



Reinforcement Learning: An Introduction

强化学习导论第二版翻译

作者：Sutton & Barto

组织：UESTC

时间：March 2, 2020

译者：吕昀琏



Victory won't come to us unless we go to it. — M. Moore

目录

第二版前言	vi
第一版前言	x
符号总结	xii
翻译说明	xvi
1 介绍	1
1.1 强化学习	1
1.2 例子	3
1.3 强化学习的要素	4
1.4 局限和范围	5
1.5 拓展例子：井字游戏	6
1.6 总结	10
1.7 强化学习的早期历史	10
1.8 书目评论	17
第一部分 表格解决方法	18
2 多臂赌博机	19
2.1 k 臂赌博机问题	19
2.2 动作值方法	20
2.3 10 臂试验	21
2.4 渐增实现	23
2.5 非平稳问题	25
2.6 乐观初始值	26
2.7 置信上限动作选择	27
2.8 梯度赌博机算法	28
2.9 关联搜索（上下文赌博机）	32
2.10 总结	32
2.11 书目与历史评论	34
3 有限马尔可夫决策过程	36
3.1 Agent-环境接口	36
3.2 目标和奖励	41

3.3 回报和 episode	41
3.4 回合和连续任务的统一符号	43
3.5 策略和值函数	44
3.6 最优策略和最优值函数	48
3.7 最优和近似	52
3.8 总结	53
3.9 书目与历史评论	54
4 动态规划	56
4.1 策略评估	56
4.2 策略改进	60
4.3 策略迭代	62
4.4 值迭代	64
4.5 异步动态规划	66
4.6 广义策略迭代	67
4.7 动态规划效率	68
4.8 总结	68
4.9 书目与历史评论	69
5 蒙特卡洛方法	71
5.1 蒙特卡洛预测	71
5.2 动作值的蒙特卡洛估计	75
5.3 蒙特卡洛控制	75
5.4 无探索起点的蒙特卡洛控制	78
5.5 重要性采样的 off-policy 预测	81
5.6 渐增实现	86
5.7 Off-policy 蒙特卡洛控制	87
5.8 * 折扣的重要性采样	89
5.9 * 每决策的重要性采样	90
5.10 总结	91
5.11 书目与历史评论	92
6 时间差分学习	94
6.1 TD 预测	94
6.2 TD 预测方法的优势	98
6.3 TD(0) 的最优性	99
6.4 Sarsa: on-policy TD 控制	102
6.5 Q-learning: off-policy TD 控制	104
6.6 期望 Sarsa	105

6.7	最大化偏差与 Double Learning	106
6.8	游戏、后期状态和其他特殊情况	108
6.9	总结	109
6.10	书目与历史评论	110
7	n 步自举	112
7.1	n 步 TD 预测	112
7.2	n 步 Sarsa	116
7.3	n 步 off-policy 学习	118
7.4	* 控制变量的每决策方法	120
7.5	无重要性采样的 off-policy 学习: n 步树备份算法	121
7.6	* 一种统一算法: $Q(\sigma)$	123
7.7	总结	126
7.8	书目与历史评论	126
8	表格法进行规划和学习	128
8.1	模型和规划	128
8.2	Dyna: 综合规划, 行动和学习	130
8.3	当模型错误时	133
8.4	优先扫描	136
8.5	期望与采样更新	138
8.6	轨迹采样	141
8.7	实时动态规划	143
8.8	决策时规划	145
8.9	启发式搜索	146
8.10	展开算法	148
8.11	蒙特卡洛树搜索	149
8.12	本章总结	151
8.13	第一部分总结: 维度	152
8.14	书目与历史评论	154
9	Elegant\LaTeX 系列模板介绍	157
9.1	ElegantBook 更新说明	157
9.2	模板安装与更新	158
9.2.1	在线使用模板	158
9.2.2	本地免安装使用	158
9.2.3	发行版安装使用	158
9.2.4	更新问题	158
9.2.5	其他发行版本	159

9.3 用户作品计划	159
9.4 关于提交	160
9.5 协作人员招募	160
9.6 致谢	160
9.7 捐赠	160
10 ElegantBook 设置说明	162
10.1 语言模式	162
10.2 设备选项	162
10.3 颜色主题	162
10.4 封面	163
10.4.1 封面个性化	163
10.4.2 封面图	164
10.4.3 徽标	164
10.4.4 自定义封面	164
10.5 章标题	164
10.6 数学环境简介	164
10.6.1 定理类环境的使用	165
10.6.2 其他环境的使用	165
10.7 装饰物	166
10.8 列表环境	166
10.9 参考文献	166
10.10 添加序章	167
10.11 目录选项与深度	167
10.12 章节摘要	167
10.13 章后习题	168
第十章 习题	168
10.14 旁注	168
11 字体选项	170
11.1 数学字体选项	170
11.2 使用 newtx 系列字体	170
11.2.1 连字符	170
11.2.2 宏包冲突	170
11.3 中文字体选项	171
11.3.1 方正字体选项	171
11.3.2 其他中文字体	171



12 ElegantBook 写作示例	173
12.1 Lebesgue 积分	173
12.1.1 积分的定义	173
第十二章 习题	175
13 常见问题集	176
14 版本更新历史	178
A 基本数学工具	180
A.1 求和算子与描述统计量	180



第二版前言

自本书第一版出版以来的 20 年里，人工智能取得了巨大的进步，这在很大程度上是由机器学习的进步推动的，包括强化学习的进步。虽然令人印象深刻的可用的计算能力是一部分这些进步中的原因，但理论和算法方面的新发展也一直是推动力量。面对这一进展，我们 1998 年出版的书的第二版早就应该出版了，我们终于在 2012 年开始了这个项目。我们第二版的目标与第一版的目标相同：提供清晰而简单的强化学习的关键思想和算法的描述，所有相关学科的读者都可以接触到这些关键思想和算法。这个版本仍然是一个介绍性的版本，我们将重点放在核心的在线学习算法上。这一版包括了一些新的话题，这些话题在过去的几年里变得越来越重要，我们扩大了对现在更好理解的话题的覆盖范围。但我们没有试图提供该领域的全面覆盖，它已经在许多不同的方向爆发。很抱歉，我们不得不删除除了少数几个贡献之外的所有内容。

与第一版一样，我们选择不对强化学习进行严格的正式处理，或者用最一般的术语来表示它。然而，自从第一版以来，我们对一些主题的深入理解需要更多的数学知识来解释；我们在阴影框内设置了更多数学部分，不喜欢数学的人可以选择跳过。我们还使用了与第一版略有不同的符号。在教学中，我们发现新的记法有助于解决一些常见的混乱之处。它强调随机变量（用大写字母表示）和它们的实例（用小写表示）之间的差异。例如，时间步 t 处的状态、动作和奖励分别表示为 S_t 、 A_t 和 R_t ，而它们可能的值可以表示为 s 、 a 和 r 。与此同时，对于值函数（例如 v_π ）使用小写字母并将大写字母限制为它们的表格估计值（例如 $Q_t(s, a)$ ）是很自然的。近似值函数是随机参数的确定性函数，因此也是小写形式（例如 $\hat{v}(s, \mathbf{w}_t) \approx v_\pi(s)$ ）。诸如权重向量 \mathbf{w}_t （以前记为 $\boldsymbol{\theta}_t$ ）和特征向量 \mathbf{x}_t （以前记为 ϕ_t ）等向量即使是随机变量，也会以粗体并以小写形式书写。矩阵保留使用大写粗体。在第一版中，我们使用了特殊的符号 $\mathcal{P}_{s,s'}^a$ 和 $R_{s,s'}^a$ 来表示转移概率和期望奖励。这种符号的一个缺点是，它仍然没有完全描述奖励的动态性，只给出他们的期望，这对于动态规划来说是足够的，但是对于强化学习来说却不是。另一个缺点是下标和上标过多。在这个版本中，我们使用显式符号 $p(s', r|s, a)$ 来表示给定当前状态和动作的下一个状态和奖励的联合概率。第 xii 页的表格总结了符号的所有变化。

第二版有了很大的扩充，顶层组织也发生了变化。在第一章绪论之后，第二版分为三个新的部分。第一部分（第 2-8 章）尽可能多地讨论强化学习，但不超出可以找到精确解的表格情况。我们涵盖了表格情况的学习和规划方法，以及它们在 n 步方法和 Dyna 中的统一。这一部分中介绍的许多算法对第二版来说都是新的，包括 UCB、期望 Sarsa、双重学习、树备份、 $Q(\sigma)$ 、RTDP 和 MCTS。首先彻底地完成表格情况，可以在最简单的背景中开发出核心思想。然后，本书的第二部分（第 9-13 章）致力于将思想扩展到函数逼近。它在人工神经网络，傅立叶基础，LSTD，基于核的方法，Gradient-TD 和 Emphatic-TD 方法，平均奖励方法，真实在线 TD(λ) 和策略梯度方法方面有新的章节。第二版大大扩展了对离线策略学习的处理方式，首先是在第 5-7 章中讨论表格情况，然后在第 11 章和

第 12 章中讨论函数逼近。另一个变化是第二版将 n 步自举的前向观点（现在在第 7 章中更全面地讨论）与资格迹的后向观点（现在在第 12 章中独立讨论）分开。这本书的第三部分有大量关于强化学习与心理学（第 14 章）和神经科学（第 15 章）的关系的新章节，以及一个更新的案例研究章节，包括 Atari 游戏、Watson 的下注策略，以及围棋程序 AlphaGo 和 AlphaGo Zero（第 16 章）。尽管如此，出于必要我们只包括了该领域已完成的所有工作中的一小部分。我们的选择反映了我们对廉价的免模型方法的长期兴趣，这些方法应该能够很好地扩展到大型应用中。最后一章现在包括关于强化学习的未来社会影响的讨论。不管好坏，第二版篇幅大约是第一版的两倍。

这本书被设计用来作为一个或两个学期的强化学习课程的主要文本。对于一个学期的课程，前十章应该按顺序覆盖并形成一个良好的核心，根据口味，可以从其他章节、其他书籍（如 Bertsekas 和 Tsitsiklis (1996)、Wiering 和 van Otterlo (2012) 和 Szepesvari (2010)）或文献中添加材料。根据学生的背景，一些关于在线监督学习的额外材料可能会有所帮助。Option 和 option 模型的思想是很自然的补充 (Sutton, Precup 和 Singh, 1999)。一门为期两学期的课程可以涵盖所有章节以及补充材料。这本书也可以作为机器学习、人工智能或神经网络等更广泛课程的一部分。在这种情况下，可能只希望涵盖材料的一部分。建议您简要介绍第 1 章，第 2 章至第 2.4 节，第 3 章，然后根据时间和兴趣从其余各章中选择各节。第 6 章对于本主题和本书的其余部分来说是最重要的。侧重于机器学习或神经网络的课程应涵盖第 9 章和第 10 章，而侧重于人工智能或规划的课程则应涵盖第 8 章。在整本书中，对于本书其余部分而言那些较为困难和不重要的部分和章节都标有 * 号。这些可以在第一次阅读时省略，而不会在以后造成问题。有些练习也标有 * 号，表示它们更高级，对理解本章的基本材料不是必需的。

大多数章节的结尾都有一个标题为“书目和历史评论”的章节，在这一章节中，我们对这一章中提出的观点的来源给予了信任，为进一步阅读和正在进行的研究提供了指导，并描述了相关的历史背景。尽管我们试图使这些部分具有权威性和完整性，但我们无疑遗漏了一些重要的前期工作。为此，我们再次表示歉意，我们欢迎将更正和扩展纳入该书的电子版。

像第一版一样，这本书的这个版本是为了纪念 A. Harry Klopf。是 Harry 把我们介绍给彼此的，也是他关于大脑和人工智能的想法让我们开始了长时间的强化学习之旅。Harry 受过神经生理学训练，长期对机器智能感兴趣，是俄亥俄州莱特帕特森空军基地空军科学研究中心 (AFOSR) 航空电子董事会的一名资深科学家。他不满意平衡寻求过程，包括稳态和纠错模式分类方法在解释自然智能和为机器智能提供基础方面的重要性。他指出，试图最大化某些东西（无论是什么）的系统与寻求均衡的系统在性质上是不同的，他认为最大化系统是理解自然智能的重要方面和构建人工智能的关键。Harry 在从 AFOSR 获得资助的一个项目中发挥了重要作用，该项目旨在评估这些想法和相关想法的科学价值。该项目于 20 世纪 70 年代末在马萨诸塞大学阿默斯特分校 (UMass Amherst) 进行，最初的指导是 MichaelArbib、William Kilmer 和 Nico Spinelli，他们是阿默斯特大学计算机和信息科学系的教授，也是该大学系统神经科学控制论中心的创始成员，这是一个专注于神经科学和人工智能交叉的有远见的组织。最近从密歇根大学获得博士学位的 Barto

被聘为该项目的博士后研究员。与此同时，在斯坦福大学攻读计算机科学和心理学的本科生萨顿 Sutton 一直在与 Harry 通信，讨论他们对刺激计时在经典条件反射中的作用的共同兴趣。Harry 向马萨诸塞州大学的小组建议，Sutton 将对这个项目起到很好的补充作用。就这样，Sutton 成为了麻省大学的研究生，他的博士学位是由 Barto 指导的，Barto 已经成为一名副教授。本书中提出的强化学习研究理所当然地是该项目的成果，该项目由 Harry 发起，并受到他的想法的启发。此外，Harry 负责使我们，作者们之间进行了长期而愉快的互动。通过将本书献给 Harry，我们不仅在强化学习领域，而且在我们的合作中都对他的重要贡献表示敬意。我们也感谢 Arbib、Kilmer 和 Spinelli 教授为我们提供了开始探索这些想法的机会。最后，我们感谢 AFOSR 在我们的研究初期给予的大力支持，并感谢 NSF 在接下来的许多年中给予的大力支持。

我们要感谢很多人对第二版的启发和帮助。我们感谢每一个为第一版提供灵感和帮助的人，也应该为这一版得到我们最深切的感谢，如果没有他们对第一版的贡献，这一版就不会存在。在这份长长的名单中，我们必须加上许多其他专门为第二版做出贡献的人。我们的学生在我们教授这些材料的许多年里在很多方面做出了贡献：揭露错误，修正错误，以及尤其是在感到困惑的地方我们可以更好地解释这些事情。我们特别感谢 Martha Steenstrup 通篇阅读并提供详细的评论。如果没有许多心理学和神经科学领域专家的帮助，心理学和神经科学的章节是不可能写出来的。我们感谢 John Moore 多年来对动物学习实验、理论和神经科学的耐心指导，感谢他仔细阅读了第 14 章和第 15 章的多个草稿。我们也感谢 Matt Botvinick、Nathaniel Daw、Peter Dayan 和 Yael Niv 对这些章节草稿的深刻评论，他们在大量文献中的重要指导，以及他们拦截了我们早期草稿中的许多错误。当然，这些章节中的其余错误，而且肯定还有一些，完全是我们自己的错误。我们感谢 Phil Thomas 帮助我们让非心理学家和非神经学家能够接触到这些章节，我们感谢 Peter Sterling 帮助我们改进了论述。我们感谢 Jim Houk 向我们介绍基底神经节的信息处理这一主题，并提醒我们注意神经科学的其他相关方面。Jose MartNez、Terry Sejnowski、David Silver、Gerry Tesauro、Georgios Theoccharous 和 Phil Thomas 慷慨地帮助我们了解他们的强化学习应用的详细信息，并将其纳入案例研究一章，他们对这些部分的草稿提供了有用的意见。特别感谢 David Silver 帮助我们更好地理解 Monte Carlo Tree search 和 DeepMind 围棋程序。我们感谢 George Konidaris 在傅立叶基础上的章节帮助。Emilio Cartoni、Thomas Cederborg、StefanDernbach、Clemens Rosenbaum、Patrick Taylor、Thomas Colin 和 Pierre-Luc Bacon 在一些重要的方面帮助了我们，对此我们非常感激。

Sutton 还想感谢艾伯塔大学强化学习和人工智能实验室的成员为第二版做出的贡献。他欠 Rupam Mahmood 一笔特别的债，因为他在第 5 章中对 off-policy Monte Carlo 方法的处理做出了重要贡献。他特别感谢 Rupam Mahmood 在第 5 章中对 off-policy 蒙特卡罗方法的处理所做的重要贡献，感谢 Hamid Maei 在第 11 章中提出的 off-policy 学习观点，感谢 Eric Graves 在第 13 章中进行的实验，感谢张尚同（Shangtong Zhang）对几乎所有实验结果的复制和验证，感谢 Kris De Asis 改进了第 7 章和第 12 章的新技术内容，感谢 Harm van Seijen 的深刻见解，这导致了 n 步方法与资格迹的分离，以及（与 Hadovan Hasselt 一起）第 12 章中提出的资格迹的前向和后向视图的精确等价的思想。Sutton 还感谢艾伯

塔省政府和加拿大国家科学与工程研究委员会在第二版构思和撰写期间给予他的支持和自由。他特别要感谢 Randy Goebe 为艾伯塔省的研究创造了一个支持性和有远见的环境。他还想感谢 DeepMind 在写这本书的最后六个月里对他的支持。

最后，我们要感谢在互联网上发布的第二版草稿的许多细心的读者。他们发现了我们遗漏的许多错误，并提醒了我们一些可能的困惑之处。



第一版前言

我们最初在 1979 年末开始关注于现在所谓的强化学习。我们俩都在马萨诸塞大学工作，致力于最早项目之一，以复兴像适应性元素这样的神经元网络可能被证明是有前途的人工适应性智能的方法的想法。该项目探索了由 A. Harry Klopf 开发的“自适应系统的静态理论”。Harry 的工作是丰富的思想源泉，我们被允许对其进行批判性的探索，并将其与自适应系统中以前的悠久历史进行比较。我们的任务变成了将这些想法分开并理解它们之间的关系和相对重要性。这种情况一直持续到今天，但是在 1979 年，我们开始意识到，也许很久以来就被认为是最简单的想法，从计算的角度来看却很少受到关注。这仅仅是一个学习系统的理想，它需要某种东西，它可以调整其行为，以便最大化来自其环境的特殊信号。这就是“享乐主义”学习系统的理想，或者正如我们现在所说的，强化学习的思想。

和其他人一样，我们感觉强化学习在控制论和人工智能的早期就已经进行了彻底的探索。然而，经过仔细检查，我们发现它只被略微探索过。尽管强化学习显然推动了一些最早的学习计算研究，但这些研究人员中的大多数已经转向其他方面，如模式分类、监督学习和自适应控制，或者他们完全放弃了学习研究。结果，涉及学习如何从环境中获取东西的特殊问题受到相对较少的关注。回过头来看，关注这个思想是启动这一研究分支的关键一步。强化学习的计算研究进展甚微，直到人们意识到这一基本思想还没有被彻底探索。

从那时起，该领域已经走了很长一段路，并在多个方向上发展和成熟。强化学习已逐渐成为机器学习、人工智能和神经网络研究中最活跃的研究领域之一。该领域已经发展出坚实的数学基础和令人印象深刻的广泛应用。强化学习的计算研究现已成为一个广阔的领域，在世界各地有数百名活跃的研究人员，他们来自不同的学科，如心理学、控制理论、人工智能和神经科学。特别重要的是建立和发展了与最优控制和动态规划理论的关系。从互动中学习以实现目标的整体问题仍远未解决，但我们对它的理解已经有了显著提高。我们现在可以将组成部分的思想，如时间差分学习、动态规划和函数逼近，放在一个关于整体问题的连贯视角中。

我们写这本书的目的是对强化学习的关键思想和算法提供一个清晰而简单的描述。我们想让所有相关学科的读者都能了解我们的论述方法，但我们不能详细介绍所有这些观点。在很大程度上，我们的论述采取了人工智能和工程学的观点。覆盖与其他领域的联系，我们将其留给其他人或其他时间。我们还选择不对强化学习进行严格的形式化处理，我们没有达到数学抽象的最高水平，也没有依赖于定理证明格式。我们试图选择一定程度的数学细节，以使数学朝正确的方向倾斜，而又不会分散基础思想的简单性和潜在的普遍性。

从某种意义上说，我们致力于这本书已有三十年，我们有很多人要感谢。首先，我们感谢那些亲自帮助我们发展本书中提出的总体观点的人：Harry Klopf 帮助我们认识到

需要复兴强化学习；Chris Watkins, Dimitri Bertsekas, John Tsitsiklis 和 Paul Werbos 帮助我们看到关系对动态规划的价值；John Moore 和 Jim Kehoe 来自动物学习理论的见解和启发；Oliver Selfridge 强调适应的广度和重要性；更一般地说，我们的同事和学生做出了无数的贡献：Ron Williams, Charles Anderson, Satinder Singh, Sridhar Mahadevan, Steve Bradtke, Bob Crites, Peter Dayan 和 Leemon Baird。通过与 Paul Cohen, Paul Utgoff, Martha Steenstrup, Gerry Tesauro, Mike Jordan, Leslie Kaelbling, Andrew Moore, Chris Atkeson, Tom Mitchell, Nils Nilsson, Stuart Russell, Tom Dietterich, Tom Dean 的讨论，我们的强化学习观得到了极大的丰富。我们感谢 Michael Littman, Gerry Tesauro, Bob Crites, Satinder Singh 和 Wei Zhang 分别提供了第 4.7、15.1、15.4、15.4 和 15.6 节的详细信息。我们感谢空军科学研究所，美国国家科学基金会和 GTE 实验室的长期和有远见的支持。

我们还要感谢阅读本书草稿并提供宝贵意见的许多人，包括 Tom Kalt, John Tsitsiklis, Paweł Cichosz, Olle Gallmo, Chuck Anderson, Stuart Russell, Ben Van Roy, Paul Steenstrup, Paul Cohen, Sridhar Mahadevan, Jette Randlov, Brian Sheppard, Thomas O'Connell, Richard Coggins, Cristina Versino, John H. Hiett, Andreas Badelt, Jay Ponte, Joe Beck, Justus Piater, Martha Steenstrup, Satinder Singh, Tommi Jaakkola, Dimitri Bertsekas, Torbjörn Ekman, Christina Björkman, Jakob Carlstrom 和 Olle Palmgren。最后，我们感谢 Gwyn Mitchell 在许多方面提供的帮助，并感谢 Harry Stanton 和 Bob Prior 成为麻省理工学院出版社的冠军。

符号总结

大写字母用于随机变量，而小写字母用于随机变量的值和标量函数。要求是实值向量的量以粗体和小写形式书写（即使是随机变量）。矩阵是粗体大写字母。

\doteq : 定义为真的等价关系

\approx : 近似相等

\propto : 成正比

$\Pr\{X = x\}$: 随机变量 X 取值 x 的概率

$X \sim p$: 随机变量 X 选自分布 $p(x) \doteq \Pr\{X = x\}$

$\mathbb{E}[X]$: 随机变量 X 的期望，即 $\mathbb{E}[X] \doteq \sum_x p(x)x$

$\arg \max_a f(a)$: $f(a)$ 取最大值时的 a 值

$\ln x$: x 的自然对数

$e^x, \exp(x)$: 自然对数的底取 x 次方， $e \approx 2.71828$, $e^{\ln x} = x$

\mathbb{R} : 实数集

$f : \mathcal{X} \rightarrow \mathcal{Y}$: 从集合 \mathcal{X} 的元素到集合 \mathcal{Y} 的元素的函数 f

\leftarrow : 赋值

$(a, b]$: a 和 b 之间的实际间隔，包括 b 但不包括 a

ε : 在 ε 贪婪策略中采取随机动作的概率

α, β : 步长参数

γ : 折扣率参数

λ : 资格迹的衰减率参数

$\mathbb{I}_{predicate}$: 指示函数（如果 $predicate$ 为真，则 $predicate \doteq 1$ ，否则为 0）

多臂赌博机问题:

k : 动作(臂)数

t : 离散时间步或者玩耍次数

$q_*(a)$: 动作 a 的真实值 (期望奖励)

$Q_t(a)$: $q_*(a)$ 在时间 t 处的估计

$N_t(a)$: 在时间 t 之前选择动作 a 的次数

$H_t(a)$: 在时间 t 选择动作 a 的习得偏好

$\pi_t(a)$: 在时间 t 选择动作 a 的概率

\bar{R}_t : 给定 π_t ，在时间 t 期望奖励的估计

马尔可夫决策过程:

s, s' : 状态

a : 动作

- r : 奖励
- \mathcal{S} : 所有非终结状态的集合
- \mathcal{S}^+ : 所有状态的集合，包括终结状态
- $\mathcal{A}(s)$: 状态 s 中所有可用动作的集合
- \mathcal{R} : 所有可能的奖励的集合， \mathbb{R} 的有限子集
- \subset : 子集，例如， $\mathcal{R} \subset \mathbb{R}$
- \in : 元素，例如， $s \in \mathcal{S}$, $r \in \mathcal{R}$
- $|\mathcal{S}|$: 集合 \mathcal{S} 中的元素数
- t : 离散时间步
- $T, T(t)$: 一个回合的最后时间步，或包括时间步 t 的回合的最后时间步
- A_t : t 时刻的行动
- S_t : t 时刻的状态，通常是随机的，归因于 S_{t-1} 和 A_{t-1}
- R_t : t 时刻的奖励，通常是随机的，归因于 S_{t-1} 和 A_{t-1}
- π : 策略（决策规则）
- $\pi(s)$: 在确定性策略 π 下在状态 s 中采取的动作
- $\pi(a|s)$: 在随机策略 π 下在状态 s 采取动作 a 的概率
- G_t : 时间 t 后的回报
- h : 水平线，向前看的时间步长
- $G_{t:t+n}, G_{t:h}$: 从 $t+1$ 到 $t+n$ 或到 h 的 n 步回报（折扣和修正）
- $\bar{G}_{t:h}$: $t+1$ 至 h 的平滑回报（未折扣和未修正）（第 5.8 节）
- G_t^λ : λ 回报（第 12.1 节）
- $G_{t:h}^\lambda$: 截断，修正的 λ 回报（第 12.3 节）
- $G_t^{\lambda s}, G_t^{\lambda a}$: 按估计状态或作用，值修正的 λ 回报（第 12.8 节）
- $p(s', r|s, a)$: 从状态 s 和动作 a 转移到具有奖励 r 的状态 s' 的概率
- $p(s'|s, a)$: 从状态 s 采取动作 a 转换到状态 s' 的概率
- $r(s, a)$: 采取动作 a 后状态 s 期望立即回报
- $r(s, a, s')$: 在动作 a 下从 s 转移到 s' 的期望立即回报
- $v_\pi(s)$: 策略 π 下状态 s 的值（期望回报）
- $v_*(s)$: 最优策略 π 下状态 s 的值
- $q_\pi(s, a)$: 策略 π 下状态 s 中采取动作 a 的值
- $q_*(s, a)$: 最优策略 π 下状态 s 中采取动作 a 的值
- V, V_t : 状态值函数 v_π 或 v_* 的数组估计
- Q, Q_t : 动作值函数 q_π 或 q_* 的数组估计
- $\bar{V}_t(s)$: 近似动作值的期望，比如， $\bar{V}_t(s) \doteq \sum_a \pi(a|s) Q_t(s, a)$
- U_t : 在时间 t 估计的目标
- δ_t : t (随机变量) 处的时间差分 (TD) 误差（第 6.1 节）
- δ_t^s, δ_t^a : TD 误差的状态和动作特定形式（第 12.9 节）
- n : 在 n 步方法中， n 是自举的步数

- d : 维度—— \mathbf{w} 的组成数量
 d' : 替代维度—— $\boldsymbol{\theta}$ 的组成数量
 \mathbf{w}, \mathbf{w}_t : 近似值函数下的权重的 d 维向量
 $w_i, w_{t,i}$: 习得的权重向量的第 i 个分量
 $\hat{v}(s, \mathbf{w})$: 给定权重向量 \mathbf{w} 的状态 s 的近似值
 $v_{\mathbf{w}}(s)$: $\hat{v}(s, \mathbf{w})$ 的替代符号
 $\hat{q}(s, a, \mathbf{w})$: 给定权重向量 \mathbf{w} 的状态动作对 s, a 的近似值
 $\nabla \hat{v}(s, \mathbf{w})$: $\hat{v}(s, \mathbf{w})$ 相对于 \mathbf{w} 的偏导数的列向量
 $\nabla \hat{q}(s, a, \mathbf{w})$: $\hat{q}(s, a, \mathbf{w})$ 相对于 \mathbf{w} 的偏导数的列向量
 $\mathbf{x}(s)$: 处于状态 s 时可见的特征向量
 $\mathbf{x}(s, a)$: 在状态 s 采取动作 a 时可见的特征向量
 $x_i(s), x_i(s, a)$: 向量 $\mathbf{x}(s)$ 或 $\mathbf{x}(s, a)$ 的第 i 个分量
 \mathbf{x}_t : $\mathbf{x}(S_t)$ 或 $\mathbf{x}(S_t, A_t)$ 的简写
 $\mathbf{w}^\top \mathbf{x}$: 向量的内积, $\mathbf{w}^\top \mathbf{x} \doteq \sum_i w_i x_i$, 例如, $\hat{v}(s, \mathbf{w}) \doteq \mathbf{w}^\top \mathbf{x}(s)$
 \mathbf{v}, \mathbf{v}_t : 用于学习 \mathbf{w} 的权重的次 d 维向量 (第 11 章)
 \mathbf{z}_t : 时间 t 时资格迹的 d 维向量 (第 12 章)
 $\boldsymbol{\theta}, \boldsymbol{\theta}_t$: 目标策略的参数向量 (第 13 章)
 $\pi(a|s, \boldsymbol{\theta})$: 给定参数向量 $\boldsymbol{\theta}$, 在状态 s 下采取动作 a 的概率
 $\pi_{\boldsymbol{\theta}}$: 对应于参数 $\boldsymbol{\theta}$ 的策略
 $\nabla \pi(a|s, \boldsymbol{\theta})$: $\pi(a|s, \boldsymbol{\theta})$ 相对于 $\boldsymbol{\theta}$ 的偏导数的列向量
 $J(\boldsymbol{\theta})$: 策略 $\pi_{\boldsymbol{\theta}}$ 的性能度量
 $\nabla J(\boldsymbol{\theta})$: $J(\boldsymbol{\theta})$ 相对于 $\boldsymbol{\theta}$ 的偏导数的列向量
 $h(s, a, \boldsymbol{\theta})$: 基于 $\boldsymbol{\theta}$ 选择状态 s 中的动作 a 的偏好
 $b(a|s)$: 学习目标策略 π 时用于选择动作的行为策略
 $b(s)$: 策略梯度方法的基线函数 $b : \mathcal{S} \mapsto \mathbb{R}$
 b : MDP 或搜索树的分支因子
 $\rho_{t:h}$: 从时间 t 到时间 h 的重要性采样率 (第 5.5 节)
 ρ_t : 仅时间 t 的重要性采样率, $\rho_{t:t}$
 $r(\pi)$: 策略 π 的平均奖励 (奖励率) (第 10.3 节)
 \bar{R}_t : t 时刻 $r(\pi)$ 的估计
 $\mu(s)$: 状态 s 上的在线策略分布 (第 9.2 节)
 $\boldsymbol{\mu}$: 所有 $s \in S$ 的 $\mu(s)$ 的 $|\mathcal{S}|$ 维向量
 $\|v\|_\mu^2$: 值函数 v 的 μ 加权平方范数, 即 $\|v\|_\mu^2 \doteq \sum_{s \in \mathcal{S}} \mu(s) v(s)^2$
 $\eta(s)$: 每回合访问状态 s 的期望次数 (第 199 页)
 Π : 值函数的投影算子 (第 268 页)
 B_π : 值函数的贝尔曼算子 (第 11.4 节)
 \mathbf{A} : $d \times d$ 矩阵 $\mathbf{A} \doteq \mathbb{E}[\mathbf{x}_t (\mathbf{x}_t - \gamma \mathbf{x}_{t+1})^\top]$
 \mathbf{b} : d 维向量 $\mathbf{b} \doteq \mathbb{E}[R_{t+1} \mathbf{x}_t]$

w_{TD}: TD 不动点 $\mathbf{w}_{\text{TD}} \doteq \mathbf{A}^{-1}\mathbf{b}$ (一个 d 维向量, 第 9.4 节)

I: 单位矩阵

P: π 下的状态转移概率的 $|\mathcal{S}| \times |\mathcal{S}|$ 维矩阵

D: 对角线上有 μ 的 $|\mathcal{S}| \times |\mathcal{S}|$ 维对角矩阵

X: 以 $x(s)$ 为行的 $|\mathcal{S}| \times |d|$ 维矩阵

$\bar{\delta}_{\mathbf{w}}(s)$: 状态 s 的 $v_{\mathbf{w}}$ 的贝尔曼误差 (期望 TD 误差) (第 11.4 节)

$\bar{\delta}_{\mathbf{w}}$, BE: 贝尔曼误差向量, 分量为 $\bar{\delta}_{\mathbf{w}}(s)$

$\overline{\text{VE}}(\mathbf{w})$: 均方值误差 $\overline{\text{VE}}(\mathbf{w}) \doteq \|v_{\mathbf{w}} - v_{\pi}\|_{\mu}^2$

$\overline{\text{BE}}(\mathbf{w})$: 均方贝尔曼误差 $\overline{\text{BE}}(\mathbf{w}) \doteq \|\bar{\delta}_{\mathbf{w}}\|_{\mu}^2$

$\overline{\text{PBE}}(\mathbf{w})$: 均方投影贝尔曼误差 $\overline{\text{PBE}}(\mathbf{w}) \doteq \|\Pi \bar{\delta}_{\mathbf{w}}\|_{\mu}^2$

$\overline{\text{TDE}}(\mathbf{w})$: 均方时间差分误差 $\overline{\text{TDE}}(\mathbf{w}) \doteq \mathbb{E}_b[\rho_t \delta_t^2]$ (第 11.5 节)

$\overline{\text{RE}}(\mathbf{w})$: 均方回报误差 (第 11.6 节)



翻译说明

强化学习领域目前国外研究比较多，所以我们常常需要看英文资料才能了解到最新的前沿动态。为了兼顾英文资料而不引起混淆，本人不对强化学习中的非熟知人名、特殊词汇以及强化学习中的方法和理论词汇进行翻译。另外，翻译的页码对应于原书的页码。

由于英文和中文各方面的差异，每个人有不同的翻译习惯，一个单词或者词语可以被翻译成各个形式。因此，很有必要列出翻译过程中遇到的特殊单词或词语，这种良好的方式可以帮助读者看其他文献或者书籍时迅速进行转换。下面列出本人在翻译过程中意译的特殊单词或词汇：

- action: 动作
- associations: 关联
- agent: 译成 agent，可译成智能体，代理
- artificial neural network: 人工神经网络
- actor-critic: 译成 actor-critic，可译成演员-评论家
- action-value: 动作值
- Alan Turing: 阿兰·图灵（人名）
- arrays: 数组
- active exploration: 主动探索
- approximation: 近似，逼近
- absorbing state: 吸收状态
- action-value function: 动作值函数
- afterstates: 后期状态
- behavior/behaving/behavioral/behave: 行为，表现
- backgammon: 双陆棋
- Bellman: 贝尔曼（人名）
- bellman equation: 贝尔曼方程
- backwards: 向后
- bootstrap/bootstrapping: 自举
- blackjack: 21 点
- backpropagation neural networks: 反向神经网络
- backward-view: 后向观点
- bias: 有偏
- baseline: 基线，基准线
- behavior policy: 行为策略
- backward focusing: 向后聚焦

- background planning: 背景规划
curse of dimensionality: 维度灾难, 维度诅咒
credit: 信用
control: 控制
controller: 控制器
context: 上下文
conditioned reflexes: 条件反射
Claude Shannon: 克劳德 · 香农
critic: 评论家
continuing tasks: 连续任务
consistency condition: 一致性条件
certainty-equivalence estimate: 确定性等价估计
control variate: 控制变量
dynamical: 动态
dynamic programming : 动态规划
dopamine: 多巴胺
Double learning: 双重学习, double learning
Double Q-learning:
discounting: 折扣
discount rate: 折扣率
distribution models: 分布模型
decision-time planning: 决策时规划
exploration/explore/exploring: 探索
exploitation/exploit/exploiting: 利用
exploring starts: 探索起点
environment: 环境
events: 事件
evolutionary: 进化
episodes: 回合
episodic tasks: 回合任务
 ε -greedy: ε -greedy
exponential recency-weighted average: 指数近期加权平均
evaluates: 评价, 评估
experience: 经验
expected update: 期望更新
eligibility traces: 资格迹
function: 函数
forward: 向前

forward-view: 前向观点
flat partial returns:
goal: 目标
goal-directed: 目标导向
genetic algorithms: 遗传算法
genetic programming: 遗传规划
generalize: 泛化
game theory: 博弈论
greedy: 贪婪
generalized policy iteration: 广义策略迭代
hierarchical: 层级
heuristic search: 启发式搜索
interactive: 互动, 交互
immediate: 即时
iterative policy evaluation: 迭代策略评估
importance sampling: 重要性采样
importance-sampling ratio: 重要性采样率
k-armed bandit: k 臂赌博机
label: 标签
look ahead: 预见
Law of Effect: 效果定律
Least-Mean-Square: 最小均方
map: 映射
markov decision process: 马尔可夫决策过程
model: 模型
model-based: 基于模型
model-free: 免模型, 无模型
Monte Carlo: 蒙特卡罗, 蒙特卡洛
Monte Carlo Tree search: 蒙特卡罗树搜索, 蒙特卡洛树搜索
Monte Carlo methods: 蒙特卡罗方法, 蒙特卡洛方法
memory: 内存, 存储
organisms: 有机体
optimal: 最优
optimal control: 最优控制
optimal policy: 最优策略
optimal state-value function: 最优状态值函数
offline: 离线
online: 在线

- one-armed bandit: 单臂赌博机
one-step: 一步, 单步
option: 译成 option
planning: 规划
psychology: 心理学
policy: 策略
policy evaluation: 策略评估
Policy Improvement: 策略改进
perceived: 感知
prior knowledge: 先验知识
player: 玩家
Pavlov: 巴甫洛夫 (人名)
pseudocode: 伪代码
prioritized sweeping: 优先扫描
Q-learning: 译成 Q-learning, 可译成 Q-学习
reinforcement learning: 强化学习
rules: 规则
reward: 奖励
return: 回报
rollout algorithms: 展开算法
state: 状态
supervised learning: 监督学习
search: 搜索
signal: 信号
simulated annealing: 模拟退火
static: 静态
space: 空间
step: 步
step-size: 步长
slot machine: 老虎机
state-value function: 状态值函数
self-consistency condition: 自治条件
sample models: 样本模型
trial-and-error: 试错
tic-tac-toe: 井字游戏
temporal-difference: 时间差分
tables/tabular: 表格
time step: 时间步

Thompson: 汤普森

terminal state: 终止状态

trails: 试验

target policy: 目标策略

tree-backup algorithm: 树备份算法

unsupervised learning: 非监督学习

uncertainty: 不确定性

value function: 值函数

value: 值, 价值

value iteration: 值迭代



第一章 介绍

当我们思考学习的本质时，我们通过与环境互动来学习的想法很可能首先出现在我们的脑海中。当婴儿玩耍、挥舞手臂或环顾四周时，他没有明确的老师，但他确实与周围的环境有直接的感觉运动联系。运用这种联系会产生大量关于因果关系，动作后果以及为了实现目标应该做些什么等方面的信息。在我们的一生中，这样的互动无疑是关于我们的环境和我们自身的主要知识来源。无论我们是在学习开车还是在进行对话，我们都敏锐地意识到我们的环境对我们所做的事情如何反应，我们试图通过我们的行为来影响所发生的事情。从互动中学习是几乎所有学习和智能理论的基本理念。

在这本书中，我们探索了一种从互动中学习的计算方法。我们不是直接对人或动物如何学习进行理论推导，而是主要探索理想化的学习情境，并评估各种学习方法的有效性¹。也就是说，我们采用人工智能研究人员或工程师的观点。我们探索能有效解决科学或经济利益的学习问题的机器设计，通过数学分析或计算实验来评估设计。我们探索的方法被称为强化学习，与其他机器学习方法相比，它更注重从互动中进行目标导向学习。

1.1 强化学习

强化学习是学习做什么——如何将情境映射到动作——以便最大化数字奖励信号。学习者不会被告知要采取哪些动作，而是必须通过尝试发现哪些动作产生的奖励最大。在最有趣和最具挑战性的情况下，动作不仅会影响到眼前的奖励，还会影响到下一情境，并通过这种情况影响到所有随后的奖励。试错搜索和延迟奖励这两个特征是强化学习的两个最重要的区别性特征。

强化学习，就像许多以“ing”结尾的主题一样，比如机器学习和登山，同时也是一个问题，一类在这个问题上很有效的解决方法，以及研究这个问题及其解决方法的领域。对这三件事使用一个名字是很方便的，但同时保持三者在概念上的分离也很重要。特别是，问题和解决方法的区分在强化学习中非常重要；未能做出这种区分是许多困惑的根源。

我们利用动态系统理论的思想将强化学习问题形式化，特别是作为不完全已知的马尔可夫决策过程的最优控制。这种形式化的细节必须等到第3章，但基本思想只是捕捉学习 agent 随着时间的推移与环境互动以实现目标所面临的实际问题的最重要方面。学习 agent 必须能够在一定程度上感知其环境的状态，并且必须能够采取影响该状态的动作。agent 还必须有一个或多个与环境状态相关的目标。马尔可夫决策过程的目的是只包括这三个方面——感觉、行动和目标——以其最简单的可能形式，而不会使它们中的任何一个变得琐碎。任何适合于解决这类问题的方法都被称为强化学习方法。

强化学习不同于监督学习，监督学习是目前机器学习领域研究最多的一种学习方式。监督学习是从知识渊博的外部监督者提供的带标签示例的训练集中学习。每个示例都是

¹ 第14章和第15章总结了与心理学和神经科学的关系。

对情况的描述以及系统应针对该情况采取的正确动作的规范（标签），该规范通常用于标识该情况所属的类别。这种学习的目的是使系统推断或概括其对策，以使其在训练集中不存在的情况下正确行动。这是一种重要的学习方式，但单靠它并不足以从互动中学习。在互动问题中，要获得既正确又代表 agent 必须采取行动的所有情况的期望行为的示例通常是不切实际的。在未知领域——人们期望学习是最有益的——agent 必须能够从自己的经验中学习。

强化学习也不同于机器学习研究人员所说的无监督学习，无监督学习通常是关于发现隐藏在未标记数据集中的结构。监督学习和无监督学习这两个术语似乎对机器学习范式进行了详尽的分类，但事实并非如此。虽然有人可能会认为强化学习是一种无监督的学习，因为它不依赖于正确行为的示例，但强化学习试图最大化奖励信号，而不是试图发现隐藏的结构。揭示 agent 经验中的结构在强化学习中当然是有用的，但本身并不能解决最大化奖励信号的强化学习问题，因此我们认为强化学习是第三种机器学习范式，与监督学习、无监督学习以及其他范式并驾齐驱。

强化学习而不是其他类型学习中会出现的挑战之一是探索与利用之间的权衡。为了获得更多的奖励，强化学习 agent 必须偏爱过去尝试过并且发现能有效产生奖励的行为。但要发现这样的动作，它必须尝试以前没有选择过的动作。Agent 必须利用它已经经历过的来获得奖励，但是它也必须探索以便在将来做出更好的行动选择。进退两难的境地是，无论是探索还是利用，都不可能在不失败的情况下专心致志地进行。Agent 必须尝试各种方法，并逐渐偏爱那些看起来最好的方法。对于随机任务，每个动作都必须多次尝试才能获得期望回报的可靠估计。探索-利用问题是数学家们几十年来一直在深入研究的问题，但至今仍未得到解决。目前，我们只是注意到，平衡探索和利用的整个问题甚至不会出现在监督和无监督的学习中，至少在这些范式的最纯粹形式中是不会出现的。

强化学习的另一个关键特征是它明确地考虑了目标导向的 agent 与不确定环境交互的整个问题。这与许多考虑子问题的方法形成鲜明对比，这些方法没有考虑它们可能如何适应更大的图景。例如，我们已经提到，许多机器学习研究都与监督学习有关，但没有明确说明这种能力最终将如何发挥作用。其他研究人员发展了有总体目标的规划理论，但没有考虑规划在实时决策中的作用，也没有考虑规划所需的预测模型从何而来的问题。虽然这些方法已经产生了许多有用的结果，但它们对孤立的子问题的关注是一个很大的限制。

强化学习采取相反的策略，从一个完整的、互动的、寻求目标的 agent 开始。所有的强化学习 agent 都有明确的目标，能够感知环境的各个方面，并能够选择动作来影响环境。此外，通常从一开始就假定 agent 必须运行，尽管它所面临的环境有很大的不确定性。当强化学习涉及规划时，它必须解决规划和实时动作选择之间的相互作用，以及如何获取和改进环境模型的问题；当强化学习涉及监督学习时，它这样做是出于特定的原因，这些原因决定了哪些能力是关键的，哪些能力不是关键的。为了使学习研究取得进展，重要的子问题必须被分离和研究，但它们应该是在完整的、互动的、寻求目标的 agent 中发挥明确作用的子问题，即使完整 agent 的所有细节还不能被填充。

我们所说的完整的、互动的、寻找目标的 agent 并不总是指像完整的有机体或机器人

这样的东西。这些都是很明显的例子，但是一个完整的、互动的、寻求目标的 agent 也可以是一个更大的行为系统的组成部分。在这种情况下，agent 直接与较大系统的其余部分交互，并间接与较大系统的环境交互。一个简单的例子是一个 agent，它监控机器人电池的电量，并向机器人的控制架构发送命令。这个 agent 理的环境是机器人的其余部分以及机器人的环境。人们必须超越最明显的 agent 及其环境的例子来理解强化学习框架的普遍性。

现代强化学习最令人兴奋的方面之一是它与其他工程和科学学科的大量和富有成效的互动。强化学习是人工智能和机器学习数十年来与统计学、最优化和其他数学学科更紧密结合的趋势的一部分。例如，一些强化学习方法使用参数化逼近器学习的能力解决了运筹学和控制理论中经典的“维数灾难”。更明显的是，强化学习还与心理学和神经科学产生了强烈的互动，具有双向的实质性好处。在机器学习的所有形式中，强化学习是最接近于人类和其他动物所做的那种学习，强化学习的许多核心算法最初都是受到生物学习系统的启发。强化学习也带来了回报，既通过动物学习的心理学模型更好地匹配了一些经验数据，也通过大脑奖励系统的部分有影响力的模型。本书的主体提出了与工程和人工智能有关的强化学习的思想，并在第 14 章和第 15 章中概述了与心理学和神经科学的联系。

最后，强化学习也是人工智能向简单通用原则回归的更大趋势的一部分。自 20 世纪 60 年代末以来，许多人工智能研究人员假设没有可以发现的一般原理，相反，智能是由于拥有大量特殊目的的技巧、程序和启发式方法。人们有时会说，如果我们能把足够多的相关事实输入机器，比方说一千万或十亿，那么它就会变得智能。基于一般原则的方法，如搜索或学习，被描述为“弱方法”，而那些基于特定知识的方法被称为“强方法”。这种观点在今天仍然很普遍，但并不占主导地位。从我们的观点来看，现在就下结论还为时过早：在寻找一般原则方面投入的努力太少，无法得出没有任何一般原则的结论。现代人工智能现在包括许多研究寻找学习、搜索和决策的一般原则。目前还不清楚钟摆会往回摆动多久，但强化学习研究肯定是朝着更简单、更少的人工智能一般原理摇摆的一部分。

1.2 例子

理解强化学习的一个好方法是考虑一些指导它发展的例子和可能的应用。

- 一个象棋大师出棋。这种选择是由规划——预期可能的回复和反回复——以及对特定位置和移动的期望的即时、直觉判断所决定。
- 自适应控制器实时调整炼油厂的操作参数。控制器根据指定的边际成本优化产量/成本/质量权衡，而不严格遵守工程师最初建议的设定点。
- 一只小瞪羚在出生几分钟后挣扎着站起来。半小时后，它以每小时 20 英里的速度运行。
- 移动机器人决定它是否应该进入一个新房间来收集更多的垃圾，还是开始试图找到回到电池充电站的路。它根据电池当前的充电水平以及过去找到充电器的速度和容

易程度做出决定。

- Phil 在准备他的早餐。仔细观察，即使是这种看似平凡的活动也揭示了一个复杂的条件行为和连锁的目标-子目标关系的网络：走到橱柜前，打开它，选择一个麦片盒，然后伸手去拿，抓住，再拿出盒子。要获得碗、勺子和牛奶盒，还需要其他复杂的、可调的、互动的行为序列。每一步都包括一系列的眼球运动，以获取信息并指导到达和运动。人们会不断地快速判断如何搬运这些物品，或者在拿到其他物品之前先把其中一些摆放到餐桌上是否更好。每一步都有目标的引导，比如抓住勺子或拿到冰箱，并服务于其他目标，比如一旦谷类食品准备好，就用勺子吃，并且最终获得营养。无论他是否意识到这一点，Phil 都在获取关于他身体状况的信息，这些信息决定了他的营养需求、饥饿程度和食物偏好。

这些例子都有一些非常基本的特点，因此很容易被忽略。所有这些都涉及到一个积极的决策 agent 和其环境之间的互动，在这种互动中，agent 寻求实现一个目标，尽管其环境是不确定的。agent 的动作被允许影响环境的未来状态（例如，下一个象棋位置、精炼厂的蓄水池水位、机器人的下一个位置和其电池的未来充电水平），从而影响 agent 在以后时间可用的动作和机会。正确的选择需要考虑到动作的间接、延迟后果，因此可能需要远见或规划。

与此同时，在所有这些例子中，动作的效果无法完全预测；因此，agent 必须频繁地监视其环境并做出适当的反应。例如，Phil 必须注意他倒进麦片碗里的牛奶，以防止牛奶溢出。所有这些例子都涉及明确的目标，即 agent 可以根据它能直接感觉到的东西来判断朝着目标的进展。棋手知道自己是否赢了，炼油厂的控制器知道正在生产多少石油，小羚羊知道它何时倒下，移动机器人知道它的电池什么时候耗尽，Phil 知道他是否正在享受他的早餐。

在所有这些例子中，随着时间的推移 agent 可以利用其经验来提高其性能。棋手提炼他用来评估位置的直觉，从而提高他的棋艺；瞪羚幼仔提高了它奔跑的能力；Phil 学会简化早餐的制作。Agent 从一开始就带给任务的知识（无论是从以前的相关任务经验还是通过设计或演化而来）会影响有用的或易于学习的内容，但是与环境的交互对于调整行为以利用任务的特定特性是至关重要的。

1.3 强化学习的要素

除了 agent 和环境之外，还可以确定强化学习系统的四个主要子要素：策略、奖励信号、值函数，以及可选的环境模型。

策略定义学习 agent 在给定时间的行为方式。粗略地说，策略是从感知的环境状态到处于这些状态时要采取的动作的映射。它对应于心理学上所说的一套刺激——反应规则或关联。在某些情况下，策略可以是简单的函数或查找表，而在其他情况下，它可能涉及诸如搜索过程之类的大量计算。策略是强化学习 agent 的核心，因为只有它才能决定行为。一般来说，策略可能是随机的，为每个动作指定概率。

奖励信号定义了强化学习问题的目标。在每个时间步，环境都会向强化学习 agent 发

送一个称为奖励的单个数值。Agent 的唯一目标是使其长期获得的总奖励最大化。因此，奖励信号定义了对 agent 来说什么是好的和不好的事件。在生物系统中，我们可能认为奖励是快乐或痛苦经历的衍生物。它们是 agent 面临的问题的直接和决定性特征。奖励信号是改变策略的主要依据；如果策略选择的动作之后是低奖励，那么策略可能会改变，以选择未来在这种情况下的其他动作。一般来说，奖励信号可能是环境状态和所采取动作的随机函数。

奖励信号表明什么是即时好的，而值函数则指明什么是长期好的。粗略地说，状态的值是 agent 从该状态开始，在未来期望积累的奖励总额。奖励决定了环境状态的即时、内在的可取性，而价值则表明了在考虑到可能跟随的状态和那些状态可获得的奖励之后，状态的长期可取性。例如，一个状态可能总是产生较低的即时奖励，但仍然有很高的价值，因为其他产生高奖励的状态经常紧随其后。反之亦然。打个比方，奖励有点像快乐（如果高的话）和痛苦（如果低的话），而价值观对应于一种更精致和更有远见的判断，即我们对环境处于特定状态的满意或不满程度。

奖励在某种意义上是首要的，而价值作为对奖励的预测是次要的。没有奖励就没有价值，估计价值的唯一目的就是获得更多的奖励。然而，我们在制定和评估决策时最关心的是价值。动作选择是基于价值判断做出的。我们寻求能带来最高价值而非最高奖励的动作，因为从长远来看，这些动作为我们获得了最大量的奖励。不幸的是，确定价值要比确定奖励困难得多。奖励基本上是由环境直接给出的，但价值必须根据 agent 在其整个生命周期内所做的观察序列来估计和重新估计。实际上，我们考虑的几乎所有强化学习算法中最重要的组成部分是一种科学估算价值的方法。价值评估的中心作用可以说过去六十年来有关强化学习的最重要的知识。

一些强化学习系统的第四个也是最后一个要素是环境模型。这是一种模仿环境行为的方法，或者更笼统地说，它允许对环境将如何表现做出推断。例如，给定状态和动作，模型可以预测所得的下一个状态和下一个奖励。模型是用于规划，我们所说的规划是指在实际经历之前，通过考虑未来可能发生的情况来决定动作方针的任何方式。使用模型和规划来解决强化学习问题的方法被称为基于模型的方法，这与简单的免模型方法相反，后者明确地说是试错学习者的方法，被视为几乎与规划对立。在第 8 章中，我们探索了强化学习系统，它通过试错，学习环境的模型，并使用该模型进行规划，同时学习。现代强化学习涵盖了从低级的试错学习到高级的深思熟虑的规划。

1.4 局限和范围

强化学习在很大程度上依赖于状态的概念，其作为策略和值函数的输入，以及作为模型的输入和输出。非正式地，我们可以认为状态是向 agent 传达特定时间“环境如何”的某种意义的信号。我们在这里使用的状态的正式定义由第 3 章中给出的马尔可夫决策过程框架给出。然而，更一般地，我们鼓励读者遵循非正式的含义，并将状态视为 agent 可以获得的关于其环境的任何信息。实际上，我们假设状态信号是由一些预处理系统产生的，这些预处理系统名义上是 agent 环境的一部分。我们不会在本书中讨论构造、改变或

