymbol{\theta}$的连续函数来拟合和近似，即$\pi(s,a) \approx \pi_{\boldsymbol{\theta}}(s,a)$，$V(s) \approx V_{\boldsymbol{\theta}}(s)$，或$Q(s,a) \approx Q_{\boldsymbol{\theta}}(s,a)$。

以状态值函数为例，其函数近似方法如图 10.16所示。对特定状态$\{S_0,S_1,S_2\}$，其状态值的精确估计为$\{V(S_0),V(S_1),V(S_2)\}$；函数近似用$V_{\boldsymbol{\theta}}(S)$来逼近$V(S)$。从图中可见，在原离散模型的特定状态$\{S_i\}$上，曲线$V_{\boldsymbol{\theta}}(s)$接近离散模型的状态值。

图 10.16所示的函数近似方法带来一个重要变化，即原本离散的状态S_i变成了距离可度量的连续状态。这种可度量性的产生可分为两种情况：第一种是原学习任务的状态本身就是可度量的，对这种情况，函数近似只需基于原任务的可度量输入作为函数$V_{\boldsymbol{\theta}}(s)$的状态输入即可。第二种情况是原学习任务的状态本身就是离散的，如在围棋任务中的棋盘状态。对这种情况，可对离散状态进行连续空间嵌入（State Embedding），如AlphaGo基于CNN将棋盘状态嵌入到连续低维子空间中，在该空间中状态即具有度量性。

状态的可度量性和连续性具有重要意义。首先，状态之间距离的可度量性使得不同状态的学习互相促进，因为一个训练样本可以通过更新函数参数来学习所有状态。第二，状态的连续性意味着任何新状态都可以被估计，极大提高了系统的可扩展性。第三，函数近似基于参数优化，参数的数量一般远少于原离散系统的状态数，意味着系统泛化能力的提高。

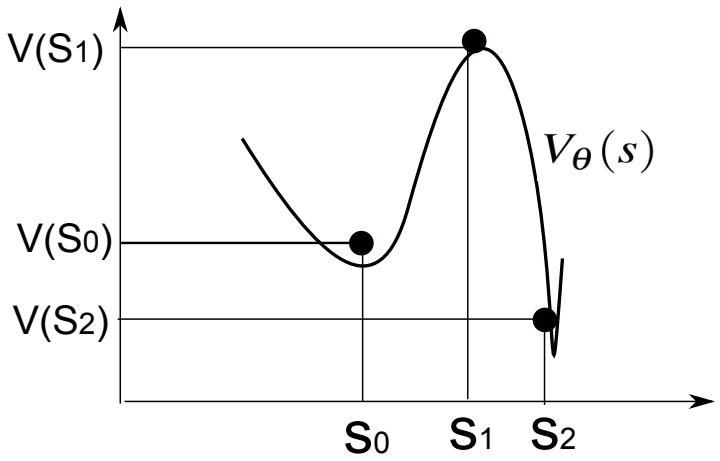

Fig. 10.16 状态值函数近似。曲线为基于参数$\boldsymbol{\theta}$的函数$V_{\boldsymbol{\theta}}(s)$，$\{S_1, S_2, S_3\}$为原离散模型的状态点。

上述对状态值函数的近似方法对动作值函数$Q(s,a)$同样适用，稍有不同的是近似函数需要考虑状态和动作两个因素。可采用图 10.17所示的两种近似方法。显然，第一种近似方法的通用性更强，第二种方法仅适用于离散状态，且状态数不多的情况。

可以采用各种参数模型来实现$V_{\boldsymbol{\theta}}(s)$或$Q_{\boldsymbol{\theta}}(s,a)$，常用的包括线性模型和神经网络模型。我们后面将讨论基于这些模型的参数更新方法，现在让我们假设该更新方法已知，即对一个交互采样$(S_t, A_t, R_{t+1}, S_{t+1})$，可通过更