学习状态信号的问题(除了第 17.3 节中的简要介绍)。我们采取这种方法不是因为我们认为状态表示不重要，而是为了充分关注决策问题。换句话说，我们在这本书中关注的不是设计状态信号，而是根据任何可用的状态信号来决定采取什么动作。

我们在这本书中考虑的大多数强化学习方法都是围绕估计值函数来构建的，但并不一定需要这样做来解决强化学习问题。例如，诸如遗传算法、遗传规划、模拟退火和其他优化方法的求解方法从不估计值函数。这些方法应用多个静态策略，每个静态策略都在较长时间内与单独的环境实例交互。获得最多奖励的策略及其随机变化被传递到下一代策略，并重复这一过程。我们称这些方法为进化方法，因为它们的操作类似于生物进化产生具有熟练行为的有机体的方式，即使它们在自己的一生中没有学习。如果策略空间非常小，或者可以被构造成使得好的策略很常见或者很容易找到，或者如果有很多时间用于搜索，那么进化方法可能是有效的。此外，进化方法在学习 agent 不能感知其环境的完整状态的问题上具有优势。

我们的重点是强化学习方法，这种方法在与环境互动的同时进行学习，而进化方法则不然。在许多情况下，能够利用个体行为互动细节的方法比进化方法更有效。进化方法忽略了强化学习问题的许多有用结构：它们没有利用这样一个事实，即它们正在寻找的策略是从状态到动作的函数；他们没有注意到个体在一生中经历了哪些状态，或者选择了哪些动作。在某些情况下，这种信息可能会产生误导(例如，当状态被错误感知时)，但更常见的情况是，它应该能够实现更有效的搜索。虽然进化和学习有许多共同的特点，并且可以自然地协同工作，但是我们并不认为进化方法本身特别适合强化学习问题，因此，我们在本书中不涉及它们。

1.5 拓展例子：井字游戏

为了说明强化学习的一般思想，并将其与其他方法进行比较，我们接下来将更详细地考虑一个例子。

想一想我们熟悉的孩子玩的井字游戏。两个玩家轮流在一块三乘三的棋盘上玩。一个玩家玩 X，另一个玩家玩 O，直到一个玩家通过水平、垂直或对角连续放置三个标记排成一行来获胜，就像 X 玩家在右边的游戏中所做的那样。如果棋盘填满了，没有一个玩家连续三个一行，那么这场比赛就是一场平局。因为一个技术娴熟的玩家可以玩得永远不会输，让我们假设我们的对手是一个不完美的玩家，他的玩法有时是不正确的，可以让我们获胜。就目前而言，事实上，让我们考虑平局和失败对我们同样不利。我们如何才能打造出一个能够发现对手游戏中的不完美之处，并学会最大化取胜机会的玩家呢？

X	O	O
O	X	X
		X

虽然这是一个简单的问题，但无法通过经典的技术以令人满意的方式轻松解决。例如，博弈论中经典的“极小极大”解决方案在这里是不正确的，因为它假设对手玩的是一种特殊的方式。例如，极小极大玩家永远不会达到它可能会输的游戏状态，即使实际上它总是因为对手的不正确游戏而从这种状态中获胜。序列决策问题的经典优化方法，如

动态规划，可以为任何对手的计算出最优解，但需要输入对手的完整说明，包括对手在每个棋盘状态下的每一步棋的概率。让我们假设这个问题没有先验的信息，因为它不适用于绝大多数实际感兴趣的问题。另一方面，这样的信息可以从经验中估计出来，在这种情况下，可以通过与对手进行多次游戏来估计。在这个问题上，我们能做的最好的事情是首先学习对手的行为模型，达到一定的置信度，然后应用动态规划来计算给定近似对手模型的最优解。最后，这与我们在本书后面讨论的一些强化学习方法没有什么不同。

应用于这个问题的进化方法将直接搜索可能的策略空间，以寻找赢得对手的可能性很高的策略。这里，策略是一种规则，它告诉玩家在游戏的每一种状态要走什么棋，状态是指三乘三的棋盘上 **X** 和 **O** 的每一种可能的配置。对于所考虑的每种策略，其获胜概率的估计将通过与对手进行一定数量的博弈来获得。然后，该评估将指示下一步考虑哪种或者哪些策略。一种典型的进化方法是在策略空间中攀爬，连续生成和评估策略，试图获得渐进的改进。或者，也许可以使用遗传方式的算法来维护和评估一组政策。实际上可以应用数百种不同的优化方法。

下面是如何利用值函数来处理井字游戏问题的方法。首先，我们会建立一个数字表格，一个代表游戏的每个可能状态。每个数字都是我们在该状态获胜的可能性的最新估计。我们把这个估计值当作状态的值，整个表就是学习值函数。状态 **A** 的值比状态 **B** 高，或者被认为比状态 **B**“更好”，如果我们当前估计的获胜概率比状态 **B** 高。假设我们总是玩 **X**，那么对于所有有三个 **X** 的状态来说，获胜的概率是 1，因为我们已经赢了。类似地，对于连续有三个 **O** 的所有状态，或者已经填满的状态，正确的概率都是 0，因为我们不能从他们那里获胜。我们将所有其他状态的初始值设置为 0.5，表示我们有 50% 的获胜机会。

然后我们和对手玩很多场游戏。为了选择我们的移动，我们检查每个可能的移动产生的状态（棋盘的每个空格对应一个状态），并在表中查找它们的当前值。大多数情况下，我们都在贪婪地移动，选择通向价值最大的状态的移动，也就是说，往具有最高的估计获胜概率移动。然而，有时我们会从其他动作中随机选择。这些被称为探索性的移动，因为它们使我们体验到我们本来可能永远看不到的状态。游戏中做出和考虑的一系列动作如图1.1所示。

当我们在玩的时候，我们会改变我们在游戏中发现自己所处的状态的值。我们试图让他们更准确地估计获胜的可能性。为此，我们在每次贪婪地移动之后将状态值“备份”到移动之前的状态，如图1.1中的箭头所示。更准确地说，较早状态的当前值被更新为更接近较晚状态的值，这可以通过将较早状态的值向较晚状态的值移动一小部分来实现。如果我们让 S_t 表示贪婪移动之前的状态，让 S_{t+1} 表示移动后的状态，则对 S_t 的估计值（表示为 $V(S_t)$ ）的更新可以写为

$$V(S_t) \leftarrow V(S_t) + \alpha[V(S_{t+1}) - V(S_t)],$$

其中 α 是一个小的正分数，称为步长参数，它影响学习速率。此更新规则是时间差分学习方法的一个例子，之所以这样命名是因为它的更改是基于两个连续时间的差异 $V(S_{t+1}) - V(S_t)$ 。

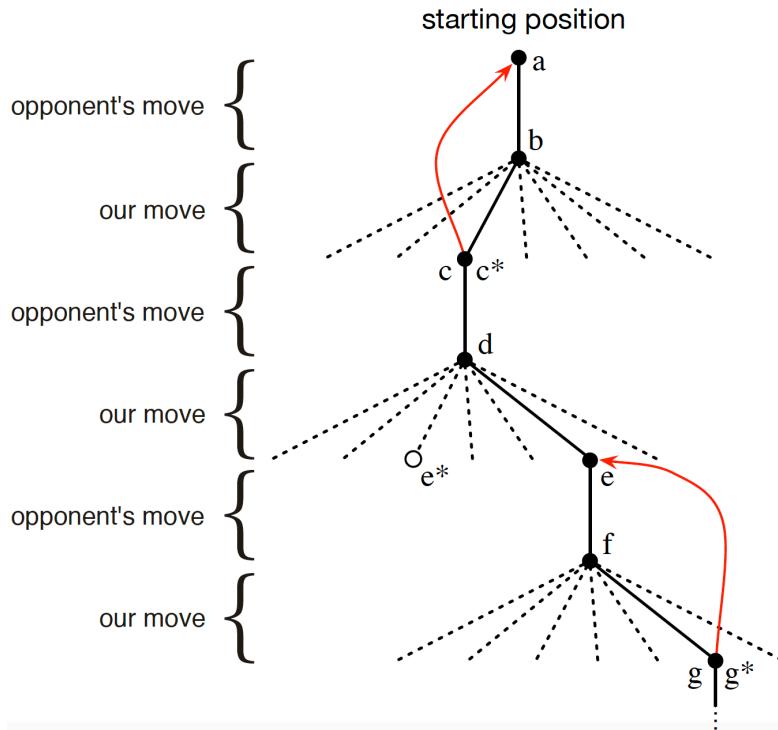


图 1.1: 井字游戏的一系列移动。实心的黑线表示在游戏中采取的移动；虚线表示我们（我们的强化学习玩家）考虑过但没有采取的移动。我们的第二个移动是探索性的移动，这意味着即使另一个兄弟移动，即导致 e^* 的那个，排名更高，我们还是采取了这个移动。探索性移动不会导致任何学习，但我们的每一次其他的移动可以，如红色箭头所示它们都会导致更新，其中估计值从树的后面节点移动到前面的节点，如正文中所详细描述的那样。

上面描述的方法在这项任务上执行得相当好。例如，如果步长参数随着时间的推移适当减小，那么对于任何固定的对手，这个方法都会收敛到我们的玩家在给定最优玩法的情况下从每个状态取胜的真实概率。此外，随后采取的移动（除探索移动外）实际上是针对这个（不完美的）对手的最优移动。换句话说，该方法收敛于与该对手进行博弈的最优策略。如果步长参数没有随着时间的推移一直减少到零，那么这个玩家也能很好地对付那些慢慢改变他们游戏方式的对手。

这个例子说明了进化方法和学习值函数的方法之间的区别。为了评估策略，进化方法保持策略不变，并与对手玩许多游戏，或者使用对手的模型来模拟许多游戏。获胜的频率给出了该策略获胜概率的无偏估计，并可用于指导下一个策略选择。但是每一个策略的改变都是在许多游戏之后才做出的，并且只使用每一个游戏的最终结果：游戏期间发生的事情被忽略。例如，如果玩家赢了，那么它在游戏中的所有行为都被给予信用，与对于获胜至关重要的特定移动无关。甚至对从未发生的移动也给予了肯定！相反，值函数方法允许评估各个状态。最后，进化方法和值函数方法都搜索策略空间，但是学习值函数会利用游戏过程中可用的信息。

这个简单的例子说明了强化学习方法的一些关键特性。首先，强调在与环境互动的同时学习，在这种情况下是与对手玩家互动。其次，有一个明确的目标，考虑到选择的延迟效应，正确的行为需要规划或远见。例如，简单的强化学习玩家将学会为短见的对手设置多步移动陷阱。强化学习解决方案的一个显著特点是不需要使用对手的模型，也不需要对可能的未来状态和动作序列进行显式搜索，就可以实现规划和前瞻的效果。

虽然这个例子说明了强化学习的一些关键特性，但它非常简单，可能会给人一种强化学习比实际情况更有限的印象。虽然井字棋是一种双人游戏，但强化学习也适用于没有外部对手的情况，即“与自然对抗的游戏”。强化学习也不局限于行为被分成不同回合的问题，就像井字游戏那样，只有在每一回合结束时才能获得奖励。当行为无限期地持续和任何时候都能收到各种数量的奖励时，它也同样适用。强化学习也适用于那些甚至不能分解成离散时间步长的问题，比如井字游戏。一般原理也适用于连续时间问题，尽管理论变得更加复杂，我们在这个介绍性的讨论中省略了它。

井字游戏具有相对较小的有限状态集，然而强化学习可以在状态集非常大甚至无限时使用。例如，Gerry Tesauro (1992, 1995) 将上述算法与人工神经网络相结合，学习下具有大约 1020 个状态的双陆棋。拥有如此众多的状态，不可能只经历其中的一小部分。Tesauro 的程序学会了比以前的任何程序玩得更好，最终也比世界上最好的人类玩家打得更好（第 16.1 节）。人工神经网络为程序提供了从其经验中进行泛化的能力，因此在新的状态中，它根据其网络所确定的从过去面临的相似状态中保存的信息来选择移动。强化学习系统在具有如此大的状态集的问题中能够工作得多好，与它从过去的经验中进行概括的恰当程度密切相关。正是在这一角色中，我们最需要有强化学习的监督学习方法。人工神经网络和深度学习（第 9.6 节）不是唯一的，也不一定是最好的方法。

在这个井字游戏的例子中，学习从没有超出游戏规则的先验知识开始，但强化学习绝不意味着对学习和智力的全盘看法。相反，先验信息可以以各种方式结合到强化学习中，这些方式对于特定的学习至关重要（例如参见第 9.5、17.4 和 13.1 节）。在井字游戏的例子中，我们也可以进入真实状态，而强化学习也可以应用于部分状态被隐藏的情况，或者不同的状态对学习者来说是相同的情况。

最后，井字游戏玩家能够预见，并知道它的每一步可能会导致的状态。要做到这一点，它必须有一个游戏模型，让它能够预见环境将如何变化，以应对它可能永远不会采取的移动。许多问题都是这样的，但在其他问题中，甚至缺乏动作效果的短期模型。强化学习可以在任何一种情况下应用。模型不是必需的，但是如果模型是可用的或可以学习，那么可以很容易地使用它们（第 8 章）。

另一方面，有一些强化学习方法根本不需要任何环境模型。免模型系统甚至无法考虑它们的环境将如何改变以响应单个动作。在这个意义上，井字游戏玩家对于它的对手来说是免模型的：它没有任何类型的对手模型。因为模型必须相当精确才能有用，所以当解决问题的真正瓶颈是建立一个精确的环境模型时，免模型方法比更复杂的方法更有优势。免模型方法也是基于模型方法的重要构件。在这本书里，我们在讨论如何将免模型方法作为更复杂的基于模型的方法的组成部分之前，专门用了几章来讨论免模型方法。

在一个系统中，强化学习既可以在高级别上使用，也可以在低级别上使用。尽管井字游戏玩家只学到了游戏的基本动作，但没有什么能阻止强化学习在更高的水平上发挥作用，在那里，每个“动作”本身都可能是一种可能精心设计的解决问题的方法的应用。在层级学习系统中，强化学习可以在多个层次上同时工作。

 **练习 1.1** : *Self-Play* 假设上面描述的强化学习算法不是与随机对手对战，而是与自身对战，双方都在学习。你认为在这种情况下会发生什么？它会学习一个不同的策略来选择

动作吗? □

- ✍ **练习 1.2 :** *Symmetries* 许多井字游戏的位置看起来不同, 但由于对称性实际上是一样的。我们如何修改上述学习过程来利用这一点? 这种变化会在哪些方面改善学习过程? 现在再想想。假设对手没有利用对称性。那样的话, 我们应该吗? 那么, 对称相等的位置必然具有相同的值, 这是真的吗? □
- ✍ **练习 1.3 :** *Greedy Play* 假设强化学习玩家是贪婪的, 也就是说, 他总是做出让他达到最佳位置的移动。它会比不贪婪的玩家学得更好或更差吗? 可能会发生什么问题? □
- ✍ **练习 1.4 :** *Learning from Exploration* 假设学习更新发生在所有移动之后, 包括探索移动。如果随时间逐步减小步长参数 (而不是探索的趋势), 则状态值将收敛到一组不同的概率。当我们从或者不从探索性动作中学习时对应的两组概率是什么 (概念上)? 假设我们确实在继续进行探索移动, 那么哪一组概率可能更好学习? 哪个会带来更多胜利? □
- ✍ **练习 1.5 :** *Other Improvements* 你还能想出其他方法来提高强化学习玩家吗? 你能想出更好的办法来解决所提出的井字游戏问题吗? □

1.6 总结

强化学习是一种理解和自动化目标导向学习和决策的计算方法。它与其他计算方法的不同之处在于, 它强调 agent 通过与其环境的直接交互进行学习, 而不需要示例性监督或完整的环境模型。在我们看来, 强化学习是第一个认真解决在与环境交互中学习以实现长期目标时出现的计算问题的领域。

强化学习使用马尔可夫决策过程的形式框架来定义学习 agent 与其环境之间的状态、动作和奖励方面的交互。该框架旨在成为一种简单的方式来表示人工智能问题的基本特征。这些特征包括因果, 不确定性和非决定性, 以及明确目标的存在。

值和值函数的概念是我们在本书中考虑的大多数强化学习方法的关键。我们的立场是, 值函数对于在策略空间进行有效搜索非常重要。值函数的使用将强化学习方法与进化方法区分开来, 进化方法在整个策略评估的指导下直接在策略空间进行搜索。

1.7 强化学习的早期历史

强化学习的早期历史有两条主线, 既漫长又丰富, 在与现代强化学习交织在一起之前, 这两条主线都是独立追求的。一条主线关注的是在试错中学习, 它起源于动物学习的心理学。这条主线贯穿了人工智能的一些最早的工作, 并导致了强化学习在 20 世纪 80 年代初的复兴。第二条主线关注的是最优控制问题及其利用值函数和动态规划的解决方案。在大多数情况下, 这条主线并不涉及学习。这两条主线大部分是独立的, 但在某种程度上, 围绕着第三条不太明显的主线变得相互关联, 这第三条主线关注的是时间差分方法, 比如本章中井字游戏的例子。这三条主线在 20 世纪 80 年代末汇集在一起, 产生了我们在本书中介绍的现代强化学习领域。

关注于试错学习的主线是我们最熟悉的, 也是我们在这段短暂的历史中最有发言权的。然而, 在此之前, 我们先简要讨论一下最优控制主线。



术语“最优控制”在 20 世纪 50 年代末开始使用，用来描述设计控制器以最小化或最大化动态系统随时间的行为量度的问题。解决这一问题的方法之一是由理查德·贝尔曼 (Richard Bellman) 等人在 20 世纪 50 年代中期通过推广 19 世纪的哈密顿 (Hamilton) 和雅各比 (Jacobi) 理论而发展起来的。这种方法使用动力系统状态和值函数或“最优回报函数”的概念来定义函数方程，现在通常称为贝尔曼方程。通过求解这个方程来解决最优控制问题的一类方法被称为动态规划 (Bellman, 1957a)。贝尔曼 (1957b) 还引入了最优控制问题的离散随机版本，称为马尔可夫决策过程 (MDPs)。Ronald Howard (1960) 提出了 MDP 的策略迭代方法。所有这些都是现代强化学习理论和算法的基本要素。

动态规划被广泛认为是解决一般随机最优控制问题唯一可行的方法。它受到贝尔曼所谓的“维数灾难”的困扰，这意味着它的计算需求随着状态变量的数量呈指数增长，但它仍然比任何其他通用方法更精确、更广泛地适用。自 20 世纪 50 年代后期以来，动态规划得到了广泛的发展，包括对部分可观察的 MDPs 的扩展 (Lovejoy, 1991 年调查)、许多应用 (White, 1985 年, 1988 年, 1993 年调查)、近似方法 (Rust, 1996 年调查)) 和异步方法 (Bertsekas, 1982, 1983)。有许多关于动态规划的出色的现代方法可供使用 (例如 Bertsekas, 2005, 2012; Puterman, 1994; Ross, 1983; 和 Whittle, 1982, 1983)。Bryson (1996) 提供了最优控制的权威历史。

一方面是最优控制和动态规划，另一方面是学习，这两者之间的联系很难识别。我们不能确定是什么导致了这种分离，但它的主要原因可能是所涉及的学科和它们不同的目标之间的分离。另一个贡献可能是流行的观点，即动态规划是一种离线计算，本质上依赖于精确的系统模型和贝尔曼方程的解析解。此外，动态规划最简单的形式是一种在时间上向后进行的计算，这使得很难理解它如何参与必须向前进行的学习过程。动态规划的一些最早的工作，如贝尔曼和 Dreyfus (1959) 的工作，现在可以归类为遵循学习的方法。Witten (1977) 的工作 (下面讨论) 无疑是学习和动态编程思想的结合。Werbos (1987) 明确地论证了动态规划和学习方法之间更大的相互关系，以及动态规划与理解神经和认知机制的相关性。对我们来说，直到 1989 年 Chris Watkins 的工作才将动态规划方法与在线学习完全结合起来，他使用 MDP 形式对强化学习的处理已经被广泛采用。从那时起，这些关系被许多研究人员广泛发展，特别是 Dimitri Bertsekas 和 John Tsitsiklis (1996)，他们创造了术语“神经动态规划”来指动态规划和人工神经网络的结合。目前使用的另一个术语是“近似动态规划”，这些不同的方法强调了该学科的不同方面，但它们都与强化学习有共同的兴趣，即避开动态规划的经典缺点。

我们认为最优控制的所有工作在某种意义上也是强化学习的工作。我们将强化学习方法定义为解决强化学习问题的任何一种有效的方法，现在可以清楚地看到，这些问题与最优控制问题密切相关，特别是随机最优控制问题，如 MDP 形式的随机最优控制问题。因此，我们必须考虑最优控制的求解方法，如动态规划，也是强化学习方法。因为几乎所有的传统方法都需要完整的系统知识才能被控制，所以说它们是强化学的一部分有点不自然。另一方面，许多动态规划算法是增量和迭代的。就像学习方法一样，它们通过逐次逼近逐渐得到正确的答案。正如我们在这本书的其余部分所展示的那样，这些相似之处远远不是表面上的。完全知识和不完全知识的理论和解决方法是如此紧密地

联系在一起，以至于我们觉得必须把它们作为同一主题的一部分来考虑。

现在让我们回到通向现代强化学习领域的另一条主线，这条主线以试错学习的思想为中心。我们在这里只涉及主要的接触点，在第 14.3 节中会更详细地讨论这个主题。根据美国心理学家 R.S.Woodworth (1938) 的说法，试错学习的概念可以追溯到 19 世纪 50 年代 Alexander Bain 关于通过“摸索和实验”来学习的讨论，更明确地说，是英国行为学家和心理学家 Conway Lloyd Morgan 在 1894 年用这个术语来描述他对动物行为的观察。也许第一个简明扼要地将试错学习的本质表达为学习原则的人是 Edward Thorndike：

在对同一情况做出的几种反应中，伴随或紧随其后的是对动物满意的反应，在其他条件相同的情况下，它们与这种情况的联系更紧密，因此，当它再次出现时，它们更有可能再次出现；那些伴随或紧随其后的是动物的不适，在其他条件相同的情况下，它们与这种情况的联系将被削弱，因此，当它再次发生时，它们发生的可能性就更小。满足感或不适感越大，纽带越强或越弱。（Thorndike, 1911 年，第 244 页）

Thorndike 称之为“效果定律”，因为它描述了强化事件对选择行为的趋势的影响。Thorndike 后来修改了定律，以更好地解释动物学习的后续数据（如奖励和惩罚的效果之间的差异），定律的各种形式在学习理论家中产生了相当大的争议（例如，参见 Gallistel, 2005; Herrnstein, 1970; Kimble, 1961, 1967; Mazur, 1994 年）。尽管如此，效果法则——以一种或另一种形式——被广泛认为是许多行为背后的基本原则（例如，Hilgard and Bower, 1975; Dennett, 1978; Campbell, 1960; Cziko, 1995）。它是 Clark Hull (1943, 1952) 有影响的学习理论和 B. F. Skinner (1938) 有影响的实验方法的基础。

在动物学习的上下文中，术语“强化”是在 Thorndike 表达效果定律之后就开始很好地使用，据我们所知，它最早出现在巴甫洛夫条件反射专著的 1927 年英译本中。巴甫洛夫将强化描述为一种行为模式的增强，这是由于动物接受了刺激——一种强化——与另一种刺激或反应有适当的时间关系。一些心理学家将强化的概念扩展到包括行为的削弱和增强，并将强化的概念扩展到可能包括省略或终止刺激。要被认为是强化，增强或削弱必须在强化退出后持续存在；仅仅吸引动物注意或激励其行为而不产生持久变化的刺激将不被认为是强化。

在计算机中实施试错学习的想法出现在关于人工智能可能性的最早想法中。在 1948 年的一份报告中，阿兰·图灵描述了一种“愉悦-痛苦系统”的设计，它遵循效果定律：

当到达动作未确定的配置时，将对丢失的数据进行随机选择，并在描述中临时应用适当的条目。当疼痛刺激发生时，所有暂定条目都会被取消，当快乐刺激发生时，它们都会被永久化。（图灵，1948 年）

许多巧妙的机电机器被建造出来验证了试错学习。最早的可能是 Thomas Ross (1933) 建造的一台机器，它能够通过一个简单的迷宫找到自己的路，并记住通过开关设置的路径。1951 年，W.Grey Walter 建造了能够进行简单学习的“机械乌龟”(Walter, 1950) 的一个版本。1952 年，克劳德·香农演示了一只名叫 Teseus 的迷宫老鼠通过试错找到迷宫中的

路，迷宫本身通过地板下的磁铁和继电器记住成功的方向（另见香农，1951）。J.A.Deutsch（1954）基于他的行为理论（Deutsch, 1953）描述了一种迷宫求解机器，它与基于模型的强化学习的一些共同性质（第8章）。在他的博士学位论文中，Marvin Minsky（1954）讨论了强化学习的计算模型，并描述了他构建的模拟机器，该机器由他称为 SNARC（Stochastic Neural-Analog Reinforcement Calculators，随机神经模拟强化计算器）的组件组成，旨在模拟类似于大脑中可修改的突触连接（第15章）。网站cyberneticzoo.com包含了大量关于这些和许多其他机电学习机器的信息。

建造机电学习机器让位于对数字计算机编程来完成各种类型的学习，其中一些实现了试错学习。Farley 和 Clark（1954）描述了一个通过试错学习的神经网络学习机的数字模拟。但是他们的兴趣很快就从试错学习转移到泛化和模式识别，也就是说，从强化学习转移到监督学习（Clark 和 Farley, 1955）。这开始了一种关于这些学习类型之间关系的混乱模式。许多研究人员似乎认为他们在研究强化学习，但实际上是在研究监督学习。例如，人工神经网络的先驱，如 Rosenblatt（1962）和 Widrow and Hoff（1960），显然受到强化学习的激励——他们使用奖励和惩罚的语言——但是他们研究的系统是适用于模式识别和知觉学习的监督学习系统。即使在今天，一些研究者和教科书也淡化或模糊了这些学习类型之间的区别。例如，一些人工神经网络教科书使用“试错”一词来描述从训练样本中学习的网络。这是一种可以理解的混淆，因为这些网络使用错误信息来更新连接权重，但这忽略了试错学习的本质特征，即基于评估性反馈选择动作，而不依赖于正确动作的知识。

部分由于这些困惑，在20世纪60年代和70年代，对真正的试错学习的研究变得稀少，尽管也有明显的例外。在20世纪60年代，“强化”和“强化学习”这两个术语第一次被用在工程文献中来描述试错学习的工程应用（例如，Waltz 和 Fu, 1965；Mendel, 1966；Fu, 1970；Mendel 和 McLaren, 1970）。特别有影响的是 Minsky 的论文《迈向人工智能的步骤》(Minsky, 1961)，它讨论了几个与试错学习相关的问题，包括预测、期望，以及他所说的复杂强化学习系统的基本信用分配问题：如何在可能涉及到成功的许多决策中分配成功的信用？在某种意义上，我们在这本书中讨论的所有方法都是为了解决这个问题。Minsky 的论文今天很值得阅读。

在接下来的几个段落中，我们讨论其他一些例外情况和部分例外情况，这些例外情况相对忽视了20世纪60年代和70年代对真正的试错学习的计算和理论研究。

一个例外是新西兰研究人员 John Andreae 的工作，他开发了一种名为 STELLA 的系统，该系统在与环境的互动中通过试错来学习。这个系统包括一个世界的内部模型，以及后来处理隐藏状态问题的“内部独白”(Andreae, 1963, 1969a, b)。Andreae 后来的工作（1977）更多地强调向老师学习，但仍然包括在试错中学习，产生新事件是该系统的目标之一。这项工作的一个特点是“回漏过程”，在 Andreae（1998）中进行了更详细的阐述，它实现了一种类似于我们描述的备份更新操作的信用分配机制。不幸的是，他的开创性研究并不广为人知，也没有对后续的强化学习研究产生重大影响。最近的总结可用（Andreae, 2017a, b）。

更有影响力的是 Donald Michie 的工作。在 1961 年和 1963 年，他描述了一个简单的

试错学习系统，用于学习如何玩井字游戏（或 nought 和 cross），称为 MENACE (for Matchbox Educable Naughts and Crosses Engine)。它由每个可能的游戏位置的火柴盒组成，每个火柴盒包含许多彩色珠子，从那个位置的每个可能的移动都有不同的颜色。通过从对应于当前游戏位置的火柴盒中随机抽取一个珠子，可以确定 MENACE 的移动。当游戏结束时，珠子会被添加到游戏中使用的盒子中或从盒子中移除，以奖励或惩罚 MENACE 的决定。Michie 和 Chambers (1968) 描述了另一种井字游戏强化学习器，称为 GLEE (Game Learning Expectimax Engine, 游戏学习预期引擎) 和强化学习控制器，称为 BOXES。他们将 BOXES 应用于学习平衡铰接在可移动手推车上的杆子的任务，这是基于只有当杆子掉落或手推车到达轨道末端时才会出现的失败信号。这项任务是根据 Widrow and Smith (1964) 的早期工作改编的，他使用监督学习方法，并假设已经能够平衡杆子的老师进行了指导。Michie 和 Chambers 的杆子平衡版本是在知识不完全的情况下强化学习任务的最好的早期例子之一。它影响了后来的强化学习工作，从我们自己的一些研究开始 (Barto, Sutton 和 Anderson, 1983; Sutton, 1984)。Michie 一直强调试错以及学习是人工智能必不可少的方面 (Michie, 1974)。

Widrow, Gupta 和 Maitra (1973) 修改了 Widrow 和 Hoff (1960) 的最小均方 (LMS) 算法，以生成强化学习规则，该规则可以从成功和失败信号中学习，而不必从训练样本中学习。他们称这种学习形式为“选择性自举适应”，并将其描述为“与批评家一起学习”，而不是“与老师一起学习”。他们分析了这个规则，并展示了如何学习 21 点。这是 Widrow 对强化学习的一次孤立尝试，他对监督学习的贡献更具影响力。我们对“评论家”一词的使用源自 Widrow, Gupta 和 Maitra 的论文。Buchanan, Mitchell, Smith 和 Johnson (1978) 在机器学习的上下文中独立使用了“批评家”一词（另请参见 Dietterich 和 Buchanan, 1984 年），但是对于他们来说，批评家是一个专家系统，能够做的不仅仅是评估性能。

学习自动机的研究对导致现代强化学习研究的试错主线有更直接的影响。这些方法用于解决非关联的纯选择性学习问题，称为 k 臂赌博机，类似于老虎机或除了使用 k 杠杆外的“单臂赌博机”（参见第 2 章）。学习自动机是简单的，低内存的机器，可以提高在这些问题中获得奖励的概率。学习自动机起源于 20 世纪 60 年代俄国数学家和物理学家 M.L.Tsetlin 及其同事（于 1973 年在 Tsetlin 去世后出版），此后在工程领域得到了广泛的发展（参见 Narendra 和 Thathachar, 1974 年, 1989 年）。这些发展包括对随机学习自动机的研究，这是一种基于奖励信号更新动作概率的方法。尽管不是按照随机学习自动机的传统进行发展，但 Harth 和 Tzanakou (1974) 的 Alopex 算法 (for Algorithm of pattern extraction, 用于模式提取算法) 是一种随机方法，用于检测动作和强化之间的相关性，这影响了我们的一些早期研究 (Barto, Sutton 和 Brouwer, 1981)。随机学习自动机被心理学的早期工作所预示，首先是 William Estes (1950) 致力于学习的统计理论，后来又被其他人进一步发展（例如，Bush 和 Mosteller, 1955; Sternberg, 1963）。

在心理学领域发展起来的统计学习理论被经济学研究人员所采用，从而导致了该领域致力于强化学习的研究主线。这项工作始于 1973 年，当时将 Bush 和 Mosteller 的学习理论应用于一系列经典经济模型中 (Cross, 1973 年)。这项研究的一个目标是研究比传统理想化的经济 agent 更像真实人的人工 agent (Arthur, 1991)。这种方法扩展到博弈

论背景下的强化学习研究。经济学中的强化学习在很大程度上独立于人工智能中的强化学习的早期工作，尽管博弈论仍然是这两个领域都感兴趣的话题（超出了本书的范围）。Camerer (2011) 讨论了经济学中的强化学习传统，而 Nowe, Vrancx 和 De Hauwere (2012) 从多 agent 扩展的角度对本书中介绍的方法进行了概述。博弈论中的强化与用于玩井字游戏，跳棋和其他休闲游戏的强化学习的主题大不相同。有关强化学习和游戏这方面的概述，请参见，例如，Szita (2012)。

John Holland (1975) 概述了基于选择原理的自适应系统的一般理论。他的早期工作主要涉及非关联形式的试错，例如进化方法和 k 臂赌博机。1976 年，更全面的是 1986 年，他引入了分类器系统，这是真正的强化学习系统，包括关联和值函数。Holland 分类器系统的一个关键组成部分是用于信用分配的“bucket-brigade”算法，该算法与我们的井字游戏示例中使用的时间差分算法紧密相关，并在第 6 章中进行了讨论。另一个关键组成部分是遗传算法，一种进化方法，其作用是发展有用的表示形式。分类器系统已被许多研究人员广泛发展，形成了强化学习研究的主要分支 (Urbanowicz 和 Moore, 2009 年进行了综述)，但是遗传算法，我们不认为它们本身就是强化学习系统，受到了更多关注，就像其他用于进化计算的方法一样（例如，Fogel, Owens 和 Walsh, 1966; Koza, 1992）。

在人工智能中，最有责任将试错的主线重新引向强化学习的人是 Harry Klopf (1972, 1975, 1982)。Klopf 认识到，随着学习研究者几乎完全专注于监督学习，适应性行为的基本方面正在消失。Klopf 认为，缺少的是行为的快乐方面，从环境中获得某种结果的动力，控制环境达到期望的目标而远离不期望的目标(见第 15.9 节)。这是反复试验学习的基本思想。Klopf 的思想对作者们尤其有影响，因为我们对它们的评估 (Barto and Sutton, 1981a) 使我们对监督学习和强化学习之间的区别有所了解，并最终使我们着重于强化学习。我们和同事完成的许多早期工作都是为了表明强化学习和监督学习确实是不同的 (Barto, Sutton 和 Brouwer, 1981; Barto 和 Sutton, 1981b; Barto 和 Anandan, 1985)。其他研究表明强化学习如何解决人工神经网络学习中的重要问题，特别是它如何产生多层网络的学习算法 (Barto, Anderson 和 Sutton, 1982; Barto 和 Anderson, 1985; Barto, 1985, 1986; Barto, 1985, 1986)。Barto 和 Jordan, 1987 年；请参阅第 15.10 节)。

我们现在转到强化学习历史的第三条主线，那就是关于时间差分学习。时间差分学习方法的独特之处在于受同一量的时间连续估计值之间的差异所驱动，例如在井字游戏示例中获胜的概率。这条主线比其他两条主线更小、更不明显，但它在该领域发挥了特别重要的作用，部分原因是时间差分方法似乎是强化学习的新方法和独特的方法。

时间差分学习的起源部分源于动物学习心理学，尤其是在辅助补强的概念中。次要强化是一种与主要强化（例如食物或疼痛）配对的刺激物，因此具有类似的强化特性。Minsky (1954) 可能是第一个意识到这种心理学原理对人工学习系统很重要的人。Arthur Samuel (1959) 是第一个提出并实现一种学习方法的人，该方法包括时间差分思想，这是他著名的跳棋比赛程序的一部分（第 16.2 节）。

Samuel 没有提及 Minsky 的工作，也没有提及与动物学习的可能联系。他的灵感显然来自克劳德 · 香农 (1950 年) 的建议，即可以对一台计算机进行编程，使其使用评估函数下棋，并且可以通过在线修改此函数来改善其游戏性。（香农的这些想法可能也影响

了贝尔曼，但我们对此没有任何证据)。Minsky (1961) 在他的“步骤”论文中广泛讨论了 Samuel 的工作，提出了与自然和人为的次要强化理论的联系。

正如我们已经讨论过的，在 Minsky 和 Samuel 工作之后的十年中，关于试错学习的计算工作很少，并且显然没有针对时间差分学习进行任何计算工作。1972 年，Klopf 将试错学习与时间差分学习的重要组成部分结合在一起。Klopf 感兴趣的是可以扩展到大型系统中学习的原理，因此对局部强化的概念很感兴趣，因此整个学习系统的各个子组件可以相互强化。他提出了“广义强化”的概念，其中每个组成部分（名义上是每个神经元）都以强化术语看待其所有输入：兴奋性输入为奖励，抑制性输入为惩罚。这与我们现在称为时间差分学习的想法不同，回想起来，它比 Samuel 的研究还远。另一方面，Klopf 将这一想法与试错学习联系起来，并将其与庞大的动物学习心理学经验数据库联系起来。

Sutton (1978a, b, c) 进一步发展了 Klopf 的思想，特别是与动物学习理论的联系，描述了由时间连续预测的变化驱动的学习规则。他和 Barto 完善了这些观点，并建立了基于时间差分学习的经典条件作用的心理学模型 (Sutton 和 Barto, 1981a; Barto 和 Sutton, 1982)。随后是基于时间差分学习的经典条件作用的其他几种有影响力的心理学模型（例如 Klopf, 1988; Moore 等, 1986; Sutton 和 Barto, 1987, 1990)。此时发展的一些神经科学模型在时间差分学习方面得到了很好的解释 (Hawkins 和 Kandel, 1984; Byrne, Gingrich 和 Baxter, 1990; Gelperin, Hopfield 和 Tank, 1985; Tesauro, 1986; Friston 等, 1994 年)，尽管在大多数情况下没有历史联系。

我们关于时间差分学习的早期工作受到动物学习理论和 Klopf 工作的强烈影响。与 Minsky 的“步骤”论文和 Samuel 的跳棋玩家之间的关系直到后来才被认可。但是，到 1981 年，我们已经完全意识到上述所有先前的工作都是时间差分和试错主线的一部分。目前，我们开发了一种将时间差分学习与试错学习相结合的方法，称为 actor-critic 体系结构，并将该方法应用于 Michie 和 Chambers 的杠杆平衡问题 (Barto, Sutton 和 Anderson, 1983)。这种方法在 Sutton (1984) 博士学位论文中得到了广泛的研究，并扩展到在 Anderson (1986) 博士学位论文中使用反向传播神经网络。大约在这个时候，Holland (1986) 以他的 bucket-brigade 算法的形式，将时间差分思想明确地纳入了他的分类器系统。Sutton (1988) 采取了关键步骤，将时间差分学习与控制分离开来，将其作为通用的预测方法。该论文还介绍了 TD(λ) 算法并证明了其收敛性。

当我们在 1981 年完成关于 actor-critic 体系结构的工作时，我们发现了 Ian Witten (1977, 1976a) 的一篇论文，该论文似乎是时间差分学习规则的最早出版物。他提出了我们现在称为表格 TD(0) 的方法，该方法用作解决 MDP 的自适应控制器的一部分。这项工作于 1974 年首次提交给期刊出版，也出现在 Witten 的 1976 年博士学位论文中。Witten 的工作是 Andreae 早期使用 STeLLA 和其他试错学习系统进行实验的后继型工作。因此，Witten 于 1977 年发表的论文涵盖了强化学习研究的两个主要主线，即试错学习和最优控制，同时为时间差分学习做出了明显的早期贡献。

时间差分和最优控制主线在 1989 年与 Chris Watkins 的 Q-learning 发展完全结合在一起。这项工作扩展并整合了强化学习研究的所有三个主线的先前工作。Paul Werbos (1987) 通过争论试错学习和动态规划的收敛性从 1977 年开始为这种集成做出了贡献。

Watkins 工作之时，强化学习研究已经有了巨大的发展，主要是在人工智能的机器学习子领域，也包括人工神经网络和更广泛的人工智能。1992 年，Gerry Tesauro 的双陆棋游戏程序 TD-Gammon 取得了巨大的成功，给这个领域带来了更多的关注。

自本书第一版出版以来，神经科学的一个蓬勃发展的子领域开始关注强化学习算法与神经系统强化学习之间的关系。造成这种情况的最主要原因是时间差分算法的行为与大脑中产生多巴胺的神经元活动之间的不可思议的相似性，正如许多研究人员所指出的那样 (Friston 等, 1994; Barto, 1995a; Houk, Adams, 和 Barto, 1995 年; Montague, Dayan 和 Sejnowski, 1996 年; Schultz, Dayan 和 Montague, 1997 年)。第 15 章介绍了强化学习的这一令人兴奋的方面。在强化学习的近代历史中，其他重要的贡献太多，在这个简短的叙述中无法提及；我们在它们出现的各个章节的末尾引用了其中的许多内容。

1.8 书目评论

有关强化学习的其他一般知识，请读者参考 Szepesvari (2010), Bertsekas 和 Tsitsiklis (1996), Kaelbling (1993a) 以及 Sugiyama, Hachiya 和 Morimura (2013) 的书。从控制或运筹学的角度来看的书籍包括 Si, Barto, Powell 和 Wunsch (2004), Powell (2011), Lewis 和 Liu (2012)，和 Bertsekas (2012)。Cao (2009) 的综述将强化学习置于随机动态系统的学习和优化的其他方法的上下文中。《机器学习》期刊上的三期特刊侧重于强化学习：Sutton (1992a), Kaelbling (1996) 和 Singh (2002)。Barto (1995b) 提供了有用的综述；Kaelbling, Littman 和 Moore (1996)；以及 Keerthi 和 Ravindran (1997)。Weiring 和 van Otterlo (2012) 编辑的卷很好地概述了最近的发展。

1.2 本章中 Phil 早餐的例子是受 Agre (1988) 启发的。

1.5 井字游戏示例中使用的时间差分方法在第 6 章进行了阐述。

第一部分

表格解决方法

在本书的这一部分中，我们以最简单的形式描述了强化学习算法的几乎所有核心思想：其中状态和动作空间足够小，可以将近似值函数表示为数组或表格。在这种情况下，这些方法通常可以找到精确的解决方案，也就是说，它们通常可以精确地找到最优值函数和最优策略。这与本书下一部分中描述的近似方法形成对比，后者只能找到近似解，但反过来可以有效地应用于更大的问题。

本书这一部分的第一章描述了强化学习问题的一种特殊情况的解决方法，在这种情况下，只有一种状态，称为赌博机问题。第二章描述了贯穿本书其余部分的一般问题表述——有限马尔可夫决策过程——以及它的主要思想，包括贝尔曼方程和值函数。

接下来的三章描述了用于解决有限马尔可夫决策问题的方法的三个基本类别：动态规划，蒙特卡洛方法和时间差分学习。每类方法都有其优点和缺点。动态规划方法在数学上已经很好地开发，但是需要完整且准确的环境模型。蒙特卡洛方法不需要模型，概念上也很简单，但并不适合逐步进行增量计算。最后，时间差分方法不需要模型，并且是完全增量的，但分析起来更复杂。这些方法在效率和收敛速度方面也有一些不同。

剩下的两章描述了如何将这三类方法结合起来，以获得每种方法的最佳特性。在其一章中，我们描述了如何通过多步自举方法将蒙特卡罗方法的优点与时间差分方法的优点结合起来。在本书这部分的最后一章中，我们将展示时间差分学习方法如何与模型学习和规划方法（如动态规划）相结合，以完整和统一地解决表格强化学习问题。

第二章 多臂赌博机

强化学习区别于其他类型的学习的最重要特征是，它使用的训练信息是评价所采取的动作，而不是通过给出正确的动作来进行指导。这就产生了主动探索和明确搜索良好行为的需要。纯粹的评价性反馈表明所采取的动作有多好，但并不能说明它是可能的最好或最差的动作。另一方面，纯粹的指导性反馈指出了要采取的正确动作，而与实际采取的动作无关。这种反馈是监督学习的基础，包括模式分类、人工神经网络和系统识别的大部分内容。以它们的纯粹形式，这两种反馈是截然不同的：评价性反馈完全取决于所采取的动作，而指导性反馈则与所采取的动作无关。

在本章中，我们将在一种简化的设置下研究强化学习的评价方面，这一方面不涉及学习在一种以上情况下的行动。这种非关联的设置是其中大多数涉及评价性反馈的先前工作都已完成的设置，它避免了全部强化学习问题的大部分复杂性。通过研究这种情况，我们可以最清楚地看到评价性反馈与指导性反馈有何不同，并且可以与之结合。

我们探讨的特定的非关联性、评价性反馈问题是 k 臂赌博机问题的一个简单版本。我们使用这个问题来介绍许多基本的学习方法，这些方法将在后面的章节中扩展以应用于全部的强化学习问题。在本章的最后，我们向全部的强化学习问题迈进一步，通过讨论当赌博机问题变得关联性时，即在不止一种情况下采取行动时会发生的情况。

2.1 k 臂赌博机问题

考虑以下学习问题。您反复面临着 k 个不同选项或动作中的一个选择。在每一次选择之后，您会得到一个从平稳概率分布中选出的数值奖励，该奖励取决于您选择的动作。您的目标是在某个时间段内最大化期望的总奖励，例如，超过 1000 个动作选择或时间步。

这是 k 臂赌博机问题的原始形式，因此被类比为老虎机或“单臂赌博机”，除了它有 k 个杠杆而不是一个。每个动作的选择都像是玩老虎机的一个杠杆的游戏，而奖励就是命中大奖的回报。通过反复的动作选择，您可以将动作集中在最佳杠杆上，从而最大程度地赢取奖金。另一个类比是医生为一系列重病患者选择实验治疗。每个动作都是治疗的选择，每个奖励都是患者的幸存或健康。如今，有时会使用“赌博机问题”术语来概括上述问题，但在本书中，我们仅将其用于指代这种简单案例。

在我们的 k 臂赌博机问题中，只要选择了该动作， k 个动作中的每个动作都有期望或平均的奖励；让我们称之为动作的价值。我们将在时间步 t 上选择的动作表示为 A_t ，并将相应的奖励表示为 R_t 。假定选择了 a ，则任意动作 a 的值 $q_*(a)$ 是期望的奖励：

$$q_*(a) \doteq \mathbb{E}[R_t | A_t = a].$$

如果您知道每个动作的价值，那么解决 k 臂赌博机问题将是微不足道的：您将始终选择

价值最高的动作。我们假设您不确定地知道动作值，尽管您可能有估计。我们将时间步 t 处动作 a 的估计值表示为 $Q_t(a)$ 。我们希望 $Q_t(a)$ 接近 $q_*(a)$ 。

如果您维护动作值的估计，则在任何时间步上至少都有一个动作的估计值最大。我们称这些为贪婪的动作。当您选择这些动作之一时，我们说您正在利用对这些动作的值的现有知识。相反，如果您选择其中一种非贪婪动作，则表示您正在探索，因为这使您可以改善对非贪婪动作价值的估计。利用是在一步之内最大化期望回报的正确做法，但从长远来看，探索可能会产生更大的总奖励。例如，假设贪婪动作的价值是确定的，而其他几项动作的评估结果几乎相同，但存在很大的不确定性。这种不确定性使得这些其他动作中的至少一个实际上可能比贪婪的动作要好，但是您不知道哪个。如果您还有很多时间步可以选择动作，那么探索非贪婪的动作并发现其中哪个比贪婪的动作更好可能会更好。在短期内，探索过程中的奖励较低，但从长期来看，奖励较高，因为发现更好的动作后，您可以多次利用它们。由于不可能通过任何一个单一的动作选择来进行探索和利用，因此人们经常提到探索与利用之间的“冲突”。

在任何特定情况下，是进行探索还是利用更好，都将以复杂的方式取决于估计的精确值，不确定性以及剩余时间步的数量。对于 k 臂赌博机和相关问题的特殊数学公式，有许多复杂的方法可以平衡探索和利用。但是，大多数这些方法都对平稳性和先验知识做出了强有力的假设，这些假设在应用以及我们将在后续章节中考虑的全面强化学习问题中被违反或无法验证。当这些方法的理论假设不适用时，对这些方法的最优性或有界损失的保证就不太满意。

在本书中，我们不担心以复杂的方式平衡探索与利用；我们只担心如何平衡它们。在本章中，我们介绍了几种用于解决 k 臂赌博机问题的简单平衡方法，并表明它们比总是利用的方法好得多。平衡探索与利用的需要是强化学习中出现的一个独特挑战；我们的 k 臂赌博机问题的简单性使我们能够以特别清晰的形式展示这一点。

2.2 动作值方法

我们首先更仔细地研究用于估计动作值的方法，以及用于使用估计值来做出动作选择决策的方法，我们统称为动作值方法。回想一下，动作的真实值是选择该动作时的平均奖励。估计这一点的一种自然方法是对实际收到的奖励进行平均：

$$Q_t(a) \doteq \frac{t \text{ 之前采取 } a \text{ 时的奖励总和}}{t \text{ 之前采取 } a \text{ 的次数}} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{I}_{A_t=a}}{\sum_{i=1}^{t-1} \mathbb{I}_{A_t=a}}, \quad (2.1)$$

其中 $\mathbb{I}_{predicate}$ 表示随机变量，如果 $predicate$ 为真则为 1，否则为 0。如果分母为零，那么我们将 $Q_t(a)$ 定义为某个默认值，例如 0。随着分母趋于无穷大，根据大数定律， $Q_t(a)$ 收敛至 $q_*(a)$ 。我们称其为估计动作价值的样本平均方法，因为每个估计都是相关奖励样本的平均值。当然，这只是估计动作价值的一种方法，不一定是最佳方法。尽管如此，现在让我们继续使用这种简单的估计方法，然后转向如何将估计值用于选择动作的问题。

最简单的动作选择规则是选择估计值最高的动作之一，也就是上一节定义的贪婪动作之一。如果有多个贪婪动作，则以某种任意方式（可能是随机的）在其中进行选择。我



们将此贪婪动作选择方法写为

$$A_t \doteq \arg \max_a Q_t(a), \quad (2.2)$$

其中 $\arg \max_a$ 表示动作 a , 该动作让其后的表达式被最大化 (同样, 平局任意)。贪婪的动作选择总是利用当前的知识来最大化立即的奖励。它不花时间对所有明显劣等的动作进行抽样, 以查看它们是否会真的更好。一个简单的替代方法是在大多数时间贪婪地表现, 但每隔一段时间, 比如说小概率 ε , 而是从所有具有相等概率的动作中随机选择, 而与动作值估计无关。我们将这种接近贪婪的动作选择规则的方法称为 ε -greedy 方法。这些方法的一个优点是, 在极限中, 随着步数的增加, 每个动作将被采样无限次, 从而确保所有 $Q_t(a)$ 收敛到 $q_*(a)$ 。当然, 这意味着选择最优动作的概率收敛到大于 $1-\varepsilon$, 即接近确定性。然而, 这些只是渐近保证, 而对于这些方法的实际有效性却很少提到。

 **练习 2.1** 在 ε -greedy 动作选择中, 对于两个动作和 $\varepsilon = 0.5$ 的情况, 选择贪婪动作的概率是多少? □

2.3 10 臂试验

为了粗略地评估贪婪和 ε -greedy 动作值方法的相对有效性, 我们在一组测试问题上对它们进行了数值比较。这是一组 2000 个随机生成的 k 臂赌博机问题, $k = 10$ 。对于每个赌博机问题, 如图2.1所示, 动作值 $q_*(a)$, $a = 1, \dots, 10$ 根据均值为 0 和方差为 1 的正态 (高斯) 分布进行选择。然后, 当学习方法应用于在时间步 t 选择动作 A_t 的问题时, 从均值为 $q_*(A_t)$ 和方差为 1 的正态分布中选择实际奖励 R_t 。这些分布在图2.1中以灰色表示。我们称这套测试任务为 10 臂试验。对于任何一种学习方法, 我们都可以测量它的性能和行为, 因为它在应用于其中一个赌博机问题时, 随着 1000 多个时间步的经验而有所改善。这就构成了一次运行。重复进行 2000 次独立运行, 每次运行都有一个不同的赌博机问题, 我们得到了学习算法的平均行为的度量。

图2.2在 10 臂试验上比较了贪婪方法和上述两种贪婪方法 ($\varepsilon = 0.01$ 和 $\varepsilon = 0.1$)。所有这些方法均使用样本平均技术形成其动作值估算值。上方的图显示了预期奖励随着经验的增加。贪婪方法在开始时的改进速度比其他方法稍快, 但随后在较低的水平上趋于平稳。它获得的每一步奖励只有大约 1, 相比之下, 在这个试验上, 最好的可能是大约 1.55。从长远来看, 贪婪方法的表现要差得多, 因为它经常陷入执行次优动作的困境。下图显示了贪婪方法仅在大约三分之一的任务中找到了最优动作。在其他三分之二中, 其最佳动作的初始样本令人失望, 并且从未恢复。 ε -greedy 方法最终表现更好, 因为它们不断探索并提高了识别最优动作的机会。 $\varepsilon = 0.1$ 方法进行了更多探索, 通常会更早地找到最优动作, 但它从未选择该动作的时间超过了 91%。 $\varepsilon = 0.01$ 方法的改进速度较慢, 但在图中所示的两个性能指标上最终都比 $\varepsilon = 0.1$ 方法更好。还可以随着时间的推移而减小 ε , 以尝试获得高值和低值中的最佳值。

ε -greedy 方法胜于贪婪方法的优势取决于任务。例如, 假设奖励方差更大, 比如 10 而不是 1。在奖励更嘈杂的情况下, 需要更多的探索才能找到最优动作, ε -greedy 方法应