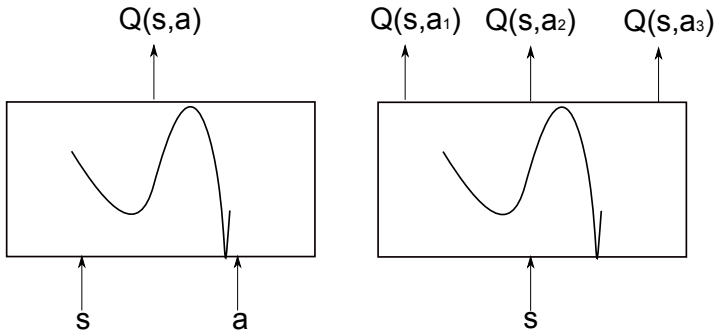

Fig. 10.17 动作值函数两种近似方法。

新$\boldsymbol{\theta}$得到一个更优化的$Q_{\boldsymbol{\theta}}$，则基于GPI，策略优化可以基于ε-贪心原则通过迭代实现。图 10.18给出一个在线GPI策略优化过程。在每次更新完$\boldsymbol{\theta}$以后，基于ε-贪心原则，由$Q_{\boldsymbol{\theta}}(s,a)$ 得到策略π，基于π生成下一次交互...如此迭代更新，得到优化的$Q_{\boldsymbol{\theta}}(s,a)$。注意在线学习中每一步迭代仅依赖当前观察值对$\boldsymbol{\theta}$进行更新，因此是一种随机更新，得到的$Q$函数也不是基于当前$\pi$的真正动作值函数。仅管如此，GPI框架依然保证对优化策略的学习。

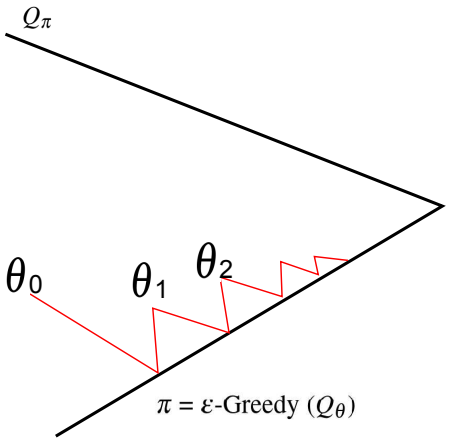

Fig. 10.18 基于Q函数近似的策略优化。

10.7.2 基于梯度的参数优化

我们以动作值函数$Q(s,a)$近似为例，介绍如何对参数进行更新优化。总体来说，该优化的目的是基于每一个交互采样(S_t,A_t,R_T,S_{t+1})更新参数$\boldsymbol{\theta}$，使得$Q_{\boldsymbol{\theta}}$在采样点(S_t,A_t)更趋向其长期收益G_t。

设优化目标是$Q_{\boldsymbol{\theta}}(S_t,A_t)$与$G_t$的均方差（Mean Square Value Error, MSVE），即:

$$L(\boldsymbol{\theta}) = \sum_{s,a} d_\pi(s,a)(Q_{\boldsymbol{\theta}}(s,a) - Q_\pi(s,a))^2,$$

其中$d(s,a)$表示状态-动作对(s,a)被访问的比例。基于采样方法，该目标可写成：

$$L(\boldsymbol{\theta}) = \sum_t (Q_{\boldsymbol{\theta}}(S_t,A_t) - G_t)^2.$$

采用随机梯度下降法（SGD），对每个采样点做一次参数更新，有:

$$\begin{aligned}\boldsymbol{\theta}_t &= \boldsymbol{\theta}_{t-1} - \alpha \frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \\ &= \boldsymbol{\theta}_{t-1} - \alpha (Q_{\boldsymbol{\theta}}(S_t,A_t) - G_t) \frac{\partial Q_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}},\end{aligned} \tag{10.14}$$

其中α为SGD的学习步长，一般随时间t增长而下降。上式中的G_t依学习算法的不同而有不同形式。对于MC方法，有:

$$G_t = R_{t+1} + \gamma R_{t+2} + ... \gamma^{T-1} R_T.$$

对于TD(0)（Sarsa方式），有：

$$G_t = R_{t+1} + \gamma Q_{\boldsymbol{\theta}}(S_{t+1},A_{t+1}),$$

其中A_{t+1}为下次采样选择的动作。注意上式中的Bootstrapping部分$Q_{\boldsymbol{\theta}}(S_{t+1},A_{t+1})$也是$\boldsymbol{\theta}$函数，但在梯度计算时并未考虑，因此上述TD(0)公式并非确切意义上的SGD，因而有时也称为Semi-SGD。