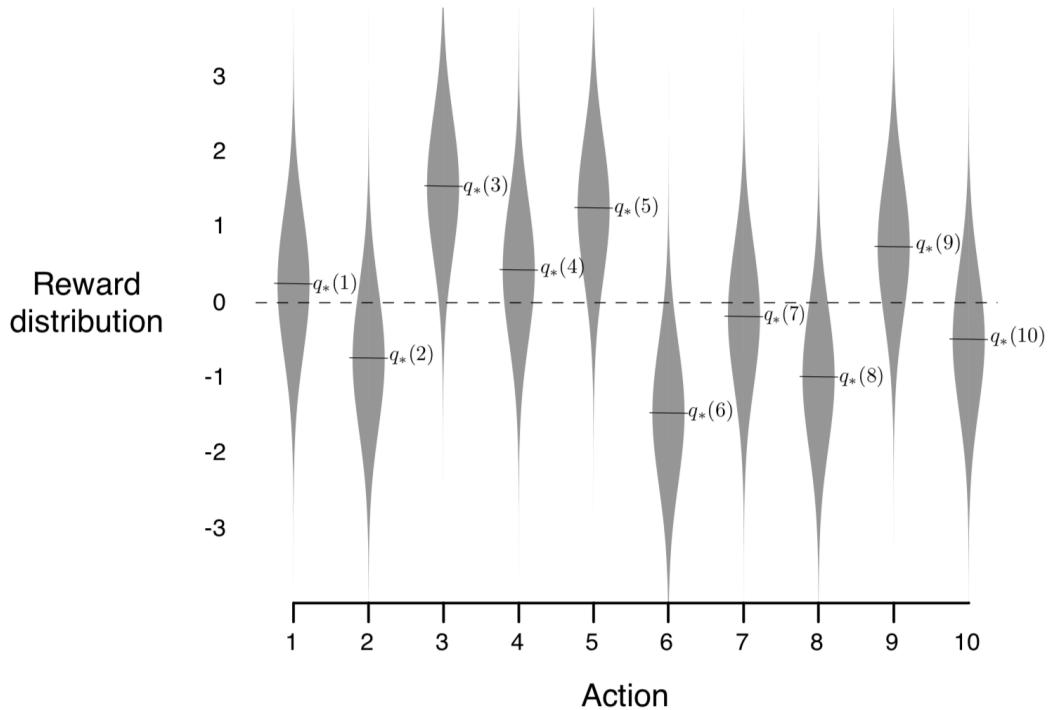


图 2.1: 10 臂试验中的赌博机问题示例。根据均值为零和单位方差的正态分布选择十个动作中每个动作的真实值 $q_*(a)$ ，然后根据这些灰色分布所表明的均值为 $q_*(a)$ 和单位方差的正态分布选择实际奖励。

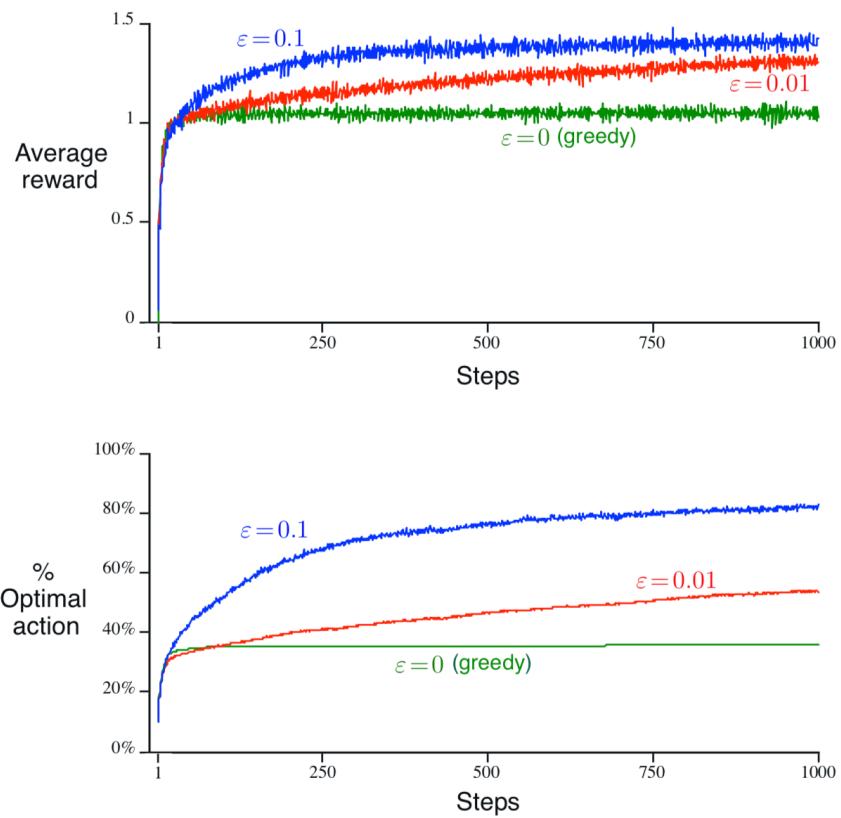


图 2.2: ε -greedy 动作值方法在 10 臂试验上的平均性能。这些数据是具有不同的赌博机问题的 2000 多次运行的平均值。所有方法都使用样本平均值作为它们的动作值估计。

该比贪婪方法更好。另一方面，如果奖励方差为零，那么贪婪的方法在尝试一次之后就会知道每个动作的真实价值。在这种情况下，贪婪的方法实际上可能执行得最好，因为它很快就会找到最优动作，然后再也不用去探索。但是，即使在确定性的情况下，如果我们弱化一些其他假设，探索也有很大的优势。例如，假设赌博机任务是非平稳的，也就是说，动作的真实值随着时间的推移而变化。在这种情况下，甚至在确定性的情况下也需要探索，以确保非贪婪的动作之一不会变得比贪婪的动作更好。正如我们将在接下来的几章中看到的，非平稳性是强化学习中最常见的情况。即使潜在的任务是平稳的和确定的，学习者也面临着一组赌博机般的决策任务，随着学习的进行和智能体的决策策略的变化，每个任务都会随着时间的推移而变化。强化学习需要在探索和利用之间取得平衡。

- ✍ **练习 2.2 :** *Bandit example* 考虑一个具有 $k = 4$ 个动作的 k 臂赌博机问题，分别表示为 1、2、3 和 4。考虑对该问题应用赌博机算法，该算法使用 ε -greedy 动作选择，样本平均动作值估计和对于所有 a , $Q_1(a) = 0$ 。假设动作和奖励的初始序列为 $A_1 = 1, R_1 = -1, A_2 = 2, R_2 = 1, A_3 = 2, R_3 = -2, A_4 = 2, R_4 = 2, A_5 = 3, R_5 = 0$ 。在某些时间步上， ε 情况可能已经发生，导致随机选择一个动作。这肯定发生在哪些时间步？在哪些时间步这可能已经发生？□
- ✍ **练习 2.3** 在图2.2所示的比较中，就累积奖励和选择最佳动作的概率而言，哪种方法在长期内表现最好？它会好多少？量化地表达你的答案。□

2.4 演增实现

到目前为止，我们已经讨论过的动作值方法都将动作值估计为观察到的奖励的样本平均值。现在，我们转向如何以计算有效的方式来计算这些平均值的问题，尤其是在恒定内存和恒定的每时间步计算的情况下。

为了简化表示法，我们集中在单个动作上。现在让 R_i 表示第 i 次选择该动作后收到的奖励，让 Q_n 表示其被选择 $n - 1$ 次后其动作值的估计，我们现在可以简单地写为

$$Q_n \doteq \frac{R_1 + R_2 + \dots + R_{n-1}}{n - 1}.$$

显而易见的实现是维护所有奖励的记录，然后在需要估计值的时候执行此计算。然而，如果这样做了，那么内存和计算需求将随着时间的推移而增长，正如我们所看到的那样。每一笔额外的奖励都需要额外的内存来存储它，并需要额外的计算来计算分子中的总和。

正如您可能怀疑的那样，这并不是真正必要的。利用处理每个新奖励所需的小而恒定的计算来设计更新平均值的递增公式很容易。给定 Q_n 和第 n 次奖励 R_n ，所有 n 个奖励的新平均值可由下式计算

$$Q_{n+1} = \frac{1}{n} \sum_{i=1}^n R_i$$



$$\begin{aligned}
&= \frac{1}{n} \left(R_n + \sum_{i=1}^{n-1} R_i \right) \\
&= \frac{1}{n} \left(R_n + (n-1) \frac{1}{n-1} \sum_{i=1}^{n-1} R_i \right) \\
&= \frac{1}{n} (R_n + (n-1)Q_n) \\
&= \frac{1}{n} (R_n + nQ_n - Q_n) \\
&= Q_n + \frac{1}{n} [R_n - Q_n], \tag{2.3}
\end{aligned}$$

对于 $n = 1$, 它仍然成立, 对于任意 Q_1 , 获得 $Q_2 = R_1$ 。此实现只需要 Q_n 和 n 的内存, 每个新的奖励只需要很小的计算 (2.3)。

此更新规则 (2.3) 的形式在本书中经常出现。一般形式是

$$NewEstimate \leftarrow OldEstimate + StepSize [Target - OldEstimate]. \tag{2.4}$$

表达式 $[Target - OldEstimate]$ 是估计中的错误。它是通过向“Target”迈进一步来减少。假定 `target` 可能指示移动的理想方向, 尽管它可能会有噪声。在上述情况下, 例如, `target` 是第 n 个奖励。

请注意, 增量方法 (2.3) 中使用的步长参数 (*StepSize*) 随时间步而变化。在处理动作 a 的第 n 个奖励时, 该方法使用步长参数 $\frac{1}{n}$ 。在本书中, 我们用 α 或更一般地用 $\alpha_t(a)$ 表示步长参数。

下面的框中显示了使用递增计算的样本平均和 ε -greedy 动作选择的完整赌博机算法的伪代码。假定函数 `bandit(a)` 采取动作并返回相应的奖励。

一个简单的赌博机算法

初始化, a 从 1 到 k :

$$Q(a) \leftarrow 0$$

$$N(a) \leftarrow 0$$

永久循环:

$$A \leftarrow \begin{cases} \arg \max_a Q(a) & \text{概率为 } 1 - \varepsilon \\ \text{随机动作} & \text{概率为 } \varepsilon \end{cases}$$

$$R \leftarrow \text{bandit}(A)$$

$$N(A) \leftarrow N(A) + 1$$

$$Q(A) \leftarrow Q(A) + \frac{1}{N(A)} [R - Q(A)]$$

2.5 非平稳问题

到目前为止讨论的平均方法适用于平稳的赌博机问题，即奖励概率不随时间变化的赌博机问题，如前所述，我们经常遇到实际上完全非平稳的强化学习问题。在这种情况下，将更多的权重放在近期的奖励上，而不是放在过去的长期奖励上，这是有意义的。最流行的方法之一是使用恒定步长参数。例如，用于更新 $n - 1$ 个过去奖励的平均 Q_n 的增量更新规则（2.3）被修改为

$$Q_{n+1} \doteq Q_n + \alpha [R_n - Q_n], \quad (2.5)$$

其中步长参数 $\alpha \in (0, 1]$ 是常量。这导致 Q_{n+1} 是过去奖励和初始估计 Q_1 的加权平均值：

$$\begin{aligned} Q_{n+1} &= Q_n + \alpha [R_n - Q_n] \\ &= \alpha R_n + (1 - \alpha) Q_n \\ &= \alpha R_n + (1 - \alpha) [\alpha R_{n-1} + (1 - \alpha) Q_{n-1}] \\ &= \alpha R_n + (1 - \alpha) \alpha R_{n-1} + (1 - \alpha)^2 Q_{n-1} \\ &= \alpha R_n + (1 - \alpha) \alpha R_{n-1} + (1 - \alpha)^2 \alpha R_{n-2} + \cdots + (1 - \alpha)^{n-1} \alpha R_1 + (1 - \alpha)^n Q_1 \\ &= (1 - \alpha)^n Q_1 + \sum_{i=1}^n \alpha (1 - \alpha)^{n-i} R_i. \end{aligned} \quad (2.6)$$

我们称其为加权平均值，是因为权重之和为 $(1 - \alpha)^n + \sum_{i=1}^n \alpha (1 - \alpha)^{n-i} = 1$ ，您可以自己检查一下。请注意，给予奖励 R_i 的权重 $\alpha (1 - \alpha)^{n-i}$ 取决于 $n - i$ 之前观察到的奖励数量。 $1 - \alpha$ 小于 1，因此给予 R_i 的权重随着介入奖励数量的增加而减少。实际上，权重根据 $1 - \alpha$ 的指数呈指数衰减。（如果 $1 - \alpha = 0$ ，则由于权重为 $0^0 = 1$ ，所有的权重都会在最后一个奖励 R_n 上）。因此，有时将其称为指数近期加权平均。

有时可以很方便地逐步改变步长参数。令 $\alpha_n(a)$ 表示步长参数，该参数用于处理在第 n 次选择动作 a 后收到的奖励。正如我们已经注意到的那样，选择 $\alpha_n(a) = \frac{1}{n}$ 会导致样本平均方法，通过大数定律可以保证该方法收敛到真实的动作值。但是，当然不能保证对序列 $\{\alpha_n(a)\}$ 的所有选择都收敛。随机逼近理论中的一个著名结果为我们提供了以概率 1 收敛的条件：

$$\sum_{n=1}^{\infty} \alpha_n(a) = \infty \quad \text{和} \quad \sum_{n=1}^{\infty} \alpha_n^2(a) < \infty \quad (2.7)$$

第一个条件以确保步幅足够大，以最终克服任何初始条件或随机波动。第二个条件保证最终步长变得足够小以确保收敛。

注意，对于样本平均情况， $\alpha_n(a) = \frac{1}{n}$ ，这两个收敛条件都满足，但是对于恒定步长参数， $\alpha_n(a) = \alpha$ ，则情况不满足。在后一种情况下，不满足第二个条件，这表明估计值从未完全收敛，而是随着最近收到的奖励而不断变化。如上所述，这在非平稳环境中实际上是可取的，有效非平稳问题是强化学习中最常见的问题。此外，满足条件（2.7）的步长参数序列通常会收敛很慢，或者需要进行大量调整才能获得令人满意的收敛速度。尽管满足这些收敛条件的步长参数序列在理论工作中经常使用，但是在应用和实证研究中



却很少使用它们。

练习 2.4 如果步长参数 α_n 不恒定，则估计值 Q_n 是先前接收的奖励的加权平均值，其权重与 (2.6) 给出的权重不同。就步长参数的序列而言，对于一般情况，类似于 (2.6)，每个先前奖励的权重是多少？□

练习 2.5 (编程) 设计并进行实验，以证明样本平均方法对于解决非平稳问题的困难。使用 10 臂试验的修改版本，其中起初所有 $q_*(a)$ 均相等，然后进行独立的随机游走（比如在每一步对所有 $q_*(a)$ 加上均值为零且标准差为 0.01 的正态分布增量）。绘制类似图 2.2 所示的图，为使用样本平均值进行增量计算的动作值方法，和另一使用恒定步长参数 $\alpha = 0.1$ 的动作值方法去准备图。使用 $\varepsilon = 0.1$ 和更长的运行时间，比如 10,000 步。□

2.6 乐观初始值

到目前为止，我们讨论的所有方法都在某种程度上依赖于初始动作值估计 $Q_1(a)$ 。按统计学的说法，这些方法与它们最初的估计是有偏的。对于样本平均方法，只要至少选择了一次所有动作，偏差就会消失，但对于 α 恒定的方法，偏差是永久性的，尽管如 (2.6) 所示，随着时间的推移，偏差会减少。在实践中，这种偏差通常不是问题，有时会非常有帮助。缺点是，初始估计实际上变成了必须由用户选择的一组参数，哪怕只是将它们全部设置为零。好处是，它们提供了一种简单的方法来提供一些关于期望奖励水平的先验知识。

初始动作值也可以用作鼓励探索的简单方式。假设我们没有像在 10 臂试验上那样将初始动作值设置为零，而是全部设置为 5。回想一下，这个问题中的 $q_*(a)$ 是从均值为 0 和方差为 1 的正态分布中选择的。因此，初始估计值为 5 是非常乐观的。但这种乐观主义鼓励对动作值方法的探索。无论最初选择哪种动作，奖励都会低于初始估计；学习者会切换到其他动作，对所获得的奖励感到“失望”。结果是，在值估计收敛之前，所有动作都会尝试几次。即使总是选择贪婪的动作，系统也会进行大量的探索。

图 2.3 显示了对于所有 a 在使用 $Q_1(a) = +5$ 的贪婪方法的 10 臂赌博机试验中的性能。为了进行比较，还显示了 $Q_1(a) = 0$ 的 ε -greedy 方法。起初，乐观方法的性能较差，因为它会进行更多的探索，但最终效果会更好，因为其探索会随着时间的推移而减少。我们认为这是一种鼓励探索的技术为乐观的初始值。我们认为这是一个简单的技巧，可以有效解决平稳问题，但远不是鼓励探索的普遍有用方法。例如，它不适用于非平稳问题，因为它的探索动力本来就是暂时的。如果任务发生变化，从而产生了新的探索需求，则此方法无济于事。确实，任何以任何特殊方式关注初始条件的方法都不太可能解决一般的非平稳情况。时间的开始只发生一次，因此我们不应该过多地关注它。这种批评也适用于样本平均方法，该方法也将时间的开始视为特殊事件，用相同的权重对所有随后的奖励进行平均。但是，所有这些方法都非常简单，并且其中一种（或它们的一些简单组合）在实践中通常就足够了。在本书的其余部分中，我们将频繁使用这些简单的探索技术中的几种。

练习 2.6 : *Mysterious Spikes* 图 2.3 中显示的结果应该是相当可靠的，因为它们是 2000 多



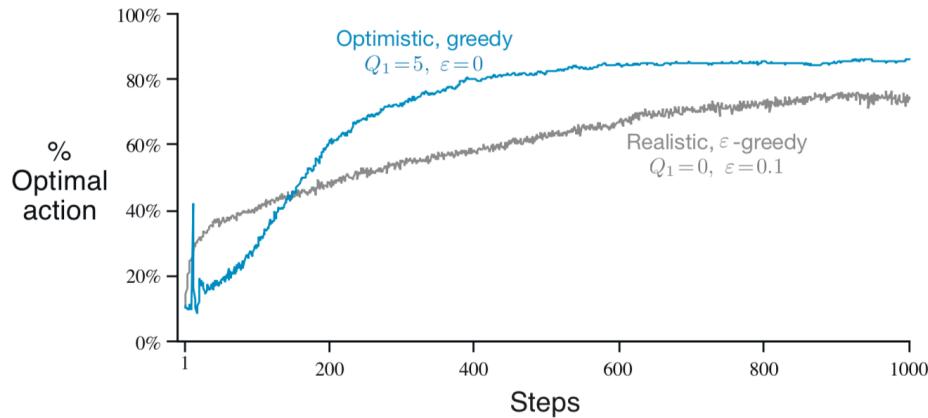


图 2.3: 乐观的初始动作值估计对 10 臂试验的影响。两种方法都使用恒定的步长参数 $\alpha = 0.1$ 。

个单独的、随机选择的 10 臂赌博机任务的平均值。那么，为什么乐观方法的曲线的早期部分会有振荡和尖峰呢？换句话说，是什么让这种方法在特定的早期步骤上表现得更好或更差呢？

练习 2.7 : Unbiased Constant-Step-Size Trick 在本章的大部分内容中，我们使用样本平均来估计动作值，因为样本平均不会产生恒定步长所产生的初始偏差（参见导致 (2.6) 的分析）。然而，样本平均并不是一个完全令人满意的解决方案，因为它们在非平稳问题上的表现可能很差。是否有可能避免固定步长的偏差，同时保持它们在非平稳问题上的优势？一种方法是使用步长为

$$\beta_n \doteq \alpha / \bar{o}_n, \quad (2.8)$$

来处理特定动作的第 n 次奖励，其中 $\alpha > 0$ 是常规的恒定步长，而 \bar{o}_n 是从 0 开始的跟踪：

$$\bar{o}_n \doteq \bar{o}_{n-1} + \alpha(1 - \bar{o}_{n-1}), \quad \text{对于 } n \geq 0, \quad \bar{o}_0 \doteq 0. \quad (2.9)$$

进行类似 (2.6) 中的分析，表明 Q 值是没有初始偏差的指数近期加权平均。

2.7 置信上限动作选择

探索是必要的，因为动作值估计的准确性总是存在不确定性。贪婪的动作是那些目前看起来最好的动作，但其他一些动作实际上可能更好。 ϵ -greedy 的动作选择迫使人们尝试非贪婪的动作，但不加区别，不偏好那些近乎贪婪或特别不确定的动作。最好根据非贪婪动作的实际最优潜力进行选择，同时考虑到它们的估计与最大值有多接近，以及这些估计中的不确定性。这样做的一种有效方法是根据

$$A_t \doteq \arg \max_a \left[Q_t(a) + c \sqrt{\frac{\ln t}{N_t(a)}} \right], \quad (2.10)$$

选择动作，其中 $\ln t$ 表示 t 的自然对数（必须将 $e \approx 2.71828$ 的数字提高到等于 t ）， $N_t(a)$ 表示在时间 t 之前选择动作 a 的次数（(2.1) 中的分母），而数字 $c > 0$ 控制探索的程度。如果 $N_t(a) = 0$ ，那么 a 被认为是最大化动作。

此置信上限 (upper confidence bound, UCB) 动作选择的思想是平方根项是对 a 的值估计中的不确定性或方差的度量。因此，最大化的量是动作 a 可能真值的一种上界，并且 c 确定置信水平。每选择 a 一次，不确定度都可能减小： $N_t(a)$ 递增，并且如分母所示，不确定度项减小。另一方面，每次选择除 a 以外的动作时， t 增加，但 $N_t(a)$ 不增加；因为 t 出现在分子中，所以不确定性估计增加。使用自然对数意味着增加随着时间的推移变小，但是无界的；所有动作最终都将被选择，但是具有较低估计值的动作，或者已经被频繁选择的动作，将随着时间的推移以递减的频率被选择。

图2.4显示了在 10 臂试验上使用 UCB 的结果。UCB 通常表现良好，如此处所示，但是要比 ε -greedy 从赌博机扩展到本书其余部分中考虑的更一般的强化学习环境要困难得多。一个难题在于处理非平稳问题；需要比第2.5节中介绍的方法更复杂的方法。另一个困难是处理大型状态空间，尤其是在使用本书第二部分中开发的函数逼近时。在这些更高级的设置中，UCB 动作选择的想法通常不切实际。

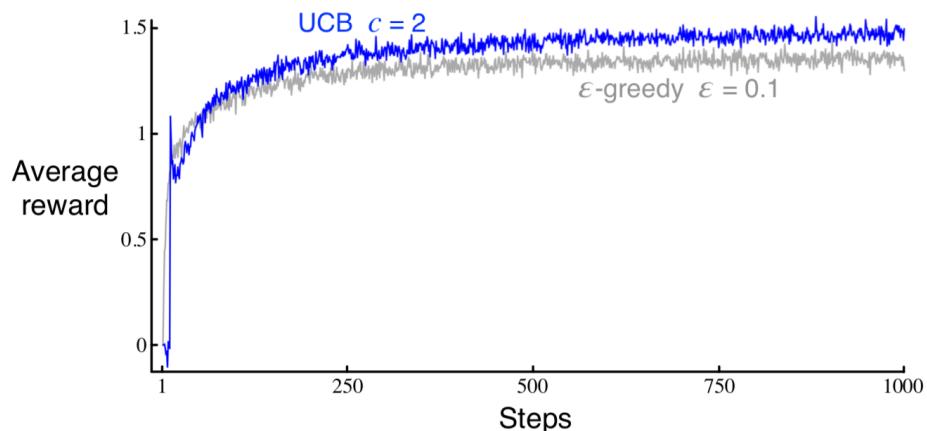


图 2.4：10 臂试验上 UCB 动作选择的平均性能。如图所示，UCB 通常比 ε -greedy 动作选择表现得更好，除了在前 k 个步中，它在尚未尝试的动作中随机选择。

✍ **练习 2.8 :** *UCB Spikes* 在图2.4中，UCB 算法在第 11 步显示了明显的峰值性能。这是为什么？请注意，为了让你的答案完全令人满意，必须解释为什么奖励在第 11 步增加，为什么在随后的步减少。提示：如果 $c = 1$ ，则尖峰不那么突出。□

2.8 梯度赌博机算法

到目前为止，在本章中，我们已经考虑了估计动作值并使用这些估计来选择动作的方法。这通常是一种很好的方法，但不是唯一可能的方法。在本节中，我们考虑学习每个动作 a 的数值偏好 (preference)，我们将其命名为 $H_t(a)$ 。偏好越大，采取该动作的频率就越高，但偏好没有奖励方面的解释。只有一个动作相对于另一个动作的相对偏好是重要的；如果我们将 1000 加到所有动作偏好上，则不会影响动作概率，这些概率是根据

如下的 soft-max 分布（即 Gibbs 分布或 Boltzman 分布）确定的：

$$\Pr\{A_t = a\} \doteq \frac{e^{H_t(a)}}{\sum_{b=1}^k e^{H_t(b)}} \doteq \pi_t(a), \quad (2.11)$$

这里我们还引入了一个有用的新符号， $\pi_t(a)$ ，用来表示在时间 t 采取动作 a 的概率。最初，所有动作偏好都是相同的（例如，对于所有 a , $H_1(a) = 0$ ），使得所有动作被选择的概率相等。

 **练习 2.9** 证明了在两种动作情况下，soft-max 分布与统计学和人工神经网络中常用的 logistic 函数或 sigmoid 函数的分布相同。 \square

基于随机梯度上升的思想，给出了 oft-max 动作偏好的自然学习算法。在每个步骤中，在选择动作 A_t 并接收奖励 R_t 之后，将通过以下方式更新动作偏好：

$$\begin{aligned} H_{t+1}(A_t) &\doteq H_t(A_t) + \alpha(R_t - \bar{R}_t)(1 - \pi_t(A_t)), & \text{and} \\ H_{t+1}(a) &\doteq H_t(a) - \alpha(R_t - \bar{R}_t)\pi_t(a), & \text{for all } a \neq A_t, \end{aligned} \quad (2.12)$$

其中 $\alpha > 0$ 是步长参数， $\bar{R}_t \in \mathbb{R}$ 是到时间 t （但不包括时间 t ）为止的所有奖励的平均值，它可以按照第2.4节（如果问题是非平稳的，则是第2.5节）中描述的方式递增计算。 \bar{R}_t 用作比较奖励的基准线。如果奖励高于基线，那么在未来采取 A_t 的概率就会增加，如果奖励低于基线，那么这种可能性就会降低。未选择的动作以相反的方向移动。

图2.5显示了在 10 臂试验的变体上使用梯度赌博机算法的结果，在该变型中，根据均值为 4 而不是零的正态分布选择真实的期望奖励（并且与以前一样具有单位方差）。由于奖励基线项的存在，所有奖励的这种上移对梯度赌博机算法完全没有影响，它会立即适应新的水平。但是，如果省略基线（即，如果在 (2.12) 中将 \bar{R}_t 视为常量零），则性能将显著降低，如图所示。

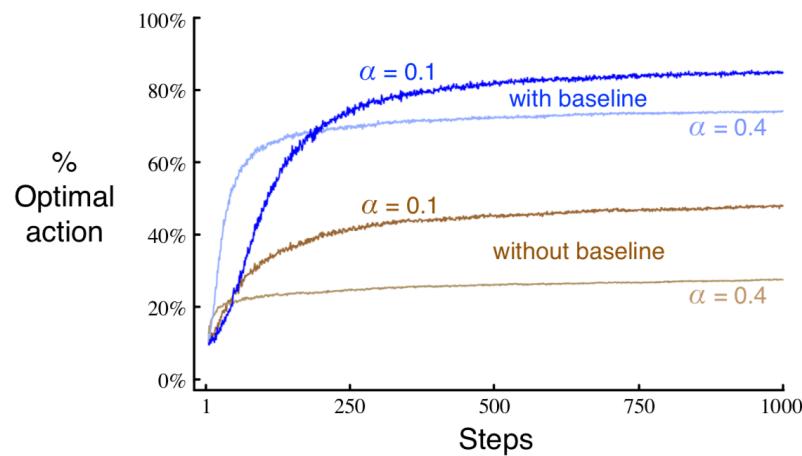


图 2.5：在 10 臂试验上，当 $q_*(a)$ 被选择为接近 4 而不是接近零时，具有和不具有奖励基线的梯度赌博机算法的平均性能。

随机梯度上升的赌博机梯度算法

通过将梯度赌博机算法理解为梯度上升的随机近似，可以更深入地了解梯度赌博机算法。在精确梯度上升中，每个动作偏好 $H_t(a)$ 将与增量对性能的影响成正比地递增：

$$H_{t+1}(a) \doteq H_t(a) + \alpha \frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)}, \quad (2.13)$$

这里的性能衡量标准是期望奖励：

$$\mathbb{E}[R_t] = \sum_x \pi_t(x) q_*(x), \quad (2.14)$$

增量影响的度量是该性能度量相对于动作偏好的偏导数。当然，由于假设我们不知道 $q_*(x)$ ，因此在我们的情况下不可能完全实现梯度上升，但是实际上我们的算法（2.12）的更新在期望值上等于（2.13），使该算法成为随机梯度上升的一个实例。计算表明，此过程仅需进行初次演算，但需要执行几个步骤。首先，我们仔细研究一下确切的性能梯度：

$$\begin{aligned} \frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} &= \frac{\partial}{\partial H_t(a)} \left[\sum_x \pi_t(x) q_*(x) \right] \\ &= \sum_x q_*(x) \frac{\partial \pi_t(x)}{\partial H_t(a)} \\ &= \sum_x (q_*(x) - B_t) \frac{\partial \pi_t(x)}{\partial H_t(a)}, \end{aligned}$$

其中 B_t ，称为基线，可以是不依赖于 x 的任何标量。我们可以在这里包含一个基线，而不改变等式，因为在所有的动作中梯度总和为零， $\sum_x \frac{\partial \pi_t(x)}{\partial H_t(a)} = 0$ ；当 $H_t(a)$ 改变时，一些动作的概率上升，一些下降，但是变化的总和一定为零，因为概率的总和总是 1。

接下来，我们将总和的每一项乘以 $\pi_t(x)/\pi_t(x)$ ：

$$\frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} = \sum_x \pi_t(x) (q_*(x) - B_t) \frac{\partial \pi_t(x)}{\partial H_t(a)} / \pi_t(x).$$

方程现在是期望的形式，将随机变量 A_t 的所有可能值 x 相加，然后乘以取这些值的概率。因此：

$$\begin{aligned} &= \mathbb{E} \left[(q_*(A_t) - B_t) \frac{\partial \pi_t(A_t)}{\partial H_t(a)} / \pi_t(A_t) \right] \\ &= \mathbb{E} \left[(R_t - \bar{R}_t) \frac{\partial \pi_t(A_t)}{\partial H_t(a)} / \pi_t(A_t) \right], \end{aligned}$$

这里我们选择基线 $B_t = \bar{R}_t$ ，并用 R_t 代替 $q_*(A_t)$ ，这是允许的，因为 $\mathbb{E}[R_t | A_t] =$

$q_*(A_t)$ 。不久，我们将建立 $\frac{\partial \pi_t(x)}{\partial H_t(a)} = \pi_t(x)(\mathbb{I}_{a=x} - \pi_t(a))$ ，如果 $a = x$ ，则将 $\mathbb{I}_{a=x}$ 定义为 1，否则定义为 0。现在，我们有

$$\begin{aligned} &= \mathbb{E} [(R_t - \bar{R}_t) \pi_t(A_t) (\mathbb{I}_{a=A_t} - \pi_t(a)) / \pi_t(A_t)] \\ &= \mathbb{E} [(R_t - \bar{R}_t) (\mathbb{I}_{a=A_t} - \pi_t(a))] \end{aligned}$$

回想一下，我们的计划一直是将性能梯度作为我们可以在每一步中采样的期望，就像我们刚才所做的那样，然后在每一步中与样例成比例地更新。用上述期望的样例代替 (2.13) 中的性能梯度：

$$H_{t+1}(a) = H_t(a) + \alpha(R_t - \bar{R}_t)(\mathbb{I}_{a=A_t} - \pi_t(a)), \quad \text{for all } a,$$

您可能会认为它等同于我们的原始算法 (2.12)。

因此，正如我们假设的那样，它仅仅是为了表明 $\frac{\partial \pi_t(x)}{\partial H_t(a)} = \pi_t(x)(\mathbb{I}_{a=x} - \pi_t(a))$ 。回忆一下导数的标准商规则：

$$\frac{\partial}{\partial x} \left[\frac{f(x)}{g(x)} \right] = \frac{\frac{\partial f(x)}{\partial x} g(x) - f(x) \frac{\partial g(x)}{\partial x}}{g(x)^2}$$

使用这个，我们可以写出

$$\begin{aligned} \frac{\partial \pi_t(x)}{\partial H_t(a)} &= \frac{\partial}{\partial H_t(a)} \pi_t(x) \\ &= \frac{\partial}{\partial H_t(a)} \left[\frac{e^{H_t(x)}}{\sum_{y=1}^k e^{H_t(y)}} \right] \\ &= \frac{\frac{\partial e^{H_t(x)}}{\partial H_t(a)} \sum_{y=1}^k e^{H_t(y)} - e^{H_t(x)} \frac{\partial \sum_{y=1}^k e^{H_t(y)}}{\partial H_t(a)}}{(\sum_{y=1}^k e^{H_t(y)})^2} \quad (\text{by the quotient rule}) \\ &= \frac{\mathbb{I}_{a=x} e^{H_t(x)} \sum_{y=1}^k e^{H_t(y)} - e^{H_t(y)} e^{H_t(a)}}{(\sum_{y=1}^k e^{H_t(y)})^2} \quad (\text{because } \frac{\partial e^x}{\partial x} = e^x) \\ &= \frac{\mathbb{I}_{a=x} e^{H_t(x)}}{\sum_{y=1}^k e^{H_t(y)}} - \frac{e^{H_t(x)} e^{H_t(a)}}{(\sum_{y=1}^k e^{H_t(y)})^2} \\ &= \mathbb{I}_{a=x} \pi_t(x) - \pi_t(x) \pi_t(a) \\ &= \pi_t(x) (\mathbb{I}_{a=x} - \pi_t(a)) \end{aligned}$$

我们刚刚证明了梯度赌博机算法的期望更新等于期望奖励的梯度，从而证明了该算法是随机梯度上升的一个实例。这保证了算法具有鲁棒的收敛性。

请注意，除了不依赖于所选动作之外，我们不需要奖励基线的任何属性。例如，我们可以将它设置为 0 或 1000，算法仍将是随机梯度上升的实例。基线的选择不会影响算法的期望更新，但会影响更新的方差，从而影响收敛速度(如图2.5中所示)。选择它作为奖励的平均值可能不是最好的，但它很简单，在实践中效果很好。



2.9 关联搜索（上下文赌博机）

到目前为止，我们在本章中只考虑了非关联任务，即不需要将不同的动作与不同的情况相关联的任务。在这些任务中，学习者要么在任务平稳时试图找到单一的最佳动作，要么在任务非平稳时试图跟踪随着时间的推移而变化的最佳动作。然而，在一般的强化学习任务中，有不止一种情况，学习策略的目标是：从情况到在这些情况下最好的动作的映射。为了为整个问题做好准备，我们简要讨论将非关联任务扩展到关联环境的最简单方法。

举个例子，假设有几个不同的 k 臂赌博机任务，每一步你都会随机选择其中一个。因此，赌博机任务一步一步地随机变化。在您看来，这是一个单一的非平稳 k 臂赌博机任务，其真实动作值随步骤的不同而随机变化。您可以尝试使用本章中介绍的可以处理非平稳性的方法，但除非真正的动作值缓慢更改，否则这些方法不会很好地工作。但是，现在假设，当为您选择了一个赌博机任务时，您会得到一些关于其身份（但不是其动作值）的独特线索。也许您面对的是一台实际的老虎机，它会随着动作值的改变而改变其显示颜色。现在，您可以学习一种策略，它将每个任务（由您看到的颜色表示）与面对该任务时要采取的最佳动作相关联-例如，如果是红色，则选择臂 1；如果是绿色，则选择臂 2。在没有任何区分赌博机任务的信息的情况下，使用正确的策略通常可以做得更好。

这是关联搜索任务的一个例子，之所以被称为关联搜索任务，是因为它既涉及到寻找最佳动作的试错学习，也涉及到将这些动作与它们最好的情况相关联。在文献中，关联搜索任务现在通常被称为上下文赌博机。关联搜索任务介于 k 臂赌博机问题和完全强化学习问题之间。它们类似于完全强化学习问题，因为它们涉及到学习策略，但又像我们版本的 k 臂赌博机问题，因为每个动作只影响立即奖励。如果动作被允许影响在下一种情况以及奖励，那么我们就有了完全的强化学习问题。我们将在下一章提出这个问题，并在本书的其余部分考虑其影响。

✍ **练习 2.10** 假设你面对的是一个 2 臂赌博机任务，其真实动作值随时间步而随机变化。具体地说，假设对于任何时间步，动作 1 和 2 的真值分别为 0.1 和 0.2，概率为 0.5（情况），以及 0.9 和 0.8，概率为 0.5（情况 B）。如果你在任何一步都不能说出你面对的是哪一种情况，你能达到的最好的成功期望是什么，你应该如何行动来实现它？现在假设在每一步中都被告知您面对的是情况 A 还是情况 B（尽管您仍然不知道真实的动作值）。这是一个关联搜索任务。在这个任务中，你能达到的最好的成功期望是什么？你应该如何行动才能达到这个目标？□

2.10 总结

在本章中，我们介绍了几种平衡探索和利用的简单方法。 ε -greedy 方法是一小部分时间随机选择动作，而 UCB 方法是确定性地选择，但通过在每个步骤巧妙地偏爱到目前为止收到的样本较少的动作来实现探索。梯度赌博机算法不是估计动作值，而是动作偏好，并且使用 softmax 分布以分级的、概率的方式偏爱更喜欢的动作。乐观地初始化估

计的简单计策甚至会导致贪婪的方法进行显著的探索。

自然会问这些方法中哪种方法最好。尽管这通常是一个很难回答的问题，但我们可以在这章中使用 10 胳试验运行它们，并比较它们的性能。麻烦的是它们都有一个参数。为了获得有意义的比较，我们必须将其性能视为其参数的函数。到目前为止，我们的图形显示了每种算法和参数设置随时间的学习过程，以生成该算法和参数设置的学习曲线。如果我们为所有算法和所有参数设置绘制学习曲线，则该图将过于复杂且拥挤而无法进行清晰的比较。相反，我们根据 1000 步的平均值总结出完整的学习曲线。该值与学习曲线下的面积成比例。图 2.6 显示了针对本章中的各种赌博机算法的度量，每种度量都是其自身参数的函数，在 x 轴上以单个刻度显示。这种图称为参数研究。请注意，参数值相差 2 倍，并以对数刻度显示。还要注意每种算法性能的倒 U 形特征；所有算法在其参数的中间值上表现最好，既不能过大也不能过小。在评估一种方法时，我们不仅应关注其最佳参数设置的性能，而且还应关注其对参数值的敏感程度。所有这些算法都是相当不敏感的，它们在大约相差一个数量级的参数值范围内表现良好。总体而言，在这个问题上，UCB 似乎表现最好。

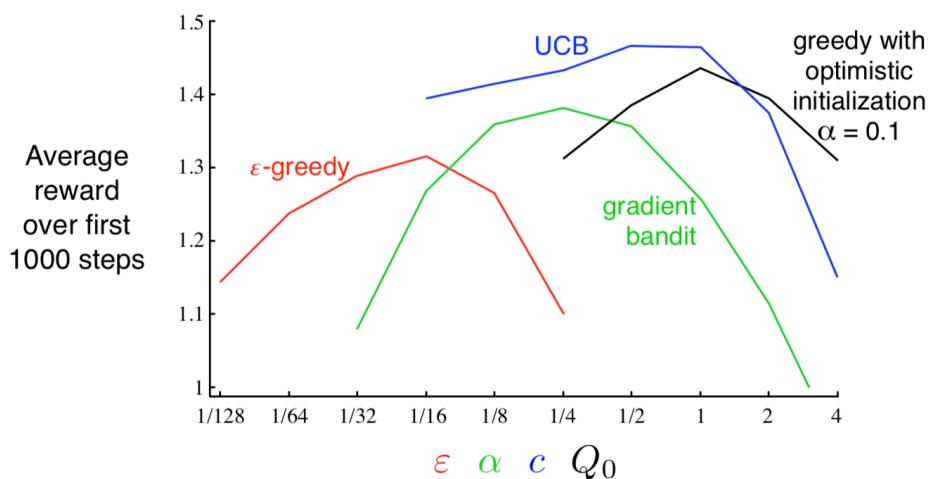


图 2.6：本章介绍的各种赌博机算法的参数研究。每一个点都是在特定的参数设置下，用特定的算法通过 1000 步获得的平均奖励。

尽管它们很简单，但我们认为本章中介绍的方法可以被认为是最先进技术。有更复杂的方法，但是它们的复杂性和假设使其对于我们真正关注的完全强化学习问题不切实际。从第 5 章开始，我们将介绍解决完全强化学习问题的学习方法，这些方法部分使用了本章中探讨的简单方法。

虽然本章探讨的简单方法可能是我们目前所能做的最好的，但它们远不是平衡探索和利用问题的完全令人满意的解决方案。

一种经过充分研究的平衡 k 胳赌博机问题中的探索与利用的方法是计算一种特殊的作用值，称为 Gittins 指数。在某些重要的特殊情况下，尽管确实需要对可能出现的问题的先验分布的完整知识，但我们通常认为这是不可用的，但这种计算是易于处理的，并直接导致最优解。另外，这种方法的理论和计算可扩展性似乎都不能推广到我们在本书其余部分中考虑的完全强化学习问题。

Gittins 指数方法是贝叶斯方法的一个实例，贝叶斯方法假设动作值的初始分布是已知的，然后在每个步骤之后精确地更新分布（假设真实的动作值是固定的）。通常，更新计算可能非常复杂，但是对于某些特殊分布（称为共轭先验），它们很容易。一种可能性是，然后根据每个步骤的动作是最佳动作的后验概率来选择它们。这种方法有时称为后验采样或汤普森采样，其执行效果通常与本章介绍的最佳无分布方法相似。

在贝叶斯环境中，甚至可以考虑计算探索与利用之间的最佳平衡。可以为任何可能的动作计算每个可能的立即奖励的概率以及动作值的后验分布。这种不断变化的分布成为问题的信息状态。给定 1000 个步骤的时间范围，可以考虑所有 1000 个步骤的所有可能动作，所有可能的结果奖励，所有可能的下一个动作，所有下一个奖励，等等。有了这些假设，就可以确定每个可能事件链的奖励和概率，并且只需要选择最佳事件即可。但是，树的可能性展得非常迅速。即使只有两个动作和两个奖励，树也会有 22000 片叶子。精确地执行这种庞大的计算通常是不可行的，但是也许可以进行科学地近似。这种方法将有效地将赌博机问题变成完全强化学习问题的一个实例。最后，我们也许可以使用近似的强化学习方法（如本书第二部分中介绍的方法）来寻求最佳解决方案。但这是属于研究的范畴，不在本入门书的讨论范围之内。

- ✍ **练习 2.11** (编程) 对于练习 2.5 中概述的非平稳情况，制作类似于图 2.6 的图。包括 $\alpha = 0.1$ 的步长不变的 ϵ -greedy 算法。使用 200,000 步的运行，并作为每个算法和参数设置的性能度量，使用最近 100,000 步的平均奖励。 □

2.11 书目与历史评论

2.1 在统计、工程和心理学领域都研究过赌博机问题。从统计学上讲，赌博机问题属于“实验的顺序设计”，由 Thompson (1933, 1934) 和 Robbins (1952) 提出，并由 Bellman (1956) 研究。Berry 和 Fristedt (1985) 从统计学的角度提供了对赌博机问题的广泛论述。Narendra 和 Thathachar (1989) 从工程学的角度来看待赌博机问题，对关注它们的各种理论传统进行了很好的讨论。在心理学中，赌博机问题在统计学习理论中发挥了作用（例如，Bush 和 Mosteller, 1955 年；Estes, 1950 年）。

启发式搜索文献中经常使用贪婪一词（例如 Pearl, 1984 年）。在控制工程中，探索与利用之间的冲突称为识别（或估计）与控制之间的冲突（例如 Witten, 1976b）。Feldbaum (1965) 将其称为双重控制问题，指的是在试图控制不确定性系统时需要同时解决识别和控制两个问题。在讨论遗传算法的各个方面时，Holland (1975) 强调了这种冲突的重要性，称其为开发需求与新信息需求之间的冲突。

2.2 Thathachar 和 Sastry (1985) 首次提出了解决我们的 k 臂赌博机问题的动作值方法。在自动机学习文献中，这些通常称为估计算法。术语动作值归因于 Watkins (1989)。最早使用 ϵ -greedy 方法的人可能也是 Watkins (1989, 187 页)，但是这个想法是如此简单，以至于可能会更早地使用。

2.4-2.5 该材料属于随机迭代算法的一般标题，Bertsekas 和 Tsitsiklis (1996) 对此进行了很好的介绍。

2.6 Sutton (1996) 在强化学习中使用了乐观初始化。

2.7 Lai 和 Robbins (1985), Kaelbling (1993b) 和 Agrawal (1995) 进行了使用置信度上界估计来选择动作的早期工作。我们在本文中介绍的 UCB 算法在文献中称为 UCB1，最早由 Auer, Cesa-Bianchi 和 Fischer (2002) 开发。

2.8 梯度赌博机算法是 Williams (1992) 引入的基于梯度的强化学习算法的特例，后来又发展为我们在本书稍后将讨论的 *actor-critic* 和策略梯度算法。Balaraman Ravindran (个人交流) 对我们的发展产生了影响。那里以及 Greensmith, Bartlett, Baxter (2002, 2004) 和 Dick (2015) 提供了有关基线选择的进一步讨论。此类算法的早期系统研究是由 Sutton (1984) 完成的。

动作选择规则 (2.11) 的术语“soft-max”归因于 Bridle (1990)。该规则似乎是 Luce (1959) 首次提出的。

2.9 Barto, Sutton 和 Brouwer (1981) 引入了术语“关联搜索”和相应的问题。关联强化学习这个词也已经用于关联搜索 (Barto 和 Anandan, 1985)，但是我们更喜欢保留这个术语作为完全强化学习问题的同义词 (如 Sutton, 1984)。(而且，正如我们指出的那样，现代文献也使用“上下文赌博机”一词来解决这个问题。) 我们注意到，Thorndike 的“效果定律”(在第 1 章中引用) 通过提及情境 (状态) 之间的关联链接的形成来描述关联搜索。) 和动作。根据操作性或工具性调节的术语学 (例如，Skinner, 1938)，判别性刺激是发信号通知存在特定的强化意外事件的刺激。用我们的术语来说，不同的鉴别刺激对应于不同的状态。

2.10 Bellman (1956) 首次展示了如何使用动态规划来在问题的贝叶斯公式内计算探索与利用之间的最佳平衡。Gittins 指数方法归功于 Gittins 和 Jones (1974)。Duff (1995) 展示了如何通过强化学习来学习针对赌博机问题的 Gittins 指数。Kumar (1985) 的调查很好地讨论了贝叶斯方法和非贝叶斯方法解决这些问题。信息状态一词来自有关部分可观测的 MDP 的文献。参见，例如 Lovejoy (1991)。

其他理论研究则集中在探索的效率上，通常表示为一种算法能够以多快的速度处理最佳决策策略。形式化探索效率的一种方式是，通过适应性学习来监督学习算法的样本复杂性，这是算法在学习目标函数时需要达到所需准确度的训练示例数量。强化学习算法的探索样本复杂度的定义是时间步数，在该时间步中算法没有选择接近最优的动作 (Kakade, 2003 年)。Li (2012) 在对强化学习中探索科学的理论方法的调查中讨论了这种方法和其他几种方法。Russo, VanRoy, Kazerouni, Osband 和 Wen (2018) 提供了汤普森采样的现代方法。

第三章 有限马尔可夫决策过程

在本章中，我们介绍有限马尔可夫决策过程或有限 MDPs 的形式问题，我们将在本书的其余部分中尝试解决这些问题。这个问题涉及评估反馈，就像赌博机一样，同时也涉及关联方面，即在不同情况下选择不同的动作。MDPs 是序列决策的经典形式，其动作不仅影响立即奖励，而且影响后续情况或状态，并影响这些未来奖励。因此，MDPs 涉及延迟奖励，并且需要权衡立即奖励和延迟奖励。在赌博机问题中，我们估计每个动作 a 的值 $q_*(a)$ ，在 MDPs 中，我们估计每个状态 s 中每个动作 a 的值 $q_*(s, a)$ ，或者在给定最优动作选择的情况下我们估计每个状态的值 $v_*(s)$ 。这些依赖于状态的量对于准确地将长期后果的信用分配给单独的动作选择至关重要。

MDPs 是强化学习问题的数学理想形式，可以针对其进行精确的理论陈述。我们介绍了问题的数学结构的关键要素，例如回报，价值函数和 Bellman 方程。我们试图传达可被定义为有限 MDPs 的广泛应用。如同在所有人工智能中一样，在应用广度和数学易处理性之间存在着紧密关系。在本章中，我们将介绍这种紧密关系，并讨论它所隐含的一些权衡和挑战。在第 17 章中讨论了可以在 MDPs 之外进行强化学习的一些方法。

3.1 Agent-环境接口

MDPs 旨在直接解决从交互中学习以实现目标的问题。学习者和决策者称为 agent，它所交互的包括 agent 以外的所有事物都称为环境。它们不断地相互作用，agent 选择动作，环境对这些动作作出响应并向 agent 呈现新的情况¹。环境还产生了奖励，即 agent 通过选择动作而力求使之最大化的特殊数值。

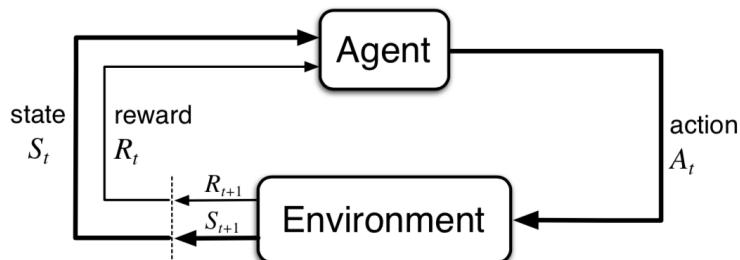


图 3.1：马尔可夫决策过程中 agent 与环境的交互。

更具体地说，agent 和环境在一系列离散的时间步 ($t = 0, 1, 2, 3, \dots$) 中进行交互²。在每个时间步 t ，agent 都接收到环境状态 $S_t \in \mathcal{S}$ 的某种表示形式，并在此基础上选择

¹我们使用术语 agent，环境和动作来代替工程师的术语控制器，受控系统（或工厂）和控制信号，因为它们对广泛的读者更有意义。

²我们将注意力集中在离散时间上，以使事情尽可能简单，即使许多想法可以扩展到连续时间情况（例如，参见 Bertsekas 和 Tsitsiklis, 1996; Doya, 1996）。

一个动作 $A_t \in \mathcal{A}(s)$ ³。一步之后，作为动作结果的一部分，agent 会获得一个数值奖励 $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$ ，并处于一个新状态 S_{t+1} ⁴，MDP 和 agent 一起产生一个如下开始的序列或轨迹：

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \dots \quad (3.1)$$

在有限的 MDP 中，状态、动作和奖励 (S , A 和 R) 的集合都具有有限数量的元素。在这种情况下，随机变量 R_t 和 S_t 具有明确定义的离散概率分布，其仅取决于先前的状态和动作。也就是说，给定先前状态和动作的特定值，对于这些随机变量的特定值， $s' \in \mathcal{S}$ 和 $r \in \mathcal{R}$ ，这些值在时间 t 出现的概率为：

$$p(s', r | s, a) \doteq \Pr\{S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a\}, \quad (3.2)$$

对于所有 $s', s \in \mathcal{S}$, $r \in \mathcal{R}$ 和 $a \in \mathcal{A}$ 。函数 p 定义了 MDP 的动态。等式中等号上的点提醒我们这是一个定义（在此函数 p 情况下），而不是根据前面的定义得出的事实。动态函数 $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ 是带四个参数的普通确定性函数。中间的“|”来自条件概率的表示法，但这只是在提醒我们 p 为 s 和 a 的每个选择指定了概率分布，即

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r | s, a) = 1, \quad \text{对于所有 } s \in \mathcal{S}, a \in \mathcal{A}(s). \quad (3.3)$$

在马尔可夫决策过程中，由 p 给出的概率完全表征了环境的动态。也就是说， S_t 和 R_t 的每个可能值的概率仅取决于紧邻的先前状态和动作 S_{t-1} 和 A_{t-1} ，并且在给定它们的情况下，完全不取决于早期的状态和动作。最好将其视为不是对决策过程的限制，而是对状态的限制。状态必须包括有关过去 agent 与环境交互的各个方面信息，这些信息将对未来有所影响。如果是这样，则称该状态具有马尔可夫性质。在本书中，我们将假定马尔可夫性质，尽管从第二部分开始，我们将考虑不依赖于马尔可夫性质的近似方法，在第 17 章中，我们将考虑如何从非马尔可夫观测值中学习和构造马尔可夫状态。

从四参数动态函数 p 中，可以计算出人们可能想要了解的有关环境的任何其他信息，例如状态转移概率（我们将其略微滥用地用符号表示为三参数函数 $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ ），

$$p(s' | s, a) \doteq \Pr\{S_t = s' | S_{t-1} = s, A_{t-1} = a\} = \sum_{r \in \mathcal{R}} p(s', r | s, a). \quad (3.4)$$

我们还可以将状态-动作对的期望奖励计算为两个参数的函数 $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ ：

$$r(s, a) \doteq \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r | s, a), \quad (3.5)$$

以及对状态-动作-下一个状态三元组的期望奖励作为三参数函数 $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ ，

³为了简化表示法，我们有时会假设一种特殊情况，即所有状态下的动作集都相同，并将其简单地写为 \mathcal{A} 。

⁴我们使用 R_{t+1} 而不是 R_t 表示由于 A_t 产生的奖励，因为它强调下一个奖励和下一个状态， R_{t+1} 和 S_{t+1} 是共同确定的。不幸的是，这两种约定都在文献中被广泛使用。



$$r(s, a, s') \doteq \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r \in \mathcal{R}} r \frac{p(s', r | s, a)}{p(s' | s, a)}. \quad (3.6)$$

在这本书里，我们通常使用四参数 p 函数 (3.2)，但是这些其他的符号有时也很方便。

MDP 框架是抽象且灵活的，可以以许多不同的方式应用于许多不同的问题。例如，时间步长不必指实时的固定间隔。它们可以指代决策和动作的任意连续阶段。这些动作可以是低级控制（例如，施加到机械臂的电机上的电压），也可以是高级决策（例如，是否吃午餐或去读研究生）。同样，状态可以采取多种形式。它们可以完全由低级别的感知（例如直接的传感器读数）确定，也可以是更高级别的抽象（例如房间中物体的符号描述）。构成状态的一些因素可能是基于对过去感知的记忆，甚至可能完全是思维或主观。例如，一个 agent 可能处于不确定物体位置的状态，或者只是在某种明确定义的意义上感到惊讶。同样，某些动作可能完全是思维或计算。例如，某些动作可能会控制 agent 选择思考的内容或将注意力集中在何处。一般而言，动作可以是我们想要学习如何做出的任何决定，状态可以是我们知道的可能对做出决定有用的东西。