公式 10.14中的梯度$\frac{\partial Q_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}$取决于函数$Q_{\boldsymbol{\theta}}$的具体形式。例如，如果采用线性模型，则该梯度简化为输入特征向量，如果采用神经网络模型，一般要采用BP算法对梯度进行反向传递。值得说明的是，如果基于Bootstrap方法求值函数，则在GPI中用Off-Policy策略优化方法可能会导致学习不稳定，但On-Policy方法一般是稳定的。

10.7.3 基于函数近似的策略学习

前面我们已经讨论了在学习任务中对值函数和模型进行学习。有了函数近似方法，现在我们可以讨论对策略进行学习了。所谓策略学习，即是学习在状态s下选择动作a的概率函数$\pi(a|s)$。同值函数学习类似，我们希望得到一个基于参数w的概率函数$\pi_w(a|s)$，使得在某一准则$\eta(\cdot)$下性能最好。和值函数学习相比，策略学习有如下好处：

- 不同值函数可能对应相同的策略，因此策略学习可能比值函数学习更容易收敛。
- 基于值函数学习一般采用贪心原则得到策略，不容易处理包括随机动作的策略。
- 策略学习在处理连续动作上更有优势。

近似策略函数可以是任何形式，如线性模型、神经网络等。对策略学习而言，这些函数一般都包括一个Soft-max 层，以得到动作的后验概率。对于准则$\eta(\cdot)$，在多轮任务中，可选择起始状态的状态值函数：

$$\eta_1(\pi) = V_\pi(S_0).$$

在连续任务中，可选择平均值收益：

$$\eta_v(\pi) = \sum_s d^\pi(s) V_\pi(s),$$

或平均一步收益：

$$\eta_r(\pi) = \sum_s d^\pi(s) \sum_a \pi(a|s) R_{s,a},$$

其中$d^\pi(s)$ 是基于π的MDP过程在状态s上的分布, $R_{s,a} = \sum_r rP(r|s,a)$为在状态$s$选择动作$a$得到的收益期望。

我们以平均值收益为例，推导近似策略函数的梯度下降公式。设$\pi_{\mathbf{w}}$为π的函数近似并设其为连续可导的，则有：

$$J_v(\mathbf{w}) = \eta_v(\pi_{\mathbf{w}}) = \sum_s d(s) \sum_a \pi_{\mathbf{w}}(a|s) Q_{\pi_{\mathbf{w}}}(s,a).$$

取对$\mathbf{w}$的偏导，可得：

$$\frac{\partial J_v(\mathbf{w})}{\partial \mathbf{w}} = \sum_s d(s) \sum_a \pi_{\mathbf{w}}(a|s) \frac{\partial log(\pi_{\mathbf{w}}(a|s)}{\partial \mathbf{w}} Q_{\pi_{\mathbf{w}}}(s,a) \tag{10.15}$$

$$= \mathbb{E}_{\pi_{\mathbf{w}}} \frac{\partial log(\pi_{\mathbf{w}}(a|s))}{\partial \mathbf{w}} Q_{\pi_{\mathbf{w}}}(s,a). \tag{10.16}$$

可以证明，上式所表达的梯度形式同样适用于起始状态值函数收益η_1和平均一步收益的$\frac{1}{1-\gamma}\eta_r$，称为‘策略梯度定理’。基于该定理，即可对参数$\mathbf{w}$进行学习。以多轮任务为例，基于MC采样的策略学习方法如算法 7 所示，其中以本轮的长期收益G_t来近似$Q(S_t,A_t)$。该算法称为REINFORCE。注意其中偏微分部分的求解取决于所选择函数的具体形式。

Input: I: Max episodes
Output: $\pi_{\mathbf{w}}(a|s)$
Initialize $\mathbf{w}$;
for *i:=0* **to** *I* **do**
  Sample episode $[S_0,A_0,R_1,S_1,A_1,...,R_{T_i},S_{T_i}]$ by $\pi_{\mathbf{w}}$;
  for *t:=0* **to** T_i*-1* **do**
    $G_t = R_{t+1} + ...R_T$;
    $\mathbf{w} \leftarrow \mathbf{w} + \beta \frac{\partial log(\pi_{\mathbf{w}}(A_t|S_t))}{\partial \mathbf{w}} G_t$;
  end
end

Algorithm 7: 多轮任务中的REINFORCE 算法，用长期收益G_t的采样代替长期收益的期望$Q(s,a)$，β为学习率。

10.7.4 Actor-Critic方法

根据策略梯度定理，策略学习需要估计状态值函数$Q_{\pi_{\mathbf{w}}}(s,a)$。在REINFORCE算法中，我们用一次交互过程中的G_t来对其做近似，更好的办法是用另一个函数来近似，即：

$$Q_{\pi_{\mathbf{w}}}(s,a) \approx Q_{\boldsymbol{\theta}}(s,a).$$

该值函数可以用前面介绍过的函数近似方法学习。这种同时学习值函数和策略的方法称为Actor-Critic方法，其中Actor 学习策略$\pi_{\mathbf{w}}$, Critic学习该策略的近似值函数$Q_{\boldsymbol{\theta}}(s,a)$。图 10.19给出了该学习方法的示意图。从图中可以看到，Actor-Critic事实上是一种对值函数和策略同时做函数近似和优化的混合学习

方法。算法 8给出一个连续任务中用TD(0)进行值函数学习（Critic），用梯度下降进行策略学习（Actor）的算法流程。

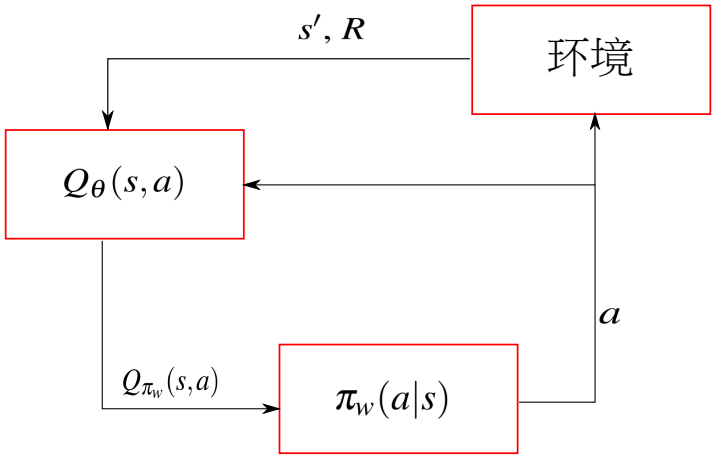

Fig. 10.19 Actor-Critic学习。Actor是一个策略学习器，并依策略生成动作和环境进行交互。Critic是一个值函数学习器，用来对策略进行评估，并将评估结果交给策略学习器进行下一步改进。注意，对Actor（策略）学习来说，其学习信号完全由Critic（值函数）给出，并不需要接收从环境的直接反馈。

Output: $\pi_{\mathbf{w}}(a|s)$

Initialize $\mathbf{w}$, $\boldsymbol{\theta}$, s;

Sample $a \sim \pi_{\mathbf{w}}(a|s)$ **for** *Each Step* **do**

  Sample s', R by perform a

  Sample $a' \sim \pi_{\mathbf{w}}(a|s)$

  $\delta = R + \gamma Q_{\boldsymbol{\theta}}(s',a') - Q_{\boldsymbol{\theta}}(s,a)$;

  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\delta\frac{\partial Q_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}$

  $\mathbf{w} \leftarrow \mathbf{w} + \beta\frac{\partial log(\pi_{\mathbf{w}}(a|s))}{\partial \mathbf{w}} Q_{\boldsymbol{\theta}}(s,a)$;

  $s \leftarrow s'$;

  $a \leftarrow a'$;

end

Algorithm 8: 连续任务Actor-Critic学习算法。Critic学习基于TD(0)，Actor学习基于策略梯度定理。

Actor-Critic可以用来学习连续动作，如在机械臂操作中的扭矩。在连续动作任务中，我们一般假设动作符合某种分布，通过学习该分布的参数来确

定连续动作的规律。以高斯分布为例，其参数为均值μ和方差σ，则对策略的函数近似方法可转化为对μ和σ的函数近似:

$$\pi_{\mathbf{w}}(a|s) = \frac{1}{\sqrt{2}\sigma(s;\mathbf{w})} exp\{\frac{(a-\mu(s;\mathbf{w}))^2}{\sigma(s;\mathbf{w})^2}\}$$