特别是，agent 与环境之间的边界通常不同于机器人或动物身体的物理边界。通常，边界比这更接近 agent。例如，通常应将机器人及其感应硬件的电机和机械链接视为环境的一部分，而不是 agent 的一部分。同样，如果我们将 MDP 框架应用于人或动物，则肌肉，骨骼和感觉器官应该被视为环境的一部分。奖励也可能在自然和人工学习系统的物理内部计算，但被认为是 agent 的外部。

我们遵循的一般规则是，任何不能由 agent 随意更改的东西都被视为在其外部，因此是其环境的一部分。我们并不认为环境中的一切对 agent 都未知。例如，agent 通常很了解如何根据其动作和采取这些动作的状态的函数来计算奖励。但我们始终认为奖励计算处于 agent 的外部，因为它定义了 agent 所面临的任务，因此必须超出其任意更改的能力。实际上，在某些情况下，agent 可能知道有关其环境如何运作的所有信息，但仍然面临着困难的强化学习任务，就像我们可能确切地知道像魔方一样的难题如何工作，但仍无法解决它。agent 与环境的边界代表了 agent 的绝对控制权，而不是其知识的极限。

agent 与环境的边界可以出于不同的目的而位于不同的位置。在复杂的机器人中，许多不同的 agent 可能会同时运行，每个 agent 都有自己的边界。例如，一个 agent 可以做出高级决策，这些决策构成了实现高级决策的较低级 agent 所面对的状态的一部分。在实践中，一旦选择了特定的状态、动作和奖励，就确定了 agent 与环境的边界，从而确定了感兴趣的特定决策任务。

MDP 框架是从交互中学习目标导向问题的一个相当抽象的概念。它提出，无论感觉、记忆和控制装置的细节如何，以及人们试图实现什么目标，任何学习目标导向行为的问题都可以简化为在 agent 与其环境之间来回传递的三个信号：一种信号代表 agent 做出的选择（动作），一种信号代表做出选择的依据（状态），另一种信号定义 agent 的目标（奖励）。该框架可能不足以有效地表示所有决策学习问题，它具有广泛的实用性和适用性。

当然，特定状态和动作因任务而异，并且它们的表示方式会极大地影响性能。在强化学习中，就像在其他类型的学习中一样，这种代表性的选择目前更多的是艺术而不是



科学。在本书中，我们提供了一些有关表示状态和动作的良好方法的建议和示例，但是我们主要关注的是选择了表达方式之后，学习如何行动的一般原则。

例 3.1 : Bioreactor 假设正在应用强化学习来确定生物反应器的瞬间温度和搅拌速率（一大桶营养物质和细菌用于生产有用的化学物质）。这种应用中的动作可能是目标温度和目标搅拌速率，这些目标温度和目标搅拌速率会传递到较低级别的控制系统，进而直接激活加热元件和电机以达到目标。状态可能是热电偶和其他感官读数，可能是经过过滤和延迟，加上代表桶中成分和目标化学物质的符号输入。奖励可能是生物反应器生产有用化学物质的速率的瞬时测量。请注意，这里的每个状态都是传感器读数和符号输入的列表或向量，每个动作是由目标温度和搅拌速度组成的向量。具有这种结构化表示的状态和动作是强化学习任务的典型特征。另一方面，奖励始终是一个数字。 ■

例 3.2 : Pick-and-Place Robot 考虑使用强化学习在重复的放置任务中来控制机器人手臂的运动。如果我们想学习快速、平稳的运动，学习 agent 将不得不直接控制电动机，并且拥有关节连杆的当前位置和速度的低延迟信息。在这种情况下，动作可能是在每个关节处施加到每个电动机的电压，状态可能是关节角度和速度的最新读数。对于成功拾取并放置的每个对象，奖励可能为 +1。为了鼓励平稳的运动，可以在每个时间步长上根据动作的瞬间“急动”给予较小的负面奖励。 ■

✍ **练习 3.1** 设计三个符合 MDP 框架的您自己的示例任务，为每个任务确定状态、动作和奖励。尽可能使这三个例子各不相同。该框架是抽象且灵活的，可以以多种不同的方式应用。在你的至少一个例子中，以某种方式扩展它的限制。 □

✍ **练习 3.2** MDP 框架是否足以有效地代表所有以目标为导向的学习任务？你能想出任何明确的例外吗？ □

✍ **练习 3.3** 考虑一下驾驶问题。您可以根据油门、方向盘和制动器来定义动作，即身体与机器接触的地方。或者您可以将它们定义为，例如橡胶与道路相接的地方，将您的动作视为轮胎扭矩。或者您还可以定义它们为，例如大脑与身体接触的地方，这些动作是肌肉抽搐来控制您的四肢。或者您可以提高到一个很高的层次，并说您的动作是您选择开车前往的地方。在 agent 与环境之间划清界限的正确层次和正确地方是什么？在什么基础上线路的一个位置优于另一个位置？有没有什么基本的理由让你更喜欢一个位置，或者这是一个自由的选择？ □

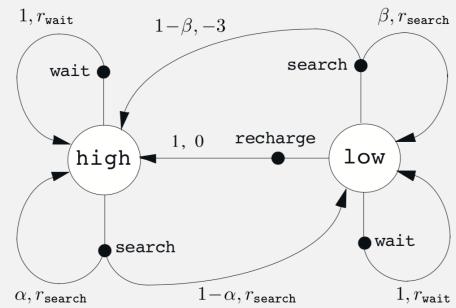
例 3.3 Recycling Robot

例 3.3 Recycling Robot

移动机器人的任务是在自然环境中收集空的汽水罐。它具有用于检测罐头的传感器，以及可以拾起它们并将其放置在板载垃圾箱中的手臂和抓取器；它使用可充电电池运行。机器人的控制系统具有用于解释感官信息，进行导航以及控制手臂和抓手的组件。强化学习 agent 根据电池的当前电量确定有关如何搜索罐的高级决策。为了举一个简单的例子，我们假设只能区分两个电荷水平，包括一个小的状态集 $\mathcal{S} = \{\text{high}, \text{low}\}$ 。在每种状态下，agent 可以决定是否 (1) 在特定时间段内主动搜索罐子，(2) 保持静止并等待某人带来一个罐子，或者 (3) 回到基地给电池充电。当

电量为 **high** 时，充电总是愚蠢的，因此我们不将其包括在为此状态设置的动作中。然后，动作集为 $\mathcal{A}(\text{high}) = \{\text{search}, \text{wait}\}$ 和 $\mathcal{A}(\text{low}) = \{\text{search}, \text{wait}, \text{recharge}\}$ 。大部分时间奖励为零，但是当机器人得到空罐时奖励为正，如果电池电量耗尽则奖励为很大的负数。找到罐头的最好方法是主动寻找它们，但这会消耗机器人的电池，而等待则不会。每当机器人搜索时，都有可能耗尽其电池。在这种情况下，机器人必须关闭并等待救援（产生低奖励）。如果电量水平为 **high**，则可以始终完成一段时间的主动搜索，而不会有耗尽电池的风险。以高能级开始的搜索时间以概率 α 离开能级 **high**，而以概率 $1 - \alpha$ 降低到 **low**。另一方面，在能量水平为 **low** 时进行的搜索时间使其具有低概率 β ，并且耗尽电池概率为 $1 - \beta$ 。在后一种情况下，必须解救机器人，然后将电池充电回高电平。机器人收集到的每个罐子都计为单位奖励，而每当需要救援机器人时，奖励为 -3。设 r_{search} 和 r_{wait} ，其中 $r_{\text{search}} > r_{\text{wait}}$ ，分别表示机器人在搜索和等待期间将收集的预期罐头数量（以及期望奖励）。最后，假设在基地充电期间无法收集任何罐子，并且耗尽电池也无法收集到罐子。那么这个系统是有限的 MDP，我们可以记下转移概率和期望奖励，其动态性如左表所示：

s	a	s'	$p(s' s, a)$	$r(s, a, s')$
high	search	high	α	r_{search}
high	search	low	$1 - \alpha$	r_{search}
low	search	high	$1 - \beta$	-3
low	search	low	β	r_{search}
high	wait	high	1	r_{wait}
high	wait	low	0	-
low	wait	high	0	-
low	wait	low	1	r_{wait}
low	recharge	high	1	0
low	recharge	low	0	-



请注意，表中针对当前状态 s ，动作 $a \in \mathcal{A}(s)$ 和下一个状态 s' 的每种可能组合都有一行。有些转移发生的可能性为零，因此没有为它们指定期望奖励。右图是总结有限 MDP 动态的另一种有用方法，即转移图。节点有两种类型：状态节点和动作节点。每个可能状态都有一个状态节点（用状态名称标记的大空心圆圈），每个状态-动作对都有一个动作节点（用动作名称标记的小实心圆圈，并用一条线连接到状态节点）。从状态 s 开始并执行动作 a 将从状态节点 s 移动到动作节点 (s, a) 。然后，环境通过离开动作节点 (s, a) 的箭头之一转移到下一个状态节点作为响应。每个箭头对应一个三元组 (s, s', a) ，其中 s' 是下一个状态，我们用转移概率 $p(s'|s, a)$ 以及该转移的期望奖励 $r(s, a, s')$ 来标记该箭头。请注意，标记离开动作节点的箭头的转移概率总和为 1。

练习 3.4 针对 $p(s', r|s, a)$ 给出一个类似于例 3.3 中的表。它应该具有 s, a, s', r 和 $p(s', r|s, a)$ 的列，以及 $p(s', r|s, a) > 0$ 的每个 4 元组的行。

3.2 目标和奖励

在强化学习中，根据从环境传递到 agent 的特殊信号（称为奖励）来形式化 agent 的目的或目标。在每个时间步，奖励都是一个简单的数字 $R_t \in \mathbb{R}$ 。非正式地说，agent 的目标是使所获得的奖励总额最大化。这意味着不是最大化立即奖励，而是最大化从长远来看的累积奖励。我们可以将这种非正式的想法明确地表述为奖励假设：

我们所指的目标和目的都可以很好地理解为所接收标量信号（称为奖励）的累加总和的期望值的最大化。

使用奖励信号来形式化目标的概念是强化学习最显著的特征之一。

尽管首先根据奖励信号制定目标似乎有些局限，但在实践中它已被证明是灵活且广泛适用的。了解这一点的最好方法是考虑它如何被使用或可能被使用的示例。例如，为了使机器人学会走路，研究人员在每个时间步上都提供了与机器人的向前运动成比例的奖励。在使机器人学会如何从迷宫中逃脱时，在逃脱之前经过的每个时间步长的奖励通常为-1。这鼓励 agent 尽快逃脱。为了使机器人学会寻找并收集空的汽水罐以进行回收，大多数时间给它的是零奖励，然后收集每个汽水罐后奖励 +1。当机器人碰到东西或有人大吼大叫时，可能还会想给它负面的奖励。对于学习西洋跳棋或国际象棋的 agent，自然的奖励是：获胜 +1，失败-1，平局和所有非终局位置为 0。

您可以看到所有这些示例中发生的事情。agent 总是学会最大化其奖励。如果我们希望它为我们做一些事情，我们必须以这种方式向它提供奖励，即通过最大化地它们，agent 也将实现我们的目标。因此，至关重要的是，我们设定的奖励能够真正表明我们想要实现的目标。特别是，奖励信号不是向 agent 传授有关如何实现我们想要做的事的先验知识的地方⁵。例如，一个下国际象棋的 agent 应该只因实际获胜而获得奖励，而不是因为实现了子目标，例如夺取对手的棋子或控制棋盘的中心。如果对实现这些子目标有所奖励，那么 agent 可能会找到一种方法来实现它们，而不是实现真正的目标。例如，它可能找到一种方法来夺取对手的棋子，甚至以输掉比赛为代价。奖励信号是您向机器人传达您想要实现的目标的方式，而不是您希望如何实现的方式⁶。

3.3 回报和 episode

到目前为止，我们已经非正式地讨论了学习的目标。我们已经说过，agent 的目标是获得从长远来看的最大累积奖励。如何正式定义呢？如果在时间步长 t 之后收到的奖励序列表示为 $R_{t+1}, R_{t+2}, R_{t+3}, \dots$ ，那么我们希望最大化该序列的哪个精确方面？通常，我们寻求最大化期望回报，其中回报（表示为 G_t ）定义为奖励序列的某些特定函数。在最简单的情况下，回报就是奖励的总和：

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \dots + R_T, \quad (3.7)$$

⁵传授这类先验知识的较好地方是初始策略函数或初始值函数，或者是对这些函数的影响。

⁶17.4 节进一步探讨了设计有效的奖励信号的问题。



其中 T 是最后的时间步长。这种方法在存在最终时间步长自然概念的应用中很有意义，也就是说，当 agent 与环境之间的交互自然分解为子序列时，我们将其称为 episodes⁷，例如玩游戏，穿越迷宫，或任何形式的重复交互。每个 episode 以称为终止状态的特殊状态结束，然后重置为标准起始状态或从起始状态的标准分布中抽样。即使您认为 episodes 以不同的方式结束（例如赢得或输掉一场比赛），下一个 episode 的开始也将独立于上一个 episode 的结束。因此，可以将这些 episodes 全部视为以相同的终止状态结束，并为不同的结果提供不同的奖励。具有此类 episode 的任务称为回合任务。在回合任务中，有时我们需要将所有非终止状态的集合（表示为 \mathcal{S} ）与所有状态加上终止状态的集合（表示为 \mathcal{S}^+ ）区分开。终止时间 T 是一个随机变量，通常随 episode 而变化。

另一方面，在许多情况下，agent 与环境之间的交互不会自然地分解为可识别的 episode，而是无限制地持续进行。例如，这是制定正在进行的过程控制任务的自然方法，或者是使用寿命长的机器人的应用。我们把这些称为连续任务。因此，公式 (3.7) 对于连续任务是有问题的，因为最终时间步长为 $T = \infty$ ，而我们试图最大化的回报本身很容易是无限的（例如，假设 agent 在每个时间步长都获得 +1 的奖励）。因此，在本书中，我们通常使用一个概念上稍微复杂，但在数学上简单得多的回报定义。

我们需要的另一概念是折扣。根据这种方法，agent 尝试选择动作，以便将来获得的折扣奖励的总和最大化。特别地，它选择 A_t 以最大化期望地折扣回报：

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \quad (3.8)$$

其中 γ 是 $0 \leq \gamma \leq 1$ 的参数，称之为折扣率。

折扣率决定了未来奖励的当前值：未来 k 个时间步长收到的奖励的价值仅相当于立即收到的奖励的 γ^{k-1} 倍。如果 $\gamma < 1$ ，则只要奖励序列 $\{R_k\}$ 有界，(3.8) 中的无限和就具有极限值。如果 $\gamma = 0$ ，则 agent 是“近视”的，其仅关注最大化立即奖励：在这种情况下，其目标是学习如何选择 A_t 以便仅最大化 R_{t+1} 。如果 agent 的每个动作都只影响立即奖励，而不影响未来的奖励，那么近视的 agent 可以通过分别最大化每个立即奖励来最大化 (3.8)。但一般而言，采取最大化立即奖励的动作会减少获得未来奖励的机会，从而减少回报。当 γ 接近 1 时，回报目标会更强烈地考虑未来的奖励；agent 变得更有远见。

连续时间步长的回报相互关联，这对于强化学习的理论和算法很重要：

$$\begin{aligned} G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned} \quad (3.9)$$

请注意，如果我们定义 $G_T = 0$ ，则这适用于所有时间步长 $t < T$ ，即使终止发生在 $t+1$ 。这通常使从奖励序列计算回报变得容易。

请注意，尽管回报 (3.8) 是无穷多个项的总和，但如果 $\gamma < 1$ ，奖励不为零且为常

⁷在文献中 episodes 有时称为 trials。



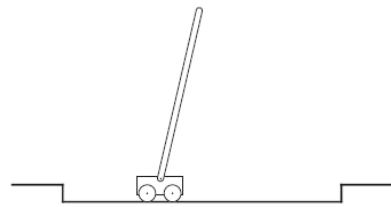
数，则仍然是有限的。例如，如果奖励是常数 +1，那么回报为：

$$G_t = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma} \quad (3.10)$$

- 练习 3.5 3.1 节中的方程式适用于连续的情况，需要进行修改（非常轻微）以适用于回合任务。通过给出 (3.3) 的修改版本，表明您知道所需的修改。 \square

例 3.4 :Pole-Balancing

此任务的目的是向沿轨道移动的手推车施加力，以防止铰接在手推车上的电线杆掉落：如果电线杆从垂直方向跌落超过给定角度，或者手推车偏离轨道，则称为失败。每次失败后，电线杆将重置为垂直。这项任务可以看作是回合性的，自然的回合是反复尝试以平衡杆子。在这种情况下，对于没有发生失败的每个时间步长，奖励可能是 +1，因此每个时间的回报就是直到失败为止的步数。在这种情况下，永远成功的平衡将意味着无限的回报。或者，我们可以使用折扣将杆子平衡视为一项连续的任务。在这种情况下，每次失败的奖励为 -1，其他所有时间的奖励为 0。然后，每次的回报将与 $-\gamma^K$ 相关，其中 K 是失败之前的时间步数。在任何一种情况下，通过使杆子保持尽可能长时间的平衡来最大化回报。



- 练习 3.6 假设您将杆子平衡视为回合任务，且使用折扣，除了失败奖励设为 -1，所有其他奖励设为 0。那么每个时间的回报是什么？这个回报与这个任务的折扣的、连续的形式有什么不同？ \square

- 练习 3.7 想象一下，您正在设计一个运行迷宫的机器人。您决定逃脱迷宫时给予它 +1 的奖励，在其他所有时间给予零的奖励。这项任务似乎自然地分解为 episodes，即连续穿过迷宫，因此您决定将其视为回合性任务，目标是最大化期望的总奖励 (3.7)。在运行学习 agent 一段时间后，您发现它在逃离迷宫方面没有任何改善。这出了什么问题？您是否已有效地向 agent 传达了您想要实现的目标？ \square

- 练习 3.8 假设 $\gamma = 0.5$ ，以及收到 $T = 5$ 的如下奖励序列， $R_1 = -1, R_2 = 2, R_3 = 6, R_4 = 3, R_5 = 2$ 。那么 G_0, G_1, \dots, G_5 是什么？提示：向后工作。 \square

- 练习 3.9 假设 $\gamma = 0.9$ ，以及奖励序列为 $R_1 = 2$ ，然后是 7s 的无限序列。那么 G_1 和 G_0 是什么？ \square

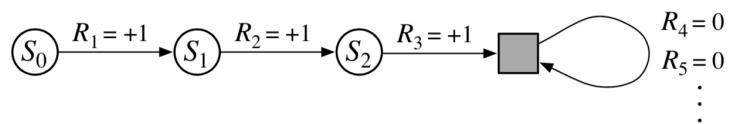
- 练习 3.10 证明 (3.10) 中的第二个等式。 \square

3.4 回合和连续任务的统一符号

在上一节中，我们描述了两种强化学习任务，一种是将 agent 与环境之间的交互自然分解为一系列单独的 episodes (回合任务)，另一种则是连续任务。从数学上讲，前一种情况更简单，因为每个动作仅影响随后 episode 期间获得的有限数量的奖励。在这本书中，我们有时考虑一种问题，有时会考虑另一种问题，但常常同时考虑这两种问题。因此，建立一个使我们能够准确地同时讨论这两种情况的符号是有用的。

为了精确地执行回合任务，需要一些附加的符号。我们需要考虑的不是一个漫长的时间步长序列，而是一系列 episodes，每个 episode 都包含一个有限的时间步长序列。我们从零开始重新编号每个 episode 的时间步长。因此，我们不仅要参考 S_t ，即在时间 t 的状态表示，而且要参考 $S_{t,i}$ ，第 i 个 episode 的时间 t 的状态表示（以及类似的 $A_{t,i}, R_{t,i}, \pi_{t,i}, T_i$ 等）。然而，事实证明，当我们讨论回合任务时，我们几乎不必区分不同的 episodes。我们几乎总是在考虑特定的单个 episode，或者对所有 episode 都适用的东西。因此，实际上，我们几乎总是通过删除对 episode 编号的明确引用来略微滥用符号。也就是说，我们用 S_t 来指代 $S_{t,i}$ ，依此类推。

我们需要另一种约定来获得涵盖回合任务和连续任务的单一符号。在一种情况下，我们将回报定义为有限项数之和（3.7），在另一种情况下，我们将回报定义为无限项数之和（3.8）。可以通过将 episode 终止视为一种特殊吸收状态的进入来统一这两者，该吸收状态仅转移到自身并且仅产生零奖励。例如，考虑状态转移图：



在这里，实心方块表示与 episode 结束相对应的特殊吸收状态。从 S_0 开始，我们得到奖励序列 $+1, +1, +1, 0, 0, 0, \dots$ 。将这些求和，我们都得到相同的回报，无论我们是对第一个 T （这里 $T = 3$ ）奖励求和，或是整个无限序列。即使我们引入折扣，也是如此。因此，根据（3.8），我们通常可以使用以下惯例来定义回报：在不需要时省略 episode 编号，并包括如果总和确定的情况下 $\gamma = 1$ 的可能性（例如，因为所有 episode 终止）。或者，我们可以写成

$$G_t \doteq \sum_{k=t+1}^T \gamma^{k-t-1} R_k, \quad (3.11)$$

包括 $T = \infty$ 或 $\gamma = 1$ （但不是全部）的可能性。在本书的其余部分中，我们将使用这些约定来简化表示法，并表述回合任务与连续任务之间的相似之处。（稍后，在第 10 章中，我们将介绍一个既连续又无折扣的表述。）

3.5 策略和值函数

几乎所有的强化学习算法都涉及估计值函数，即状态（或状态-动作对）的函数，用于估计 agent 在给定状态下的性能（或在给定状态下执行给定动作的性能）。这里的“有多好”的概念是根据预期的未来奖励，或者更确切地说是根据期望回报来定义。当然，agent 将来期望获得的奖励取决于其将采取的动作。因此，值函数根据特定的行为方式（称为策略）来定义。

正式地，策略是从状态到选择每个可能动作的概率的映射。如果 agent 在时间 t 遵循策略 π ，则 $\pi(a|s)$ 是在 $S_t = s$ 时 $A_t = a$ 的概率。像 p 一样， π 是一个普通函数； $\pi(a|s)$

中间的“|”仅提醒它定义了每个 $s \in \mathcal{S}$ 在 $a \in \mathcal{A}(s)$ 上的概率分布。强化学习方法指定了根据其经验来更改 agent 策略的方式。

练习 3.11 如果当前状态为 S_t , 并且根据随机策略 π 选择动作, 那么根据 π 和四参数函数 p (3.2), R_{t+1} 的期望是什么? \square

策略 π 下的状态 s 的值函数, 表示为 $v_\pi(s)$, 是从 s 开始并在其后跟随 π 时的期望回报。对于 MDP, 我们可以通过以下方式正式定义 v_π

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s \right], \text{ for all } s \in \mathcal{S}, \quad (3.12)$$

其中 $\mathbb{E}_\pi[\cdot]$ 表示给定 agent 遵循策略 π 的随机变量的期望值, t 是任意时间步长。请注意, 终止状态的值 (如果有的话) 始终为零。我们将函数 v_π 称为策略 π 的状态值函数。

同样, 我们定义在策略 π 下在状态 s 采取动作 a 的值, 表示为 $q_\pi(s, a)$, 作为从 s 开始, 采取动作 a , 然后遵循策略 π 的期望回报:

$$q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a \right] \quad (3.13)$$

我们称 q_π 为策略 π 下的动作值函数。

练习 3.12 用 q_π 和 π 给出 v_π 的等式。 \square

练习 3.13 用 v_π 和四个参数 p 给出 q_π 的等式。

可以根据经验估算值函数 v_π 和 q_π 。例如, 如果 agent 遵循策略 π , 并为遇到的每个状态维护该状态之后的实际回报的平均值, 则当遇到该状态的次数接近于无穷大时, 该平均值将收敛为该状态的值 $v_\pi(s)$ 。如果在每种状态下为采取的每个动作保留单独的平均值, 则这些平均值将类似地收敛到动作值 $q_\pi(s, a)$ 。我们称这种估算方法为蒙特卡罗方法, 因为它们涉及对实际回报的许多随机样本求平均。这些类型的方法在第 5 章中介绍。当然, 如果有很多状态, 则为每个状态单独保留独自的平均值可能不切实际。相反, agent 必须将 v_π 和 q_π 作为参数化函数 (参数少于状态) 来维护, 并调整参数以更好地匹配观察到的回报。尽管这很大程度上取决于参数化函数逼近器的性质, 但这也可以产生准确的估计。在本书的第二部分中讨论了这些可能性。

在强化学习和动态规划中使用的值函数的基本属性是, 它们满足类似于我们已经为回报建立的递归关系 (3.9)。对于任何策略 π 和任何状态 s , 以下一致性条件在 s 的值与其可能的后续状态的值之间成立:

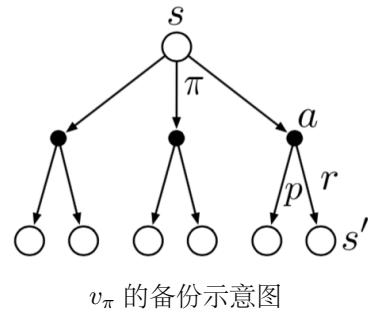
$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t | S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] \quad (\text{by (3.9)}) \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r | s, a) [r + \gamma \mathbb{E}_\pi[G_{t+1} | S_{t+1} = s']] \\ &= \sum_a \pi(a|s) \sum_{s', r} p(s', r | s, a) [r + \gamma v_\pi(s')], \quad \text{for all } s \in \mathcal{S}, \end{aligned} \quad (3.14)$$

其中暗含的是动作 a 是从集合 $\mathcal{A}(s)$ 中获取的, 下一个状态 s' 是从集合 \mathcal{S} 中获取的 (如



果是回合问题，则是从 \mathcal{S}^+ 中获取的），并且奖励 r 是从集合 \mathcal{R} 中获取的。还请注意，在最后一个方程式中，我们如何将两个和（一个在 s' 的所有值上，另一个在 r 的所有值上）合并成一个在两者的所有可能的值上的和。我们经常使用这种合并总和来简化公式。注意如何将最终表达式轻松地理解为期望值。它实际上是三个变量 a, s' 和 r 的所有值的总和。对于每个三元组，我们计算它的概率 $\pi(a|s)p(s', r|s, a)$ ，用该概率加权括号中的量，然后对所有可能性求和以获得期望值。

方程 (3.14) 是 v_π 的 Bellman 方程。它表达了一个状态的值与其后继状态的值之间的关系。如右图所示，设想从一个状态向前看到可能的后继状态。每个空心圆代表一个状态，每个实心圆代表一个状态-动作对。从状态 s (顶部的根节点) 开始，agent 可以根据其策略 π 采取任何一组动作 (图中显示了三个) 中的任何一个。根据这些情况，环境可能会以几个下一个状态 s' (图中显示两个状态) 之一以及奖励 r 做出响应，这取决于函数 p 给定的动态。Bellman 方程 (3.14) 对所有可能性进行平均，并根据其发生概率对每个可能性进行加权。它指出开始状态的值必须等于期望的下一个状态的 (折扣) 值，加上沿途期望的奖励。



v_π 的备份示意图

值函数 v_π 是其 Bellman 方程的唯一解。我们将在随后的章节中展示这个 Bellman 方程如何构成许多计算、近似和学习 v_π 的基础。我们将上述类似的图称为“备份图”，是因为这些关系图构成了强化学习方法的核心的更新或备份操作的基础。这些操作将值信息从其后继状态（或状态-动作对）传回到状态（或状态-动作对）。在整本书中，我们都使用备份图来提供所讨论算法的图形总结。（请注意，与转移图不同，备份图的状态节点不一定代表不同的状态；例如，状态可以是其自己的后继。）

例 3.5 : Gridworld 图3.2 (左) 显示了一个简单的有限 MDP 的矩形 gridworld 表示。网格的单元格对应于环境状态。在每个单元格上，可能有四个动作：north, south, east, west，这确定性地导致 agent 在网格上的相应方向上移动一个单元格。使 agent 移出网格的动作不会更改其位置，并且会导致-1 的奖励。除将 agent 从特殊状态 A 和 B 移出的那些动作以外，其他动作导致的奖励为 0。从状态 A 开始，所有四个动作产生的奖励为 +10 并将 agent 带到 A' 。从状态 B 开始，所有动作都会产生 +5 的奖励，并将 agent 带到 B' 。

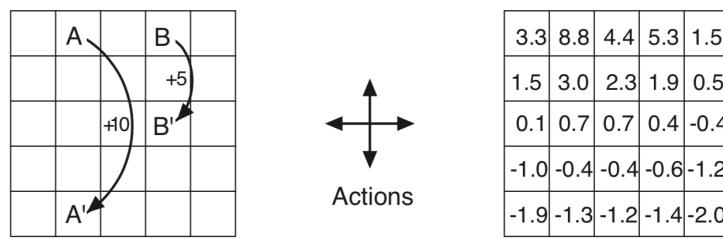


图 3.2: Gridworld 示例：期望奖励动态 (左) 和等价随机策略的状态值函数 (右)。

假设 agent 在所有状态下均以相同的概率选择所有四个动作。图3.2 (右) 显示了该

策略针对 $\gamma = 0.9$ 的折扣奖励情况下的价值函数 v_π 。该值函数是通过求解线性方程组 (3.14) 来计算的。注意下边缘附近的负值；这些是在随机策略下极有可能碰到网格边缘的结果。状态 A 是此策略下的最佳状态，但是其期望回报小于 10（立即奖励），因为 agent 从 A 被带到 A'，很可能从中运行到网格边缘。另一方面，状态 B 的价值大于 5，即它的立即奖励，因为状态 B 将 agent 带到具有正值的 B'。从 B' 开始，可能会撞到边缘的期望惩罚（负奖励）被可能走到 A 或 B 上的期望收益所补偿。 ■

- ✍ **练习 3.14** 对于例3.5的图3.2（右）中所示的值函数 v_π ，对于每个状态，Bellman 方程 (3.14) 必须成立。从数字上显示此方程适用于值为 +0.7 的中心状态，相对于它的四个相邻状态，值分别为 +2.3、+0.4、-0.4 和 +0.7。（这些数字仅精确到小数点后一位。） □
- ✍ **练习 3.15** 在 Gridworld 示例中，奖励对于目标为正，对于进入世界的边缘为负，其余时间为零。这些奖励的标记是否重要，或者只是它们之间的间隔重要？使用 (3.8) 证明，将常数 c 添加到所有奖励中会为所有状态的值添加常数 v_c ，因此不会影响任何策略下任何状态的相对值。就 c 和 γ 而言， v_c 是什么？ □
- ✍ **练习 3.16** 现在考虑将常数 c 添加到回合任务（例如迷宫赛跑）中的所有奖励中。这会产生影响吗，还是会像上面的连续任务中那样使任务保持不变？为什么或者为什么不？举个例子。 □

例 3.6：Golf 为了将打高尔夫球作为强化学习任务，我们将每次击球的罚分（负奖励）定为 -1，直到将球击入洞中为止。状态是球的位置。状态的值是从该位置到洞的击球数的负数。当然，我们的动作是我们如何瞄准和挥动球，以及我们选择哪种球杆。让我们把前者看作是给定的，仅考虑球杆的选择，我们假定它是推杆还是发球杆。图3.3的上部显示了一个可能的状态值函数 $v_{putt}(s)$ ，用于始终使用推杆的策略。洞内的终止状态的值为 0。我们假设在绿地上任何地方都可以进行推杆；这些状态的值为 -1。离开绿地我们无法通过推杆到达洞口，并且该值更大。如果我们可以从某个状态到达绿地，则该状态的值必须比绿色的值小一，即 -2。为简单起见，让我们假设我们可以非常精确和确定性地推杆，但范围有限。这给了我们在图中标记为 -2 的轮廓线。该线和绿地之间的所有位置都需要精确地进行两次击球才能进入洞中。同样，在 -2 等高线的推杆范围内的任何位置都必须具有 -3 的值，依此类推，以获得图中所示的所有等高线。推杆不会使我们脱离沙地陷阱，因此它们的值为 $-\infty$ 。总体而言，从发球区到球洞需要六杆。

- ✍ 动作值，即 q_π 的 Bellman 方程是什么？它必须根据状态-动作对 (s, a) 的可能后继动作值 $q_\pi(s', a')$ 给出动作值 $q_\pi(s, a)$ 。提示：右边的备份图与此方程式相对应。展示类

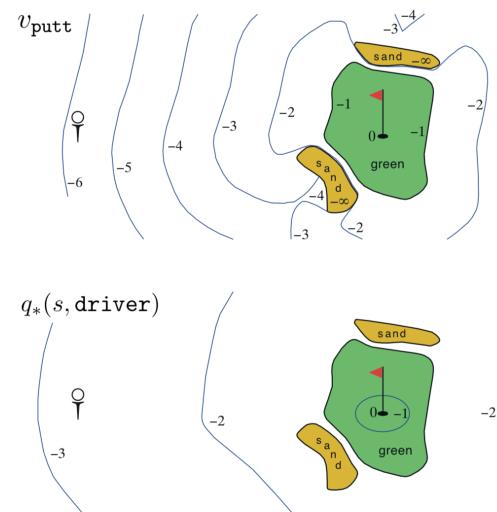
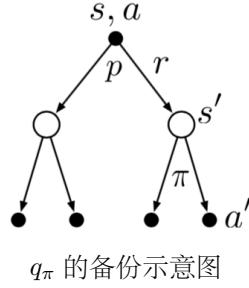


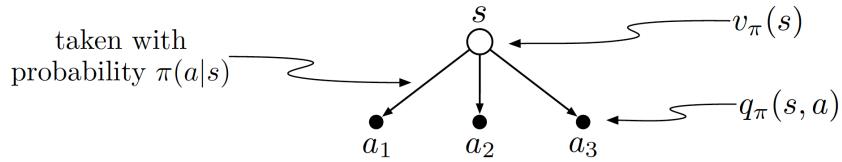
图 3.3：高尔夫示例：推杆的状态值函数（上图）和使用发球杆的最优动作值函数（下图）。

似于 (3.14) 的方程序列，但针对动作值。

□



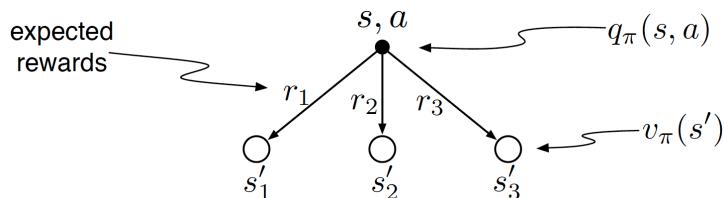
- 练习 3.18 状态的价值取决于该状态下可能采取的动作的价值，以及取决于在当前策略下采取每种动作的可能性。我们可以从植根于该状态的小备份图来考虑这一点，并考虑每个可能的动作：



给定 $S_t = s$ ，根据期望叶节点的值 $q_\pi(s, a)$ ，给出与该直觉和示意图相对应的等式，以表示根节点的值 $v_\pi(s)$ 。这个等式包括以遵循策略 π 为条件的期望。然后给出第二个等式，其中用 $\pi(a|s)$ 明确写出期望值，使得等式中不出现期望值符号。

□

- 练习 3.19 动作的值 $q_\pi(s, a)$ 取决于期望的下一个奖励和剩余奖励的期望总和。同样，我们可以用一个小的备份图来考虑这一点，该备份图扎根于一个动作（状态-动作对）并分支到可能的下一个状态：



给定 $S_t = s$ 和 $A_t = a$ ，根据期望的下一个奖励 R_{t+1} 和期望的下一个状态值 $v_\pi(S_{t+1})$ ，给出与该直觉和示意图相对应的等式的动作值 $q_\pi(s, a)$ 。此等式应包括不应以遵循该策略为条件的期望。然后给出第二个方程，用 (3.2) 定义的 $p(s', r|s, a)$ 显式写出期望值，这样方程中就不会出现期望值符号。

□

3.6 最优策略和最优值函数

粗略地说，解决强化学习任务意味着找到一种可以在长期内获得大量奖励的策略。对于有限的 MDP，我们可以通过以下方式精确定义最优策略。值函数定义了对策略的部分排序。如果策略 π 的期望回报在所有状态下均大于或等于 π' ，则将其定义为优于或等于策略 π' 。换句话说，当且仅当对于所有 $s \in \mathcal{S}$ 的 $v_\pi(s) \geq v_{\pi'}(s)$ 时， $\pi \geq \pi'$ 。始终存在

至少一个策略优于或等于所有其他策略。这是一个最优策略。尽管可能不止一个，但我们用 π_* 表示所有最优策略。它们共享相同的状态值函数，称为最优状态值函数，表示为 v_* ，定义为

$$v_*(s) \doteq \max_{\pi} v_{\pi}(s), \quad \text{for all } s \in \mathcal{S}. \quad (3.15)$$

最优策略还共享相同的最优动作值函数，表示为 q_* ，定义为

$$q_*(s, a) \doteq \max_{\pi} q_{\pi}(s, a), \quad \text{for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A}(s). \quad (3.16)$$

对于状态-动作对 (s, a) ，此函数给出在状态 s 中采取动作 a 并随后遵循最优策略的期望回报。因此，我们可以用 v_* 来写 q_* ，如下所示：

$$q_*(s, a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a]. \quad (3.17)$$

例 3.7: Optimal Value Functions for Golf 图3.3的下部显示了可能的优动作值函数 $q_*(s, driver)$ 的轮廓。如果我们首先用发球打一杆，然后选择发球还是推杆，以较优者为准，这些就是每个状态的值。发球可以使我们把球打得更远，但准确性较低。只有当我们已经非常接近的情况下，我们才能使用发球一杆打进洞。因此， $q_*(s, driver)$ 的-1 轮廓仅覆盖绿地的一小部分。但是，如果我们有两次击球，则可以从更远的地方到达洞，如-2 轮廓所示。在这种情况下，我们不必一直发球到小-1 轮廓内，而只需发球到绿地上的任何位置即可；从那里我们可以使用推杆。在本例中，最优动作值函数在执行特定的第一个动作之后，将值赋给发球，但随后使用最优动作。-3 轮廓仍更远，包括起始球台。从发球区开始，最好的动作顺序是两次发球和一次推杆，用三杆将球击入洞中。 ■

因为 v_* 是策略的值函数，所以它必须满足 Bellman 方程对状态值 (3.14) 给出的自洽条件。但是，由于 v_* 是最优值函数，因此它的一致性条件可以用特殊形式编写，而无需参考任何特定策略。这是 v_* 的 Bellman 方程或 Bellman 最优性方程。直观上，Bellman 最优性方程表示以下事实：最优策略下的状态值必须等于该状态下最优动作的期望回报。

$$\begin{aligned} v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s, a) \\ &= \max_a \mathbb{E}_{\pi_*}[G_t | S_t = s, A_t = a] \\ &= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \quad (\text{by (3.9)}) \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] \end{aligned} \quad (3.18)$$

$$= \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_*(s')]. \quad (3.19)$$

最后两个方程是 v_* 的 Bellman 最优方程的两种形式。 q_* 的 Bellman 最优方程为

$$q_*(s, a) = \mathbb{E} \left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \middle| S_t = s, A_t = a \right]$$

$$= \sum_{s',r} p(s',r|s,a) \left[r + \gamma \max_{a'} q_*(s',a') \right] \quad (3.20)$$

下图中的备份图以图形方式显示了 v_* 和 q_* 的 Bellman 最优方程中考虑的未来状态和动作的范围。这些与先前提供的 v_π 和 q_π 备份图相同，不同之处在于，在 agent 的选择点添加了圆弧，以表示采用了该选择中的最大值，而不是某些策略下的期望值。左侧的备份图以图形方式表示 Bellman 最优方程 (3.19) 和右侧的备份图的图形表示 (3.20)。

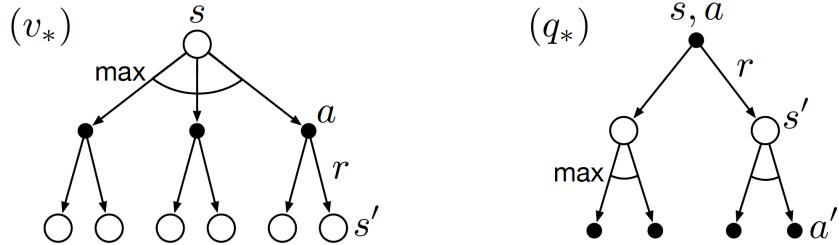


图 3.4: v_* 和 q_* 的备份示意图

对于有限的 MDP， v_* (3.19) 的 Bellman 最优方程具有唯一的解。Bellman 最优方程实际上是一个方程组，每个状态一个，因此，如果有 n 个状态，则有 n 个未知的 n 个方程。如果环境的动态 p 已知，则原则上可以使用多种求解非线性方程组的方法中的任何一种来求解 v_* 的方程组。同样，也可以求解 q_* 的一组相关方程。

一旦有了 v_* ，确定最优策略就相对容易了。对于每个状态 s ，在 Bellman 最优方程中将有一个或多个获得最大值的动作。任何仅将非零概率分配给这些动作的策略都是最优策略。您可以将其视为一步搜索。如果您有最佳值函数 v_* ，则在一步搜索后显示最优的动作将是最优动作。换句话说，任何对最优评估函数 v_* 贪婪的策略都是最优策略。贪婪一词在计算机科学中用于描述仅基于局部或直接考虑因素来选择替代方案的任何搜索或决策过程，而不考虑这种选择可能阻止将来获得更好替代方案的可能性。因此，它描述了仅根据动作的短期后果选择动作的策略。 v_* 的美妙之处在于，如果用它来评估动作的短期后果，特别是一步的后果，那么贪婪的策略实际上在我们感兴趣的长远意义上是最佳的，因为 v_* 已经考虑到了所有未来可能行为的奖励后果。借助 v_* ，最优的期望长期回报转化为每个状态都能局部可立即获得的量。因此，超前一步的搜索会产生长期的最优动作。

拥有 q_* 使选择最优动作变得更加容易。使用 q_* ，agent 甚至不必进行超前一步的搜索：对于任何状态 s ，它都可以简单地找到使 $q_*(s,a)$ 最大的任何动作。动作值函数有效地缓存所有超前一步搜索的结果。它提供最优的期望长期回报，作为每个状态-动作对局部可立即获得的值。因此，以表示状态-动作对的函数（而不是仅仅表示状态）为代价，最优动作值函数允许选择最优动作，而不必了解任何有关可能的后继状态及其值的任何信息，即不必了解有关环境动态的任何信息。

例 3.8 : Solving the Gridworld 假设我们为例3.5中引入的简单网格任务求解 v_* 的 Bellman 方程，并在图3.5 (左) 中再次显示。回想一下，状态 A 之后是 +10 奖励并转换到状态 A'，而状态 B 之后是 +5 奖励并转换到状态 B'。图3.5 (中) 显示了最优值函数，图3.5 (右) 显示了相应的最优策略。当一个单元格中有多个箭头时，所有相应的动作都是最优的。■

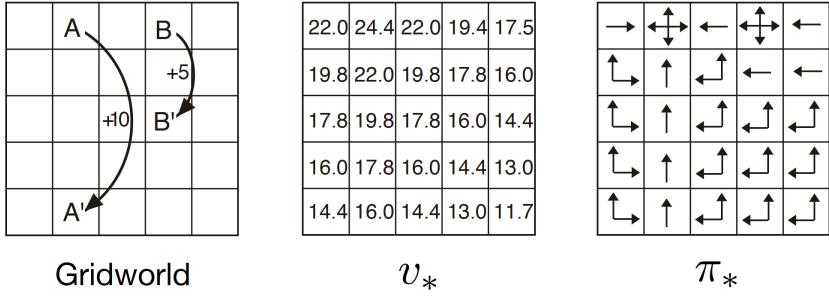


图 3.5: 网格示例的最优解。

例 3.9 : Bellman Optimality Equations for the Recycling Robot 使用 (3.19)，我们可以明确给出用于回收机器人示例的 Bellman 最优方程。为了更简洁，我们将状态 high 和 low，以及动作 search、wait、recharge 分别缩写为 h、l、s、w 和 re。因为只有两个状态，所以 Bellman 最优方程由两个方程组成。 $v_*(h)$ 的等式可写为：

$$\begin{aligned} v_*(h) &= \max \left\{ \begin{array}{l} p(h|h, s)[r(h, s, h) + \gamma v_*(h)] + p(l|h, s)[r(h, s, l) + \gamma v_*(l)], \\ p(h|h, w)[r(h, w, h) + \gamma v_*(h)] + p(l|h, w)[r(h, s, l) + \gamma v_*(l)] \end{array} \right\} \\ &= \max \left\{ \begin{array}{l} \alpha[r_s + \gamma v_*(h)] + (1 - \alpha)[r_s + \gamma v_*(l)], \\ 1[r_w + \gamma v_*(h)] + 0[r_w + \gamma v_*(l)] \end{array} \right\} \\ &= \max \left\{ \begin{array}{l} r_s + \gamma[\alpha v_*(h) + (1 - \alpha)v_*(l)], \\ r_w + \gamma v_*(h) \end{array} \right\}. \end{aligned}$$

对 $v_*(l)$ 遵循相同的过程可得到公式

$$v_*(l) = \max \left\{ \begin{array}{l} \beta r_s - 3(1 - \beta) + \gamma[(1 - \beta)v_*(h) + \beta v_*(l)] \\ r_w + \gamma v_*(l) \\ \gamma v_*(h) \end{array} \right\}.$$

对于 $0 \leq \gamma < 1$ 和 $0 \leq \alpha, \beta \leq 1$ 的 $r_s, r_w, \alpha, \beta, \gamma$ 的任意选择，正好有一对 $v_*(h)$ 和 $v_*(l)$ 同时满足这两个非线性方程。 ■

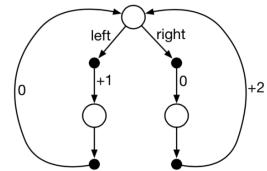
显式求解 Bellman 最优方程为找到最优策略，从而解决强化学习问题提供了一条途径。然而，这种解决方案很少直接有用。这类似于穷举搜索，向前看所有可能性，计算其发生的概率和它们对期望奖励的期望值。该解决方案至少依赖于三个在实践中很少成立的假设：(1) 我们准确地了解环境的动态；(2) 我们有足够的计算资源来完成解的计算；(3) 马尔可夫性质。对于我们感兴趣的任務类型，通常不能完全实现此解决方案，因为违反了这些假设的各种组合。例如，尽管第一个和第三个假设对 backgammon 游戏没有问题，但是第二个是主要障碍。由于该游戏包含约 10^{20} 个状态，因此在当今最快的计算机上求解 v_* 的 Bellman 方程将花费数千年，而对于 q_* 的求解也是如此。在强化学习中，通常不得不求取近似解。

许多不同的决策方法可以看作是近似求解 Bellman 最优方程的方法。例如，启发式搜索方法可以看作是将 (3.19) 的右侧扩展多次，直到一定深度，形成可能的“树”，然后在

“叶”节点使用启发式评估函数来逼近 v_* （诸如 A* 这样的启发式搜索方法几乎总是基于回合任务。）。动态规划的方法与 Bellman 最优方程紧密相关。许多强化学习方法可以清楚地理解为使用实际经历的转移代替期望转移的知识来近似解决 Bellman 最优方程。在以下各章中，我们将考虑各种此类方法。

- ✍ **练习 3.20** 画出或描述 golf 示例中的最优状态值函数。 □
- ✍ **练习 3.21** 画出或描述 golf 示例中用于推杆的最优动作值函数 $q_*(s, \text{putter})$ 轮廓。 □
- ✍ **练习 3.22**

考虑右图所示的连续 MDP。唯一要做出的决定是顶处状态，其左右有两个动作可用即 left 和 right，数字表示每次动作后确定收到的奖励。有两种确定性策略， π_{left} 和 π_{right} 。如果 $\gamma = 0$ ，哪种策略最优？如果 $\gamma = 0.9$ 呢？如果 $\gamma = 0.5$ 呢？ □



- ✍ **练习 3.23** 给出 recycling robot 中 q_* 的 Bellman 方程。 □
- ✍ **练习 3.24** 图3.5给出了 gridworld 最优状态的最优值为 24.4，保留了一位小数。使用您对最优策略的知识，并使用 (3.8) 以符号方式表示该值，然后将其计算到小数点后三位。□
- ✍ **练习 3.25** 给出用 q_* 表示的 v_* 方程。 □
- ✍ **练习 3.26** 给出用 v_* 和四参数 p 表示的 q_* 方程。 □
- ✍ **练习 3.27** 给出用 q_* 表示的 π_* 方程。 □
- ✍ **练习 3.28** 给出用 v_* 和四参数 p 表示的 π_* 方程。 □
- ✍ **练习 3.29** 根据三参数函数 p (3.4) 和两参数函数 r (3.5) 重写四个值函数 (v_π, v_*, q_π, q_*) 的四个 Bellman 方程。 □

3.7 最优和近似

我们定义了最优值函数和最优策略。显然，学习最优策略的 agent 做得很好，但是在实践中这种情况很少发生。对于我们感兴趣的的任务类型，只有以极高的计算成本才能生成最优策略。最优定义的概念组织了我们在本书中描述的学习方法，并提供了一种理解各种学习算法的理论特性的方法，但理想的情况是 agent 只能在不同程度上近似。如上所述，即使我们拥有完整而准确的环境动态模型，通常也无法通过解决 Bellman 最优方程来简单地计算最优策略。例如，象棋这样的棋盘游戏只是人类经验的一小部分，但是大型的、定制设计的计算机仍然无法计算出最优走法。agent 所面临的问题的关键方面始终是 agent 可用的计算能力，特别是 agent 可以在单个时间步中执行的计算量。

可用内存也是一个重要的约束。通常需要大量内存来构建值函数、策略和模型的近似值。在具有有限状态集的小型任务中，可以使用每个状态（或状态-动作对）只有一个条目的数组或表来形成这些近似值。我们称其为表格情况，相应的方法称为表格方法。但是，在许多实际感兴趣的情况下，表中的状态远远多于可能的条目。在这些情况下，必须使用某种更紧凑的参数化函数表示来近似函数。

我们对强化学习问题的框架迫使我们选择近似。但是，它也为我们提供了一些实现

有用近似的独特机会。例如，在逼近最优行为时，可能存在许多状态，其 agent 面对的可能性很低，以至于为他们选择次优动作对 agent 获得的奖励数量影响很小。例如，Tesauro 的 backgammon 玩家会发挥出色的技能，尽管它可能会在棋盘配置上做出非常糟糕的决定，而这在对抗专家的游戏中从来不会出现这种情况。实际上，TD-Gammon 可能会在游戏状态集的很大一部分中做出错误的决定。强化学习的在线性质使得可以通过在学习中做出更多努力来为经常遇到的状态做出好的决策来近似最优策略，而以减少对很少遇到的状态的干扰为代价。这是强化学习不同于其他近似求解 MDP 的方法的一个关键性质。

3.8 总结

让我们总结一下本章介绍的强化学习问题的要素。强化学习是从交互中学习如何行动以实现目标。强化学习 agent 及其环境在一系列离散的时间步长上进行交互。他们的接口的规范定义了一个特定的任务：动作是 agent 做出的选择；状态是做出选择的基础；奖励是评估选择的基础。agent 内部的一切都是完全已知的，并且可由 agent 控制。外部的所有事物都是无法完全控制的，但可能是完全已知，也不可能完全已知。策略是一种随机规则，其中 agent 通过该规则选择作为状态函数的动作。agent 的目标是随着时间的推移使其获得的奖励数量最大化。

当上述强化学习设置用明确定义的转移概率表示时，就构成了马尔可夫决策过程 (MDP)。有限的 MDP 是具有有限状态、动作和（如我们在此处所述）奖励集的 MDP。当前的强化学习理论大部分都局限于有限的 MDP，但其方法和思想具有更广泛的适用性。

回报是 agent 寻求最大化（期望价值）的未来奖励的函数。它有几种不同的定义，具体取决于任务的性质以及是否希望折扣延迟的奖励。没有折扣的形式适用于回合任务，在这种情景中，agent 与环境之间的交互自然而然地分解为 episodes。折扣后的形式适用于连续任务，在这种任务中，交互不会自然地分解为 episodes，而是可以无限制地继续。我们尝试定义两种任务的回报，使得一组方程既可以适用于回合情况，也可以适用于连续情况。

给定 agent 使用的策略，则策略的值函数分配给每个状态或状态-动作对，从该状态或状态-动作对的期望回报。最优值函数分配给每个状态或状态-动作对，任何策略可实现的最大期望回报。值函数最优的策略是最优策略。对于给定的 MDP，状态和状态-动作对的最优值函数是唯一的，但是可以有许多最优策略。任何对最优值函数贪婪的策略都一定是最优策略。Bellman 最优方程是最优函数必须满足的特殊一致性条件，并且原则上可以针对最优函数进行求解，从而可以相对轻松地确定最优策略。

强化学习问题可以通过各种不同的方式提出，这取决于对 agent 最初可获得的知识水平的假设。在知识完全的问题中，agent 具有完整而准确的环境动态模型。如果环境是 MDP，则此模型由完整的四参数动态函数 p (3.2) 组成。在知识不完全的问题中，没有完整且完美的环境模型。

即使 agent 具有完整且准确的环境模型，agent 通常也无法在每个时间步执行足够的计算以充分使用它。可用内存也是一个重要的约束。可能需要内存来建立值函数、策略

和模型的精确近似值。在大多数实际感兴趣的情况下，状态远远多于表中可能包含的条目，因此必须进行近似计算。

最优定义的概念组织了我们在本书中描述的学习方法，并提供了一种理解各种学习算法的理论特性的方法，但是理想的情况是，强化学习 agent 只能在不同程度上近似。在强化学习中，我们非常关注无法找到最优解但必须以某种方式近似的情况。

3.9 书目与历史评论

强化学习问题源于最优控制领域的马尔可夫决策过程 (MDP) 的思想。这些心理学上的历史影响和其他主要影响在第1章给出的简要历史中进行了描述。强化学习为 MDP 增加了对现实大问题的近似和不完全信息的关注。MDP 和强化学习问题与人工智能中的传统学习和决策问题只有微弱的联系。但是，人工智能现在正从多种角度积极探索用于规划和决策的 MDP 公式。MDP 比以前在人工智能中使用的公式更通用，因为它们允许更通用的目标和不确定性。

例如，Bertsekas (2005), White (1969), Whittle (1982、1983) 和 Puterman (1994) 处理了 MDP 的理论。Ross (1983) 对有限情况进行了特别紧凑的处理。MDPs 也在随机最优控制的标题下进行了研究，其中自适应最优控制方法与强化学习最密切相关（例如 Kumar, 1985; Kumar 和 Varaiya, 1986）。

MDPs 的理论源于努力理解在不确定性下做出序列决策的问题，其中每个决策都可以取决于先前的决策及其结果。它有时被称为多阶段决策过程理论或序列决策过程理论，它起源于关于顺序抽样的统计文献，从 Thompson (1933, 1934) 和 Robbins (1952) 的论文开始，我们在第2章中引用了关于赌博机问题（如果被表述为多情形问题，则是典型的 MDP）的这些文献。

我们知道使用 MDP 形式讨论强化学习的最早实例是 Andreae (1969b) 对学习机统一视图的描述。Witten 和 Corbin (1973) 试验了一种强化学习系统，后来由 Witten (1977, 1976a) 使用 MDP 形式对其进行了分析。尽管他没有明确提及 MDP，但 Werbos (1977) 提出了与现代强化学习方法相关的随机最优控制问题的近似求解方法（另请参见 Werbos, 1982、1987、1988、1989、1992）。尽管当时 Werbos 的想法尚未得到广泛认可，但他们强调近似解决各种领域（包括人工智能）的最优控制问题的重要性方面具有先见之明。强化学习和 MDPs 的最有影响的整合归功于 Watkins (1989)。

3.1 我们用 $p(s', r|s, a)$ 来描述 MDP 动态的特性有些不寻常。在 MDP 文献中，更常见的是根据状态转移概率 $p(s'|s, a)$ 和期望的下一个奖励 $r(s, a)$ 来描述。但是，在强化学习中，我们通常不得不参考单个的实际或样本奖励（而不仅仅是其期望值）。我们的表示法还使人更容易理解， S_t 和 R_t 通常是共同确定的，因此必须具有相同的时间索引。在强化学习教学中，我们发现我们的记法在概念上更直白，更容易理解。

有关状态的系统理论概念的直观讨论，请参见 Minsky (1967)。

生物反应器的例子是基于 Ungar (1990) 和 Miller and Williams (1992) 的工作。回收机器人的例子是由 Jonathan Connell (1989) 制造的集罐机器人所启发。Kober 和 Peters (2012)

提出了强化学习的机器人应用集合。

3.2 奖励假说由 Michael Littman（个人交流）提出。

3.3-3.4 回合和连续任务的术语与 MDP 文献中通常使用的术语不同。在该文献中，通常区分三种类型的任务：(1) 有限时长任务，其中交互在特定的固定时间步长后终止。(2) 无限期任务，其中交互可以持续任意长时间，但最终必须终止；(3) 无限时长任务，其中交互不会终止。我们的回合任务和连续任务分别类似于无限期任务和无限时长任务，但是我们更希望强调交互性质的不同。这种差异似乎比通常术语所强调的目标函数的差异更为根本。回合任务通常使用无限期目标函数，连续任务使用无限时长目标函数，但是我们将其视为常见的巧合，而不是根本的区别。

杆子平衡的例子来自 Michie 和 Chambers (1968) 和 Barto, Sutton 和 Anderson (1983)。

3.5-3.5 从长远来看，根据好坏来分配价值具有悠久的历史。在控制理论中，将状态映射到代表控制决策的长期后果的数值是最优控制理论的关键部分，最优控制理论是在 1950 年代通过扩展 19 世纪经典力学的状态函数理论而发展的（参见例如 Schultz 和 Melsa (1967)）。在描述如何编程计算机来下棋时，Shannon (1950) 建议使用评估函数，该函数考虑了国际象棋位置的长期优缺点。

Watkins (1989) 的 Q-learning 算法（第 6 章）将动作值函数作为强化学习的重要组成部分，因此这些函数通常被称为“Q 函数”。但是动作值函数的概念远不止于此。Shannon (1950) 提出，下棋程序可以使用函数 $h(P, M)$ 来确定位置 P 上的移动 M 是否值得研究。Michie (1961, 1963) 的 MENACE 系统和 Michie 和 Chambers (1968) 的 BOXES 系统可以理解为估计动作值函数。在古典物理学中，Hamilton 的主函数是一个动作值函数。牛顿动态对于该函数是贪婪的（例如，Goldstein, 1957）。动作值函数在 Denardo (1967) 关于动态规划的理论上以收缩映射的方式也起着核心作用。

Bellman 最优方程（对于 v_* ）由 Richard Bellman (1957a) 推广，他将其称为“基本函数方程”。连续时间和状态问题的 Bellman 最优方程的对应物被称为 Hamilton–Jacobi–Bellman 方程（或通常只是 Hamilton–Jacobi 方程），表明其起源于古典物理学（例如 Schultz 和 Melsa, 1967）。

Chris Watkins 提出了高尔夫的例子。

第四章 动态规划

动态规划 (dynamic programming, DP) 一词是指一组算法，这些算法可用于在给定环境完美模型（如马尔可夫决策过程 (MDP)）的情况下计算最优策略。经典的 DP 算法在强化学习中的作用有限，这既是因为它们假设了一个完美的模型，又因为它们的计算量很大，但是它们在理论上仍然很重要。DP 为理解本书其余部分中介绍的方法提供了必要的基础。实际上，所有这些方法都可以看作是试图实现与 DP 几乎相同的效果，只是需要较少的计算，且无需假设理想的环境模型。

我们通常假定环境是有限的 MDP。也就是说，我们假设其状态、动作和奖励集，即 $\mathcal{S}, \mathcal{A}, \mathcal{R}$ ，是有限的，并且对于所有 $s \in \mathcal{S}, a \in \mathcal{A}(s), r \in \mathcal{R}, s' \in \mathcal{S}^+$ （如果问题是回合的，则 \mathcal{S}^+ 是 \mathcal{S} 加上终止状态），其动态均由一组概率 $p(s', r|s, a)$ 给定。尽管 DP 想法可以应用于具有连续状态和动作空间的问题，但只有在特殊情况下才可能得到精确的解。获得具有连续状态和动作的任务的近似解的常用方法是量化状态和动作空间，然后应用有限状态 DP 方法。我们在第二部分中探索的方法适用于连续问题，并且是该方法的重要扩展。

DP 以及一般意义上的强化学习的关键思想是使用值函数来组织和构造对良好策略的搜索。在本章中，我们说明如何使用 DP 来计算第3章中定义的值函数。如此处所讨论的，一旦找到满足 Bellman 最优方程的最优值函数 v_* 或 q_* ，就可以轻松获得最优策略：

$$\begin{aligned} v_*(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] \\ &= \max_a \sum_{s', r} p(s', r|s, a) [r + \gamma v_*(s')], \text{ or} \end{aligned} \quad (4.1)$$

$$\begin{aligned} q_*(s, a) &= \mathbb{E} \left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \middle| S_t = s, A_t = a \right] \\ &= \sum_{s', r} p(s', r|s, a) \left[r + \gamma \max_{a'} q_*(s', a') \right], \end{aligned} \quad (4.2)$$

对于所有 $s \in \mathcal{S}, a \in \mathcal{A}(s), s' \in \mathcal{S}^+$ 。正如我们将看到的那样，DP 算法是通过将诸如此类的 Bellman 方程转化为赋值，即转化为用于改善所需值函数的近似值的更新规则而获得的。

4.1 策略评估

首先，我们考虑如何为任意策略 π 计算状态值函数 v_π 。在 DP 文献中，这称为策略评估。我们也将其称为预测问题。回顾第3章，对于所有 $s \in \mathcal{S}$ ，

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

$$\begin{aligned}
 &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \quad (\text{from (3.9)}) \\
 &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s]
 \end{aligned} \tag{4.3}$$

$$= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma v_\pi(s')], \tag{4.4}$$

其中 $\pi(a|s)$ 是在策略 π 下在状态 s 下采取动作 a 的概率，期望值用 π 下标以表示它们是遵循 π 的条件。只要在策略 π 下所有状态都保证 $\gamma < 1$ 或最终终止，就可以保证 v_π 的存在和唯一性。

如果环境动态完全已知，则 (4.4) 是 $|\mathcal{S}|$ 个未知的 $|\mathcal{S}|$ 组联立线性方程组成的系统 $(v_\pi(s), s \in \mathcal{S})$ 。原则上，它的解是一种简单的计算，即使很乏味。就我们的目的而言，迭代求解方法是最合适的。考虑一系列近似值函数 v_0, v_1, v_2, \dots ，每个函数将 \mathcal{S}^+ 映射到 \mathbb{R} （实数）。初始近似值 v_0 是任意选择的（除了终止状态，如果有的话，必须给定值 0），并且通过使用 v_π (4.4) 的 Bellman 方程作为更新规则来获得每个逐次的近似值：

$$\begin{aligned}
 v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1})|S_t = s] \\
 &= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')]
 \end{aligned} \tag{4.5}$$

对于所有 $s \in \mathcal{S}$ 。显然， $v_k = v_\pi$ 是此更新规则的固定点，因为在这种情况下， v_π 的 Bellman 方程可确保我们相等。事实上，在保证 v_π 存在的相同条件下，一般可以证明序列 v_k 当 $k \rightarrow \infty$ 时收敛到 v_π 。该算法称为迭代策略评估。

为了从 v_k 产生每个逐次逼近 v_{k+1} ，迭代策略评估将相同的操作应用于每个状态 s : 在被评估的策略下可能的所有一步转移，它将 s 的旧值替换为从 s 的后继状态的旧值中获得的新值，以及期望的立即奖励。我们称这种操作为期望更新。迭代策略评估的每次迭代都会更新每个状态的值一次，以产生新的近似值函数 v_{k+1} 。有几种不同的期望更新，这取决于状态（如此处）或状态-动作对是否正在更新，以及取决于合并后继状态的估计值的精确方式。DP 算法中完成的所有更新都称为期望更新，因为它们基于对所有可能的下一状态的期望，而不是基于样本下一状态。更新的性质可以用上面的公式表示，也可以用第3章介绍的备份图表示。例如，第 59 页上显示了与迭代策略评估中使用的期望更新相对应的备份图。

要编写一个顺序计算机程序来实现由 (4.5) 给出的迭代策略评估，您将必须使用两个数组，一个数组用于旧值 $v_k(s)$ ，一个数组用于新值 $v_{k+1}(s)$ 。使用两个数组，可以在不更改旧值的情况下从旧值逐个计算新值。当然，使用一个数组并“in place”更新值会更容易，也就是说，每个新值都会立即覆盖旧值。然后，根据状态的更新顺序，有时会使用新值代替 (4.5) 右侧的旧值。此 in-place 算法也收敛到 v_π ；实际上，它的收敛速度通常比两数组版本快，正如您可能期望的那样，因为它会在新数据可用时立即使用它们。我们认为更新是在整个状态空间中完成的。对于 in-place 算法，状态在扫描期间更新其值的顺序对收敛速度有显着影响。我们通常会在考虑 DP 算法时考虑到 in-place 版本。

下方框中的伪代码显示了完整的迭代策略评估 in-place 版本。请注意它如何处理终止。形式上，迭代策略评估仅在极限处收敛，但在实践中，它必须在这之前停止。伪代

码在每次扫描后测试量 $\max_{s \in \mathcal{S}} |v_{k+1}(s) - v_k(s)|$, 当其足够小时停止。

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

输入 π , 即要评估的策略

算法参数: 确定估计精度的小阈值 $\theta > 0$

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

循环:

$$\Delta \leftarrow 0$$

对于每个 $s \in \mathcal{S}$ 循环:

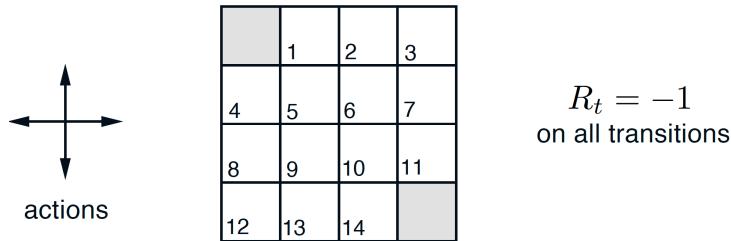
$$v \leftarrow V(s)$$

$$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$$

$$\Delta \leftarrow \max(\Delta, |v - V(s)|)$$

直到 $\Delta < \theta$

例 4.1 考虑如下所示的 4×4 的 gridworld。



非终止状态为 $\mathcal{S} = \{1, 2, \dots, 14\}$ 。每个状态中可能有四个动作, $\mathcal{A} = \{\text{up}, \text{down}, \text{right}, \text{left}\}$, 这确定性地导致相应的状态转移, 除了使 agent 离开网格的动作实际上使状态保持不变。因此, 例如, 对于所有 $r \in \mathcal{R}$, $p(6, -1|5, \text{right}) = 1, p(7, -1|7, \text{right}) = 1, p(10, r|5, \text{right}) = 0$ 。这是一项非折扣的回合任务。在所有转移中, 直到达到终止状态, 奖励均为-1。终止状态在图中用阴影表示 (尽管在两个地方都显示了它, 但在形式上是一个状态)。因此, 对于所有状态 s, s' 和动作 a , 期望奖励函数为 $r(s, a, s') = -1$ 。假设 agent 遵循等概率的随机策略 (所有动作的可能性均等)。图4.1的左侧显示了通过迭代策略评估计算出的值函数 $\{v_k\}$ 的序列。最终的估计值实际上是 v_π , 在这种情况下, 它为每个状态提供了从该状态到终止的期望步数。 ■

- ✍ **练习 4.1** 在例4.1中, 如果 π 是等概率随机策略, 那么 $q_\pi(11, \text{down})$ 是什么? $q_\pi(7, \text{down})$ 是什么? □
- ✍ **练习 4.2** 在例4.1中, 假设在状态 13 下方的网格世界中添加了一个新状态 15, 并且其动作 (left, up, right, down) 分别使该 agent 进入状态 12、13、14 和 15。假设从原始状态的转移未更改。那么, 等概率随机策略的 $v_\pi(15)$ 是什么? 现在假设状态 13 的动态也发生了变化, 以至于从状态 13 向下执行的动作将使 agent 到达新的状态 15。在这种情况下, 等概率随机策略的 $v_\pi(15)$ 是什么? □
- ✍ **练习 4.3** 对于动作值函数 q_π 及其通过序列函数 q_0, q_1, q_2, \dots 进行的逐次逼近, 类似于 (4.3), (4.4) 和 (4.5) 的方程是什么? □

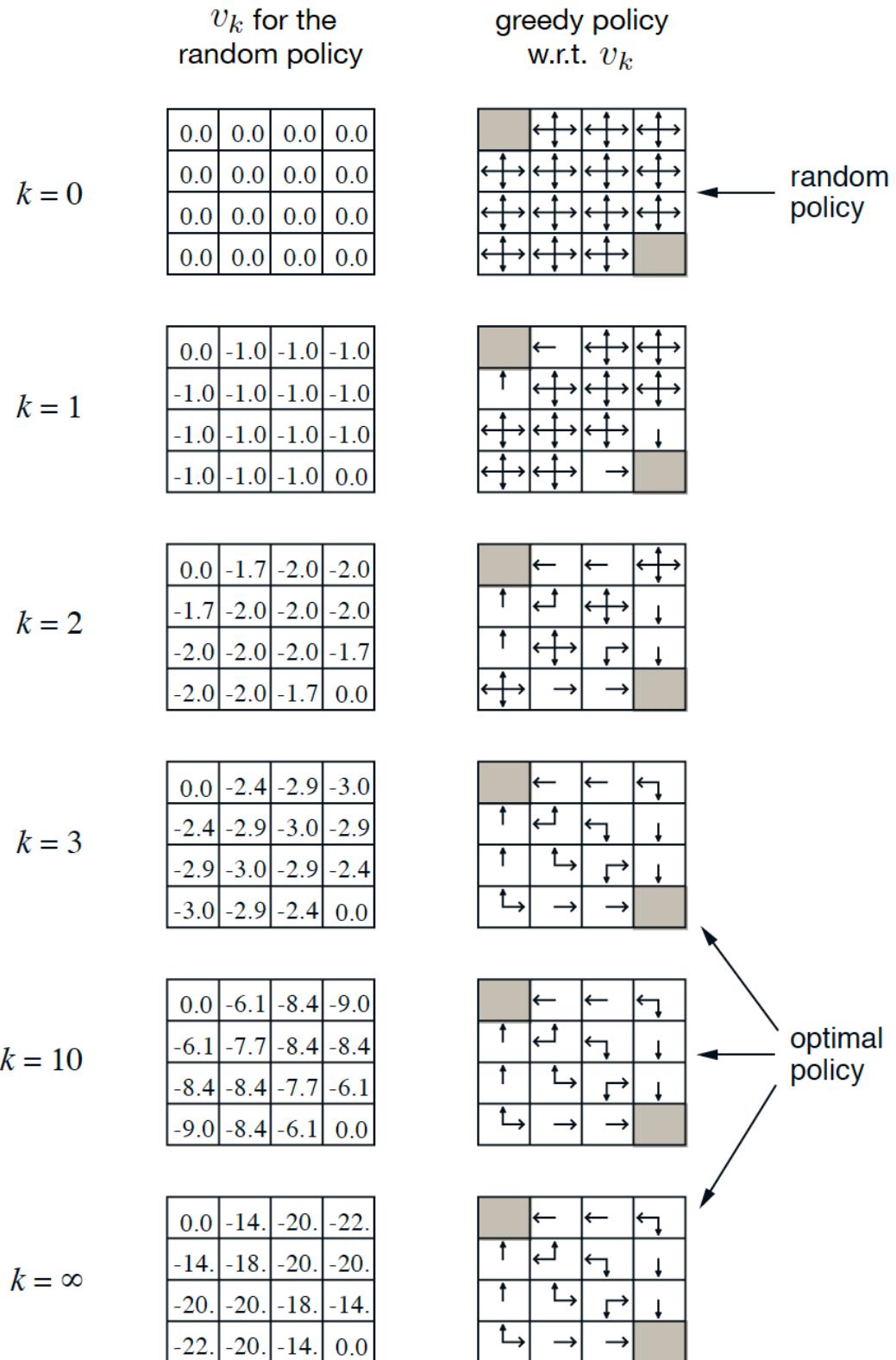


图 4.1: 小型网格世界上进行迭代策略评估的收敛性。左列是随机策略的状态值函数的近似序列 (所有动作的可能性均等)。右列是与值函数估计相对应的贪婪策略的序列 (所有达到最大值的动作均显示箭头, 所显示的数字四舍五入为两位有效数字)。最后的策略只能保证是对随机策略的改进, 但是在这种情况下, 它以及第三次迭代之后的所有策略都是最优的。

4.2 策略改进

我们计算策略价值函数的原因是为了帮助找到更好的策略。假设我们已经为任意确定性策略 π 确定了值函数 v_π 。对于某些状态 s , 我们想知道是否应该更改策略以确定性地选择 $a \neq \pi(s)$ 的动作。我们知道从 s (即 $v_\pi(s)$) 遵循当前策略有多好, 但是更改为新策略会更好还是更坏? 回答此问题的一种方法是考虑在 s 选择 a , 然后遵循现有策略 π 。这种行为方式的价值为

$$\begin{aligned} q_\pi(s, a) &\doteq \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s, A_t = a] \\ &= \sum_{s', r} p(s', r|s, a) [r + \gamma v_\pi(s')] . \end{aligned} \quad (4.6)$$

关键标准是它是否大于或小于 $v_\pi(s)$ 。如果它更大, 也就是说, 如果一次在 s 中选择 a , 然后跟随 π 比一直跟随 π 更好, 那么人们会希望每次遇到 s 都选择 a 更好, 而新策略实际上总体上将是更好的策略。

这是一个被称为策略改进定理的一般结果的特例。设 π 和 π' 是任意一对确定性策略, 使得对于所有 $s \in \mathcal{S}$

$$q_\pi(s, \pi'(s)) \geq v_\pi(s). \quad (4.7)$$

则策略 π' 必须等于或优于 π 。也就是说, 它必须从所有状态 $s \in \mathcal{S}$ 获得更大或相等的期望回报:

$$v_{\pi'}(s) \geq v_\pi(s). \quad (4.8)$$

此外, 如果在任何状态下均存在 (4.7) 的严格不等式, 则在该状态下必须存在 (4.8) 的严格不等式。

策略改进定理适用于本节开始时考虑的两个策略, 即原始确定性策略 π 和更改后的策略 π' , 其除了 $\pi'(s) = a \neq \pi(s)$, 与 π 相同。对于除 s 之外的其他状态, 因为两边相等, 所以 (4.7) 成立。因此, 如果 $q_\pi(s, a) > v_\pi(s)$, 则更改后的策略为确实比 π 好。

策略改进定理的证明背后的思想很容易理解。从 (4.7) 开始, 我们继续用 (4.6) 扩展 q_π 一边, 然后重新应用 (4.7) 直到得到 $v_{\pi'}(s)$:

$$\begin{aligned} v_\pi(s) &\leq q_\pi(s, \pi'(s)) \\ &= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s, A_t = \pi'(s)] \quad (\text{by (4.6)}) \\ &= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] \\ &\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1}))|S_t = s] \quad (\text{by (4.7)}) \\ &= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}[R_{t+2} + \gamma v_\pi(S_{t+2})|S_{t+1}, A_{t+1} = \pi'(S_{t+1})]|S_t = s] \\ &= \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2})|S_t = s] \end{aligned}$$

$$\begin{aligned}
& \vdots \\
\leq & \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots | S_t = s] \\
= & v_{\pi'}(s)
\end{aligned}$$

到目前为止，我们已经看到，在给定策略及其价值函数的情况下，我们可以轻松地评估单个状态下策略对特定动作的更改。考虑所有状态和所有可能动作的变化是自然的扩展，根据 $q_{\pi}(s, a)$ 选择在每个状态下表现最优的动作。换句话说，要考虑新的贪婪策略 π'

$$\begin{aligned}
\pi'(s) & \doteq \arg \max_a q_{\pi}(s, a) \\
& = \arg \max_a \mathbb{E}[R_{t+1} + \gamma v_{\pi}(S_{t+1}) | S_t = s, A_t = a] \\
& = \arg \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_{\pi}(s')],
\end{aligned} \tag{4.9}$$

其中 $\arg \max_a$ 表示 a 的值，在该值处后面的表达式被最大化（关系被任意打破）。根据 v_{π} ，贪婪策略会采取在短期内（在先行一步之后）看起来最好的动作。通过构建，贪婪策略满足策略改进定理 (4.7) 的条件，因此我们知道它与原始策略一样好，或更好。通过相对于原始策略的价值函数贪婪地制定新策略以对原始策略进行改进的过程称为策略改进。

假设新的贪婪策略 π' 与旧策略 π 一样好，但并不比旧策略 好。那么 $v_{\pi} = v_{\pi'}$ ，从 (4.9) 得出，对于所有 $s \in \mathcal{S}$:

$$\begin{aligned}
v_{\pi'}(s) & = \max_a \mathbb{E}[R_{t+1} + \gamma v_{\pi'}(S_{t+1}) | S_t = s, A_t = a] \\
& = \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_{\pi'}(s')].
\end{aligned}$$

但这与 Bellman 最优方程 (4.1) 相同，因此 v_{π} 必须是 v_* ，并且 π 和 π' 都必须是最优策略。因此，策略改进必须给我们严格更好的策略，除非原始策略已经是最佳的。

到目前为止，在本节中，我们已经考虑了确定性策略的特殊情况。在一般情况下，随机策略 π 指定在每种状态 s 中采取每个动作 a 的概率 $\pi(a|s)$ 。我们不会详细介绍，但实际上，本节中的所有思想很容易扩展到随机策略。特别地，策略改进定理如针对随机情况所陈述的那样进行。此外，如果在策略改进步骤中有联系，例如 (4.9)，也就是说，如果有多个动作可以达到最大效果，那么在随机情况下，我们就无需从中选择一个动作。取而代之的是，可以为每个最大化动作提供在新的贪婪策略中被选择的概率的一部分。只要所有次最大动作的概率为零，就可以使用任何分配方案。

图4.1的最后一行显示了针对随机策略的策略改进示例。在此，原始策略 π 是等概率随机策略，而新策略 π' 相对于 v_{π} 是贪婪的。左下图显示了值函数 v_{π} ，右下图显示了可能的 π' 集。 π' 图中带有多个箭头的状态是指几个动作在 (4.9) 中达到最大值的状态。允许在这些动作之间进行任何概率的分配。通过检查可以看出，任何此类策略的值函数 $v_{\pi'}(s)$ 在所有状态 $s \in \mathcal{S}$ 处均为-1，-2 或-3，而 $v_{\pi}(s)$ 最多为-14。因此，对于所有 $s \in \mathcal{S}$ ，

$v_{\pi'}(s) \geq v_{\pi}(s)$, 说明了策略的改进。尽管在这种情况下, 新策略 π' 恰好是最优的, 但通常只能保证改进。

4.3 策略迭代

一旦使用 v_{π} 改进了策略 π 以产生更好的策略 π' , 我们便可以计算 $v_{\pi'}$ 并再次对其进行改进以得到更好的 π'' 。因此, 我们可以获得一系列单调改进的策略和价值函数:

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

其中 \xrightarrow{E} 表示策略评估, 而 \xrightarrow{I} 表示策略改进。保证每一项策略都是对前一项策略的严格改进 (除非它已经是最优的)。因为有限的 MDP 仅具有有限数量的策略, 所以此过程一定在有限数量的迭代中收敛到最优策略和最优值函数。

查找最优策略的这种方式称为策略迭代。下面的方框中提供了完整的算法。请注意, 每个策略评估本身就是一个迭代计算, 都是从前一个策略的值函数开始的。这通常会大大提高策略评估的收敛速度 (大概是因为价值函数从一个策略到另一个策略的变化很小)。

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. 初始化

对于所有 $s \in \mathcal{S}$, 任意取 $V(s) \in \mathbb{R}$ 和 $\pi(s) \in \mathcal{A}(s)$

2. 策略评估

循环:

$$\Delta \leftarrow 0$$

对于每个 $s \in \mathcal{S}$ 循环:

$$v \leftarrow V(s)$$

$$V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s))[r + \gamma V(s')]$$

$$\Delta \leftarrow \max(\Delta, |v - V(s)|)$$

直到 $\Delta < \theta$ (决定估计精度的一个小的正数)

3. 策略改进

$$policy-stable \leftarrow true$$

对于每个 $s \in \mathcal{S}$:

$$old-action \leftarrow \pi(s)$$

$$\pi(s) \leftarrow \arg \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$$

如果 $old-action \neq \pi(s)$, 那么 $policy-stable \leftarrow false$

如果 $policy-stable$, 那么停止且返回 $V \approx v_*$ 和 $\pi \approx \pi_*$; 否则转到 2

例 4.2 : Jack's Car Rental 杰克为一家全国性的汽车租赁公司管理两个地点。每天都有一定量的客户到达每个位置来租车。如果杰克有车可用, 他就会把车租出去, 国家公司会给他 10 美元。如果他在那个位置没有车, 那生意就没了。归还汽车后的第二天即可



租车。为了帮助确保有需要的地方有汽车可用，杰克可以连夜在两个地点之间移动它们，每移动一辆汽车要花费 2 美元。我们假设在每个位置请求和返回的汽车数量是泊松随机变量，这意味着该数量为 n 的概率为 $\frac{\lambda^n}{n!} e^{-\lambda}$ ，其中 λ 为期望数量。假设在第一个和第二个位置的租赁请求的 λ 是 3 和 4，在返回时的 λ 是 3 和 2。为了略微简化问题，我们假设每个位置最多只能有 20 辆汽车（任何额外的汽车都归还给全国性公司，因此从问题中消失），在一个晚上最多只能从一个位置转移五辆汽车到另一个位置。我们将折扣率设为 $\gamma = 0.9$ ，并将其公式化为连续的有限 MDP，其中时间步长是天，状态是一天结束时每个位置的汽车数量，动作是夜间在两个地点之间移动的汽车净数量。图4.2显示了从不移动任何车辆的策略开始的策略迭代找到的策略序列。■

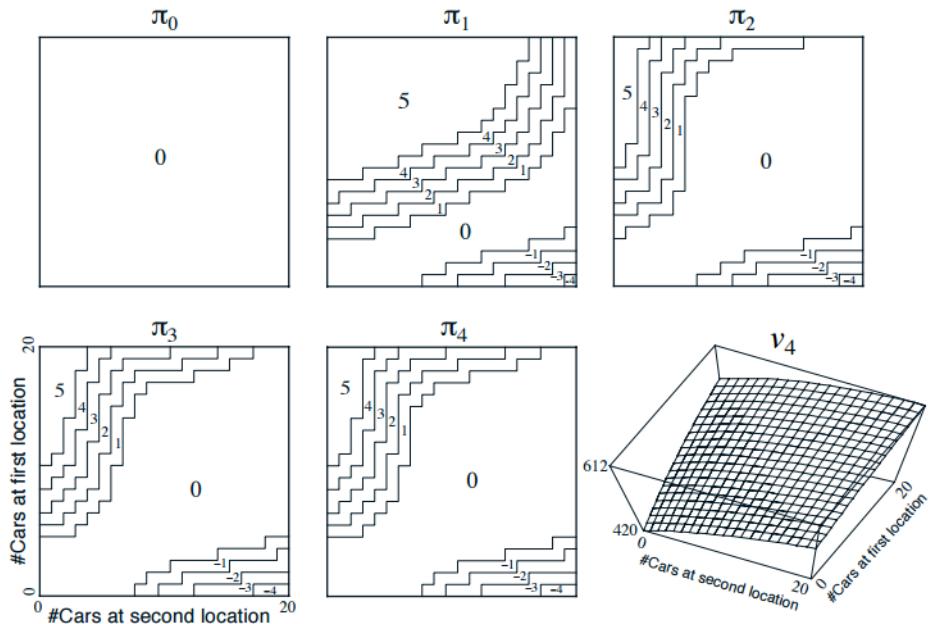


图 4.2：通过对杰克的租车问题进行策略迭代发现的策略序列，以及最终的状态值函数。前五个图显示了对于一天结束时每个位置的汽车数量，从第一个位置转移到第二个位置的汽车数量（负数表示从第二个位置转移到第一个位置）。每个后续策略都是对先前策略的严格改进，而最后一个策略是最优的。

策略迭代通常会以很少的迭代收敛，正如杰克的汽车租赁示例所说明的那样，图4.1中的示例也说明了这一点。图4.1的左下图显示了等概率随机策略的值函数，而右下图显示了此值函数的贪婪策略。策略改进定理向我们保证了这些策略比原始的随机策略要好。然而，在这种情况下，这些策略不仅更好，而且是最优的，它们以最少的步骤数进入终止状态。在此示例中，策略迭代将在一次迭代后找到最优策略。

练习 4.4 第 80 页上的策略迭代算法有一个细微的错误：如果策略在两个或两个以上同样良好的策略之间连续切换，它可能永远不会终止。这可以用于教学，但不适用于实际使用。修改伪代码，以确保收敛。□

练习 4.5 如何为动作值定义策略迭代？给出用于计算 q_* 的完整算法，类似于第 80 页的计算 v_* 的算法。请特别注意此练习，因为所涉及的思想将在本书的其余部分中使用。□

练习 4.6 假设您只考虑仅使用 ε -soft 的策略，这意味着在每个状态 s 中选择每个动作的概

率至少为 $\varepsilon/|\mathcal{A}(s)|$ 。请定性地描述第 80 页上针对 v_* 的策略迭代算法在步骤 3、2 和 1 的每个步骤中所需的更改。□

 **练习 4.7 (programming)** 编写用于策略迭代的程序，并通过以下更改重新解决杰克的租车问题。杰克在第一地点的一名员工每晚晚上乘汽车回家，且住在第二地点附近。她很乐意免费将一辆汽车送往第二个地点。每辆额外的汽车仍需花费 2 美元，所有朝另一方向行驶的汽车也是如此。另外，杰克在每个位置都有有限的停车位。如果在一个地点过夜的汽车停放了十辆以上（在任何车辆移动之后），则使用第二个停车场（与那里停放的汽车数量无关）必须产生 4 美元的额外费用。这些类型的非线性和任意动态通常发生在实际问题中，除动态规划外，其他优化方法无法轻松解决。要检查您的程序，请首先复制针对原始问题给出的结果。□

4.4 值迭代

策略迭代的一个缺点是其每次迭代都涉及策略评估，策略评估本身可能是旷日持久的迭代计算，需要对状态集进行多次扫描。如果以迭代方式进行策略评估，则仅在极限内才会发生完全收敛到 v_π 的情况。我们必须等待精确的收敛，还是可以就此止步？图4.1 中的示例确实表明，有可能截断策略评估。在该示例中，前三个之外的策略评估迭代对相应的贪婪策略没有影响。

实际上，可以以几种方式截断策略迭代的策略评估步骤，而不会丢失策略迭代的收敛性保证。一种重要的特殊情况是，仅一次扫描（每个状态一次更新）后就停止策略评估。此算法称为值迭代。可以将其编写为一种特别简单的更新操作，将策略改进和截断的策略评估步骤结合在一起：

$$\begin{aligned} v_{k+1}(s) &\doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')] , \end{aligned} \quad (4.10)$$

对于所有 $s \in \mathcal{S}$ 。对于任意 v_0 ，可以证明序列 $\{v_k\}$ 在保证 v_* 存在的相同条件下收敛到 v_* 。

了解值迭代的另一种方法是参考 Bellman 最优方程 (4.1)。请注意，只需将 Bellman 最优方程式转换为更新规则即可获得值迭代。还要注意，值迭代更新与策略评估更新 (4.5) 相同，不同之处在于它要求对所有动作都采取最大值。查看这种紧密关系的另一种方式是比较这些算法在 59 页（策略评估）和图3.4左侧（值迭代）的备份图。这是计算 v_π 和 v_* 的自然备份操作。

最后，让我们考虑值迭代如何终止。像策略评估一样，值迭代在形式上需要无限次迭代才能精确收敛到 v_* 。实际上，一旦值函数在扫描中仅发生少量变化，我们便停止。下面的方框显示了具有这种终止条件的完整算法。



Value Iteration, for estimating $\pi \approx \pi_*$

算法参数：一个小的阈值 $\theta > 0$ 决定估计精度

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

循环：

$$\Delta \leftarrow 0$$

对于每个 $s \in \mathcal{S}$ 循环：

$$v \leftarrow V(s)$$

$$V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$$

$$\Delta \leftarrow \max(\Delta, |v - V(s)|)$$

直到 $\Delta < \theta$

输出确定性策略 $\pi \approx \pi_*$, 使得 $\pi(s) = \arg \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$

值迭代有效地将策略评估的一次扫描和策略改进的一次扫描结合在一起。通常，可以通过在每个策略改进扫描之间插入多个策略评估扫描来实现更快的收敛。一般而言，可以将整类的截断策略迭代算法视为扫描序列，其中一些使用策略评估更新，而另一些使用值迭代更新。因为 (4.10) 中的最大运算是这些更新之间的唯一区别，所以这仅意味着将最大运算添加到策略评估的某些扫描中。所有这些算法都收敛到折扣的有限 MDPs 的最优策略。

例 4.3 : Gambler's Problem

赌徒有机会下注一系列掷硬币的结果。如果硬币正面朝上，那么他赢的钱与他在这一掷硬币上下注的美元一样多；如果是反面，他就输掉了赌注。当赌徒通过达到其 100 美元的目标获胜时，或由于钱用完而输时，游戏结束。每次掷硬币时，赌徒必须决定要押出他的资本的哪一部分，以整数美元计。可以将此问题表述为无折扣的回合的有限 MDP。状态是赌徒的资本 $s \in \{1, 2, \dots, 99\}$, 动作是赌注, $a \in \{0, 1, \dots, \min(s, 100 - s)\}$ 。除赌徒达到目标的交易 (+1) 外，所有交易的奖励均为零。然后，状态值函数给出从每个状态中获胜的概率。策略是从资产水平到赌注的映射。最优策略使达到目标的可能性最大化。 ph 表示硬币正

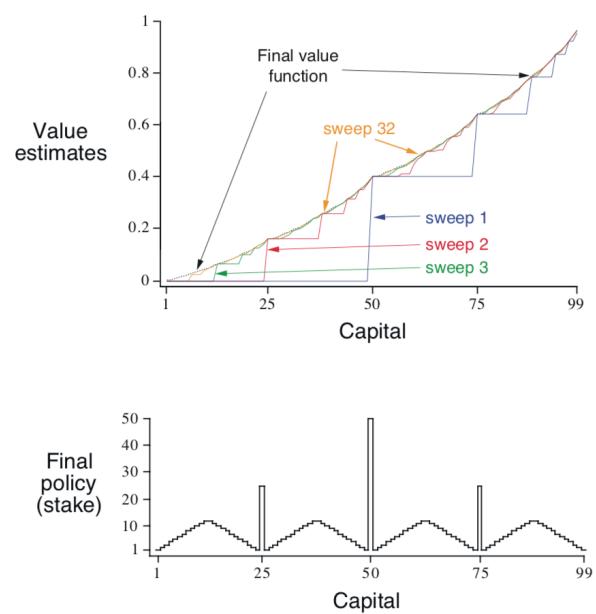


图 4.3: $ph = 0.4$ 时赌徒问题的解。上图显示了通过值迭代的连续扫描找到的值函数。下图显示了最终策略。

面的概率。如果已知 ph , 则整个问题

也已知, 并且可以通过例如值迭代来解决。图4.3显示了在 $ph = 0.4$ 的情况下, 值函数在连续的值迭代扫描中的值函数变化, 以及找到的最终策略。此策略是最优策略, 但并不是唯一的。实际上, 存在一整套最优策略, 所有策略都对应于最优化函数的 argmax 动作选择的约束。你能猜出这整套长什么样吗? ■

- ☞ **练习 4.8** 为什么赌徒问题的最优策略会有如此奇怪的形式? 特别是, 对于 50 的资本, 它将全部的赌注都押注在一个翻转上, 而对于 51 的资本, 则并非如此。为什么这是个好策略? □
- ☞ **练习 4.9 (programming)** 为赌徒的问题实施值迭代, 并针对 $ph = 0.25$ 和 $ph = 0.55$ 进行求解。在编程中, 您可能会发现引入与终止相对应的两个虚拟状态会很方便, 其资本分别为 0 和 100, 分别赋予它们 0 和 1 的值。以图形方式显示结果, 如图4.3所示。当 $\theta \rightarrow 0$ 时, 您的结果稳定吗? □
- ☞ **练习 4.10** 什么是动作值 $q_{k+1}(s, a)$ 的值迭代更新 (4.10) 的类似物? □

4.5 异步动态规划

到目前为止, 我们讨论的 DP 方法的主要缺点是它们涉及对 MDP 的整个状态集的操作, 也就是说, 它们需要对状态集进行扫描。如果状态集很大, 那么即使一次扫描也可能非常昂贵。例如, backgammon 游戏拥有 10^{20} 多个状态。即使我们可以每秒对一百万个状态执行值迭代更新, 也需要一千多年来完成一次扫描。

异步 DP 算法是 in-place 进行迭代的 DP 算法, 它们没有按照状态集的系统扫描来组织。这些算法使用碰巧可用的其他状态值, 以任何顺序更新状态值。某些状态的值可能在其他状态的值更新一次之前更新几次。但是, 要正确收敛, 异步算法必须继续更新所有状态的值: 它不能忽略计算中某个点之后的任何状态。异步 DP 算法在选择要更新的状态时具有极大的灵活性。

例如, 异步值迭代的一种版本使用值迭代更新 (4.10) 在每个步骤 k 上仅更新一个状态 s_k 的值。如果 $0 < \gamma < 1$, 则仅在所有状态在序列 $\{s_k\}$ 中出现无限次数 (该序列甚至可能是随机的) 的情况下, 才能保证 v_* 的渐近收敛 (在无折扣的情况下, 可能有一些更新顺序不会导致收敛, 但是避免这些相对容易)。同样, 可以将策略评估和值迭代更新混合在一起以产生一种异步截断的策略迭代。尽管这种和其他不常见的 DP 算法的细节不在本书的讨论范围之内, 但是很显然, 一些不同的更新构成了构建模块, 可以在各种各样的无扫描 DP 算法中灵活使用。

当然, 避免扫描并不一定意味着我们可以用更少的计算就摆脱困境。这仅意味着算法无需在改进策略之前就陷入任何无望的漫长扫描中。我们可以通过选择应用更新的状态来尝试利用这种灵活性, 以提高算法的进度。我们可以尝试对更新进行排序, 以使价值信息以一种有效的方式从一个状态传播到另一个状态。一些状态可能不需要像其他状态那样频繁地更新其值。如果某些状态与最优行为无关, 我们甚至可以尝试完全跳过某些状态的更新。在第 8 章中讨论了一些这样做的想法。

异步算法还使将混合计算与实时交互变得更加容易。为了解决给定的 MDP，我们可以在 agent 实际经历 MDP 的同时运行迭代 DP 算法。agent 的经验可用于确定 DP 算法对其应用更新的状态。同时，来自 DP 算法的最新价值和策略信息可以指导 agent 的决策。例如，当 agent 访问状态时对状态进行更新。这样就可以将 DP 算法的更新集中在状态集与 agent 最相关的部分上。这种重点是强化学习中反复出现的主题。

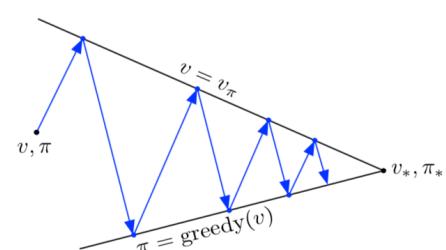
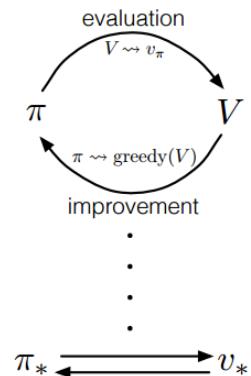
4.6 广义策略迭代

策略迭代由两个同时进行的交互过程组成，一个使值函数与当前策略一致（策略评估），另一个使策略对当前值函数变得贪婪（策略改进）。在策略迭代中，这两个过程交替进行，每个过程都在另一个过程开始之前完成，但这并不是必需的。例如，在值迭代中，每次策略改进之间仅执行一次策略评估迭代。在异步 DP 方法中，评估和改进过程以更细的粒度交织在一起。在某些情况下，在返回到另一个过程之前，会在一个过程中更新单个状态。。只要这两个过程继续更新所有状态，最终结果通常是相同的，即收敛到最优值函数和最优策略。

我们使用广义策略迭代（generalized policy iteration，GPI）一词来表示使策略评估和策略改进过程相互交互的一般思想，而与这两个过程的粒度和其他细节无关。几乎所有的强化学习方法都被很好地描述为 GPI。就是说，所有都有可识别的策略和值函数，如右图所示，策略总是相对于值函数不断改进，而值函数总是朝着策略的值函数发展。如果评估过程和改进过程都稳定下来，即不再产生变化，那么值函数和策略必须是最优的。值函数仅在与当前策略一致时才稳定，而策略只有在相对于当前值函数贪婪时才稳定，因此，这两个过程都只有在找到相对于其自身评估函数贪婪的策略时才稳定。这意味着 Bellman 最优方程（4.1）成立，因此策略和值函数是最优的。

GPI 中的评估和改进过程可以被视为竞争和合作。他们竞争的方向是相反的方向。对于值函数使策略贪婪通常会使值函数对于更改后的策略不正确，而使值函数与策略一致通常会使该策略不再贪婪。但是，从长远来看，这两个过程相互作用可以找到一个共同解：最优值函数和最优策略。

人们还可以根据两个约束或目标来思考 GPI 中评估和改进过程之间的相互作用，例如，如右图所示，是二维空间中的两条线。尽管实际几何要比这复杂得多，但该图显示了在实际情况下发生的事情。每个过程都将值函数或策略推向代表两个目标之一的解的线条上。由于两条线不正交，所以目标相互影响。直接冲向一个目标会导致远离另一个目标的移动。但是，不可避免地，联合过程更接近于



优化的总体目标。该图中的箭头与策略迭代的行为相对应，因为每个箭头都使系统完全实现两个目标中的一个。在 GPI 中，也可以针对每个目标采取较小、不完整的步骤。无论哪种情况，这两个过程都会共同实现最优的总体目标，尽管它们都没有尝试直接实现这一目标。

4.7 动态规划效率

DP 可能不适用于非常大的问题，但与其他解决 MDP 的方法相比，DP 方法实际上是相当有效的。如果我们忽略一些技术细节，那么 DP 方法发现最优策略所花费的时间（最坏情况）就是状态和动作的多项式。如果 n 和 k 表示状态和动作的数量，则意味着 DP 方法采用的计算运算法量小于 n 和 k 的多项式函数。即使（确定性）策略的总数为 k^n ，也可以确保 DP 方法在多项式时间内找到最优策略。从这个意义上讲，DP 比在策略空间中进行任何直接搜索快得多，因为直接搜索将必须穷尽地检查每个策略以提供相同的保证。线性规划方法也可以用于求解 MDP，在某些情况下，它们的最坏情况下的收敛保证比 DP 方法更好。但是，线性规划方法在状态数量远少于 DP 方法时变得不切实际（大约 100 倍）。对于最大的问题，只有 DP 方法可行。

由于维度诅咒，有时认为 DP 的适用性有限，事实上，状态数通常随状态变量的数量呈指数增长。大状态集确实会造成困难，但这是问题的固有困难，而不是作为解决方法的 DP。实际上，与直接搜索和线性规划等竞争方法相比，DP 相对更适合处理大型状态空间。

实际上，DP 方法可以与当今的计算机一起使用，以解决具有数百万个状态的 MDP。策略迭代和值迭代都被广泛使用，并且尚不清楚哪一个总体上更好。在实践中，这些方法通常比其理论上最坏的情况下运行时间快得多，特别是如果它们以良好的初始值函数或策略开始时。

对于状态空间较大的问题，通常首选异步 DP 方法。要完成一个同步方法的一次扫描，就需要针对每个状态进行计算和存储。对于某些问题，即使这么多的内存和计算也是不切实际的，但是该问题仍然可以解决，因为沿着最优解轨迹出现的状态相对较少。在这种情况下，可以使用异步方法和 GPI 的其他变体，并且与同步方法相比，可以更快地找到好的或最优策略。

4.8 总结

在本章中，我们已经熟悉了动态规划的基本思想和算法，因为它们与求解有限的 MDP 有关。策略评估是指（通常）对给定策略的值函数进行迭代计算。策略改进是指在给定策略的值函数的情况下计算改进的策略。将这两个计算放在一起，我们可以获得两种最流行的 DP 方法：策略迭代和值迭代。在完全已知 MDP 的情况下，这些方法均可用于可靠地计算有限 MDP 的最优策略和值函数。

经典的 DP 方法对状态集进行扫描，对每个状态执行期望的更新操作。每个此类操作都会根据所有可能的后继状态的值及其发生的概率来更新一个状态的值。期望的更新

与 Bellman 方程密切相关：它们只是将这些方程转换为赋值语句而已。当更新不再导致值发生任何变化时，就会发生收敛到满足相应的 Bellman 方程的值。正如有四个主值函数 ($v_{pi}, v_*, q_\pi(s, a), q_*(s, a)$) 一样，有四个对应的 Bellman 方程和四个对应的期望更新。DP 更新操作的直观视图由其备份图给出。

通过将 DP 方法视为广义策略迭代 (GPI)，可以深入了解 DP 方法以及实际上几乎所有的强化学习方法。GPI 是围绕一个近似策略和一个近似值函数进行两个交互过程的总体思想。一个过程采用给定的策略并执行某种形式的策略评估，将值函数更改为更类似于该策略的真实值函数。另一个过程采用给定的值函数并执行某种形式的策略改进，并假设该值函数为其值函数，然后对其进行更改以使其更好。尽管每个过程都改变了另一个过程的基础，但总的来说，它们一起工作可以找到一个共同的解：一个策略和值函数，该函数不会因任何一个过程而改变，因此是最优的。在某些情况下，可以证明 GPI 收敛，尤其是对于本章介绍的经典 DP 方法而言。在其他情况下，收敛性还没有得到证明，但 GPI 的思想仍然提高了我们对这些方法的理解。

不必在状态集的完整扫描中执行 DP 方法。异步 DP 方法是 *in-place* 迭代方法，它们以任意顺序更新状态，可能是随机确定的，并且使用了过时的信息。这些方法中的许多方法都可以看作是 GPI 的细粒度形式。

最后，我们注意到 DP 方法的最后一个特殊属性。它们都基于后继状态值的估计来更新状态值的估计。也就是说，他们根据其他估算值更新估算值。我们称这种一般想法为自举 (bootstrapping)。许多强化学习方法可以执行自举，甚至不需要 DP 所要求的完整且准确的环境模型。在下一章中，我们将探索不需要模型且无需自举的强化学习方法。在下下一章中，我们探索不需要模型但需要自举的方法。这些关键特征和属性是可分离的，但可以以有趣的组合进行混合。

4.9 书目与历史评论

“动态规划”一词由 Bellman (1957a) 提出，他展示了如何将这些方法应用于广泛的问题。关于 DP 的广泛处理可以在许多文本中找到，包括 Bertsekas (2005, 2012), Bertsekas 和 Tsitsiklis (1996), Dreyfus 和 Law (1977), Ross (1983), White (1969) 和 Whittle (1982, 1983)。我们对 DP 的兴趣仅限于解决 MDP 的用途，但 DP 也适用于其他类型的问题。Kumar 和 Kanal (1988) 对 DP 进行了更一般的介绍。

据我们所知，DP 和强化学习之间的第一个联系是 Minsky (1961) 在评论 Samuel 的跳棋玩家时建立的。Minsky 在一个脚注中提到，可以将 DP 应用于可以以封闭式分析形式处理 Samuel 的备份过程的问题。这句话可能使人工智能研究人员误以为 DP 仅限于分析上容易解决的问题，因此与人工智能没有太大关系。Andreae (1969b) 在强化学习的上下文中提到了 DP，特别是策略迭代，尽管他没有在 DP 和学习算法之间建立特定的联系。Werbos (1977) 提出了一种近似 DP 的方法，称为“启发式动态规划”，该方法强调了连续状态问题的梯度下降方法 (Werbos, 1982, 1987, 1988, 1989, 1992)。这些方法与我们在本书中讨论的强化学习算法密切相关。Watkins (1989) 明确地将强化学习与 DP 联系

起来，将强化学习方法的一类特征称为“增量动态规划”。

4.1-4.4 这些部分描述了已建立的 DP 算法，这些算法在上面引用的任何常规 DP 参考文献中都涉及。策略改进定理和策略迭代算法归因于 Bellman (1957a) 和 Howard (1960)。Watkins (1989) 对策略改进的局部看法影响了我们的陈述。我们将值迭代作为截断的策略迭代的一种形式的讨论是基于 Puterman 和 Shin (1978) 的方法，他们提出了一类称为修正策略迭代的算法，其中包括策略迭代和值迭代作为特例。Bertsekas (1987) 给出了一个分析，说明了如何在有限的时间内进行值迭代以找到最优策略。

迭代策略评估是用于求解线性方程组的经典逐次逼近算法的一个示例。该算法使用两个数组，其中一个保留旧值，而另一个被更新，通常称为 Jacobi 风格算法，这是因为 Jacobi 使用了这种经典方法。有时也称为同步算法，因为效果就像所有值都在同一时间更新一样。需要第二个数组来顺序模拟此并行计算。在就求解线性方程组的经典 Gauss-Seidel 算法之后，该算法的 in-place 版本通常称为 Gauss-Seidel 风格算法。除了迭代策略评估之外，其他 DP 算法也可以在这些不同的版本中实现。Bertsekas 和 Tsitsiklis (1989) 很好地介绍了这些变化及其性能差异。

4.5 异步 DP 算法归功于 Bertsekas (1982, 1983)，他们也称它们为分布式 DP 算法。异步 DP 的最初动机是其在多处理器系统上的实现，该系统具有处理器之间的通信延迟并且没有全局同步时钟。Bertsekas 和 Tsitsiklis (1989) 对这些算法进行了广泛的讨论。Jacobi 风格和 Gauss-Seidel 风格的 DP 算法是异步版本的特殊情况。Williams 和 Baird (1990) 提出了与我们之前讨论的算法相比更精细的 DP 算法：更新操作本身分为可以异步执行的步骤。

4.7 本部分是在 Michael Littman 的帮助下编写的，该部分基于 Littman, Dean 和 Kaelbling (1995)。短语“维数的诅咒”归因于 Bellman (1957a)。

Daniela de Farias (de Farias, 2002; de Farias and Van Roy, 2003) 完成了关于强化学习的线性规划方法的基础性工作。

第五章 蒙特卡洛方法

在本章中，我们将考虑我们的第一种学习方法，用于估计值函数和发现最优策略。与上一章不同，这里我们不假定您对环境有完整的了解。蒙特卡洛方法仅需要经验，即与环境的实际或模拟交互中的状态、动作和奖励的样本序列。从实际经验中学习是引人注目的，因为它不需要环境动态的先验知识，但仍然可以获得最优行为。从模拟经验中学习也很有效。尽管需要模型，但是该模型仅需要生成样本转转移，而无需生成动态规划(DP) 所需的所有可能转移的完整概率分布。令人惊讶的是，在许多情况下，很容易生成根据期望概率分布采样的经验，但无法以显式形式获得分布。

蒙特卡洛方法是基于平均样本回报来解决强化学习问题的方法。为了确保有明确定义的回报可用，这里我们仅将蒙特卡洛方法定义为回合任务。也就是说，我们假设经验被分为几个回合，并且无论选择什么动作，所有回合最终都会终止。仅在一个回合完成时，值估计和策略才会更改。蒙特卡洛方法因此可以在逐回合意义上递增，但不能在逐步(在线)意义上递增。术语“蒙特卡洛”通常更广泛地用于其运算涉及大量随机成分的任何估计方法。在这里，我们专门将其用于基于完全回报平均的方法(与下一章中讨论的从部分回报中学习的方法相反)。

蒙特卡洛方法对每个状态-动作对的抽样和平均回报非常类似于我们在第2章中探讨的赌博机方法对每个动作的采样和平均奖励。主要区别在于，现在有多个状态，每个动作都像一个不同的赌博机问题(例如，关联搜索或上下文赌博机)，而不同的赌博机问题是相互关联的。即，在一种状态下采取动作后的回报取决于同一回合中在较后状态下采取的动作。因为所有动作选择都在学习中，所以从较早状态的角度来看，问题变得不平稳。