其中$\mu(s;\mathbf{w})$和$\sigma(s;\mathbf{w})$都是以$\mathbf{w}$为参数，以s为输入的连续可导函数，因$\pi_{\mathbf{w}}$也是连续可导的，满足策略梯度定理中对梯度计算的要求。另外，设计$Q_{\boldsymbol{\theta}}(s,a)$为以$\boldsymbol{\theta}$ 为参数，以(s,a)为输入的连续函数（见图 10.17中的第一种结构），则依图 10.19 所示结构和算法 8，可解决连续动作的策略学习问题。

10.8 深度强化学习方法

上节所述的函数近似法具有重要意义。通过函数近似，可以把对外界的观察（不论是连续的还是离散的）映射到某种状态空间，在该空间上判断值函数的大小（值函数学习）或动作的选择（策略学习）。这种空间映射正是深度学习的基本思路。在深度学习中，通过对原始信号的逐层处理，可学习与任务相适应的特征，因而可有效提高系统性能。如果用深度模型（如深度神经网络）对值函数或策略进行近似，则得到深度强化学习方法。深度强化学习是当前机器学习乃至整个人工智能领域研究的热点之一，取得了一系列让人惊讶的成果，事实上直接推动了近年来人工智能的热潮。

深度强化学习包括两个部分，一是深度学习模型，利用DNN（或其它深度网络）对值函数或策略做近似，第二部分是强化学习方面，利用强化学习方法对DNN的参数进行更新。采用DNN作为近似函数可充分利用其强大的特征学习习能力，采用强化学习方法对DNN进行训练可得到具有强烈目标驱动的网络。深度强化学习在训练和推理时所用的基本方法和我们上节讨论的函数近似方法并没有根本不同，只是在求近似函数的梯度时工作量更大一些。我们将不再重复这些方法，仅通过几个例子来讨论深度强化学习的强大学习能力。

10.8.1 Atari游戏

游戏一直是强化学习擅长的领域，从最初Samuel [8]的西洋棋到Tesauro的TD-Gammon [13]。然而，在2016年以前，恐怕没人想到机器玩儿起游戏来如此

强大，不仅可以在简单游戏中战胜人类业余选手，甚至可能在极为复杂的任务中战胜人类顶尖对手。

突破从Deep Mind公司利用深度Q-learning网络（DQN）教会机器玩儿Atari游戏开始 [6]。Atari平台包括49个游戏，学习方法很简单：把游戏画面传给计算机，让它通过观察这些画面控制游戏操纵杆，像人一样操作游戏。基于学习信号的复杂性和交互性，这是一个典型的强化学习任务，其中观察值为所看到的游戏画面，动作为对游戏杆的操纵，收益为打游戏过程中得到的奖励。在这一任务中，唯一的困难是输入的观察值太过原始（游戏画面），这一观察值直接作为状态输入很难被机器理解。为此，Deep Mind的研究员们用一个包含多层CNN的深度神经网络作来近似动作值函数$Q(s,a)$，并基于Q-learning算法对该网络进行学习。经过多层CNN，原始游戏画面中关于游戏状态的信息被逐层抽象出来，基于这一状态，机构即可判断在某个状态下应采取的操作，即Q函数。图 10.20给出这一深度Q网络（DQN）的结构，包括两层卷积神经网络，两层全连接网络，输出为每个动作对应的Q值。