为了处理非平稳性，我们采用了第4章为DP开发的广义策略迭代(GPI)的思想。尽管之前我们根据MDP的知识来计算值函数，但这里我们从MDP的样本回报中学习值函数。值函数和相应的策略仍然相互作用，以基本相同的方式(GPI)达到最优。与DP章节中一样，首先我们考虑预测问题(对固定的任意策略 π 的 $v_{\pi i}$ 和 q_{π} 的计算)，然后是策略改进，最后是GPI控制问题及其解决方案。从DP中获得的所有这些想法都扩展到了仅提供样本经验的蒙特卡洛的情况。

5.1 蒙特卡洛预测

我们首先考虑蒙特卡洛方法，以学习给定策略的状态值函数。回想一下，一个状态的价值就是从该状态开始的期望回报，即期望的累计未来折扣奖励。因此，一种根据经验进行估算的明显方法就是将访问该状态后观察到的回报平均化。随着观察到更多的回报，平均值应该收敛到期望值。这一思想是所有蒙特卡洛方法的基础。

尤其是，假设给定一组跟随 π 并经过 s 的回合，假设我们希望估计 $v_{\pi}(s)$ ，即策略 π

下状态 s 的值。回合中状态 s 的每次出现都称为 s 的访问。当然，在同一回合中可能会多次访问 s ；让我们将其在回合中的首次访问称为 s 的首次访问。首次访问 MC 方法将 $v_\pi(s)$ 估计为第一次访问 s 之后的平均回报，而每次访问 MC 方法将其估计为所有访问 s 之后的平均回报。在访问 s 之后返回。这两种蒙特卡洛（MC）方法非常相似，但理论特性略有不同。首次访问 MC 的研究可以追溯到 20 世纪 40 年代，是最广泛的研究对象，也是我们在本章中重点介绍的对象。如第 9 章和第 12 章中所述，每次访问 MC 都更自然地扩展为函数近似和资格迹。每次访问 MC 与首次访问 MC 都是一样的，除了没有在回合的早些时候对 S_t 进行检查。

First-visit MC prediction, for estimating $V \approx v_\pi$

输入：要评估的策略 π

初始化：

对于所有 $s \in \mathcal{S}$, 任意选取 $V(s) \in \mathbb{R}$

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

永久循环（每回合）：

生成一个跟随 π 的回合： $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

为回合的每一步循环， $t = T-1, T-2, \dots, 0$:

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

除非 S_t 出现在 S_0, S_1, \dots, S_{t-1} :

附加 G 到 $Returns(S_t)$

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

当访问（或首次访问） s 的次数趋于无穷时，首次访问 MC 和每次访问 MC 都收敛到 $v_\pi(s)$ 。对于首次访问 MC 的情况很容易看到。在这种情况下，每个回报都是具有有限方差的 $v_\pi(s)$ 的独立同分布的估计。根据大数定律，这些估计的平均值序列收敛到其期望值。每个平均值本身就是一个无偏估计，其误差的标准偏差为 $1/\sqrt{n}$ ，其中， n 是平均回报的数量。每次访问 MC 不那么直接，但是它的估计也二次收敛到 $v_\pi(s)$ (Singh and Sutton, 1996)。

通过一个例子可以最好地说明蒙特卡洛方法的使用。

例 5.1 : Blackjack 21 点这种流行的赌场纸牌游戏的目标是获得数字总和尽可能大而不会超过 21 的纸牌。所有面牌为 10，一张王牌可以为 1 或 11。我们考虑的是每个玩家与发牌者独立竞争的版本。游戏开始时发给发牌者和玩家两张牌。发牌者的一张牌面朝上，另一张牌面朝下。如果玩家立即有 21（一张王牌和一张 10），则称为自然。然后，他将获胜，除非发牌者也具有自然，在这种情况下，游戏是平局。如果玩家没有自然，那么他可以要求额外的牌，一张接一张（命中），直到他停止（坚持）或超过 21（破产）为止。如果他破产，他就会输；如果他坚持，那么该轮到发牌者。发牌者按照固定策略没有选

择地命中或坚持：他坚持任何 17 或更大的总和，否则进行命中。如果玩家破产，则玩家获胜；否则，结果（赢，输还是平局）取决于谁的最终总和接近 21。

玩 21 点自然地被定义为回合有限的 MDP。21 点的每场游戏都是一个回合。 $+1$, -1 和 0 的奖励分别用于赢，输和平局。游戏中的所有奖励均为零，我们没有折扣 ($\gamma = 1$)；因此，这些最终奖励也是回报。玩家的动作是命中或坚持。状态取决于玩家的纸牌和发牌人的显示牌。我们假设纸牌是从无限副牌中发出来的（即更换），因此跟踪已经发出的牌没有任何优势。如果玩家拥有一张可以算作 11 而不会破产的王牌，那么该王牌就可以使用。在这种情况下，始终将其计为 11，因为将其计为 1 会使总和等于 11 或更小，在这种情况下，没有什么可做的决定，因为很明显，玩家应该总是命中。因此，玩家根据三个变量做出决定：他的当前总和 (12-21)，发牌者的一张显示牌（王牌-10），以及他是否持有可用的王牌。这使得总共有 200 个状态。

考虑这样的策略，如果玩家的总和是 20 或 21，就会坚持，否则就会命中。为了通过蒙特卡洛方法找到该策略的状态值函数，可以使用该策略模拟许多场 21 点游戏，并对每个状态后的回报进行平均。这样，我们获得了图 5.1 所示的状态值函数的估计值。具有可用王牌的状态的估计不确定性和规律性较低，因为这些状态不那么常见。无论如何，在进行了 500,000 场游戏之后，值函数能被很好近似。 ■

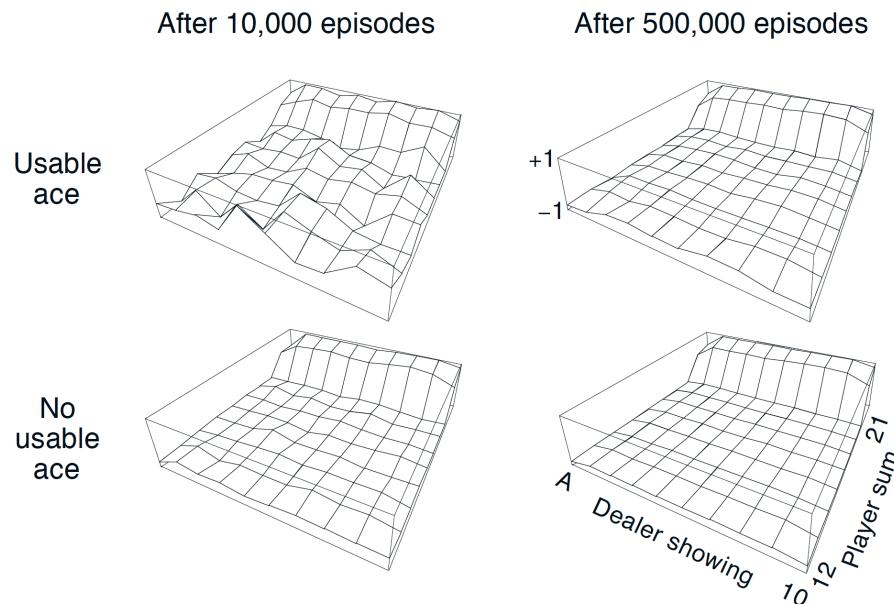


图 5.1：仅在 20 或 21 上坚持的 21 点策略的近似状态值函数，其由蒙特卡洛策略评估计算得出。

- ✍ 练习 5.1 考虑图 5.1 右侧的图。为什么估计值函数在后面的最后两行跳起来？为什么它在左侧的整个最后一行掉落？为什么在上方的图中最前面的值比在下方的图中高？ □
- ✍ 练习 5.2 假设在 21 点任务中使用了每次访问 MC 而不是首次访问 MC。您期望结果会有很大不同吗？为什么或者为什么不？ □

尽管我们在 21 点任务中对环境有完整的了解，但应用 DP 方法计算值函数并不容易。DP 方法需要下一个事件的分布，尤其是，它们需要四参数函数 p 给出的环境动态，对于 21 点而言，要确定这一点不容易。例如，假设玩家的总和为 14，而他选择坚持。根据发牌者的出示牌，他以 $+1$ 奖励结束的可能性是多少？必须先计算所有概率，然后才能应

用 DP，并且此类计算通常很复杂且容易出错。相反，生成蒙特卡洛方法所需的游戏样本很容易。令人惊讶的是经常是这种情况；即使完全了解环境动态，蒙特卡洛方法单独处理样本回合的能力也可以带来显着优势。

我们可以将备份图的思想推广到蒙特卡洛算法吗？备份图的总体思路是在顶部显示要更新的根节点，并在其下方显示其奖励和估计值对更新有贡献的所有转移和叶节点。对于 v_π 的蒙特卡洛估计，根是一个状态节点，在它的下方是沿着特定单个回合的转移的整个轨迹，在终点状态结束，如右图所示。DP 图（第 59 页）显示了所有可能的转移，而蒙特卡洛图仅显示了在一个回合中采样的转移。DP 图仅包含单步转移，而蒙特卡洛图一直到回合结束。图中的这些差异准确地反映了算法之间的基本差异。

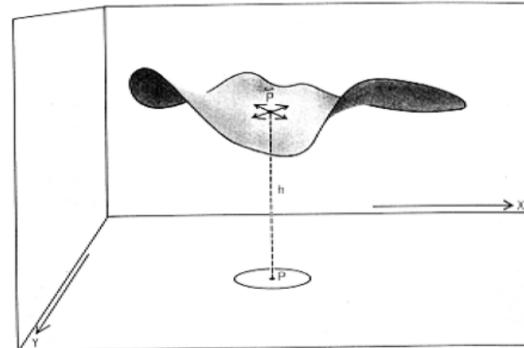
关于蒙特卡洛方法的一个重要事实是，每个状态的估计都是独立的。一个状态的估算不基于对任何其他状态的估算，像 DP 中一样。换句话说，蒙特卡洛方法不像我们在上一章中定义的那样自举。



尤其要注意，估计单个状态值的计算开销与状态数无关。当仅需要一个状态或状态子集的值时，这使得蒙特卡洛方法特别有吸引力。人们可以从感兴趣的状态开始生成许多样本回合，仅对这些状态的回报进行平均，而忽略所有其他状态。这是蒙特卡洛方法相对于 DP 方法所具有的第三个优势（能够从实际经验和模拟经验中学习）。

例 5.2 : Soap Bubble

假设将形成闭环的线框浸泡在肥皂水中，以形成在其边缘与线框相吻合的肥皂表面或气泡。如果线框的几何形状不规则但已知，则如何计算表面的形状？该形状具有以下特性：相邻点在每个点上施加的总力为零（否则形状会改变）。这意味着该表面在任意点的高度是该点周围一个小圆圈中各点的平均高度。另外，表面必须在其与线框的边界处相交。解决这类问题的常用方法是在表面覆盖的区域上放置一个网格，并通过迭代计算来求解其在网格点处的高度。边界处的网格点被强制连接到线框，所有其他网格点均朝着其四个最近邻点的平均高度进行调整。然后，此过程将进行迭代，就像 DP 的迭代策略评估一样，最终收敛到期望表面的近似值。



线框上的肥皂泡。摘自 Hersh 和 Griego (1969)，经许可复制，©1969 Scientific American, a division of Nature America, Inc. All rights reserved.

这类似于最初设计蒙特卡洛方法的问题。代替上面描述的迭代计算，想象一下站在地面上并随机行走，以相等的概率从网格点随机地移动到相邻的网格点，直到到达边界。事实证明，边界处的高度的期望值非常接近起点处所需曲面的高度（实际上，它是通过上述迭代方法计算得出的值）。因此，可以通过简单地平均从该点开始的多条行走的边界高度，来近似估计该点的表面高度。如果只对一个点或一组固定的点上的值感兴趣，则

此蒙特卡罗方法可能比基于局部一致性的迭代方法更有效。 ■

5.2 动作值的蒙特卡洛估计

如果没有模型，则估计动作值（状态-动作对的值）而不是状态值特别有用。使用模型时，仅凭状态值就足以确定策略；就像我们在 DP 章节中所做的那样，只是简单地向前一步，并选择了导致奖励和下一个状态的最佳组合的任何动作。但是，如果没有模型，仅凭状态值是不够的。必须明确估计每个动作的值，以便这些值在建议策略时很有用。因此，蒙特卡罗方法的主要目标之一就是估计 q_* 。为此，我们首先考虑动作值的策略评估问题。

动作值的策略评估问题是估计 $q_\pi(s, a)$ ，即从状态 s 开始，采取动作 a 以及随后遵循策略 π 的期望回报。为此，这方面的蒙特卡洛方法与上述关于状态值的方法基本相同，除了现在我们谈论的是访问状态-动作对而不是访问状态。一个状态-动作对 s, a 称为在一个回合中被访问过，如果状态 s 被访问过并且在其中采取了动作 a 。每次访问 MC 方法都将状态-动作对的值估计为所有访问之后的平均回报。首次访问 MC 方法对每个回合中第一次访问状态和选择动作之后的回报进行平均。随着对每个状态-动作对的访问次数接近无穷，这些方法像以前一样二次收敛到真实的期望值。

唯一的麻烦是，许多状态-动作对可能永远不会被访问。如果 π 是确定性策略，则在跟随的 π 后，仅观察到来自每个状态的一个动作的回报。由于没有回报来平均，蒙特卡洛对其他动作的估计将不会随着经验的提高而提高。这是一个严重的问题，因为学习动作值的目的是帮助选择每个状态下可用的动作。为了比较替代方案，我们需要估算每个状态的所有动作的价值，而不仅仅是目前所支持的动作。

这是保持探索的一般性问题，如第2章中的 k -臂赌博机问题所讨论的那样。要使策略评估适用于动作值，我们必须确保持续探索。一种实现方法是，指定回合以某个状态-动作对开始，并且每对都有被选择作为开始的非零概率。这保证了在无数次限制的回合情况下，可以无限次地访问所有状态-动作对。我们称此为探索起点的假设。

探索起点的假设有时是有用的，但当然不能总地依靠它，特别是当直接从与环境的实际交互中学习时。在这种情况下，起点条件不太可能有帮助。确保遇到所有状态-动作对的最常见替代方法是仅考虑随机策略，在每个状态中选择所有动作的概率不为零。我们将在后面的部分中讨论此方法的两个重要变体。目前，我们保留了探索起点的假设并完成了完整的蒙特卡洛控制方法的介绍。

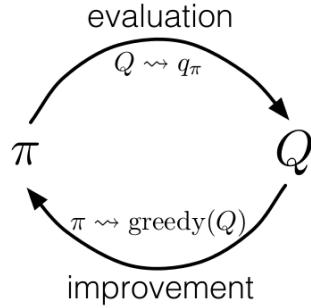
练习 5.3 蒙特卡洛估计 q_π 的备份图是什么？



5.3 蒙特卡洛控制

现在我们准备考虑如何将蒙特卡洛估计用于控制，即近似最优策略。总体思路是按照与 DP 章节中相同的模式进行，即按照广义策略迭代（GPI）的思路。在 GPI 中，既维护近似策略又保留近似值函数。如右图所示，反复更改值函数以更接近当前策略的值函

数，并相对于当前值函数反复改进策略。这两种变化在某种程度上彼此对立，因为每种变化为彼此创建了一个移动的目标，但同时使策略和价值函数都趋于最优。



首先，让我们考虑一下经典策略迭代的蒙特卡洛版本。在这种方法中，我们执行策略评估和策略改进的交替完整步骤，从任意策略 π_0 开始，到最优策略和最优动作值函数结束：

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

其中， \xrightarrow{E} 表示完整的策略评估， \xrightarrow{I} 表示完整的策略改进。完全按照上一部分中的描述进行策略评估。经历了许多回合，近似的动作值函数渐近地接近真实函数。目前，让我们假设我们确实观察到了无限多的回合，此外，这些回合是从探索起点产生的。在这些假设下，对于任意 π_k ，蒙特卡洛方法将精确计算每个 q_{π_k} 。

通过使针对当前值函数的策略贪婪来完成策略的改进。在这种情况下，我们有一个动作值函数，因此不需要模型来构建贪婪策略。对于任何动作值函数 q ，对应的贪婪策略是针对每个 $s \in \mathcal{S}$ 确定性地选择具有最大动作值的动作：

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

然后可以通过将每个 π_{k+1} 构造为关于 q_{π_k} 的贪婪策略来进行策略改进。然后，策略改进定理（第4.2节）适用于 π_k 和 π_{k+1} ，因为对于所有 $s \in \mathcal{S}$ ，

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

正如我们在上一章中讨论的那样，该定理向我们保证，每个 π_{k+1} 都统一优于 π_k ，或与 π_k 一样好，在这种情况下，它们都是最优策略。这又向我们保证，整个过程会收敛到最优策略和最优值函数。这样，仅给定采样回合，而没有其他有关环境动态的知识，就可以使用蒙特卡洛方法来找到最优策略。

上面我们做了两个不太可能的假设，以便轻松获得蒙特卡洛方法的收敛性保证。一

个是回合有探索起点，另一个是可以用无限次多的回合来进行策略评估。为了获得实用的算法，我们必须删除这两个假设。我们对第一个假设的考虑推迟到本章稍后。

现在，我们集中在这样一种假设，即政策评估是在无数次回合中进行的。这个假设相对容易消除。实际上，即使在经典的 DP 方法（例如迭代策略评估）中也出现了同样的问题，这些方法也仅渐近地收敛到真实值函数。在 DP 和蒙特卡洛情况下，都有两种方法可以解决该问题。一种方法是在每次政策评估中坚持近似于 q_{π_k} 的想法。进行测量和假设以获取估计值中错误的大小和概率的界限，然后在每次策略评估期间采取足够的步骤以确保这些界限足够小。从保证正确收敛到某种程度的近似的意义上，可以使这种方法完全令人满意。但是，它也可能需要很多的回合才能在实践中用于最小问题以外的任何问题。

还有第二种方法可以避免策略评估名义上所需的无限数量的回合，在这种情况下，我们放弃尝试在返回策略改进之前完成策略评估的尝试。在每个评估步骤中，我们将值函数移向 q_{π_k} ，但是除了许多步骤之外，我们并不期望实际接近。在第4.6节中首次引入 GPI 的概念时，便使用了这个想法。这种想法的一种极端形式值迭代，其中在策略改进的每个步骤之间仅执行一次迭代策略评估的迭代。值迭代的 in-place 版本更加极端。在那里，我们在单个状态的改进和评估步骤之间交替进行。

对于蒙特卡洛策略迭代，很自然地在逐回合的基础上在评估和改进之间交替。在每个回合之后，将观察到的回报用于策略评估，然后在该回合中访问的所有状态进行策略改进。在下一页的框中，使用伪代码给出了遵循这些思路的完整简单算法，我们将其称为蒙特卡洛探索起点，即蒙特卡洛 ES。

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

初始化：

- 对于所有 $s \in \mathcal{S}$, 任意选取 $\pi(s) \in \mathcal{A}(s)$
- 对于所有 $s \in \mathcal{S}, a \in \mathcal{A}(s)$, 任意选取 $Q(s, a) \in \mathbb{R}$
- 对于所有 $s \in \mathcal{S}, a \in \mathcal{A}(s)$, $Returns(s, a) \leftarrow$ 空列表

永久循环（对于每个回合）：

- 随机选择 $S_0 \in \mathcal{S}, A_0 \in \mathcal{A}(S_0)$, 使得所有对都具有大于 0 的概率
- 从 S_0, A_0 跟随 π 生成一个回合： $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$
- $G \leftarrow 0$

对于回合的每一步循环， $t = T-1, T-2, \dots, 0$:

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

除非 S_t, A_t 出现在 $S_0, A_0, S_1, A_1, \dots, S_{t-1}, A_{t-1}$:

附加 G 到 $Returns(S_t, A_t)$

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

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

练习 5.4 蒙特卡洛 ES 的伪代码是无效的，因为对于每个状态-动作对，它都会维护所有回报的列表，并反复计算其均值。使用与第2.4节中介绍的技术类似的技术来使得更有效，以便仅维护均值和计数（针对每个状态-行为对）并增量地更新它们。描述如何更改伪代码以实现此目的。 \square

在蒙特卡洛 ES 中，对每个状态-动作对的所有回报进行累加和平均，而与观察到的现行策略无关。显而易见，蒙特卡洛 ES 无法收敛到任何次优策略。如果确实如此，那么值函数最终将收敛到该策略的值函数，进而导致该策略发生变化。仅当策略和值函数均最优时，才能实现稳定性。随着动作值函数的变化随着时间的推移而减少，收敛到这个最佳固定点似乎是不可避免的，但是尚未得到正式证明。我们认为，这是强化学习中最基本的开放性理论问题之一（有关部分解决方案，请参见 Tsitsiklis, 2002）。

例 5.3 : Solving Blackjack 将蒙特卡洛 ES 应用于 21 点很简单。由于这些回合都是模拟游戏，因此很容易安排包含所有可能性的探索起点。在这种情况下，只需简单地等概率随机选择发牌者的牌，玩家的总和以及玩家是否有可用的王牌即可。作为初始策略，我们使用在前面的 21 点示例中评估的策略，该策略仅在 20 或 21 上坚持。对于所有状态-动作对，初始动作值函数可以为零。图5.2显示了蒙特卡洛 ES 找到的 21 点的最优策略。该策略与 Thorp (1966) 的“基本”策略相同，唯一的例外是可用王牌的策略中最左边的凹口，这在 Thorp 的策略中不存在。我们不确定出现这种差异的原因，但确信此处显示的确实是我们所描述的 21 点版本的最优策略。

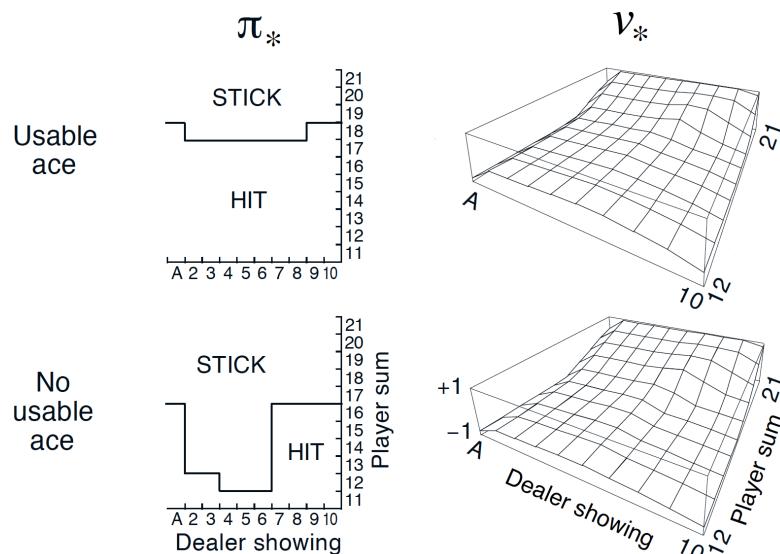


图 5.2: 蒙特卡洛 ES 发现的 21 点的最优策略和状态值函数。所示的状态值函数由蒙特卡洛 ES 找到的动作值函数计算得出。

5.4 无探索起点的蒙特卡洛控制

我们如何避免不太可能的探索起点的假设？确保无限次选择所有动作的唯一通用方法是 agent 继续选择它们。有两种方法可以确保这一点，这就是所谓的 on-policy 方法和 off-policy 方法。on-policy 方法尝试评估或改进用于决策的策略，而 off-policy 方法则评估

或改进与用于生成数据的策略不同的策略。上面开发的蒙特卡洛 ES 方法是 on-policy 方法的示例。在本节中，我们说明如何设计一种不使用不切实际的探索起点假设的 on-policy 的蒙特卡洛控制方法。下一部分将考虑 off-policy 方法。

在 on-policy 控制的方法中，策略通常是软性的，这意味着对于所有 $a \in \mathcal{S}$ 和所有 $a \in \mathcal{A}(s)$, $\pi(a|s) > 0$ ，但逐渐接近确定性最优策略。第2章讨论的许多方法都为此提供了机制。我们在本节中介绍的 on-policy 的方法使用 ε -greedy 策略，这意味着大多数情况下，他们选择具有最大估计动作值的动作，但是他们以概率 ε 随机选择一个动作。也就是说，所有非贪婪动作都被赋予了最小的选择概率 $\frac{\varepsilon}{|\mathcal{A}(s)|}$ ，而剩余的概率 $1 - \varepsilon + \frac{\varepsilon}{|\mathcal{A}(s)|}$ 被赋予给贪婪动作。 ε -greedy 策略是 ε -soft 策略的示例，定义为所有状态和动作的 $\pi(a|s) \geq \frac{\varepsilon}{|\mathcal{A}(s)|}$ 对于某些 $\varepsilon > 0$ 的策略。在所有 ε -soft 策略中， ε -greedy 是在某种意义上，其策略是最接近贪婪的策略。

on-policy 蒙特卡洛控制的总体思想仍然是 GPI 的思想。与蒙特卡洛 ES 一样，我们使用首次访问 MC 方法来估计当前策略的动作值函数。但是，如果没有探索起点的假设，我们就不能简单地通过使它相对于当前价值函数贪婪地改善策略，因为这将阻止进一步探索非贪婪动作。幸运的是，GPI 并不要求将策略一直采取贪婪策略，而只是要求将其移向贪婪策略。在我们的基于策略的方法中，我们仅将其移至 ε -greedy 策略。对于任何 ε -soft 策略 π ，任何关于 q_π 的 ε -greedy 策略都保证优于或等于 π 。完整的算法在下面的方框中给出。

On-policy first-visit MC control (for ε -soft policies), estimates $\pi \approx \pi_*$

算法参数：小的 $\varepsilon > 0$

初始化：

$\pi \leftarrow$ 一个任意的 ε -soft 策略

对于所有 $s \in \mathcal{S}, a \in \mathcal{A}(s)$, 任意选取 $Q(s, a) \in \mathbb{R}$

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

永久重复（对于每个回合）：

生成一个跟随 π 的回合： $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

对于回合的每一步循环， $t = T - 1, T - 2, \dots, 0$:

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

除非对 S_t, A_t 出现在 $S_0, A_0, S_1, A_1, \dots, S_{t-1}, A_{t-1}$:

附加 G 到 $Returns(S_t, A_t)$

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

$A^* \leftarrow \arg \max_a Q(S_t, a)$ (任意打破约束)

对于所有 $a \in \mathcal{A}(S_t)$:



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

通过策略改进定理可以确保任何关于 q_π 的 ε -greedy 策略都是对 ε -soft 策略 π 的改进。令 π' 为 ε -greedy 策略。策略改进定理的条件适用，因为对于任何 $s \in \mathcal{S}$:

$$\begin{aligned} q_\pi(s, \pi'(s)) &= \sum_a \pi'(a|s) q_\pi(s, a) \\ &= \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a) + (1 - \varepsilon) \max_a q_\pi(s, a) \\ &\geq \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a) + (1 - \varepsilon) \sum_a \frac{\pi(a|s) - \frac{\varepsilon}{|\mathcal{A}(s)|}}{1 - \varepsilon} q_\pi(s, a) \\ &\quad (\text{总和是非负权重之和为 1 的加权平均值, 因此它必须小于或等于} \\ &\quad \text{平均的最大数}) \\ &= \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a) - \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a) + \sum_a \pi(a|s) q_\pi(s, a) \\ &= v_\pi(s). \end{aligned} \tag{5.2}$$

因此，根据策略改进定理， $\pi' \geq \pi$ (即，对于所有 $s \in \mathcal{S}$, $v_{\pi'}(s) \geq v_\pi(s)$)。现在我们证明，仅当 ε -soft 策略中的 π' 和 π 均最优时，即当它们优于或等于所有其他 ε -soft 策略时，等号才能成立。

考虑一个与原始环境相似的新环境，但要求策略必须在环境中 ε -soft 移动。新环境具有与原始环境相同的动作和状态，其行为如下。如果处于状态 s 并采取动作 a ，则以概率为 $1 - \varepsilon$ 新环境的行为与旧环境完全相同。以概率为 ε 它以相等的概率随机重选动作，然后使用具有新的随机动作来表现与旧环境一样的行为。在这种具有通用策略的新环境中可以做的最好的事情与在具有 ε -soft 策略的原始环境中可以做的最好的事情一样。令 \tilde{v}_* 和 \tilde{q}_* 表示新环境的最优值函数。那么，当且仅当 $v_\pi = \tilde{v}_*$ 时，策略 π 在 ε -soft 策略中最优。根据 \tilde{v}_* 的定义，我们知道这是解决如下问题的唯一方法

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

当等号成立并且 ε -soft 不再得到改善时，那么我们也可以从 (5.2) 中知道

$$v_\pi(s) = (1 - \varepsilon) \max_a q_\pi(s, a) + \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a)$$

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

然而，这个方程式与前一个方程式相同，只是用 v_π 代替 \tilde{v}_* 。因为 \tilde{v}_* 是唯一的解，所以它必须是 $v_\pi = \tilde{v}_*$ 。

从本质上讲，我们在最后几页中显示了策略迭代可用于 ε -soft 策略。将贪婪策略的自然概念用于 ε -soft 策略，可以确保每个步骤都有改进，除非在 ε -soft 策略中找到最优策略。这种分析与在每个阶段如何确定动作值函数无关，但是它确实假定它们是精确计算的。这使我们对这一点与上一节大致相同。现在，我们仅在 ε -soft 策略中实现了最优策略，但是另一方面，我们消除了探索起点的假设。

5.5 重要性采样的 off-policy 预测

所有学习控制方法都面临一个难题：它们试图以后续的最优行为为条件来学习动作值，但是它们需要非最优行为才能探索所有动作（找到最优动作）。他们如何按照探索策略行事的同时了解最优策略？上一节中的 on-policy 的方法实际上是一种折衷方案，它不是针对最优策略而是针对仍在探索的近似最优策略来学习动作值。一种更直接的方法是使用两种策略，一种是已了解并成为最优策略的策略，另一种是更探索性的用于生成行为的策略。正在学习的策略称为目标策略，而用于生成行为的策略称为行为策略。在这种情况下，我们说学习是从目标策略的 off 数据中获取的，整个过程称为 o-policy 学习。

在本书的其余部分中，我们同时考虑 on-policy 和 off-policy 的方法。通常，on-policy 的方法比较简单，并且首先被考虑。off-policy 策略方法需要附加的概念和符号，并且由于数据是由于不同的策略所致，因此 off-policy 方法通常具有更大的方差并且收敛较慢。另一方面，off-policy 方法更强大，更通用。它们包括作为目标和行为策略相同的特殊情况的 on-policy 方法。off-policy 策略方法在应用中还具有多种其他用途。例如，它们通常可以应用于从常规的非学习控制器或人类专家生成的数据中学习。在某些人看来，off-policy 学习也被认为是学习世界动态的多步预测模型的关键（参见第 17.2 节；Sutton, 2009 年；Sutton 等人, 2011 年）。

在本节中，我们将通过考虑固定了目标和行为策略的预测问题，开始对 off-policy 的研究。也就是说，假设我们希望估计 v_π 或 q_π ，但是我们拥有的只是遵循另一策略 b 的回合，其中 $b \neq \pi$ 。在这种情况下， π 是目标策略， b 是行为策略，并且这两个策略都被固定并给出。

为了使用 b 中的回合来估计 π 的值，我们要求在 π 之下采取的每个动作也必须至少在 b 之下采取。也就是说，我们要求 $\pi(a|s) > 0$ 隐含 $b(a|s) > 0$ 。这称为覆盖率假设。从覆盖范围可以得出， b 在与 π 不同的状态下必须是随机的。另一方面，目标策略 π 可能是确定性的，实际上，这是控制应用中特别感兴趣的情况。在控制方面，相对于当前对动作值函数的估计，目标策略通常是确定性贪婪策略。该策略成为确定性的最优策略，而行为策略保持随机且更具探索性，例如 ε -greedy 策略。但是，在本节中，我们考虑 π 不

变且给定的预测问题。

几乎所有 off-policy 的方法都使用重要性采样，重要性采样是一种在给定另一种分布的样本的情况下估计一种分布下的期望值的通用技术。我们通过根据目标和行为策略下轨迹发生的相对概率对回报进行加权，将重要性采样应用于 off-policy 学习中，称为重要性采样率。给定一个起始状态 S_t ，在任何策略 π 下发生后续状态-动作轨迹 $A_t, S_{t+1}, A_{t+1}, \dots, S_T$ 的概率为

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

其中 p 是由 (3.2) 定义的状态转移概率函数。因此，目标和行为策略下轨迹的相对概率（重要性采样率）为

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

尽管轨迹概率取决于 MDP 的转移概率（通常未知），但它们在分子和分母中都相同，因此可以抵消。重要性采样率最终仅取决于两个策略和序列，而不取决于 MDP。

回想一下，我们希望估计目标策略下的期望回报（值），但是由于行为策略，我们仅有的回报 G_t 。这些回报具有错误的期望值 $\mathbb{E}[G_t | S_t = s] = v_b(z)$ ，因此无法取平均值来获得 v_π 。这就是重要性采样的来源。比率 $\rho_{t:T-1}$ 将回报转换为正确的期望值：

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

现在我们准备给出一个蒙特卡洛算法，该算法对遵循策略 b 的一批观察到的回合的回报进行平均，以估计 $v_\pi(s)$ 。在此方便地以跨回合边界增加的方式对时间步进行编号。也就是说，如果批次的第一个回合在时间 100 处于终止状态，则下一个回合在时间 $t = 101$ 处开始。这使我们能够使用时间步长编号来引用特定回合中的特定步骤。特别是，我们可以定义访问状态 s 的所有时间步长的集合，记为 $\mathfrak{T}(s)$ 。这是针对每次访问的方法；对于首次访问方法， $\mathfrak{T}(s)$ 仅包含回合中首次访问 s 的时间步长。同样，让 $T(t)$ 表示在时间 t 之后的第一次终止时间， G_t 表示从 t 之后到 $T(t)$ 的回报。则 $\{G_t\}_{t \in \mathfrak{T}(s)}$ 是与状态 s 相关的回报，而 $\{\rho_{t:T(t)-1}\}_{t \in \mathfrak{T}(s)}$ 是相应的重要采样率。要估算 $v_\pi(s)$ ，我们只需按比例缩放回报并将结果平均：

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

当以此方式将重要性采样作为简单的平均值完成时，称为普通重要性采样。

一个重要的选择是加权重要性抽样，它使用加权平均值，定义为

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

如果分母为零，则返回零。要了解这两种重要性抽样，请在观察到状态 s 的单个回报后考虑其首次访问方法的估计值。在加权平均估计中，单个回报的比率 $\rho_{t:T(t)-1}$ 在分子和分母中抵消，因此该估计值等于观察到的回报，与比率无关（假定比率为非零）。鉴于只有这种回报是观察到的，所以这是一个合理的估计，但是它的期望是 $v_b(s)$ 而不是 $v_\pi(s)$ ，并且在这种统计意义上，它是有偏差的。相比之下，普通重要性采样估计量的首次访问版本 (5.5) 总是期望中的 $v_\pi(s)$ （无偏差），但它可能是极端的。假设比率为 10，表示在目标策略下观察到的轨迹是行为策略下观察到的轨迹的十倍。在这种情况下，普通重要性抽样估计将是观察到的回报的十倍。也就是说，即使该回合的轨迹被认为可以很好地代表目标策略，但与观察到的回报相去甚远。

形式上，两种重要性采样的首次访问方法之间的差异以其偏差和方差表示。普通重要性采样是无偏的，而加权重要性采样是有偏的（尽管偏差渐近收敛到零）。另一方面，普通重要性采样的方差通常是无界的，因为比率的方差可以是无界的，而在加权估计量中，任何一次回报的最大权重都是 1。实际上，假设回报是有界的，即使比率本身的变化是无限的，加权重要性采样估计量的变化也收敛为零 (Precup, Sutton 和 Dasgupta 2001)。在实践中，加权估计通常具有明显较低的方差，因此强烈推荐使用。但是，我们不会完全放弃普通的重要性采样，因为使用本书第二部分中探讨的函数逼近可以更容易地扩展到近似方法。

普通和加权重要性采样的每次访问方法都存在偏差，不过，随着样本数量的增加，偏差也渐渐地降为零。在实践中，通常首选每次访问方法，因为它们无需跟踪已访问了哪些状态，并且因为它们更易于扩展为近似值。第 110 页的下一节提供了使用加权重要性采样的用于 off-policy 策略评估的完整的每次访问 MC 算法。

 **练习 5.5** 考虑具有单个非终止状态和单个动作的 MDP，该动作以概率 p 转移回非终止状态，并以概率 $1 - p$ 转移到终止状态。设所有转移的奖励为 +1，设 $\gamma = 1$ 。假设您观察到一个持续 10 步，回报为 10 的回合，那么非终止状态值的首次访问和每次访问估计量是什么？ □

例 5.4 : Off-policy Estimation of a Blackjack State Value 我们使用普通重要性方法和加权重要性抽样方法，从 off-policy 数据估计单个 21 点状态的值（例 5.1）。回想一下，蒙特卡洛方法的优点之一是它们可用于评估单个状态，而无需形成任何其他状态的估计。在此示例中，我们评估了发牌者显示出两点，玩家的纸牌总和为 13 且该玩家具有可用的王牌的状态（即，该玩家持有王牌和两点，或等效为三个王牌）。数据是通过在此状态下开始然后选择以相等概率随机命中或坚持（行为策略）而生成的。如例 5.1 所示，目标策略是只坚持 20 或 21。在目标策略下，此状态的值约为 -0.27726（这是通过使用目标策略分别生成一亿个回合并平均其回报来确定的）。两种 off-policy 方法都使用随机策略在 1000 个 off-policy 的回合后非常接近此值。为了确保他们做到这一点的可靠性，我们执行了 100 次独立运行，每个运行都从零的估计值开始并学习 10,000 回合。图 5.3 显示了最终的学习

曲线，每种方法的估计值的平方误差与回合次数的函数，在 100 次运行中取平均值。两种算法的误差都接近零，但是加权重要性抽样方法初始的误差就低得多，这在实践中是很典型的。

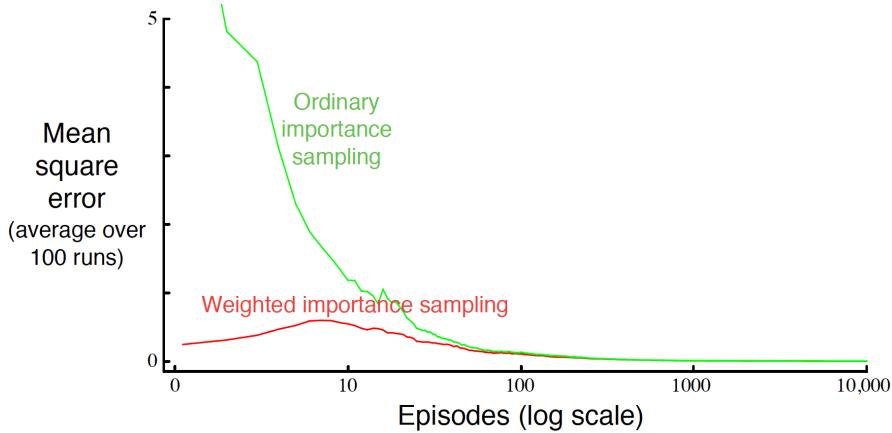


图 5.3：加权重要性采样从 off-policy 回合中对单个 21 点状态的值产生较低的误差估计。

例 5.5：Infinite Variance 普通重要性采样的估计通常会有无穷大的方差，因此当缩放后的回报具有无穷大方差时，收敛特性不会令人满意，当轨迹包含循环时，这在 off-policy 学习中很容易发生。图 5.4 显示了一个简单的示例。只有一个非终止状态 s 和两个动作，即 right 和 left。right 动作导致确定性转移到终止，而 left 动作以 0.9 的概率返回 s 或以 0.1 的概率转移到终止。在后一个转移中，奖励为 +1，否则为零。考虑始终向 left 选择的目标策略。此策略下的所有回合都包含一定数量（可能为零）的转移到 s ，然后以奖励和 +1 的回报终止。因此，目标策略下的 s 值为 1 ($\gamma = 1$)。假设我们使用行为策略从 off-policy 数据估计此值，该行为策略以相等的概率选择 right 和 left。

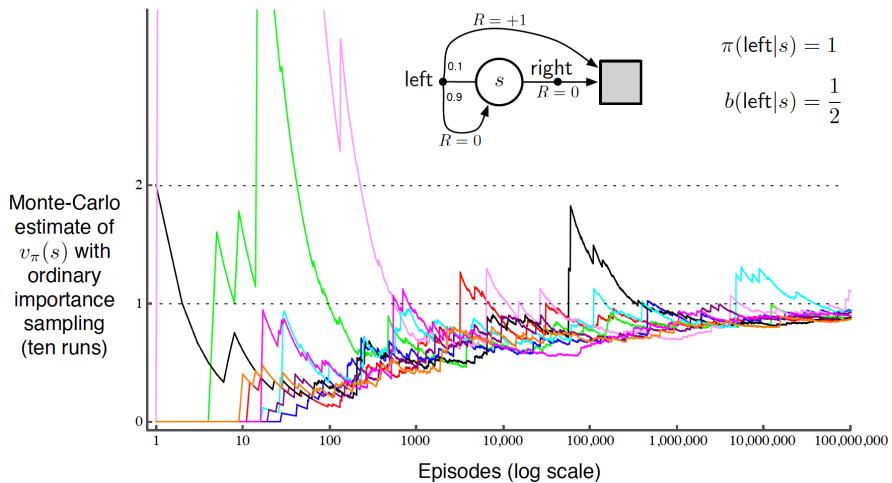


图 5.4：普通重要性采样对图中所示的单状态 MDP 产生令人惊讶的不稳定估计（例 5.5）。此处正确的估计值为 1 ($\gamma = 1$)，即使这是样本回报的期望值（在重要性抽样之后），样本的方差也是无限的，并且估计不会收敛到该值。这些结果适用于 off-policy 首次访问 MC。

图 5.4 的下半部分显示了使用普通重要性采样的 10 次首次访问 MC 算法的独立运行。即使在数百万次回合之后，估计值也无法收敛到 1 的正确值。相反，加权重要性采样算法将在以 left 动作结束的第一个回合之后永远给出精确为 1 的估计值。所有不等于 1 的

回报（即以 `right` 动作的结束）将与目标策略不一致，因此 $\rho_{t:T(t)-1}$ 为零，并且既不对分子也没有对分母（5.6）做出贡献。加权重要性采样算法仅生成与目标策略一致的回报的加权平均值，所有这些都将精确为 1。

通过简单的计算，在此示例中，我们可以验证重要性采样缩放回报的方差是无限的。任何随机变量 X 的方差是与其平均值 \bar{X} 的偏差的期望值，可以写成

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

因此，如果均值是有限的（如我们的情况），则当且仅当随机变量平方的期望是无限的时，方差才是无限的。因此，我们只需要证明重要性采样缩放回报的期望平方是无限的：

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

为了计算该期望，我们根据回合长度和终止将其分解为多种情况。首先要注意的是，对于任何以 `right` 的动作结束的回合，重要性采样率为零，因为目标策略永远不会采取这个动作；因此，这些回合对期望没有任何贡献（括号中的数量为零），可以忽略。我们只需要考虑涉及到一定数量（可能为零）的 `left` 动作的回合，这些动作会变回非终止状态，然后是一个 `left` 动作转移到终止状态。所有这些回合的回报均为 1，因此可以忽略 G_0 因子。要获得期望的平方，我们只需要考虑回合的每个长度，将回合发生的概率乘以重要性采样率的平方即可，然后将它们相加：

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

- ✍ **练习 5.6** 同样给定使用 b 生成的回报，对于动作值 $Q(s, a)$ 而不是状态值 $V(s)$ ，类似于（5.6）的方程是什么？□
- ✍ **练习 5.7** 在如图5.3所示的学习曲线中，误差通常随训练而减少，这确实是普通重要性采样方法所发生的。但是对于加权重要性采样方法，误差首先增加然后减少。您认为为什么发生这种事？□
- ✍ **练习 5.8** 例5.5的结果如图5.4所示，使用了首次访问 MC 方法。假设在同一问题上使用了每次访问 MC 方法。估计量的方差是否仍然是无限的？为什么或者为什么不？□

5.6 演增实现

可以使用第2章（第2.4节）中介绍的技术扩展，在逐回合的基础上逐步实现蒙特卡洛预测方法。在第2章中，我们平均了奖励，而在蒙特卡洛方法中，我们是平均回报。在所有其他方面，可以将与第2章完全相同的方法用于 on-policy 的蒙特卡洛方法。对于 off-policy 的蒙特卡洛方法，我们需要分别考虑那些使用普通重要性采样的方法和那些采用加权重要性采样的方法。

在普通重要性采样中，回报按重要性采样率 $\rho_{t:T(t)-1}$ (5.3) 进行缩放，然后如 (5.5) 所示简单地平均。对于这些方法，我们可以再次使用第2章的增量方法，但是使用缩放回报代替该章的奖励。剩下使用加权重要性抽样的 off-policy 方法的情况。在这里，我们必须形成回报的加权平均值，并且需要稍微不同的增量算法。

假设我们有一个回报序列 G_1, G_2, \dots, G_{n-1} ，它们都以相同的状态开始，并且每个都有对应的随机权重 W_i （例如 $W_i = \rho_{t_i:T(t_i)-1}$ ）。我们希望形成估计

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

并且在我们获得单个附加回报 G_n 的同时保持最新状态。除了跟踪 V_n 之外，我们还必须为每个状态保持第 n 次回报的权重的累加总和 C_n 。 V_n 的更新规则是

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

以及

$$C_{n+1} \doteq C_n + W_{n+1}, \quad (5.9)$$

其中 $C_0 \doteq 0$ （并且 V_1 是任意的，因此无需指定）。下一页的方框包含用于蒙特卡洛策略评估的完整的逐回合增量算法。该算法名义上适用于 off-policy 情况，使用加权重要性采样，但也适用于 on-policy 情况，只需选择相同的目标和行为策略即可（在这种情况下 ($\pi = b$)， W 总是为 1)。当根据潜在的不同策略 b 选择动作时，近似值 Q 收敛到 q_π （对于所有遇到的状态-动作对）。

Off-policy MC prediction (policy evaluation) for estimating $Q \approx q_\pi$

输入：任意目标策略 π

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

任意 $Q(s, a) \in \mathbb{R}$

$C(s, a) \leftarrow 0$

永久循环（对每个回合）：

$b \leftarrow$ 覆盖 π 的任意策略

生成一个跟随 b 的回合： $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$



$$G \leftarrow 0$$

$$W \leftarrow 1$$

对回合的每一步循环, $t = T - 1, T - 2, \dots, 0$, 直到 $W \neq 0$:

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

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

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

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

- ✍ 练习 5.9 修改首次访问 MC 策略评估的算法 (第5.1节), 以对第2.4节中描述的样本平均值使用渐增实现。□
- ✍ 练习 5.10 从 (5.7) 推导加权平均更新规则 (5.8)。遵循未加权规则 (2.3) 的推导模式。□

5.7 Off-policy 蒙特卡洛控制

现在, 我们准备介绍本书中考虑的第二类学习控制方法的示例: off-policy 方法。回想一下, on-policy 方法的显着特征是, 它们在将策略用于控制时会估计策略的价值。在 off-policy 方法中, 这两个功能是分开的。实际上, 用于生成行为的策略 (称为行为策略) 可能与被评估和改进的策略 (称为目标策略) 无关。这种分离的优点是目标策略可以是确定性的 (例如贪婪), 而行为策略可以继续对所有可能的动作进行采样。

Off-policy 蒙特卡洛控制方法使用了前两节介绍的技术之一。他们在了解和改进目标策略的同时遵循行为策略。这些技术要求行为策略具有选择目标策略 (覆盖范围) 可能选择的所有动作的非零概率。为了探索所有可能性, 我们要求行为策略是软的 (即, 它以非零概率选择所有状态下的所有动作)。

下一页的方框显示了一种基于 GPI 和加权重要性采样的 off-policy 蒙特卡洛控制方法, 用于估计 π_* 和 q_* 。目标策略 $\pi \approx \pi_*$ 是关于 Q 的贪婪策略, 它是对 q_π 的估计。行为策略 b 可以是任何东西, 但是为了确保 π 收敛到最优策略, 必须为每对状态和动作获得无限数量的回报。通过选择 b 为 ε -soft 可以确保这一点。即使根据不同的软策略 b 选择了动作, 策略 π 也会在所有遇到的状态下收敛到最优, 该软策略 b 可能在回合之间甚至在回合内发生变化。

Off-policy MC control, for estimating $\pi \approx \pi_*$

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

任意 $Q(s, a) \in \mathbb{R}$

$C(s, a) \leftarrow 0$

$\pi(s) \leftarrow \arg \max_a Q(s, a)$ (持续打破约束)

永久循环 (对每个回合):

$b \leftarrow$ 任意软策略

生成一个跟随 b 的回合: $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

$W \leftarrow 1$

对回合的每一步循环, $t = T-1, T-2, \dots, 0$:

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

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

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

$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a)$ (持续打破约束)

如果 $A_t \neq \pi(S_t)$, 那么退出内部循环 (继续下一回合)

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

一个潜在的问题是, 当回合中的所有其余动作都非常贪婪时, 此方法只能从回合的尾部学习。如果非贪婪的行为很普遍, 那么学习将会很慢, 尤其是对于长回合早期出现的状态。这可能会大大减慢学习速度。在评估 off-policy 蒙特卡洛方法对于这个问题的经验不足。如果很严重, 解决该问题的最重要方法可能是结合时间差分学习, 这是下一章中提出的算法思想。或者, 如果 γ 小于 1, 则在下一节中提出的想法可能也有很大帮助。

练习 5.11 在用于 off-policy 的 MC 控制的方框中的算法中, 您可能已经期望 W 更新涉及重要性采样率 $\frac{\pi(A_t, S_t)}{b(A_t, S_t)}$, 但是涉及 b 。为什么这是正确的? □

练习 5.12 : Racetrack (programming) 考虑如图 5.5 所示的驾驶赛车转弯。您想跑得尽可能快, 但又不能跑得太快, 以免冲出跑道。在我们简化的赛道中, 赛车位于一组离散的网格位置之一, 即图中的单元格中。速度也是离散的, 每个时间步长有许多网格单元在水平和垂直方向上移动。动作是速度分量的增量。在每一步中, 每个动作可以更改 +1、1 或 0, 总共执行九 (3×3) 个动作。两个速度分量均被限制为非负且小于 5, 并且除了起始线外, 它们都不能均为零。每个回合都以某个随机选择的起始状态开始, 其两个速度

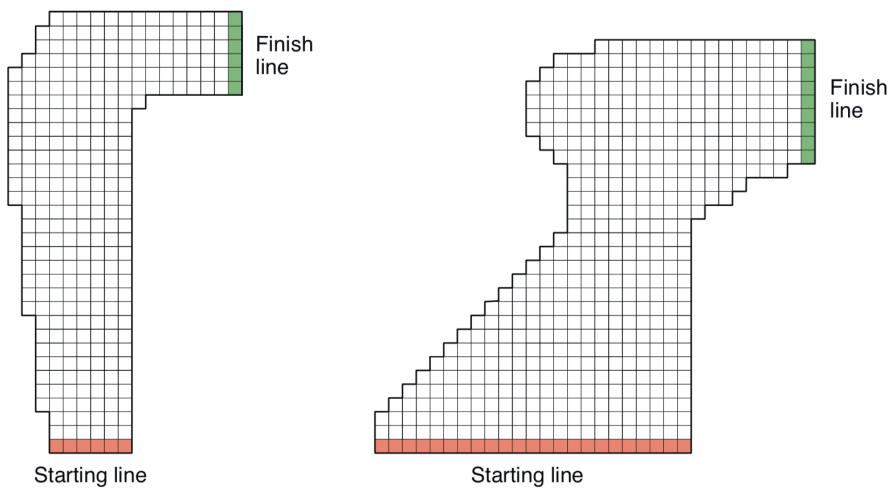


图 5.5: 右转弯可完成赛车任务。

分量均为零, 并在赛车越过终点线时结束。直到汽车越过终点线, 每一步的奖励都是 -1。如果汽车撞到了轨道边界, 则会将其移回到起跑线上的随机位置, 两个速度分量都减小

为零，并且回合继续。在每个时间步更新赛车的位置之前，请检查赛车的投影路径是否与轨道边界相交。如果与终点线相交，则回合结束；如果它与其他任何地方相交，则认为该汽车已撞到赛道边界，并被送回到起跑线。为了使任务更具挑战性，每步的概率为 0.1，速度增量均为零，与预期的增量无关。将蒙特卡洛控制方法应用于此任务，以从每个初始状态计算最优策略。展示遵循最优策略的几条轨迹（但是将这些轨迹的噪声调低）。□

5.8 * 折扣的重要性采样

到目前为止，我们考虑的 off-policy 方法是基于对被视为单一整体的回报形成重要性采样权重，而没有考虑回报的内部结构作为折扣奖励之和。现在，我们简略地考虑使用这种结构来显着减少 off-policy 估计量方差的前沿研究思路。

例如，考虑回合很长并且 $\gamma < 1$ 的情况。具体来说，假设回合持续 100 步，且 $\gamma = 0$ 。那么从时间 0 开始的回报将只是 $G_0 = R_1$ ，但其重要性采样率将是 100 个因子的乘积， $\frac{\pi(A_0|S_0)}{b(A_0|S_0)} \frac{\pi(A_1|S_1)}{b(A_1|S_1)} \dots \frac{\pi(A_{99}|S_{99})}{b(A_{99}|S_{99})}$ 。在普通重要性采样中，回报将按整个乘积进行缩放，但实际上只需要按第一个因子进行缩放，即 $\frac{\pi(A_0|S_0)}{b(A_0|S_0)}$ 。其他 99 个因子 $\frac{\pi(A_1|S_1)}{b(A_1|S_1)} \dots \frac{\pi(A_{99}|S_{99})}{b(A_{99}|S_{99})}$ 是无关紧要的，因为在第一次奖励之后，回报已经确定。这些后面的因子都与回报和期望值 1 无关。它们不会更改期望的更新，但是会极大地增加其方差。在某些情况下，他们甚至可以使方差无限。现在让我们考虑一种避免这种大的外部方差的想法。