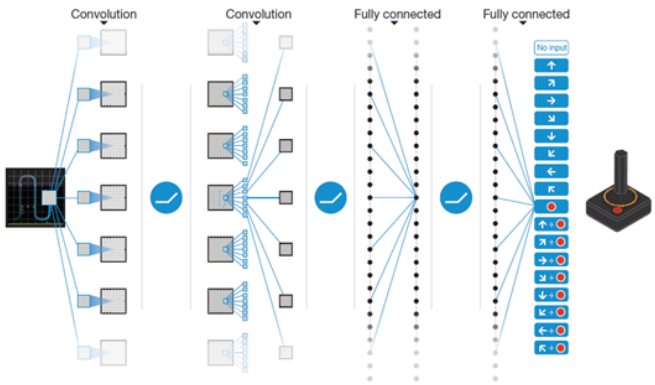

Fig. 10.20 基于DQN的Atari游戏学习框架。图片来自 [6]。

Atari游戏是多轮任务，可通过MC采样得到训练样本，依本章所介绍的回溯方法进行训练。Mnih [6] 等人在实现时采用了一种称为‘Replay’的方法，每次采样得到的样本以四元组(s,a,r,s')的形式存入经验池D，训练时从中随机选出若干样本组成一个Mini Batch进行回溯。另外，为提高训练稳定性，目标网络（即用于生成动作的DQN）每隔数次回溯后才进行更新。总结起来，DQN的回溯公式如下：

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \sum_{(s,a,r,s') \sim D} [r + \gamma \max_{a'} Q(s', a'; \boldsymbol{\theta}^-) - Q(s, a; \boldsymbol{\theta})] \frac{\partial Q(s, a; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}.$$

训练中采用ε-贪心策略，其中ε由1.0线性减小到0.1。注意上式中max符号下的Q函数以前一轮训练结果$\boldsymbol{\theta}^-$为参数，而非当前参数$\boldsymbol{\theta}$。

为考察DQN的学习能力，可以将最后一个隐藏层的激发值用t-SNE [5]映射到二维平面上。结果如图 10.21，其中颜色表示该状态的最大值函数（蓝色<红色），一些点上的截图表示该点所代表的原始游戏画面。可以看到，DQN整体上依值函数的分数将游戏画面映射到不同区域。这意味着DQN可以通过深度学习得到当前游戏状态的整体感觉。

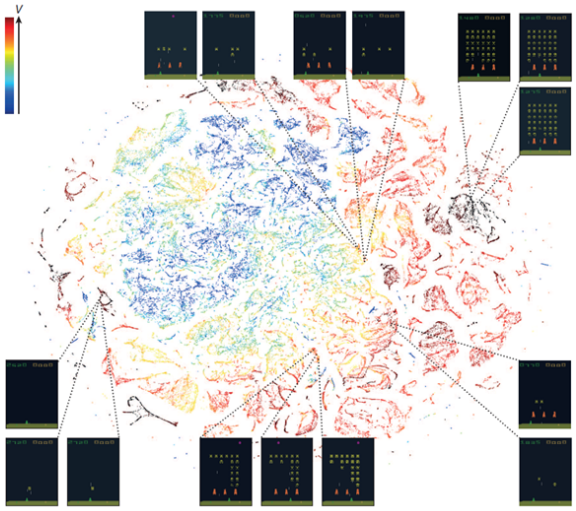

Fig. 10.21 基于DQN学习Atari游戏学习。对DQN最后一个隐藏层的激发向量用t-SNE表达在二维空间，其中颜色表示状态值函数的大小，界面截图表示某一状态点对应的游戏画面。图片来自 [6]。

10.8.2 AlphaGo

AlphaGo是将深度学习和强化学习完美结合的另一个典型 [10]。和Atari学习不同的是，AlphaGo并不完全是端对端的强化学习，还包括了

大量监督学习，不仅使用了策略网络，还使用了值网络，不仅基于贪心策略，还大量使用了MC搜索。

AlphaGo训练了三个策略网络和一个值网络。首先，基于历史上的棋局训练一个简单的策略网络p_π用于MC搜索，一个复杂的策略网络p_σ用于进行路径扩展。这两个网络都基于监督学习，其中p_π基于简单的局部模式特征和线性模型，p_σ基于13层CNN学习盘面原始特征。基于p_σ，将监督学习目标（预测每一步棋落子的准确性）变成强化学习目标（落子更有利于最终赢得胜利），通过自我对奕方式进行强化学习，得到基于强化学习的策略网络p_ρ。最后，通过基于p_ρ的自我对奕生成大量棋局，基于该数据用强化学习方法学习状态值网络v_θ。图 10.22给出了这一训练过程。