这个想法的实质是将折扣考虑为确定终止的可能性，或者等同地确定部分终止的程度。对于任何 $\gamma \in [0, 1]$ ，我们都可以将回报 G_0 视为部分终止于一步，达到 $1 - \gamma$ 的程度，产生的回报仅为第一个奖励 R_1 ，且回报部分终止于两步，达到 $(1 - \gamma)\gamma$ 的程度，产生 $R_1 + R_2$ 的回报，依此类推。后者的程度对应于终止于第二步的 $1 - \gamma$ ，而尚未终止于第一步的 γ 。因此，第三步的终止程度为 $(1 - \gamma)\gamma^2$ ，其中 γ^2 表示在前两个步骤中任一步均未发生终止。这里的部分回报称为 flat 部分回报：

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

其中 flat 表示没有折扣，“partial”表示这些回报并没有一直延伸到终止，而是停在了称为 horizon 的 h 处（而 T 是回合终止的时间）。常规的完全回报 G_t 可以看作是以上建议的 flat 部分回报之和：

$$\begin{aligned} G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots + \gamma^{T-t-1} R_T \\ &= (1 - \gamma)R_{t+1} \\ &\quad + (1 - \gamma)\gamma(R_{t+1} + R_{t+2}) \\ &\quad + (1 - \gamma)\gamma^2(R_{t+1} + R_{t+2} + R_{t+3}) \\ &\quad \vdots \\ &\quad + (1 - \gamma)\gamma^{T-t-2}(R_{t+1} + R_{t+2} + \dots + R_{T-1}) \\ &\quad + \gamma^{T-t-1}(R_{t+1} + R_{t+2} + \dots + R_T) \end{aligned}$$

$$= (1 - \gamma) \sum_{h=t+1}^{T-1} \gamma^{h-t-1} \bar{G}_{t:h} + \gamma^{T-t-1} \bar{G}_{t:T}.$$

现在，我们需要通过类似截断的重要性采样率来缩放 flat 部分回报。由于 $G_{t:h}$ 仅涉及到 horizon 为 h 的奖励，因此我们只需要概率与 h 的比率即可。类似于 (5.5)，我们定义一个普通的重要性采样估计为

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

以及类似于 (5.6)，加权重要性采样估计为

$$V(s) \doteq \frac{\sum_{t \in \mathfrak{T}(s)} \left((1 - \gamma) \sum_{h=t+1}^{T(t)-1} \gamma^{h-t-1} \rho_{t:h-1} \bar{G}_{t:h} + \gamma^{T(t)-t-1} \rho_{t:T(t)-1} \bar{G}_{t:T(t)} \right)}{\sum_{t \in \mathfrak{T}(s)} \left((1 - \gamma) \sum_{h=t+1}^{T(t)-1} \gamma^{h-t-1} \rho_{t:h-1} + \gamma^{T(t)-t-1} \rho_{t:T(t)-1} \right)}. \quad (5.11)$$

我们称这两个估计为折扣感知重要性采样估计。如果 $\gamma = 1$ ，它们考虑了折扣率，但没有影响（与 5.5 节中的 off-policy 估计量相同）。

5.9 * 每决策的重要性采样

在 off-policy 重要性采样中，还有另一种方法可以考虑作为回报总和的回报结构，该方法即使在没有折扣的情况下也可以减小方差（即，即使 $\gamma = 1$ ）。在 off-policy 估计 (5.5) 和 (5.6) 中，分子中总和的每个项本身就是一个总和：

$$\begin{aligned} \rho_{t:T-1} G_t &= \rho_{t:T-1} (R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-t-1} R_T) \\ &= \rho_{t:T-1} R_{t+1} + \gamma \rho_{t:T-1} R_{t+2} + \cdots + \gamma^{T-t-1} \rho_{t:T-1} R_T. \end{aligned} \quad (5.12)$$

在所有这些因子中，人们可能会怀疑只有第一个和最后一个（奖励）相关；其他所有因子都与奖励后发生的事件有关。此外，所有其他这些因子的期望值为 1：

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

再经过几步，可以证明，所有其他因子都对期望没有影响，换句话说，

$$\mathbb{E}[\rho_{t:T-1} R_{t+1}] = \mathbb{E}[\rho_{t:T-1} R_{t+2}] = \cdots = \mathbb{E}[\rho_{t:T-1} R_T]. \quad (5.14)$$

如果我们对 (5.11) 的第 k 个子项重复此过程，则得到

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

这样就可以写出我们对原始术语 (5.11) 的期望

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

其中

$$\tilde{G}_t = \rho_{t:t} R_{t+1} + \gamma \rho_{t:t+1} R_{t+2} + \gamma^2 \rho_{t:t+2} R_{t+3} + \cdots + \gamma^{T-t-1} \rho_{t:T-1} R_T. \quad (5.15)$$

我们称这种想法为每决策重要性采样。随之而来的是，有一个替代的重要性采样估计，与普通重要性采样估计（5.5）具有相同的无偏期望（在首次访问的情况下），使用 \tilde{G}_t ：

$$V(s) \doteq \frac{\sum_{t \in \mathfrak{T}(s)} \tilde{G}_t}{|\mathfrak{T}(s)|}, \quad (5.16)$$

有时我们可能会期望它的方差较小。

是否有加权重要性采样的每决策版本？这不太清楚。到目前为止，我们所知道的为此提出的所有估计量都不是一致的（也就是说，它们没有用有限的数据收敛到真实值）。

- ☞ **练习 5.13** 显示从（5.12）推导（5.14）的步骤。
- ☞ **练习 5.14** 修改 off-policy 蒙特卡洛控制的算法（第 111 页），以使用截断的加权平均估计（5.10）的思想。请注意，您首先需要将此方程式转换为动作值。

5.10 总结

本章介绍的蒙特卡洛方法以样本回合的形式从经验中学习价值函数和最优策略。与 DP 方法相比，这给了它们至少三种优势。首先，可以使用它们直接从与环境的交互中学习最优行为，而无需环境动态模型。其次，它们可以与模拟或样本模型一起使用。对于令人惊讶的许多应用，即使很难构造 DP 方法所需的那种转移概率的显式模型，也很容易模拟样本回合。第三，将蒙特卡洛方法集中在一小部分状态上既简单又有效。可以准确地评估一个特殊感兴趣的区域，而不必花费太多去精确评估状态集的其余部分（我们将在第 8 章中对此进行进一步探讨）。

蒙特卡罗方法的第四个优点，我们将在本书后面讨论，那就是它们可能受到违反马尔可夫性质的损害较小。这是因为它们没有根据后继状态的价值估计来更新其价值估计。换句话说，这是因为它们没有自举。

在设计蒙特卡洛控制方法时，我们遵循了第 4 章介绍的广义策略迭代（GPI）的总体方案。GPI 涉及策略评估和策略改进的交互过程。蒙特卡洛方法提供了另一种策略评估流程。他们没有使用模型来计算每个状态的值，而是简单地平均了从该状态开始的许多回报。由于状态的价值就是期望的回报，因此该平均值可以很好地逼近该价值。在控制方法中，我们对逼近动作值函数特别感兴趣，因为它们可以用于改进策略而无需环境转移动态模型。蒙特卡洛方法在逐回合的基础上混合了策略评估和策略改进步骤，并且可以逐回合地渐增实现。

保持足够的探索是蒙特卡洛控制方法中的一个问题。仅选择当前估计为最优的动作是不够的，因为那样便无法获得替代动作的回报，并且可能永远不会得知它们实际上是更好的。一种方法是通过假设回合以随机选择以涵盖所有可能性的状态-动作对开始来忽略此问题。这样的探索起点有时可以安排在带有模拟回合的应用中，但不太可能从真实



经验中学习。在 on-policy 的方法中，agent 致力于始终探索并尝试找到仍在探索的最优策略。在 off-policy 方法中，agent 还可以探索但会学习确定的最优策略，该策略可能与所遵循的策略无关。

off-policy 预测是指从不同的行为策略生成的数据中学习目标策略的价值函数。这种学习方法是基于某种形式的重要性采样，即基于对两种策略下采取观察到的行为的概率的比值对回报进行加权，从而将其期望从行为策略转换为目标策略。普通重要性采样使用加权回报的简单平均值，而加权重要性采样使用加权平均值。普通重要性采样会产生无偏估计，但具有较大的（可能是无限的）方差，而加权重要性采样始终具有有限的方差，在实践中是首选。尽管它们的概念简单，但用于预测和控制的 off-policy 蒙特卡洛方法仍未解决，并且是正在进行的研究的主题。

本章中讨论的蒙特卡洛方法与上一章中讨论的 DP 方法有两种主要区别。首先，它们基于样本经验进行操作，因此无需模型即可用于直接学习。其次，它们不自举。也就是说，他们不会基于其他价值估计来更新其价值估计。这两个区别不是紧密相连的，可以分开。在下一章中，我们将考虑从经验中学习的方法，例如蒙特卡洛方法，以及自举，例如 DP 方法。

5.11 书目与历史评论

“蒙特卡洛”一词可追溯到 20 世纪 40 年代，当时 Los Alamos 的物理学家设计了一些机会游戏，他们可以学习以帮助理解与原子弹有关的复杂数学现象。这种蒙特卡罗方法的涵盖范围可以在几本教科书中找到（例如 Kalos 和 Whitlock, 1986; Rubinstein, 1981）。
5.1-5.2 Singh 和 Sutton (1996) 区分了每次访问和首次访问 MC 方法，并证明了将这些方法与强化学习算法相关联的结果。21 点的示例基于 Widrow, Gupta 和 Maitra (1973) 使用的示例。肥皂泡的例子是经典的 Dirichlet 问题，其最早的蒙特卡洛解法是 Kakutani (1945 年；参见 Hersh 和 Griego, 1969 年；Doyle 和 Snell, 1984 年) 提出的。

Barto 和 Duff (1994) 在经典的蒙特卡洛算法的背景下讨论了线性方程组求解的策略评估。他们使用 Curtiss (1954) 的分析指出了针对大问题的蒙特卡洛策略评估的计算优势。
5.3-5.4 本书的 1998 年版中介绍了蒙特卡洛 ES。这可能是蒙特卡洛估计和基于策略迭代的控制方法之间的第一个显式联系。Michie and Chambers (1968) 早期使用蒙特卡洛方法来估计强化学习情境下的动作价值。在杆子平衡（第 56 页）中，他们使用回合持续时间的平均值来评估每种状态下每种可能动作的价值（预期的平衡“寿命”），然后使用这些评估来控制动作的选择。他们的方法在本质上类似于每次访问 MC 估计的蒙特卡洛 ES。Narendra 和 Wheeler (1986) 研究了用于遍历有限马尔可夫链的蒙特卡洛方法，该方法使用连续两次访问同一状态之间的累积回报作为调整学习自动机动作概率的奖励。

5.5 有效的 off-policy 学习已被认为是在多个领域中出现的重要挑战。例如，它与概率图（贝叶斯）模型中的“干预”和“反事实”概念密切相关（例如，Pearl, 1995; Balke 和 Pearl, 1994）。使用重要性采样的 off-policy 方法历史悠久，但尚未得到很好的理解。Rubinstein (1981), Hesterberg (1988), Shelton (2001) 和 Liu (2001) 等人讨论了加权重要性抽样，

有时也称为归一化重要性抽样（例如 Koller 和 Friedman, 2009）。

在文献中，与本书的第一版一样，off-policy 学习有时将目标策略称为估计策略。

5.7 赛道练习改编自 Barto, Bradtke 和 Singh (1995) 和 Gardner (1973)。

5.8 我们对折扣感知重要性采样的想法的处理基于对 Sutton, Mahmood, Precup 和 van Hasselt (2014) 的分析。到目前为止，Mahmood 已对此问题进行了最充分的研究 (2017; Mahmood, van Hasselt 和 Sutton, 2014)。

5.9 每决策的重要性采样由 Precup, Sutton 和 Singh (2000) 引入。他们还将 off-policy 学习与时间差分学习，资格迹和近似方法相结合，引入了我们在后续章节中考虑的细微问题。



第六章 时间差分学习

如果必须将一种思想确定为强化学习的核心和新颖性，那么毫无疑问，这将是时间差分 (temporal-difference, TD) 学习。TD 学习是蒙特卡洛思想和动态规划 (DP) 思想的结合。与蒙特卡洛方法一样，TD 方法可以直接从原始经验中学习，而无需建立环境动态模型。与 DP 一样，TD 方法也部分基于其他已获悉的估计值来更新估计值，而无需等待最终结果（它们自举）。TD, DP 和蒙特卡洛方法之间的关系是强化学习理论中反复出现的主题。本章是我们对其进行探索的开始。在完成之前，我们将看到这些思想和方法相互融合并且可以以多种方式组合。特别是，在第 7 章中，我们介绍了 n 步算法，它提供了从 TD 到蒙特卡洛方法的桥梁，在第 12 章中，我们介绍了 $\text{TD}(\lambda)$ 算法，它无缝地统一了它们。

像往常一样，我们首先关注策略评估或预测问题，即针对给定策略 π 估计值函数 v_π 的问题。对于控制问题（找到最优策略），DP, TD 和蒙特卡洛方法均使用广义策略迭代 (GPI) 的某些变体。这些方法的差异主要是它们在预测问题上的方法差异。

6.1 TD 预测

TD 和蒙特卡洛方法都使用经验来解决预测问题。给定一些遵循策略 π 的经验，这两种方法都会针对该经验中发生的非终止状态 S_t 更新其 v_π 的估计 V 。粗略地说，蒙特卡洛方法要等到访问后的回报已知，然后再将该回报用作 $V(S_t)$ 的目标。适用于非平稳环境的简单的每次访问蒙特卡洛方法是

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

其中 G_t 是时间 t 之后的实际回报，而 α 是恒定的步长参数（参见公式2.4）。让我们将此方法称为常数 α MC。蒙特卡洛方法必须等到回合结束才能确定 $V(S_t)$ 的增量（只有那时才知道 G_t ），而 TD 方法只需要等到下一个时间步。在时间 $t+1$ ，他们立即形成目标，并使用观察到的奖励 R_{t+1} 和估计值 $V(S_{t+1})$ 进行有用的更新。最简单的 TD 方法进行立即更新

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

对于转换到 S_{t+1} 并接收 R_{t+1} 。实际上，蒙特卡洛更新的目标是 G_t ，而 TD 更新的目标是 $R_{t+1} + \gamma V(S_{t+1})$ 。这种 TD 方法被称为 $\text{TD}(0)$ 或单步 TD，因为它是在第 12 章和第 7 章中开发的 $\text{TD}(\lambda)$ 和 n 步 TD 方法的特例。下面的方框完全以程序形式说明了 $\text{TD}(0)$ 。

Tabular TD(0) for estimating v_π

输入：要评估的策略 π

算法参数：步长 $\alpha \in (0, 1]$

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

对于每个回合循环：

初始化 S

对于回合的每一步循环：

$A \leftarrow$ 对于 S 由 π 给定动作

采取动作 A , 观测 R, S'

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

$S \leftarrow S'$

直到 S 是终止

因为 TD(0) 的更新部分基于现有的估计, 所以我们说它是自举方法, 例如 DP。从第3章我们知道

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

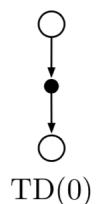
$$= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] \quad (\text{from (3.9)})$$

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

粗略地说, 蒙特卡洛方法使用 (6.3) 的估计作为目标, 而 DP 方法使用 (6.4) 的估计作为目标。蒙特卡洛目标是估计值, 因为 (6.3) 中的期望值未知。使用样本回报代替实际的期望回报。DP 目标是估计值, 不是因为期望值完全假定由环境模型提供, 而是因为 $v_\pi(S_{t+1})$ 未知且使用当前估计值 $V(S_{t+1})$ 代替。TD 目标是估计值有两个原因: 它在 (6.4) 中对期望值进行采样, 以及使用当前估计值 V 代替真实的 v_π 。因此, TD 方法将蒙特卡洛的采样与 DP 的自举相结合。正如我们将看到的那样, 经过考虑和想象, 这可以使我们在获得蒙特卡洛和 DP 两种方法的优点方面走很长一段路。

右边显示的是表格 TD(0) 的备份图。备份图顶部的状态节点的值估计是根据从其到紧随其后的状态的一个样本转换而更新的。我们将 TD 更新和蒙特卡洛更新称为样本更新, 因为它们涉及向前看样本后继状态 (或状态-动作对), 使用后继的值和沿途的奖励来计算备份的值, 并且然后相应地更新原始状态 (或状态-动作对) 的值。样本更新不同于 DP 方法的期望更新, 因为它们基于单个样本后继者, 而不是基于所有可能的后继者的完整分布。

最后, 请注意, TD(0) 更新中括号中的量是一种误差, 它测量了 S_t 的估计值与更好的估计值 $R_{t+1} + \gamma V(S_{t+1})$ 之间的差异。在强化学习中, 此量称为 TD 误差, 以多种形式出现:



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

请注意，每次 TD 误差是当时所做估算的误差。由于 TD 误差取决于下一个状态和下一个奖励，因此直到下一个时间步才实际可用。也就是说， δ_t 是 $V(S_t)$ 的误差，在时间 $t+1$ 处可用。还请注意，如果数组 V 在回合期间不发生变化（因为在蒙特卡洛方法中没有发生变化），则蒙特卡洛误差可以被写为 TD 误差的总和：

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

如果在回合期间更新 V （如 TD(0) 中一样），则此等式并不精确，但如果步长较小，则它可能仍然近似成立。这个等式的推广在时间差分学习的理论和算法中起着重要的作用。

 **练习 6.1** 如果 V 在回合中发生变化，那么 (6.6) 只能近似成立；两边之间的差异是什么？让 V_t 表示在 TD 误差 (6.5) 和 TD 更新 (6.2) 中时间 t 处使用的状态值数组。重新进行上述推导，以确定必须添加到 TD 误差之和中的附加量，以便等于蒙特卡洛误差。□

例 6.1：Driving Home 每天下班开车回家时，您都在尝试预测回家所需的时间。当您离开办公室时，请注意时间，星期几，天气以及其他可能相关的内容。假设这个星期五您正好在 6 点钟离开，估计回家需要 30 分钟。当您到达车时是 6:05，您注意到它开始下雨了。雨中的交通通常较慢，因此您估计从那时起需要 35 分钟，或总共 40 分钟。十五分钟后，您便及时完成了旅途中的高速公路部分。当您驶入次要道路时，将预计的总旅行时间减少到 35 分钟。不幸的是，此时您被困在一辆缓慢的卡车后面，道路太狭窄而无法通过。您最终不得不跟随卡车，直到在 6:40 驶入您居住的小巷。三分钟后，您回到家了。因此，状态，时间和预测的序列如下：

状态	经过的时间 (分)	预测的时间	预测的总时间
离开办公室，星期五 6 点	0	30	30
到达车，下雨	5	35	40
离开高速路	20	15	35
次道，卡车后	30	10	40
进入家庭街道	40	3	43
到家	43	0	43

在此示例中，奖励是每一段旅程的经过的时间¹。我们没有折扣 ($\gamma = 1$)，因此每个状态

¹如果这是一个以最小化旅行时间为控制问题的目标，那么我们当然会将奖励设为所花费时间的负数。但是因为这里我们只关心预测（策略评估），所以可以使用正数来简化事情。

的回报是从该状态出发的实际时间。每个状态的值是预期的运行时间。数字的第二列给出了遇到的每个状态的当前估计值。

查看蒙特卡洛方法操作的一种简单方法是在序列上绘制预测的总时间（最后一列），如图6.1（左）所示。红色箭头显示对于 $\alpha = 1$ 的常数 α MC 方法（6.1）推荐的预测变化。这些恰好是每种状态下的估计值（预计运行时间）与实际回报（实际时间）之间的误差。去）。例如，当您离开高速公路时，您以为回家只需要 15 分钟，而实际上却花了 23 分钟。公式6.1在这一点上适用，并确定离开高速公路后经过的时间的估计增量。此时的误差 $G_t - V(S_t)$ 为八分钟。假设步长参数 α 为 $1/2$ 。然后，由于这种经验，预计离开高速公路后要经过的时间将向上调四分钟。在这种情况下，这可能是一个太大的编号；车可能只是不幸的刹车。无论如何，只能离线更改，也就是说，在您到家之后。仅在这一点上，您才知道任何实际的回报。

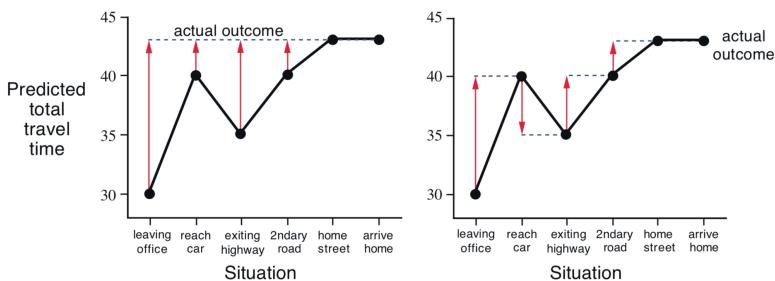


图 6.1：蒙特卡洛方法（左）和 TD 方法（右）在开车回家示例中建议的更改。

是否有必要等到最终结果出来后才能开始学习呢？假设在另一天，您再次离开办公室时估计要开车回家需要 30 分钟，但是随后您陷入了严重的交通堵塞。离开办公室 25 分钟后，您仍然在高速公路上颠簸。现在，您估计要再花费 25 分钟才能回家，总共需要 50 分钟。在交通中等待时，您已经知道您最初对 30 分钟的估计过于乐观。您是否必须等到回家才能增加对初始状态的估算？根据蒙特卡洛方法，您必须这样做，因为您尚不知道真正的回报。

另一方面，根据 TD 方法，您将立即学习，将最初的估计从 30 分钟转变为 50 分钟。实际上，每个估计都将转变为紧随其后的估计。回到驾驶的第一天，图6.1（右）显示了 TD 规则（6.2）建议的预测中的变化（如果 $\alpha = 1$ ，则是规则所做出的变化）。每个误差与预测的时间变化成正比，即与预测中的时间差异成正比。

除了让您在等待交通时可以做的事情之外，还有一些计算上的原因，为什么基于当前的预测学习而不是等到终止后知道实际回报才学习，这是有利的。我们将在下一节中简要讨论其中的一些内容。 ■

练习 6.2 这是一个练习，有助于您了解为什么 TD 方法通常比蒙特卡洛方法更有效。考虑一下开车回家的例子，以及如何用 TD 和蒙特卡洛方法解决它。您能想象一个场景，其中 TD 更新平均要比蒙特卡洛更新更好吗？举例说明过去的经验和当前的状态，您可能希望 TD 更新会更好。提示：假设您有很多下班开车回家的经验。然后，您移至新建筑物和新停车场（但您仍在同一位置进入高速公路）。现在，您开始学习有关新建筑物的预测。您能理解为什么在这种情况下，至少在最初的时候，TD 更新可能会好得多吗？在原

始场景中可能会发生同样的事情吗? □

6.2 TD 预测方法的优势

TD 方法部分基于其他估算值来更新其估算值。它们从一个猜测中学习另一个猜测，它们是自举。这是一件好事吗？TD 方法相对于蒙特卡洛和 DP 方法有什么优势？开发和回答此类问题将占用本书的其余部分及更多内容。在本节中，我们简述一些答案。

显然，TD 方法相对于 DP 方法具有一个优势，因为它们不需要环境、奖励和下一状态概率分布的模型。

TD 方法相对于蒙特卡洛方法的下一个最明显的优势是，它们自然以在线，完全增量的方式实现。对于蒙特卡洛方法，必须等到回合结束，因为只有这样才能知道回报，而对于 TD 方法，则只需要等待一个时间步。令人惊讶的是，这经常是关键的考虑因素。某些应用的回合很长，因此将所有学习延迟到回合结束时太慢。其他应用是连续的任务，根本没有任何回合。最后，正如我们在上一章中指出的那样，某些蒙特卡洛方法必须忽略或折扣对采取实验动作的回合，这会大大减慢学习速度。TD 方法不太容易受到这些问题的影响，因为它们会从每次转移中吸取教训，而不管采取了什么后续动作。

但是 TD 方法是否合理？当然，不等待实际结果就从下一步学习一个猜测很方便，但是我们仍然可以保证收敛到正确的答案吗？令人高兴的是，答案是肯定的。对于任何固定策略 π ，已经证明 $TD(0)$ 收敛到 v_π ，如果一个恒定步长参数足够小，则平均值为 v_π ；如果步长参数按照常规随机近似条件（2.7）减小，则概率为 1。大多数收敛证明仅适用于上述算法（6.2）的基于表的情况，但也适用于一般线性函数逼近的情况。这些结果将在 9.4 节中更一般地讨论。

如果 TD 和蒙特卡洛方法都渐近地收敛到正确的预测，那么自然的下一个问题是“哪个首先到达那里？”换句话说，哪种方法学得更快？哪个可以更有效地利用有限的数据？目前，这是一个悬而未决的问题，因为没有人能够从数学上证明一种方法的收敛速度比另一种方法快。实际上，甚至还不清楚用什么最正式的方式来表达这个问题！但是，实际上，在随机任务上，通常发现 TD 方法比恒定 α MC 方法收敛更快，如示例 6.2 所示。

例 6.2：Random Walk

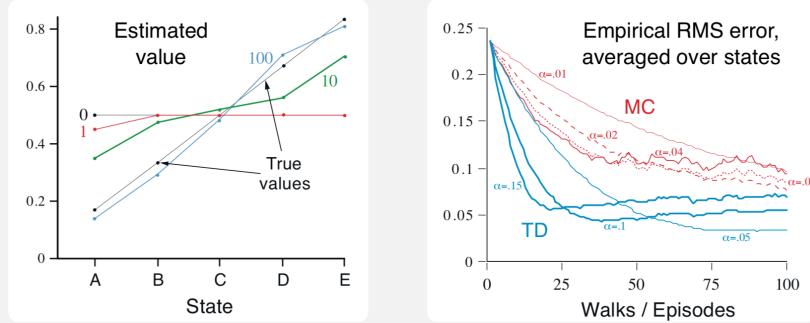
Random Walk

在此示例中，我们经验地比较了将 TD(0) 和常数 α MC 在应用于以下马尔可夫奖励过程时的预测能力：



马尔可夫奖励过程或 MRP 是没有动作的马尔可夫决策过程。当我们专注于预测问题时，我们经常会使用 MRP，在这些问题中，无需将环境造成的动态与 agent 造成的动态区分开。在此 MRP 中，所有回合都从中心状态 C 开始，然后在每一步上

以相同的概率从一个状态向左或向右进行。回合在最左端或最右端终止。当回合在右端终止时，奖励为 +1；所有其他奖励均为零。例如，一个典型的回合可能由以下状态和奖励序列组成：C, 0, B, 0, C, 0, D, 0, E, 1。因为此任务是无折扣的，所以每个状态的真实值是从该状态开始在右端终止的概率。因此，中心状态的真实值为 $v_\pi(C) = 0.5$ 。从 A 到 E 的所有状态的真实值是 $\frac{1}{6}, \frac{2}{6}, \frac{3}{6}, \frac{4}{6}, \frac{5}{6}$ 。



上方的左图显示了在 TD(0) 的一次运行中，经过各种回合后学习的值。100 个回合后的估计值几乎接近其真实值，这使用恒定的步长参数（在此示例中， $\alpha = 0.1$ ），该值会随着最近回合的结果而不确定地波动。右图显示了针对不同 α 值的两种方法的学习曲线。所示的性能度量是学习的值函数和真值函数之间的均方根 (RMS) 误差，在五个状态中取平均值，然后在 100 次运行中取平均值。在所有情况下，所有 s 的近似值函数均被初始化为中间值 $V(s) = 0.5$ 。在此任务上，TD 方法始终优于 MC 方法。 ■

- ✍ **练习 6.3** 从随机行走示例左图所示的结果可以看出，第一个回合仅导致 $V(A)$ 的变化。这告诉您第一回合发生了什么？为什么仅对该状态的估计值进行了更改？到底发生了多少变化？ □
- ✍ **练习 6.4** 随机行走示例右图所示的特定结果取决于步长参数 α 的值。您是否认为，如果使用更大范围的 α 值，将会得出关于哪种算法更好的结论？是否存在一个固定的 α 值，而每种算法在该值处的表现都明显好于所示？为什么或者为什么不？ □
- ✍ **练习 6.5** 在随机行走示例的右图中，TD 方法的 RMS 误差似乎先下降然后再上升，尤其是在高 α 值时。是什么原因造成的？您是否认为这总是发生，或者可能是近似值函数如何初始化的函数？ □
- ✍ **练习 6.6** 在例 6.2 中，我们声明了状态 A 到 E 的随机行走示例的真实值为 $\frac{1}{6}, \frac{2}{6}, \frac{3}{6}, \frac{4}{6}, \frac{5}{6}$ 。描述至少两种可以计算出的方式。您猜我们实际使用了哪一个？为什么？ □

6.3 TD(0) 的最优性

假设只有有限的经验，例如 10 个回合或 100 步。在这种情况下，采用增量学习方法的常见方法是反复展示经验，直到该方法收敛于答案为止。给定一个近似值函数 V ，对于访问非终止状态的每个时间步长 t ，都将计算由 (6.1) 或 (6.2) 指定的增量，但是该



值函数仅被所有增量的总和改变一次。然后，使用新的值函数再次处理所有可用的经验，以产生新的总体增量，依此类推，直到值函数收敛为止。之所以称为批量更新，是因为仅在处理完每批训练数据之后才进行更新。

在批量更新中，只要将 α 选为足够小，TD(0) 就确定性地收敛到单个答案，而与步长参数 α 无关。在相同条件下，常数 α MC 方法也可以确定地收敛，但得出的答案是不同的。了解这两个答案将有助于我们理解这两种方法之间的区别。在正常更新下，这些方法不会一直移动到它们各自的批量答案，但是从某种意义上说，它们会朝着这些方向采取步骤。在尝试全面理解这两个答案之前，对于所有可能的任务，我们首先来看几个示例。

例 6.3：Random walk under batch updating TD(0) 和常数 α MC 的批量更新版本按以下方式应用于随机行走预测示例（示例6.2）。在每个新的回合之后，到目前为止看到的所有回合都视为一个批。反复将它们呈现给 TD(0) 或常数 α MC， α 足够小将使得值函数收敛。然后将结果值函数与 v_π 进行比较，并绘制出五个状态（以及整个实验的 100 个独立重复）的平均均方根误差，以获得图6.2所示的学习曲线。注意，批量 TD 方法始终优于批量蒙特卡洛方法。

在批训练中，常数 α MC 收敛到值 $V(s)$ ，该值是访问每个状态 s 后实际回报的样本平均值。从它们使训练集中的实际回报最小化均方误差的意义上说，这些是最优估计。从这个意义上讲，令人惊讶的是，根据右图所示的均方根误差性能，批量 TD 方法能够执行地更好。批量 TD 的性能如何比这种优化方法更好？答案是，蒙特卡洛方法仅在有限的方式下是最优的，而 TD 在与预测回报更相关的方式上是最优的。 ■

例 6.4：You are the Predictor 现在让您自己扮演未知马尔可夫奖赏过程的回报预测者。假设您观察到以下八个回合：

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

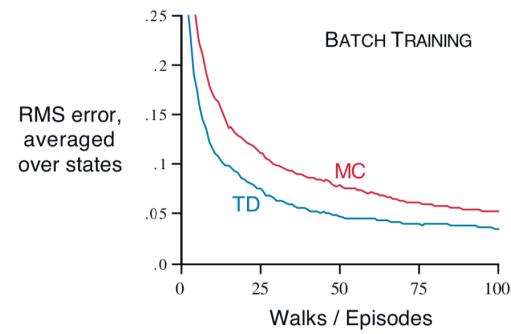
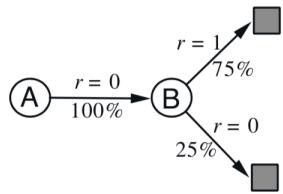


图 6.2: TD(0) 和常数 α MC 在随机行走任务批量训练下的性能。

这意味着第一个回合从状态 A 开始，以 0 的奖励转移到 B，然后以 0 的奖励从 B 终止。其他七个回合更短，从 B 开始并立即终止。给定这批数据，您认为估算 $V(A)$ 和 $V(B)$ 的最优预测和最优值是什么？每个人都可能同意 $V(B)$ 的最优值为 $\frac{3}{4}$ ，因为在状态 B 的八次中，有六次过程立即以 1 的回报终止，而在状态 B 的另外两次中，过程以 0 的回报立即终止。

但是给定该数据，估计 $V(A)$ 的最优值是多少？这里有两个合理的答案。一种是观察到流程处于状态 A 的 100% 的时间立即遍历到 B（奖励为 0）；并且由于我们已经确定 B 具有值 $\frac{3}{4}$ ，因此 A 也必须具有值 $\frac{3}{4}$ 。查看此答案的一种方法是，它基于首先对马尔可夫过程进行建模（在这种情况下如右图所示），然后在给定模型的情况下计算正确的估计值，在这种情况下确实给出 $V(A) = \frac{3}{4}$ 。这是批量 TD(0) 给出的答案。



另一个合理的答案是我们观察到 A 一次，然后返回 0。因此，我们估计 $V(A)$ 为 0。这就是批量蒙特卡洛方法给出的答案。注意，这也是在训练数据上给出最小平方误差的答案。实际上，它使数据的误差为零。但是我们仍然希望第一个答案会更好。如果过程是马尔可夫，我们希望第一个答案将对未来的数据产生较小的误差，即使在现有数据上的蒙特卡洛答案更好。 ■

例6.4说明了通过批量 TD(0) 和批量蒙特卡洛方法得出的估计值之间的一般差异。批量蒙特卡洛方法总是找到使训练集的均方误差最小的估计，而批量 TD(0) 总是找到对于马尔可夫过程的最大似然模型完全正确的估计。通常，参数的最大似然估计是其生成数据的概率最大的参数值。在这种情况下，最大似然估计是从观察到的回合以明显的方式形成的马尔可夫过程的模型：从 i 到 j 的估计转移概率是从 i 到 j 的观察到的转移的分数，并且相关期望奖励是在这些转移中观察到的奖励的平均值。给定此模型，我们可以计算出值函数的估计值，如果模型完全正确，则该估计值将完全正确。这被称为确定性等价估计，因为它等同于假设对潜在过程的估计是确定性地已知而不是被近似。通常，批量 TD(0) 收敛到确定性等价估计。

这有助于解释为什么 TD 方法比蒙特卡洛方法收敛更快。在批量形式中，TD(0) 比蒙特卡洛方法要快，因为它可以计算真实确定性等价估计。这解释了批处理结果中显示的 TD(0) 在随机行走任务上的优势（图6.2）。与确定性等价估计值的关系还可以部分解释非批处理 TD(0) 的速度优势（例如，例6.2，第 125 页，右图）。尽管非批处理方法既不能实现确定性等价，也不能实现最小平方误差估计，但是可以将其理解为大致在这些方向上移动。非批处理 TD(0) 可能会比常数 α MC 快，因为尽管它并没有完全实现，但它正在朝着更好的估计方向发展。目前，关于在线 TD 和蒙特卡洛方法的相对效率尚无定论。

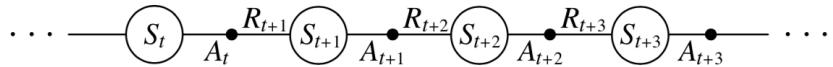
最后，值得注意的是，尽管从某种意义上说确定性等价估计是一个最优解决方案，但是直接计算它几乎是不可行的。如果 $n = |\mathcal{S}|$ 是状态数，则仅形成该过程的最大似然估计可能需要大约 n^2 个存储，如果按照常规方式进行计算，则计算相应的值函数需要大约 n^3 个计算步骤。用这些术语，确实令人惊讶的是，TD 方法可以使用不超过 n 阶的内存和在训练集上重复的计算来近似相同的解。在具有较大状态空间的任务上，TD 方法可能是逼近确定性等价解的唯一可行方法。

练习 6.7 设计 TD(0) 更新的 off-policy 版本，可与任意目标策略 π 一起使用，并覆盖行为策略 b ，并在每一步 t 都使用重要性采样率 $\rho_{t:t}$ (5.3)。 □

6.4 Sarsa: on-policy TD 控制

现在我们转向使用 TD 预测方法来解决控制问题。像往常一样，我们遵循广义策略迭代 (GPI) 的模式，仅这次是将 TD 方法用于评估或预测部分。与蒙特卡洛方法一样，我们面临着进行探索和利用平衡的需求，并且方法又分为两大类：on-policy 和 off-policy。在本节中，我们介绍一种 on-policy 的 TD 控制方法。

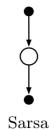
第一步是学习动作值函数，而不是状态值函数。特别地，对于 on-policy 的方法，我们必须估计当前行为策略 π 以及所有状态 s 和动作 a 的 $q_\pi(s, a)$ 。可以使用与上述用于学习 v_π 的基本相同的 TD 方法来完成此操作。回想一下，回合由状态和状态-动作对的交替序列组成：



在上一节中，我们考虑了从状态到状态的转换，并了解了状态的值。现在我们考虑从状态-动作对到状态-动作对的转移，并学习状态-动作对的值。从形式上讲，这两种情况是相同的：它们都是具有奖励过程的马尔可夫链。确保 TD(0) 下状态值收敛的定理也适用于动作值的相应算法：

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

从非终止状态 S_t 每次转移后都会执行此更新。如果 S_{t+1} 为终止状态，则 $Q(S_{t+1}, A_{t+1})$ 定义为零。该规则使用事件五元组 $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$ 中的每个元素，这些元素构成了从一个状态-动作对到下一个状态-动作对的转移。这个五元组为该算法起名为 Sarsa。Sarsa 的备份图如右图所示。



设计基于 Sarsa 预测方法的 on-policy 的控制算法很简单。像在所有 on-policy 的方法中一样，我们不断估计行为策略 π 的 q_π ，同时将 π 相对于 q_π 变为贪婪。Sarsa 控制算法的一般形式在下一页的方框中提供。

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

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

对于所有 $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$ ，任意初始化 $Q(s, a)$ ，除了 $Q(\text{terminal}, \cdot) = 0$

对于每个回合循环：

初始化 S

使用源自 Q 的策略（例如， ε -greedy）从 S 中选择 A

对于回合的每一步循环：

采取动作 A ，观测 R, S'

使用源自 Q 的策略（例如， ε -greedy）从 S' 中选择 A'

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

$S \leftarrow S'; A \leftarrow A'$
 直到 S 是终止

Sarsa 算法的收敛性取决于策略对 Q 的依赖性。例如，可以使用 ε -greedy 或 ε -soft 策略。只要对所有状态-动作对进行无穷多次访问，Sarsa 便以概率 1 收敛到最优策略和动作值函数，并且策略收敛于对贪婪策略的极限（例如，可以通过设置 $\varepsilon = 1/t$ 来实现 ε -greedy 策略）。

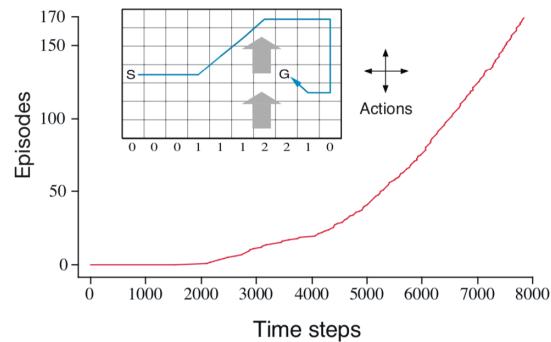
练习 6.8 证明 (6.6) 的动作值形式适用于 TD 误差 $\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$ 的动作值形式，再次假设值不会逐步改变。□

例 6.5：Windy Gridworld 下面显示的图是标准的网格世界，具有开始状态和目标状态，但有一个区别：在网格的中间有一股向上的侧风。这些动作是标准的四个动作，up, down, right, left，但在中间区域中，所产生的下一个状态通过风向上移动，其强度随列的不同而变化。风的强度在每列下方给出，以向上移动的单元数为单位。例如，如果您位于目标右边的一个单元格，那么动作向左将您带到目标上方的单元格。这是一个没有折扣的回合任务，不断获得-1 的奖励，直到达到目标状态。

右图显示将 ε -greedy 的 Sarsa 应于此任务的结果，其中 $\varepsilon = 0.1, \alpha = 0.5$ ，并且所有 s, a 的初始值 $Q(s, a) = 0$ 。图的斜率增加表明，随着时间的推移，到达目标的速度更快。到了 8000 个时间步长，贪婪策略早就是最优的（从图中可以看出它的轨迹）。持续进行的 ε -greedy 探索将平均回合长度保持在大约 17 个步骤，比最小的 15 个步骤多了两步。请注意，此处的蒙特卡洛方法不易使用，因为不能保证所有策略都可以终止。如果找到导致 agent 停留在同一状态的策略，则下一个回合将永远不会结束。诸如 Sarsa 之类的在线学习方法不存在此问题，因为在回合期间，他们会迅速学习到此类策略不佳，然后转向其他方法。■

练习 6.9：Windy Gridworld with King's Moves (programming) 假设有八种可能的动作（包括对角线移动），而不是通常的四项，重新解决风的网格世界。您可以通过这些额外的动作来改善多少？除了由风引起的动作外，您是否可以通过包含不会引起任何移动的第九个动作来做得更好呢？□

练习 6.10：Stochastic Wind (programming) 假设风的影响是随机的，有时与每列给出的平均值相差 1，则用 King 的动作重新解决有风的网格世界任务。也就是说，三分之一的时间完全按照这些值移动，就像上一个练习中一样，但是三分之一的时间将移动一个单元格到达其上方，而三分之一的时间将移动一个单元格到达其下方。例如，如果您位于目标右边一个单元格中，而您向左移动，那么三分之一的时间您将移动一个单元格移至目标上方，三分之一的时间您将移动两个单元格移至目标上方，以及三分之一的时间移至



目标。 □

6.5 Q-learning: off-policy TD 控制

强化学习的早期突破之一是开发一种称为 Q-learning 的 off-policy 的 TD 控制算法 (Watkins, 1989)，定义如下：

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

在这种情况下，学习到的动作值函数 Q 直接类似于最优动作值函数 q_* ，与遵循的策略无关。这极大地简化了算法的分析并使早期收敛性证明成为可能。该策略仍然有效，因为它可以确定访问和更新哪些状态-动作对。然而，正确收敛所需的只是所有的状态-动作对都要继续更新。正如我们在第5章中观察到的那样，从某种意义上说，这是一个最低要求，因为保证在一般情况下都能找到最优行为的任何方法都必须要求它。在这种假设和步长参数序列上通常随机逼近条件的一种变型下，已证明 Q 以概率 1 收敛到 q_* 。Q-learning 算法以程序形式显示如下。

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

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

对于所有 $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$ ，任意初始化 $Q(s, a)$ ，除了 $Q(\text{terminal}, \cdot) = 0$

对于每个回合循环：

 初始化 S

 对于回合的每一步循环：

 采取动作 A ，观测 R, S'

 使用源自 Q 的策略（例如， ϵ -greedy）从 S' 中选择 A'

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

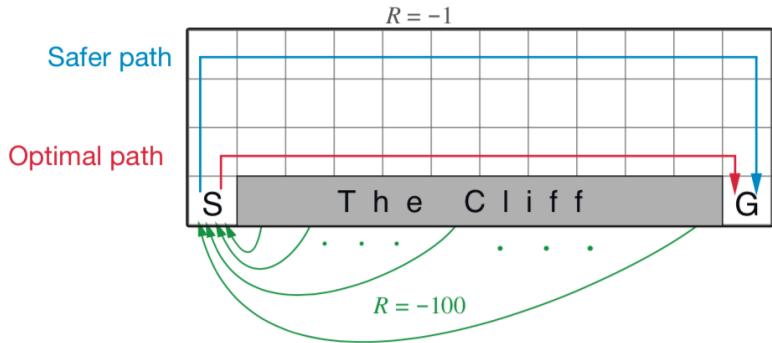
$S \leftarrow S'$

 直到 S 是终止

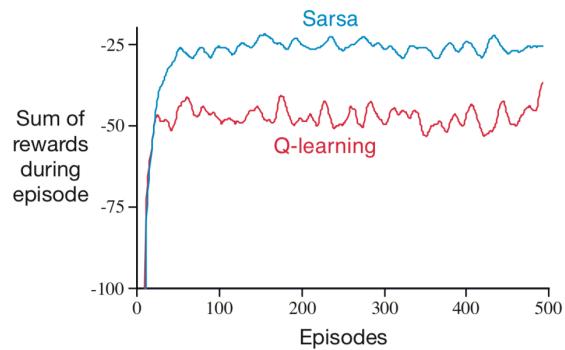
Q-learning 的备份图是什么？规则 (6.8) 更新了一个状态-动作对，因此顶层节点（更新的根）必须是一个较小的已填充的动作节点。更新也是从动作节点进行的，最大化下一个状态中所有可能的动作。因此，备份图的底部节点应该是所有这些动作节点。最后，请记住，我们用弧线表示这些“下一个动作”节点的最大值（图3.4右）。您现在能猜出图是什么吗？如果是这样，请在转至第 134 页的图6.4中的答案之前先进行猜测。

例 6.6：Cliff Walking 这个网格世界示例比较了 Sarsa 和 Q-learning，强调了 on-policy (Sarsa) 和 off-policy (Q-learning) 方法之间的区别。考虑一下右图所示的网格世界。这是一个标准的、没有折扣的回合任务，具有开始和目标状态，以及导致移动的 up, down, right, 和 left 的常规动作。除了进入标记为“The Cliff.”的区域的转移外，其他所有转移的奖励均为-1。进入该区域会产生-100 的奖励，并立即将 agent 发送回起点。





右图显示了 Sarsa 和 Q-learning 方法在 $\epsilon = 0.1$ 的 ϵ -greedy 动作选择的情况下性能。初始瞬变后，Q-learning 将学习最优策略的值，该策略沿悬崖的边缘行进。不幸的是，由于 ϵ -greedy 的动作选择，这偶尔导致掉下悬崖。另一方面，Sarsa 考虑了动作选择，并学习到通过网格上部的更长但更安全的路径。虽然 Q-learning 实际上是学习最优策略的价值，但是它的在线性能要比学习迂回策略的 Sarsa 差。当然，如果 ϵ 逐渐降低，则两种方法都将渐近收敛到最优策略。 ■



- ✍ 练习 6.11 为什么 Q-learning 被视为 off-policy 控制方法? □
- ✍ 练习 6.12 假设动作选择是贪婪的。那么 Q-learning 和 Sarsa 完全一样吗？他们会做出完全相同动作选择和权重更新吗？□

6.6 期望 Sarsa

考虑类似于 Q-learning 的学习算法，不同之处在于它使用期望值而不是下一个状态-动作对的最大值，同时考虑到每个动作在当前策略下的可能性。也就是说，考虑具有更新规则的算法

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

但这要遵循 Q-learning 模式。给定下一个状态 S_{t+1} ，此算法确定性地沿与 Sarsa 期望相同的方向移动，因此将其称为期望 Sarsa。其备份图如图6.4右侧所示。

期望 Sarsa 在计算上比 Sarsa 更为复杂，但作为回报，它消除了由于 A_{t+1} 的随机选择的方差。如果有相同的经验，我们可能会期望它的性能要比 Sarsa 稍好一些，实际上的确如此。图6.3显示了与 Sarsa 和 Q-learning 相比，使用期望 Sarsa 进行的悬崖行走任务的总结结果。期望 Sarsa 在此问题上保留了 Sarsa 的显着优势和超过 Q-learning。此外，在较大范围的步长参数 α 下，期望 Sarsa 较 Sarsa 表现出显着改进。在悬崖行走中，状态转

移都是确定性的，所有随机性都来自于策略。在这种情况下，期望 Sarsa 可以安全地设置 $\alpha = 1$ 而不会影响渐近性能的降低，而 Sarsa 只能在长期内以较小的 α 表现良好，而短期的表现则较差。在本示例和其他示例中，期望 Sarsa 相比 Sarsa 具有一致的经验优势。

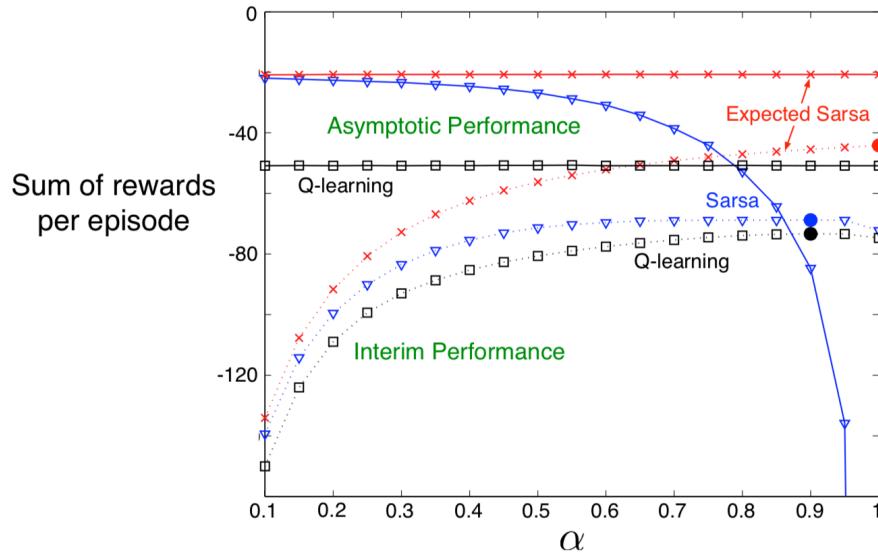


图 6.3: 作为 α 函数的 TD 控制方法在悬崖行走任务上的中值和渐近性能。所有算法都使用 $\varepsilon = 0.1$ 的 ε -greedy 策略。渐近性能是平均超过 100,000 回合，而中值性能是平均前 100 回合。这些数据分别是中值和渐近情况的平均 50,000 和 10 次运行。实心圆圈表示每种方法的最优中值性能。改编自 van Seijen 等 (2009)。



图 6.4: Q-learning 和期望 Sarsa 的备份图。

在这些悬崖行走的结果中，期望 Sarsa 是 on-policy，但是通常它可以使用与目标策略 π 不同的策略来生成行为，在这种情况下，它成为 off-policy 算法。例如，假设 π 是贪婪策略，而行为则更具探索性。那么期望 Sarsa 正是 Q-learning。从这个意义上说，期望 Sarsa 包含并拓展了 Q-learning，同时可靠地对 Sarsa 进行了改进。除了少量的额外计算成本外，期望 Sarsa 可能会完全主导其他两个更为著名的 TD 控制算法。

6.7 最大化偏差与 Double Learning

到目前为止，我们讨论的所有控制算法都在其目标策略的构建中涉及到最大化。例如，在 Q-learning 中，目标策略是给定当前动作值的贪婪策略，该策略以最大化定义，而在 Sarsa 中，该策略通常为 ε -greedy，这也涉及最大化操作。在这些算法中，将一个估计值上的最大值隐式地用作最大值的估计值，这可能导致明显的正偏差。要了解为什么，请

考虑一个状态 s , 其中存在许多动作 a , 其真实值 $q(s, a)$ 都为零, 但其估计值 $Q(s, a)$ 不确定, 因此分布高于或低于零。真实值的最大值为零, 但估计值的最大值为正, 即正偏差。我们称这种为最大化偏差。

例 6.7 : Maximization Bias Example 图6.10中显示的小 MDP 提供了一个简单的示例, 说明最大偏差如何损害 TD 控制算法的性能。MDP 具有两个非终止状态 A 和 B。回合总是从 A 开始, 在 left 和 right 两个动作之间进行选择。right 动作立即转移到终止状态, 奖励和回报为零。left 动作转移到 B, 同时也得到零奖励, 从中有许多可能的动作都导致立即终止, 并从均值-0.1 和方差 1.0 的正态分布中获得奖励。因此, 以 left 开始的任何轨迹的期望回报都是-0.1, 因此在状态 A 下采取 left 总是一个错误。然而, 由于最大偏差使 B 看起来具有正值, 我们的控制方法可能偏向 left。图6.10显示, 带有 ε -greedy 动作选择的 Q-learning 在该示例中最初学会了强烈支持的 left 动作。即使在渐近处, Q-learning 也比在我们的参数设置 ($\varepsilon = 0.1, \alpha = 0.1, \gamma = 1$) 下的最优情况多花 5% 的 left 动作。 ■

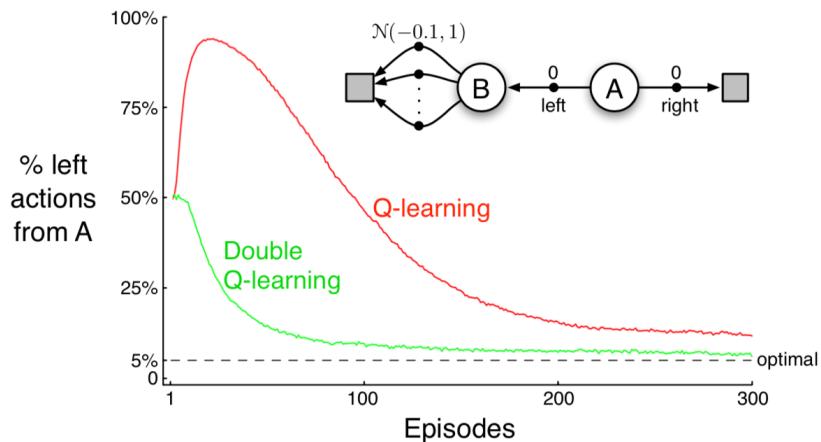


图 6.5: 在简单的回合 MDP 上进行 Q-learning 和 Double Q-learning 的比较 (图所示)。Q-learning 最初学习采取 left 动作的频率要比 right 动作的频率高, 并且总是比 $\varepsilon = 0.1$ 的 ε -greedy 动作选择强制执行的 5% 最小概率的频率高得多。相比之下, Double Q-learning 基本上不受最大化偏差的影响。这些数据平均超过 10,000 次运行。初始动作值估计为零。 ε -greedy 动作选择中的任何联系都被随机打破。

是否有避免最大化偏差的算法? 首先, 考虑一个赌博机案例, 在该案件中, 我们对每个动作的价值都进行了有噪声的估计, 该估计值是每个动作在所有游戏中获得的奖励的样本平均值。如上所述, 如果我们使用估计的最大值作为真实值的最大值的估计, 则将存在正的最大化偏差。解决问题的一种方法是, 由于使用了相同的样本 (游戏) 来确定最大化动作并估计其值。假设我们将游戏分为两组, 并使用它们学习两个独立的估计, 对所有 $a \in \mathcal{A}$, 将它们分别称为 $Q_1(a)$ 和 $Q_2(a)$, 其分别为真实值 $q(a)$ 的估计。然后我们可以使用一个估计值 Q_1 来确定最大化动作 $A^* = \arg \max_a Q_1(a)$, 另一个使用 Q_2 来提供其值的估计值 $Q_2(A^*) = Q_2(\arg \max_a Q_1(a))$ 。在 $\mathbb{E}[Q_2(A^*)] = q(A^*)$ 的意义上, 该估计将是无偏的。我们也可以重复此过程, 将两个估计的作用颠倒过来, 得出第二个无偏估计 $Q_1(\arg \max_a Q_2(a))$ 。这是 double learning 的想法。请注意, 尽管我们学习了两个估算值, 但每次游戏都只会更新一个估算值; double learning 会使内存需求增加一倍, 但不会增加每步的计算量。