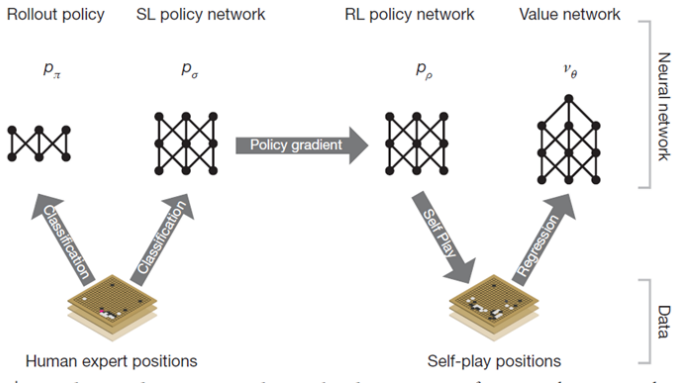

Fig. 10.22 AlaphGo的DQN网络。图片来自 [10].

在实际对奕中，AlphaGo充分利用策略网络、状态值网络和MC搜索来提高走棋精度。理论上，这三种方法都可以生成落子决策，但由于训练数据的局限，策略网络和值网络并不能保证优化，而MC搜索需要大量计算，将三者结合起来可以充分利用各自的优点，提高胜率。具体来说，在对奕时，AlphaGo 基于采样生成一个蒙特卡洛树（MCT），树上的每个结点s对应一种棋局格式（状态），每条边对应一步走棋（动作），相应的动作值函数为$Q(s,a)$。搜索过程基于异步采样，同时发起多个线程进行MC路径模拟和回溯，所有模拟完成后统计每个边所对应的$Q(s,a)$以决定落子选择。设当前状态s_0作为根结点，具体搜索和回溯过程如下所示：

- 路径选择：开始一个新的采样，从s_0到一个叶子结点s_L。在每一步$t < L$, 路径的选择基于当前MCT中每条边对应的$Q(s,a)$和由策略网络p_σ计算出的先验概率$P_\sigma(a|s)$:

$$a_t = \arg\max_a (Q(s_t,a) + u(s_t,a)),$$

其中

$$u(s,a) = cP(a|s)\frac{\sqrt{\sum_b N(s,b)}}{N(s,a)}.$$

上式中c为常数，$N(s,a)$为该条边被已经访问的次数。显然，AlphaGo倾向于选择由策略网络确定的最大概率边，但当该边被选择多次时，转而倾向选择未被访问的边，以增加路径覆盖性。每当一条边(s,a)被访问时，累加其访问次数$N(s,a)$。

- MC模拟采样：依简单策略网络p_π模拟一条对奕路径直到终局。依终局的结果确定该路径的MC 收益z_t（$z_t = +1$为赢棋，$z_t = -1$为输棋）。
- 收益累加：对当前采样路径上的树内状态$s_t; t \le L$, 更新收益如下：

$$W(s_t,a_t) = W(s_t,a_t) + (1-\lambda)(z_t) + \lambda v_\theta(s_L)$$

其中λ调整MC收益和值函数估计的相对权重，$v_\theta(s_L)$为该路径的叶子结点s_L的状态值函数，由状态值网络得出。

- 路径扩展：在搜索过程中，当发现叶子结点的某一条边(s_L,a)的累积收益$W(s,a)$大于某一阈值时（注意采样是多线程并行进行的，因此当某一线程搜索到某一叶子结点时，可能会发现某些未扩展路径的累积收益已经达到阈值），则对该边进行扩展，将其后续结点s_{L+1}加入MCT中，并初始化其访问次数、累积收益，计算其动作概率分布$P_\sigma(a|s_{L+1})$和状态值函数$v_\theta(s_{L+1})$。
- 动作值函数计算与走棋决策：所有采样结束后，对每条边(s,a)计算动作值函数$Q(s,a) = \frac{W(s,a)}{N(s,a)}$。走棋决策依根结点的动作值函数$Q(s_0,a)$做出。

由上述算法可见，AlphaGo主要采用如下三种技术提高搜索效率和精度：（1）依靠值函数网络v_θ和MC搜索对走棋决策进行价值判断；（2）依靠简单策略网络p_π生成MC采样，以增加价值判断精度；（3）依靠策略网络p_σ控制MC搜索的宽度。除此之外，Google的研究者们在工程化方面做出了一系列努力，如并行计算、异步搜索、GPU加速等，这些都是AlphaGo能取得成功的重要因素。

10.9 本章小结

本章我们总结了强化学习的基本概念和若干方法。我们讨论到强化学习既可以学习模型，也可以学习值函数和策略。相对而言，值函数学习既可以利用模型假设（MDP）的结构化知识，也具有无模型方法的灵活性，因此应用最为广泛。我们讨论了动态规划算法（DP）、蒙特卡洛算法（MC）和时序差分算法（TD）等若干值函数学习算法，并讨化了将其与模型学习结合起来的混合学习方法，如Dyna算法。为处理复杂连续状态和动作，我们讨论了函数近似方法，包括值函数近似和策略近似，以及二者结合的Actor-Critic 模型。我们特别讨论了基于深度神经网络的深度强化学习方法。归因于深度学习强大的特征表达能力，深度强化学习近年来在若干标志性任务中取得突破性进展，引起研究者的广泛关注。

我们认为强化学习是比监督学习更高级的学习方法，可以应对更复杂的反馈信息和更复杂的交互过程，因此适合很多实际任务。依靠强化学习，我们可以在不必对数据进行特别标注情况下，只依赖某一相关反馈即可学习得到较好的模型，这更像人类的学习方法：通过不精确的信号反馈，即便没有特别理性的理解，也可以学到足够合理的行为方式。特别有意思的是，当这种方法和深度学习结合在一起时，取得了令人瞩目的成就。这是因为深度学习具有强大的特征抽象能力，但并不是一种有效的过程建模方法，强化学习正好弥补了这一不足。可以想象，深度强化学习将在不久的将来取得更加让人振奋的成果，给人工智能研究带来深刻变革。

10.10 相关资源

- 本节大量参考了Sutton 和Barto 的《Reinforcement Learning: An Introduction》 [11]，包括符号惯例和某些算法举例。
- 本章参考了Dave Silver的课件 [9]，特别是在策略梯度下降一节中参考了Silver课件中关于策略梯度定理方面的内容。
- 本章参考了Kaelbling和Kober等人的综述文章 [3, 4]。
- 关于强化学习的算法实现，可以参见Szepesvari的著作 [12]。
- 关于DP方法和控制理论，可参见Bertsekas和Powell等人的著作 [1, 2, 7]。

Chapter 11
优化方法

References

[1] Bertsekas DP (2005) Dynamic Programming and Optimal Control, vol I, 3rd edn. Athena Scientific

[2] Bertsekas DP (2012) Dynamic Programming and Optimal Control, vol II, 4th edn. Athena Scientific

[3] Kaelbling LP, Littman ML, Moore AW (1996) Reinforcement learning: a survey. Journal of Artificial Intelligence Research 4(1):237–285

[4] Kober J, Peters J (2012) Reinforcement learning in robotics: A survey. In: Reinforcement Learning, Springer, pp 579–610

[5] Maaten Lvd, Hinton G (2008) Visualizing data using t-sne. Journal of Machine Learning Research 9(Nov):2579–2605

[6] Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, et al (2015) Human-level control through deep reinforcement learning. Nature 518(7540):529–533

[7] Powell WB (2009) Approximate Dynamic Programming: Solving the Curses of Dimensionality. Wiley

[8] Samuel AL (1959) Some studies in machine learning using the game of checkers. IBM Journal of research and development 3(3):210–229

[9] Silver D (2015) URL http://www0.cs.ucl.ac.uk/staff/D.Silver/web/Teaching.html

[10] Silver D, Huang A, Maddison CJ, Guez A, Sifre L, Van Den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M, et al (2016) Mastering the game of go with deep neural networks and tree search. Nature 529(7587):484–489

[11] Sutton RS, Barto AG (1999) Reinforcement Learning: An Introduction. MIT Press, URL https://mitpress.mit.edu/books/reinforcement-learning

[12] Szepesvari C (2010) Algorithms for Reinforcement Learning. Morgan & Claypool, DOI 10.2200/S00268ED1V01Y201005AIM009, URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6813120

[13] Tesauro G (1995) Temporal difference learning and td-gammon. Communications of the ACM 38(3):58–68