Double learning 的思想自然地扩展到了完整 MDP 的算法。例如，类似于 Q-learning 的 double learning 算法（称为 Double Q-learning）可以将时间步长一分为二，就像是在每个步长上掷硬币。如果硬币正面向上，则更新为

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q_2(S_{t+1}, \arg \max_a Q_1(S_{t+1}, a)) - Q_1(S_t, A_t) \right] \quad (6.10)$$

如果硬币出现反面，则在 Q_1 和 Q_2 切换时进行相同的更新，从而更新 Q_2 。两个近似值函数被完全对称地对待。行为策略可以使用两个动作值估计。例如，用于 Double Q-learning 的 ϵ -greedy 策略可以基于两个动作值估计值的平均值（或总和）。下方的方框中提供了完整的 Double Q-learning 算法。这是用于产生图6.10中结果的算法。在该示例中，double learning 似乎消除了最大化偏差带来的危害。当然，还有 Sarsa 和期望 Sarsa 的 double 版本。

Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

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

对于所有 $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, 任意初始化 $Q_1(s, a)$, $Q_2(s, a)$, 除了 $Q(\text{terminal}, \cdot) = 0$

对于每个回合循环：

初始化 S

对于回合的每一步循环：

在 $Q_1 + Q_2$ 中使用 ϵ -greedy 策略从 S 选择 A

采取动作 A , 观测 R, S'

以 0.5 的概率：

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

否则：

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

$S \leftarrow S'$

直到 S 是终止

✍ 练习 6.13 带有 ϵ -greedy 目标策略的 Double 期望 Sarsa 的更新方程是什么？



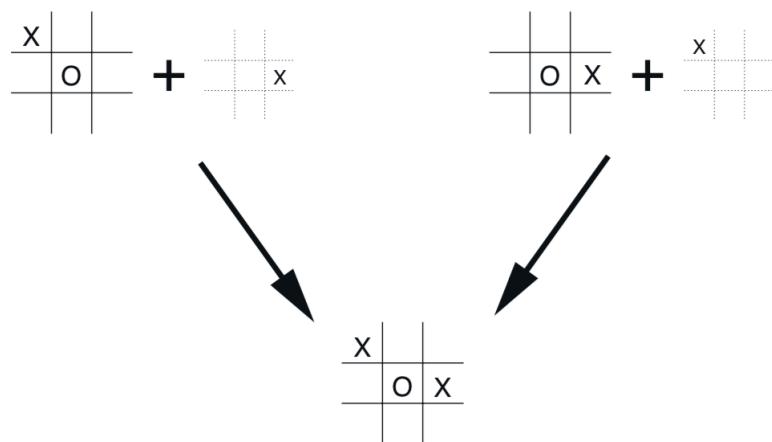
6.8 游戏、后期状态和其他特殊情况

在本书中，我们尝试为各种任务提供统一的方法，但是当然总会有一些例外的任务，可以通过专门的方式对其进行更好的处理。例如，我们的一般方法涉及学习动作值函数，但是在第1章中，我们介绍了一种 TD 方法，用于学习玩井字游戏，该方法学到的东西更像是状态值函数。如果我们仔细查看该示例，很明显，所学到的函数在通常意义上既没有动作值函数也没有状态值函数。常规状态值函数会评估 agent 可以选择动作的状态，但是井字游戏中使用的状态值函数会在 agent 采取动作后评估棋盘位置。让我们调用这些



后期状态，并在这些后期状态值函数之上调用值函数。当我们了解环境动态的初始部分但不一定了解完整动态时，后期状态很有用。例如，在游戏中，我们通常知道动作的直接效果。我们知道每一次可能的国际象棋移动都会导致什么结果，但我们的对手将如何回应。后期状态价值函数是利用此类知识并从而产生更有效的学习方法的自然方法。

从井字游戏示例可以明显看出根据后期状态设计算法更为有效的原因。传统的动作值函数会从位置和移动映射到该值的估计值。但是，许多位置-移动对产生的结果位置相同，如下例所示：



在这种情况下，位置-移动对是不同的，但产生相同的“后期位置”，因此必须具有相同的值。传统的动作值函数将不得不分别评估这两个对，而后期状态值函数将立即同等地区评估这两个对。对左边位置-移动对的任何了解，都将立即转移到右边的对上。

后期状态发生在许多任务中，而不仅仅是游戏。例如，在排队任务中，存在诸如将客户分配到服务器、拒绝客户或丢弃信息之类的工作。在这种情况下，实际上是根据动作的直接效果来定义动作的，这是完全已知的。

不可能描述所有可能的特殊问题和相应的特殊学习算法。但是，本书制定的原则应广泛应用。例如，仍然按照广义策略迭代来恰当地描述后期状态方法，其中策略和（后期状态）值函数以基本上相同的方式进行交互。在许多情况下，仍然需要在 *on-policy* 和 *off-policy* 的方法之间进行选择，以管理对持续性探索的需求。

练习 6.14 描述如何根据后期状态重新定义杰克的租车任务（例4.2）。为什么就这项特定任务而言，这样的重新制定可能会加速收敛？ □

6.9 总结

在本章中，我们介绍了一种新型的学习方法，即时间差分 (TD) 学习，并说明了如何将其应用于强化学习问题。与往常一样，我们将整个问题分为预测问题和控制问题。TD 方法是解决预测问题的蒙特卡洛方法的替代方法。在这两种情况下，控制问题的扩展都是通过从动态规划中抽象出来的广义策略迭代 (GPI) 的思想来实现的。这个想法是，近似的策略和价值函数应该以它们都朝着其最优值移动的方式交互。

组成 GPI 的两个过程之一驱使价值函数来准确预测当前策略的回报。这是预测问题。另一个过程驱使该策略在当前值函数方面进行局部改进（例如，变为 ε -greedy）。当第一个过程基于经验时，会出现保持足够探索的复杂性。我们可以根据 TD 控制方法是采用 on-policy 还是 off-policy 方法来处理这种复杂性对 TD 控制方法进行分类。Sarsa 是一种 on-policy 方法，Q-learning 是 off-policy 方法。期望 Sarsa 也是我们目前介绍的 off-policy 方法。还有第三种方法可以扩展 TD 方法到控制，这是我们在本章中未包括的方法，称为 actor-critic 方法，这些方法将在第 13 章中全面介绍。

本章介绍的方法是当今使用最广泛的强化学习方法。这可能是由于它们非常简单：以最少的计算量量，可以将在线应用到与环境交互所产生的经验；它们几乎可以用可以由小型计算机程序实现的单个方程式完全表示。在接下来的几章中，我们将扩展这些算法，使它们稍微复杂一些，并且功能更强大。所有新算法都将保留此处介绍的算法的精髓：它们将能够以相对较少的计算量在线处理经验，并且它们将受到 TD 误差的驱动。本章中介绍的 TD 方法的特殊情况应正确地称为单步、表格、无模型的 TD 方法。在接下来的两章中，我们将它们扩展为 n 步形式（与蒙特卡洛方法的联系）和包括环境模型的形式（与规划和动态规划的联系）。然后，在本书的第二部分中，我们将它们扩展为各种形式的函数近似，而不是表格（与深度学习和人工神经网络的联系）。

最后，在本章中，我们完全在强化学习问题的范围内讨论了 TD 方法，但是 TD 方法实际上比这更笼统。它们是学习对动态系统进行长期预测的通用方法。例如，TD 方法可能与预测财务数据，寿命，选举结果，天气模式，动物行为，对电站的需求或客户购买有关。只有将 TD 方法作为纯粹的预测方法进行分析，而不是在强化学习中的使用，才首先对它们的理论特性有所了解。即便如此，TD 学习方法的其他潜在应用尚未得到广泛探索。

6.10 书目与历史评论

正如我们在第 1 章中概述的那样，TD 学习的思想起源于动物学习心理学和人工智能，最著名的是 Samuel (1959) 和 Klopff (1972) 的著作。第 16.2 节将 Samuel 的工作描述为一个案例研究。与 TD 学习相关的还有 Holland (1975, 1976) 关于价值预测之间的一致性的早期想法。这些因素影响了其中一位作者 (Barto)，他是 1970 年至 1975 年在荷兰任教的密歇根大学的研究生。Holland 的想法导致了许多与 TD 相关的系统，包括 Booker (1982) 和 the bucket brigade of Holland (1986) 的工作，这与下文所述的 Sarsa 有关。

6.1-6.2 这些部分中的大多数特定材料来自 Sutton (1988)，包括 TD(0) 算法，随机行走示例和术语“时间差分学习”。Watkins (1989)，Werbos (1987) 等人影响了与动态规划和蒙特卡洛方法的关系的表征。备份图的使用是本书第一版的新增内容。

基于 Watkins 和 Dayan (1992) 的工作，Sutton (1988) 证明了表格 TD(0) 以均值收敛，而 Dayan (1992) 证明了其以 1 的概率收敛。Jaakkola, Jordan 和 Singh (1994) 和 Tsitsiklis (1994) 通过使用强大的现有随机逼近理论的扩展来推广和加强了这些结果。其他扩展和概括将在后面的章节中介绍。

6.3 Sutton (1988) 确定了批训练下 TD 算法的最优性。阐明这一结果的是 Barnard (1993) 推导的 TD 算法，它是学习马尔可夫链模型的增量方法的一个步骤和根据模型计算预测的方法的一个步骤的结合。术语“确定性等价”来自于自适应控制文献（例如，Goodwin 和 Sin，1984 年）。

6.4 Sarsa 算法由 Rummery 和 Niranjan (1994) 引入。他们结合人工神经网络对其进行了探索，并将其称为“Modified Connectionist Q-learning”。Sutton (1996) 引入了“Sarsa”这个名字。Singh, Jaakkola, Littman 和 Szepesvari (2000 年) 证明了一步表格 Sarsa (在本章中讨论的形式) 的收敛性。Tom Kalt 提出了“windy gridworld”示例。

Holland (1986) 的 bucket brigade 构想演变成与 Sarsa 密切相关的算法。Bucket brigade 的最初想法涉及相互触发的规则链。它专注于将信用从当前规则传递回触发它的规则。随着时间的流逝，bucket brigade 变得更像 TD 学习，将信用回馈给任何暂时在先的规则，而不仅仅是触发当前规则的规则。当以各种自然方式简化后，bucket brigade 的现代形式几乎与一步 Sarsa 完全相同，正如 Wilson (1994) 所详述的。

6.5 Watkins (1989) 引入了 Q-learning，Watkins 和 Dayan (1992) 提出了收敛证明的大纲。Jaakkola, Jordan 和 Singh (1994) 和 Tsitsiklis (1994) 证明了更一般的收敛结果。

6.6 期望 Sarsa 算法是由 George John (1994) 提出的，他将其称为“ \bar{Q} -learning”，并强调了它作为 off-policy 算法优于 Q-learning 的优势。当我们在本书的第一版中将期望 Sarsa 作为练习展示时我们还不知道 John 的工作；或者，当 van Seijen, van Hasselt, Whiteson 和 Weiring (2009) 确定了胜过常规的 Sarsa 和 Q-learning 的期望 Sarsa 的收敛性和条件时。我们的图6.3是根据他们的结果改编而成的。Van Seijen 等将期望 Sarsa 定义为一种 on-policy 使用的方法（就像我们在第一版中所做的那样），而现在我们将此名称用于目标和行为策略可能不同的通用算法。van Hasselt (2011) 指出了期望 Sarsa 的 off-policy 观点，他称其为“General Q-learning”。

6.7 van Hasselt (2010, 2011) 引入并广泛研究了最大化偏差和 double learning。图中的示例 MDP 改编自他的图 4.1 中的示例 (van Hasselt, 2011)。

6.8 后期状态的概念与“post-decision state”的概念相同 (Van Roy, Bertsekas, Lee 和 Tsitsiklis, 1997; Powell, 2011)。

第七章 n 步自举

??

在本章中，我们将前两章中介绍的蒙特卡洛（MC）方法和一步时间差分（TD）方法统一起来。MC 方法和一步 TD 方法都不总是最好的。在本章中，我们介绍了 n 步 TD 方法，它们对这两种方法都进行了概括，以使一种方法可以根据需要平稳地从一种方法转换到另一种方法，以满足特定任务的需求。n 步方法的一端是 MC 方法，另一端是一步 TD 方法。最好的方法通常介于这两个极端之间。

查看 n 步方法的优点的另一种方法是，它们使您摆脱了时间步的专制。使用一步 TD 方法，同一时间步长确定可以更改动作的频率以及完成自举的时间间隔。在许多应用中，人们希望能够非常快地更新动作，以考虑到已更改的所有内容，但是如果自举超过了发生了显著且可识别的状态改变的时间长度，则自举效果最好。对于一步 TD 方法，这些时间间隔是相同的，因此必须做出折衷。n 步方法使自举可以进行多个步骤，从而使我们摆脱了单个时间步长的专制。

通常使用 n 步方法的思想作为资格迹算法思想的介绍（第 12 章），该算法允许同时在多个时间间隔内进行自举。在这里，我们改为自己考虑 n 步自举的想法，将资格迹机制的处理推迟到以后。这使我们可以更好地分离问题，并在更简单的 n 步设置中处理尽可能多的问题。

像往常一样，我们首先考虑预测问题，然后考虑控制问题。也就是说，我们首先考虑 n 步方法如何帮助根据固定策略的状态函数来预测回报（即估算 v_π ）。然后，我们将思想扩展到动作值和控制方法。

7.1 n 步 TD 预测

蒙特卡洛方法和 TD 方法之间的方法空间是什么？考虑从使用 π 生成的样本回合中估计 v_π 。蒙特卡洛方法基于从该状态直到回合结束的整个观察到的奖励序列，为每个状态执行更新。另一方面，一步 TD 方法的更新仅基于下一个奖励，从状态的值自举一步，然后作为剩余奖励的代理。然后，一种中间方法将根据中间数量的奖励执行更新：奖励多于一个，但少于直到终止的全部。例如，两步更新将基于前两步奖励和两步后的状态估计值。同样，我们可以进行三步更新，四步更新，等等。图 7.1 显示了 v_π 的 n 步更新频谱的备份图，左侧为一步 TD 更新，右侧为向上直至终止的蒙特卡洛更新。

使用 n 步更新的方法仍然是 TD 方法，因为它们仍然根据其与后来的估计的差异来更改先前的估计。现在，以后的估计不是以后的一步，而是以后的 n 步。时间差异扩展到 n 步的方法称为 n 步 TD 方法。上一章介绍的 TD 方法都使用了一步更新，这就是为什么我们将它们称为一步 TD 方法。

更正式地，考虑状态-奖励序列 $S_t, R_{t+1}, S_{t+1}, R_{t+2}, \dots, R_T, S_T$ （省略动作）所导致

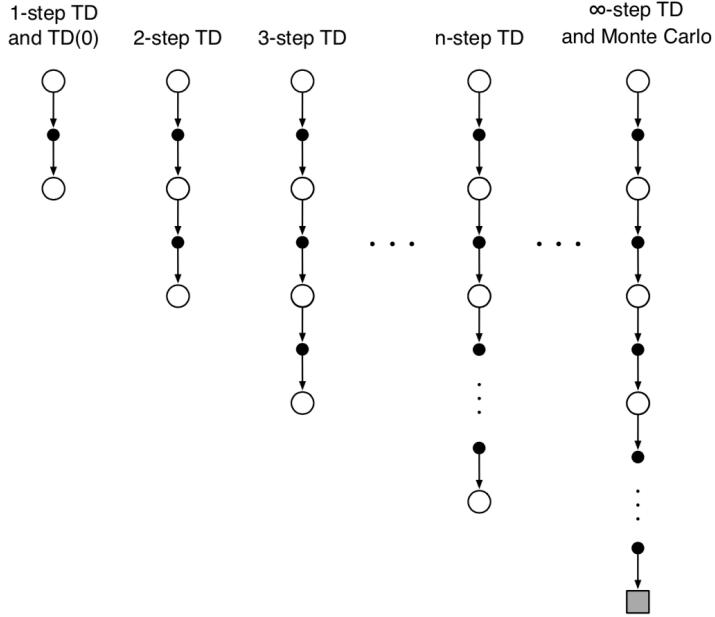


图 7.1: n 步方法的备份图。这些方法形成了从一步 TD 方法到蒙特卡洛方法的频谱范围。

的更新状态 S_t 的估计值。我们知道，在蒙特卡洛更新中，对 $v_\pi(S_t)$ 的估计朝着完全回报的方向更新：

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

其中 T 是回合的最后时间步。让我们将此量称为更新目标。在蒙特卡洛中，更新目标是回报，而在一步更新中，目标是第一个奖励加上下一个状态的折扣估算值，我们称此为一步回报：

$$G_{t:t+1} \doteq R_{t+1} + \gamma V_t(S_{t+1}),$$

其中 $V_t : \mathcal{S} \rightarrow \mathbb{R}$ 这里是 v_π 在时间 t 的估计。 $G_{t:t+1}$ 的下标表示，这是时间 t 的截断回报，使用了直到时间 $t+1$ 的奖励，如上一章所述，折扣估计 $\gamma V_t(S_{t+1})$ 代替了其他项 $\gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$ 的全部回报。现在，我们的观点是，这种想法在经过两个步之后和在一个步之后一样有意义。两步更新的目标是两步回报：

$$G_{t:t+2} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2}),$$

其中，现在 $\gamma^2 V_{t+1}(S_{t+2})$ 纠正缺少项 $\gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$ 的情况。同样，任意 n 步更新的目标是 n 步回报：

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t-n-1}(S_{t+n}), \quad (7.1)$$

对于所有 n, t 使得 $n \geq 1$ 和 $0 \leq t < T - n$ 。所有 n 步回报都可以看作是全部回报的近似值，在 n 步后被截断，然后用 $V_{t-n-1}(S_{t+n})$ 校正剩余的遗漏项。如果 $t+n \geq T$ （如果 n

步回报延伸到终止或超出终止), 则所有缺失项均视为零, 并且 n 步回报定义为等于普通的全部回报 ($G_{t:t+n} \doteq G_t$, 如果 $t + n \geq T$)。

请注意, 对于 $n > 1$ 的 n 步回报涉及从 t 转移到 $t + 1$ 时不可用的未来奖励和状态。在看到 R_{t+n} 和计算 V_{t+n-1} 之前, 任何实际算法都不能使用 n 步回报。这些可用的第一个时间是 $t + n$ 。因此, 使用 n 步回报的自然状态值学习算法是

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha[G_{t:t+n} - V_{t+n-1}(S_t)], \quad t \leq 0 < T, \quad (7.2)$$

而所有其他状态的值保持不变: 对于所有 $s \neq S_t$, $V_{t+n}(s) = V_{t+n-1}(s)$ 。我们将此算法称为 n 步 TD。请注意, 在每个回合的前 $n-1$ 个步骤中根本不做任何更改。为了弥补这一点, 在回合结束时, 终止之后和开始下一个回合之前, 进行了相等数量的其他更新。完整的伪代码在下一页的方框中提供。

n-step TD for estimating $V \approx v_\pi$

输入: 策略 π

算法参数: 步长 $\alpha \in (0, 1]$, 正整数 n

对于所有 $s \in \mathcal{S}$, 任意初始化 $V(s)$

所有存储和访问操作 (对于 S_t 和 R_t) 都可以采用其索引 $\bmod n + 1$

对每个回合循环:

初始化和存储 $S_0 \neq \text{terminal}$

$T \leftarrow \infty$

对于 $t = 0, 1, 2, \dots$ 循环:

如果 $t < T$, 那么:

根据 $\pi(\cdot | S_t)$ 采取动作

观测并存储下一个奖励 R_{t+1} 和下一个状态 S_{t+1}

如果 S_{t+1} 是终止, 那么 $T \leftarrow t + 1$

$\tau \leftarrow t - n + 1$ (τ 是更新其状态估计的时间)

如果 $\tau \geq 0$:

$$G \leq \sum_{i=\tau+1}^{\min(t+n,T)} \gamma^{i-\tau-1} R_i$$

如果 $\tau + n < T$, 那么: $G \leftarrow G + \gamma^n V(S_{\tau+n}) \quad (G_{\tau:\tau+n})$

$$V(S_\tau) \leftarrow V(S_t) + \alpha[G - V(S_\tau)]$$

直到 $\tau = T - 1$

练习 7.1 在第6章中, 我们指出, 如果价值估计在每一步之间都没有变化, 则可以将蒙特卡罗误差写为 TD 误差的总和 (6.6)。证明 (7.2) 中使用的 n 步误差也可以写为 TD 误差总和 (同样, 如果估计值不变), 可以概括先前的结果。 \square

练习 7.2 (programming) 使用 n 步方法时, 值估计的确会逐步变化, 因此使用 TD 误差之和 (参见前面的练习) 代替 (7.2) 中的误差的算法实际上将是一种稍有不同的算法。它

是更好的算法还是更差的算法？设计并编写一个小实验，以凭经验回答这个问题。□

n 步回报使用值函数 V_{t+n-1} 校正 R_{t+n} 以外的缺失奖励。在最差状态下，n 步回报的一个重要特性是，与 V_{t+n-1} 相比，可以保证对 v_π 的期望更好。也就是说，保证了 n 步回报的期望的最差误差小于或等于 γ^n 乘以 V_{t+n-1} 下的最差误差：

$$\max_s |\mathbb{E}_\pi[G_{t:t+n}|S_t = s] - v_\pi(s)| \leq \gamma^n \max_s |V_{t+n-1}(s) - v_\pi(s)|, \quad (7.3)$$

对于所有 $n \geq 1$ 。这称为 n 步回报的误差减少属性。由于减少误差的特性，可以正式证明在适当的技术条件下，所有 n 步 TD 方法都收敛于正确的预测。因此，n 步 TD 方法形成了一系列合理的方法家族，其中一步 TD 方法和蒙特卡洛方法是极端成员。

例 7.1 : n-step TD Methods on the Random Walk 考虑对例6.2（第 125 页）中所述的 5 状态随机行走任务使用 n 步 TD 方法。假设第一个回合直接从中心状态 C 向右经过 D 和 E，然后在右边终止并返回 1。回想一下，所有状态的估计值均始于中间值 $V(s)0.5$ 。由于这种经验，一步法将仅更改最后一个状态 $V(E)$ 的估计值，该估计值将向观察到的回报 1 递增。另一方面，两步方法将增加终止之前的两个状态的值： $V(D)$ 和 $V(E)$ 都将朝 1 递增。三步方法或 $n > 2$ 的任何 n 步方法，则将所有三个访问状态的值都向 1 递增相同的量。

n 的哪个值更好？图7.2显示了针对较大的随机行走过程的简单经验测试的结果，其中有 19 个状态而不是 5 个状态（左边结果为-1，所有值均初始化为 0），在本章中我们将其作为运行示例。显示了 n 步 TD 方法的结果，其中包含 n 和 α 的取值范围。每个参数设置的性能度量（在垂直轴上显示）是 19 个状态在回合结束时的预测与它们的真实值之间的平均平方误差的平方根，然后在前 10 个回合和整个实验重复 100 次中求平均值（所有参数设置均使用相同的行走集合）。注意， n 为中间值的方法效果最好。这说明了将 TD 和蒙特卡洛方法推广到 n 步方法可能比两种极端方法中的任何一种都可能表现更好。■

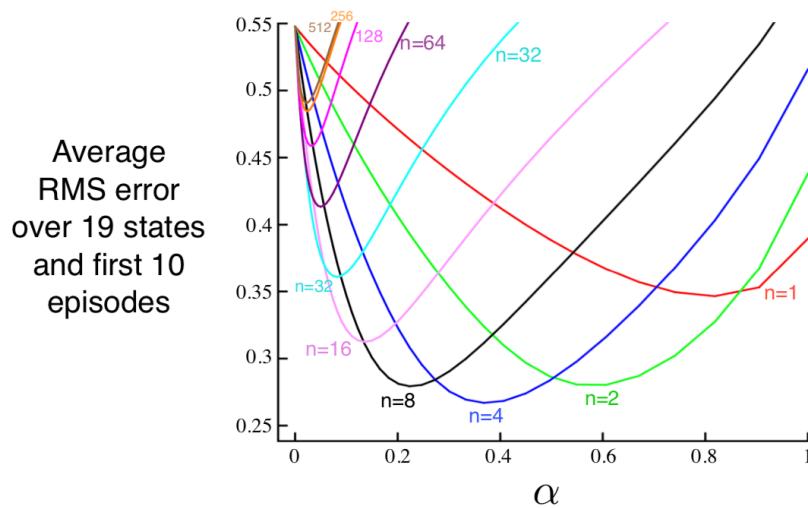


图 7.2: 在 19 状态随机行走任务中，对于 n 的各种值，关于 α 函数的 n 步 TD 方法的性能（例7.1）。

练习 7.3 您认为为什么在本章的示例中要使用较大的随机行走任务（从 19 个状态替代 5

个状态)? 较小的行走路程会将优势转移到 n 的其他值吗? 在较大步行路程中, 左边结果从 0 变为 -1 会怎么样? 您认为这对 n 的最优值有什么影响吗?

□

7.2 n 步 Sarsa

n 步方法不仅可以用于预测, 还可以用于控制吗? 在本节中, 我们说明如何将 n 步方法与 Sarsa 结合起来, 以一种直接的方式产生 on-policy 上的 TD 控制方法。Sarsa 的 n 步版本我们称为 n 步 Sarsa, 因此在上一章中介绍的原始版本此后称为一步 Sarsa 或 Sarsa(0)。

主要思想是简单地为动作 (状态-操作对) 切换状态, 然后使用 ε -greedy 策略。与 n 步 TD 的备份图 (图7.1) 类似, n 步 Sarsa 的备份图 (图7.3) 是一串交替的状态和动作, 除了 Sarsa 都以动作而不是状态开始和结束。我们根据估算的动作值来重新定义 n 步回报 (更新目标):

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}), \quad n \geq 1, 0 \leq t < T-n, \quad (7.4)$$

其中, 如果 $t+n \geq T$, 则 $G_{t:t+n} \doteq G_t$ 。那么自然算法是

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)], \quad 0 \leq t < T, \quad (7.5)$$

对于所有使得 $s \neq S_t$ 或 $a \neq A_t$ 的 s, a , 而所有其他状态的值保持不变: $Q_{t+n}(s, a) = Q_{t+n-1}(s, a)$ 。这就是我们称为 n 步 Sarsa 的算法。伪代码显示在下一页的方框中, 图7.4给出了与一步方法相比可以加快学习速度的示例。

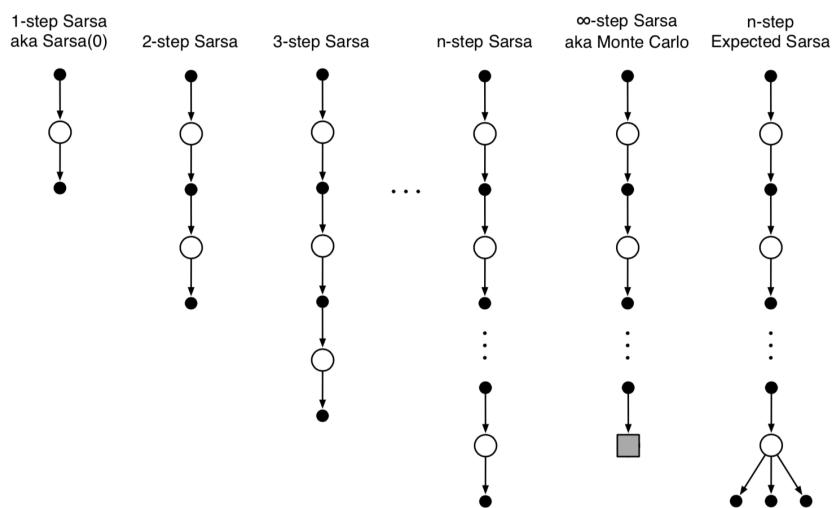


图 7.3: 状态动作值的 n 步方法的频谱备份图。它们的范围包括从 Sarsa(0) 的一步更新到蒙特卡洛方法的直到终止的更新。介于两者之间的是基于 n 步实际奖励和第 n 个下一状态-动作对的估计值的 n 步更新, 所有这些值均已适当折扣。最右边是 n 步期望 Sarsa 的备份图。

n-step Sarsa for estimating $Q \approx q_* \text{ or } q_\pi$

对于所有 $s \in \mathcal{S}, a \in \mathcal{A}$, 任意初始化 $Q(s, a)$

将 π 初始化为相对于 Q 或者给定的固定策略的 ε -greedy

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

所有存储访问操作 (对于 S_t, A_t, R_t) 可以采用其索引 $\bmod n + 1$

对于每个回合循环:

初始化和存储 $S_0 \neq \text{terminal}$

选择和存储动作 $A_0 \sim \pi(\cdot | S_0)$

$T \leftarrow \infty$

对于 $t = 0, 1, 2, \dots$ 循环:

如果 $t < T$, 那么:

采取动作 A_t

观测和存储下一个奖励 R_{t+1} 和下一个状态 S_{t+1}

如果 S_{t+1} 是终止, 那么:

$T \leftarrow t + 1$

否则:

选择和存储动作 $A_{t+1} \sim \pi(\cdot | S_{t+1})$

$\tau \leftarrow t + 1$ (τ 是更新其估计的时间)

如果 $\tau \geq 0$:

$$G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n, T)} \gamma^{i-\tau-1} R_i$$

如果 $\tau + n < T$, 那么 $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ $(G_{\tau:\tau+n})$

$$Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha[G - Q(S_\tau, A_\tau)]$$

如果 π 正在被学习, 则确保 $\pi(\cdot | S_\tau)$ 是关于 Q 的 ε -greedy

直到 $\tau = T - 1$

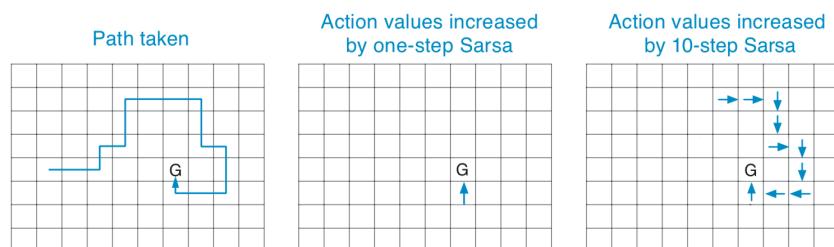


图 7.4: 因使用 n 步方法而加快了策略学习的网格世界示例。第一个面板显示了 agent 在单个回合中采取的路径, 在高奖励位置结束, 由 G 标记。在此示例中, 值最初均为 0, 除 G 处的正奖励外, 所有奖励均为零。其他两个面板中的箭头表示通过一步和 n 步 Sarsa 方法, 此路径增强了哪些动作值。一步法仅增强导致高奖励的动作序列的最后一个动作, 而 n 步法则增强序列的最后 n 个动作, 因此可以从一个回合中学到更多。

练习 7.4 证明 Sarsa (7.4) 的 n 步回报可以精确地写成新的 TD 误差, 如

$$G_{t:t+n} = Q_{t-1}(S_t, A_t) + \sum_{k=t}^{\min(t+n, T)-1} \gamma^{k-t} [R_{k+1} + \gamma Q_k(S_{k+1}, A_{k+1}) - Q_{k-1}(S_k, A_k)]. \quad (7.6)$$

□

那期望 Sarsa 呢？图7.3的最右边显示了期望 Sarsa 的 n 步版本的备份图。就像 n 步 Sarsa 中一样，它由一串线性的样本动作和状态组成，不同之处在于，它的最后一个元素是所有动作可能性的分支，该动作可能性一如既往地通过 π 下的概率加权。该算法可以用与 n 步 Sarsa（上面）相同的方程式来描述，除了将 n 步回报重新定义为

$$G_{t:t+n} \doteq R_{t+1} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{V}_{t+n-1}(S_{t+n}), \quad t+n < T, \quad (7.7)$$

（对于 $t+n \geq T$, $G_{t:t+n} \doteq G_t$ ）其中 $\hat{V}_t(s)$ 是状态 s 的期望近似值，其在目标策略下，使用时间 t 处的动作值估计：

$$\hat{V}_t(s) \doteq \sum_a \pi(a|s) Q_t(s, a), \quad \text{for all } s \in \mathcal{S}. \quad (7.8)$$

在本书的其余部分中，期望的近似值用于开发许多动作值方法。如果 s 为终止，则其期望近似值定义为 0。

7.3 n 步 off-policy 学习

回想一下 off-policy 学习是在学习一个策略 π 的价值函数，而跟随的是另一个策略 b 。通常， π 是当前动作值函数估计的贪婪策略， b 是更具探索性的策略，也许是 ε -greedy。为了使用来自 b 的数据，我们必须考虑两个策略之间的差异，使用它们采取动作的相对概率（请参阅第5.5节）。在 n 步方法中，回报是按 n 步构建的，因此我们只对这 n 个动作的相对概率感兴趣。例如，要制作 n 步 TD 的简单的 off-policy 版本，时间 t 的更新（实际上是在时间 $t+n$ 进行）可以简单地用 $\rho_{t:t+n-1}$ 加权：

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha \rho_{t:t+n-1} [G_{t:t+n} - V_{t+n-1}(S_t)], \quad 0 \leq t < T, \quad (7.9)$$

其中 $\rho_{t:t+n-1}$ ，称为重要性采样率，是在将从 A_t 变为 A_{t+n-1} 的 n 个动作的两种策略下的相对概率（参见方程5.3）：

$$\rho_{t:h} \doteq \prod_{k=t}^{\min(h, T-1)} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}. \quad (7.10)$$

例如，如果任何一个动作都不会被 π 所采取（即 $\pi(A_k|S_k) = 0$ ），则 n 步回报应该被赋予零权重，并且被完全忽略。另一方面，如果碰巧地采取了 π 的发生概率比 b 大得多的动作，那么这将增加原本给予回报的权重。这是有道理的，因为该动作是 π 的特征（因此我们想了解它），但是很少被 b 选择，因此很少出现在数据中。为了弥补这一点，我们必须在它确实发生时对其进行增加权重。请注意，如果两个策略实际上是相同的（on-policy

的情况), 则重要性采样率始终为 1。因此, 我们的新更新 (7.9) 可以概括并完全替代我们之前的 n 步 TD 更新。同样, 我们以前的 n 步 Sarsa 更新可以完全替换为简单的 off-policy 形式:

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha \rho_{t+1:t+n} [G_{t:t+n} - Q_{t+n-1}(S_t, A_t)], \quad (7.11)$$

对于 $0 \leq t < T$ 。请注意, 此处的重要性采样率比 n 步 TD (7.9) 晚一步开始和结束。这是因为在这里我们正在更新状态-动作对。我们不必关心我们选择动作的可能性。现在我们已经选择了它, 我们想从发生的事情中全面学习, 仅对后续动作进行重要性采样。完整算法的伪代码显示在下面的方框中。

Off-policy n-step Sarsa for estimating $Q \approx q_*$ or q_π

输入: 对于所有 $s \in \mathcal{S}, a \in \mathcal{A}$, 任意一个使得 $b(a|s > 0)$ 的行为策略 b

对于所有 $s \in \mathcal{S}, a \in \mathcal{A}$, 任意初始化 $Q(s, a)$

初始化 π 为关于 Q 的贪婪或者固定的给定策略

算法参数: 步长 $\alpha \in (0, 1]$, 正整数 n

所有存储和访问操作 (对于 S_t, A_t, R_t) 都可以采用它们的索引 $\bmod n + 1$

对于每个回合循环:

初始化和存储 $S_0 \neq \text{terminal}$

选择及存储一个动作 $A_0 \sim b(\cdot|S_0)$

$T \leftarrow \infty$

对于 $t = 0, 1, 2, \dots$ 循环:

如果 $t < T$, 那么:

采取动作 A_t

观测和存储下一个奖励 R_{t+1} 和下一个状态 S_{t+1}

如果 S_{t+1} 是终止, 那么:

$T \leftarrow t + 1$

否则:

选择和存储一个动作 $A_{t+1} \sim b(\cdot|S_{t+1})$

$\tau \leftarrow t - n + 1$ (τ 是更新其估计的时间)

如果 $\tau \geq 0$:

$$\rho \leftarrow \prod_{i=\tau+1}^{\min(\tau+n-1, T-1)} \frac{\pi(A_i, S_i)}{b(A_i, S_i)} \quad (\rho_{\tau+1:t+n-1})$$

$$G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n, T)} \gamma^{i-\tau-1} R_i$$

$$\text{如果 } \tau + n < T, \text{ 那么 } G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n}) \quad (G_{\tau:\tau+n})$$

$$Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha \rho [G - Q(S_\tau, A_\tau)]$$

如果 π 正在被学习, 那么确保 $\pi(\cdot|S_\tau)$ 关于 Q 是贪婪的

直到 $\tau = T - 1$

n 步期望 Sarsa 的 off-policy 版本将对 n 步 Sarsa 使用与上述相同的更新, 不同之处在



于重要性采样率要少一个因子。也就是说，上面的等式将使用 $\rho_{t+1:t+n-1}$ 而不是 $\rho_{t+1:t+n}$ ，并且当然会使用 n 步回报的期望 Sarsa 版本 (7.7)。这是因为在期望 Sarsa 中，所有可能的动作在最后一个状态下都被考虑在内；实际采取的动作没有效果，也不必进行校正。

7.4 * 控制变量的每决策方法

上一节中介绍的多步 off-policy 方法很简单，概念上也很明确，但可能不是最有效的。一种更复杂的方法将使用第5.9节中介绍的每个决策的重要性采样思想。为了理解这种方法，首先要注意的是，与所有回报一样，普通的 n 步回报 (7.1) 可以递归地编写。对于在 h 处结束的 n 个步， n 步回报则可以写为

$$G_{t:h} = R_{t+1} + \gamma G_{t+1:h}, \quad t < h < T, \quad (7.12)$$

其中 $G_{h:h} \doteq V_{h-1}(S_h)$ （回想一下，此回报是在时间 h 处使用的，先前表示为 $t+n$ ）。现在考虑遵循与目标策略 π 不同的行为策略 b 的效果。所有得到的经验，包括第一个奖励 R_{t+1} 和下一个状态 S_{t+1} ，都必须通过时间 t 的重要采样率 $\rho_t = \frac{\pi(A_t, S_t)}{b((A_t, S_t))}$ 加权。人们可能会想简单地加权上述方程式的右侧，但是可以做得更好。假设在时间 t 处的动作永远不会被 π 选择，因此 ρ_t 为零。然后，简单的加权将导致 n 步回报为零，这在将其用作目标时可能会导致高方差。取而代之的是，在这种更为复杂的方法中，使用对 n 步回报的另一种 off-policy 定义，该回报以 h 结束，如

$$G_{t:h} \doteq \rho_t [R_{t+1} + \gamma G_{t+1:h}] + (1 - \rho_t) [V_{h-1}(S_t)], \quad t < h < T, \quad (7.13)$$

其中同样 $G_{h:h} = V_{h-1}(S_h)$ 。在此方法中，如果 ρ_t 为零，则目标与估计相同且不会引起任何变化，而不是使目标缩小为零并使估计值缩小。重要采样率为零意味着我们应该忽略样本，因此保持估计不变似乎是适当的。另外，(7.13) 中的附加项称为控制变量（原因不明）。注意，控制变量不会改变期望更新；重要性采样率的期望值为 1 (第5.9节)，与估计值不相关，因此控制变量的期望值为零。还要注意，off-policy 定义 (7.13) 是对 n 步回报 (7.1) 的较早 on-policy 定义的严格概括，因为在 on-policy 情况下两者相同，其中 ρ_t 始终为 1。

对于常规的 n 步方法，与 (7.13) 结合使用的学习规则是 n 步 TD 更新 (7.2)，除了嵌入在回报中的那些之外，没有其他明确的重要性采样率。

 **练习 7.5** 编写上述 off-policy 状态值预测算法的伪代码。 □

对于动作值， n 步回报的策略定义有些不同，因为第一个动作在重要性采样中不起作用。第一步是正在学习的。在目标策略下是否不可能或者甚至不可能都没关系，已经采取了它，现在必须对奖励和遵循它的状态赋予全部的单位权重。重要性采样仅适用于其后的动作。

首先请注意，对于动作值，以 h 结尾的 n 步 on-policy 回报可以像在 (7.12) 中那样以递归方式编写期望值 (7.7)，除了对于动作值而言，递归以 $G_{h:h} \doteq \hat{V}_{h-1}(S_h)$ 结束，如

(7.8)。具有控制变量的 off-policy 形式为

$$\begin{aligned} G_{t:h} &\doteq R_{t+1} + \gamma \left(\rho_{t+1} G_{t+1:h} + \hat{V}_{h-1}(S_{t+1}) - \rho_{t+1} Q_{h-1}(S_{t+1}, A_{t+1}) \right), \\ &= R_{t+1} + \gamma \rho_{t+1} (G_{t+1:h} - Q_{h-1}(S_{t+1}, A_{t+1})) + \gamma \hat{V}_{h-1}(S_{t+1}), t < h \leq T \end{aligned} \quad (7.14)$$

如果 $h < T$, 则递归以 $G_{h:h} \doteq Q_{h-1}(S_h, A_h)$ 结束, 而如果 $h \geq T$, 则递归以 $G_{T-1:h} \doteq R_T$ 结束。所得的预测算法 (与 (7.5) 结合后) 类似于期望 Sarsa。

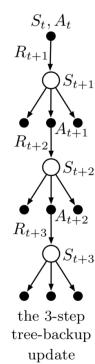
- ✍ 练习 7.6 证明上述方程中的控制变量不改变回报的期望值。 □
- ✍ 练习 7.7 编写上面刚描述的 off-policy 动作值预测算法的伪代码。在到达地平线或回合结束时, 请特别注意递归的终止条件。 □
- ✍ 练习 7.8 证明, 如果近似状态值函数不变, 则 n 步回报 (7.13) 的通用 (off-policy) 版本仍可以精确紧凑地编写为基于状态的 TD 误差之和 (6.5)。 □
- ✍ 练习 7.9 对 off-policy 的 n 步回报 (7.14) 的动作版本和期望 Sarsa 的 TD 误差 (方程6.9中括号中的量) 重复上述练习。 □
- ✍ 练习 7.10 (programming) 设计一个小的 off-policy 预测问题, 并使用它来证明, 使用 (7.13) 和 (7.2) 的 off-policy 学习算法比使用 (7.1) 和 (7.9) 的简单算法更有效。 □

我们在本节, 上一节和第5章中使用的重要性采样可以进行合理的 off-policy 学习, 但也会导致高方差更新, 从而迫使使用较小的步长参数, 从而导致学习缓慢。Off-policy 训练比 on-policy 训练慢是不可避免的, 毕竟, 数据与所学内容的相关性较低。但是, 这些方法可能还可以进行改进。控制变量是减少方差的一种方法。另一个方法是使步长快速适应所观察到的方差, 如 Autostep 方法 (Mahmood, Sutton, Degris 和 Pilarski, 2012)。另一个有希望的方法是 Karampatziakis 和 Langford (2010) 的不变更新, 由 Tian (准备中) 扩展到 TD。Mahmood 的使用技术 (2017; Mahmood 和 Sutton, 2015) 也可能是解决方案的一部分。在下一节中, 我们考虑一种不使用重要性采样的 off-policy 学习方法。

7.5 无重要性采样的 off-policy 学习: n 步树备份算法

没有重要性采样可以进行 off-policy 学习吗? 第6章中的 Q-learning 和 Expected Sarsa 针对一步情况做了这项工作, 但是有没有相应的多步算法呢? 在本节中, 我们将介绍一种这样的 n 步方法, 称为树备份算法。

该算法的思想由右图所示的 3 步树备份图提出。沿着中心脊柱往下, 在图中标注的是三个采样状态和奖励, 以及两个采样动作。这些是随机变量, 表示发生在初始状态-动作对 S_t, A_t 之后的事件。悬在每个状态的两侧是未选择的动作 (对于最后一个状态, 所有动作都被视为尚未选择)。因为我们没有未选择动作的样本数据, 所以我们自举并使用它们的价值估计来形成更新目标。这稍微扩展了备份图的概念。到目前为止, 我们始终将图顶部的节点的估计值更新为一个目标, 该目标结合沿途的奖励 (适当折扣) 和底部节点的估计值。在树备份更新中, 目标包括所有这些内容, 以及所有级别的悬



挂在两边的悬挂动作节点的估计值。这就是为什么它被称为树备份更新的原因。这是对整个估计动作值的树的更新。

更确切地说，更新来自树的叶节点的估计动作值。内部的动作节点（与实际采取的动作相对应）不参与。每个叶节点对目标的贡献权重与目标策略 π 下的发生概率成正比。因此，每个第一级动作 a 贡献的权重为 $\pi(a|S_{t+1})$ ，除了实际采取的动作 A_{t+1} 根本没有贡献。它的概率 $\pi(A_{t+1}|S_{t+1})$ 用于加权所有第二级动作值。因此，每个未选择的第二级动作 a' 贡献的权重为 $\pi(A_{t+1}|S_{t+1})\pi(a'|S_{t+2})$ 。每个第三级动作贡献的权重为 $\pi(A_{t+1}|S_{t+1})\pi(A_{t+2}|S_{t+2})\pi(a''|S_{t+3})$ ，依此类推。这就好像图中指向动作节点的每个箭头都由该动作在目标策略下被选择的概率来加权，并且，如果该动作下方有一棵树，则该权重将应用于该树中的所有叶节点。

我们可以认为 3 步树备份更新由 6 个半步组成，在从动作到后续状态的样本半步和从该状态考虑所有可能的动作及其该策略下发生的概率的期望半步之间交替。

现在让我们为 n 步树备份算法建立详细的方程。一步回报（目标）与 Expected Sarsa 相同，

$$G_{t:t+1} \doteq R_{t+1} + \gamma \sum_a \pi(a|S_{t+1})Q_t(S_{t+1}, a), \quad (7.15)$$

对于 $t < T - 1$ ，两步树备份回报为

$$\begin{aligned} G_{t:t+1} &\doteq R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1})Q_{t+1}(S_{t+1}, a) \\ &\quad + \gamma \pi(A_{t+1}|S_{t+1}) \left(R_{t+2} + \gamma \sum_a \pi(a|S_{t+2})Q_{t+1}(S_{t+2}, a) \right) \\ &= R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1})Q_{t+1}(S_{t+1}, a) + \gamma \pi(A_{t+1}|S_{t+1})G_{t+1:t+2}, \end{aligned}$$

对于 $t < T - 2$ 。后一种形式提供了对树备份 n 步回报的一般递归定义：

$$G_{t:t+n} \doteq R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1})Q_{t+n-1}(S_{t+1}, a) + \gamma \pi(A_{t+1}|S_{t+1})G_{t+1:t+n}, \quad (7.16)$$

对于 $t < T - 1, n \geq 2, n = 1$ 的情况由 (7.15) 处理，除了 $G_{T-1:T+n} \doteq R_T$ 。然后将此目标与 n 步 Sarsa 中常规动作值更新规则一起使用：

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)],$$

对于 $0 \leq t < T$ ，而所有其他状态-动作对的值保持不变： $Q_{t+n}(s, a) = Q_{t+n-1}(s, a)$ ，对于所有使得 $s \neq S_t, a \neq A_t$ 的 s, a 。此算法的伪代码显示在下一页的方框中。

 **练习 7.11** 说明如果近似动作值不变，则树备份回报 (7.16) 可以写为基于期望的 TD 误差之和：

$$G_{t:t+n} = Q(S_t, A_t) + \sum_{k=t}^{\min(t+n-1, T-1)} \delta_k \prod_{i=t+1}^k \gamma \pi(A_i|S_i),$$

其中 $\delta_t \doteq R_{t+1} + \gamma \bar{V}_t(S_{t+1}) - Q(S_t, A_t)$, 以及 \bar{V}_t 由 (7.8) 给定。 \square

n -step Tree Backup for estimating $Q \approx q_*$ or q_π

对于所有 $s \in \mathcal{S}, a \in \mathcal{A}$, 任意初始化 $Q(s, a)$

将 π 初始化为相对于 Q 的贪婪, 或固定的给定策略

算法参数: 步长 $\alpha \in (0, 1]$, 正整数 n

所有存储和访问操作都可以采用其索引 $\bmod n + 1$

对于每个回合循环:

初始化和存储 $S_0 \neq \text{terminal}$

根据 S_0 的函数任意选择一个动作 A_0 ; 存储 A_0

$T \leftarrow \infty$

对于 $t = 0, 1, 2, \dots$ 循环:

如果 $t < T$:

采取动作 A_t ; 观测并存储下一个奖励和状态 R_{t+1}, S_{t+1}

如果 S_{t+1} 是终止:

$T \leftarrow t + 1$ 否则:

根据 S_{t+1} 的函数任意选择一个动作 A_{t+1} ; 存储 A_{t+1}

$\tau \leftarrow t + 1 - n$ (τ 是更新其估计的时间)

如果 $\tau \geq 0$:

如果 $t + 1 \geq T$:

$G \leftarrow R_T$

否则:

$G \leftarrow R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a)$

从 $k = \min(t, T - 1)$ 到 $\tau + 1$ 循环:

$G \leftarrow R_k + \gamma \sum_{a \neq A_k} \pi(a|S_k) Q(S_k, a) + \gamma \pi(A_k|S_k) G$

$Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha[G - Q(S_\tau, A_\tau)]$

如果 π 正在被学习, 那么确保 $\pi(\cdot|S_\tau)$ 是关于 Q 的贪婪

直到 $\tau = T - 1$

7.6 * 一种统一算法: $Q(\sigma)$

到目前为止, 在本章中, 我们已经考虑了三种不同的动作值算法, 分别对应于图7.5中所示的前三个备份图。 n 步 Sarsa 具有所有样本转移, 树备份算法具有无需采样的完全分支的状态到动作的转移, n 步 Expected Sarsa 具有除最后一个状态到动作之外的所有样本转移, 最后一个状态到动作转移是完全分支的期望值。这些算法可以统一到什么程度?

图7.5中的第四个备份图提出了一种统一的想法。这是一种想法, 即可以逐步决定是

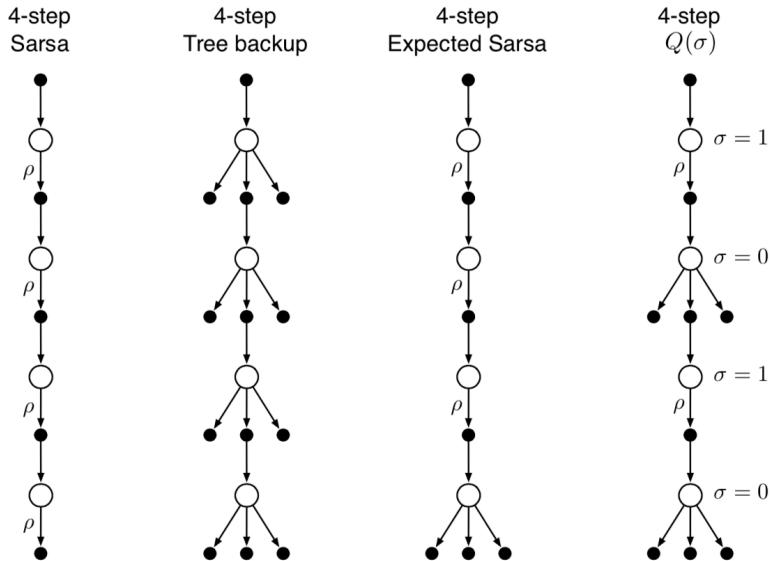


图 7.5: 本章到目前为止已经考虑的三种 n 步动作值更新的备份图 (4 步情况), 以及将它们全部统一的第四种更新的备份图。 ρ 表示在 off-policy 情况下需要进行重要性采样的半转移。第四种更新通过在每个状态下选择是否采样 ($\sigma_t = 1$) 或不采样 ($\sigma_t = 0$) 来统一所有其他更新。

要像 Sarsa 那样将动作作为样本, 还是要像对树备份更新那样考虑对所有动作的期望。然后, 如果选择始终采样, 则将获得 Sarsa, 而如果选择从不采样, 则将获得树备份算法。Expected Sarsa 会选择除最后一步以外的所有步骤进行采样。当然, 如图中最后一张图所示, 还有许多其他可能性。为了进一步增加可能性, 我们可以考虑采样和期望之间的连续变化。令 $\sigma_t \in [0, 1]$ 表示步骤 t 上的采样程度, 其中 $\sigma(1)$ 表示完全采样, 而 $\sigma(0)$ 表示没有采样的纯期望。可以在时间 t 将随机变量 σ_t 设置为状态, 动作或状态-动作对的函数。我们称这种新算法为 n 步 $Q(\sigma)$ 。

现在让我们开发 n 步 $Q(\sigma)$ 的方程。首先, 根据水平 $h = t + n$, 然后根据期望的近似值 \bar{V} (7.8), 写出树备份的 n 步回报 (7.16):

$$\begin{aligned} G_{t:h} &= R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{h-1}(S_{t+1}, a) + \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:h} \\ &= R_{t+1} + \gamma \bar{V}_{h-1}(S_{t+1}) - \gamma \pi(A_{t+1}|S_{t+1}) Q_{h-1}(S_{t+1}, A_{t+1}) + \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:h} \\ &= R_{t+1} + \gamma \pi(A_{t+1}|S_{t+1}) (G_{t+1:h} - Q_{h-1}(S_{t+1}, A_{t+1})) + \gamma \bar{V}_{h-1}(S_{t+1}), \end{aligned}$$

之后, 除了用动作概率 $\pi(A_{t+1}|S_{t+1})$ 代替重要性采样率 ρ_{t+1} 之外, 它与控制变量 (7.14) 的 Sarsa 的 n 步回报完全类似。对于 $Q(\sigma)$, 我们在这两种情况之间线性滑动:

$$\begin{aligned} G_{t:h} &\doteq R_{t+1} + \gamma (\sigma_{t+1} \rho_{t+1} + (1 - \rho_{t+1}) \pi(A_{t+1}|S_{t+1})) (G_{t+1:h} - Q_{h-1}(S_{t+1}, A_{t+1})) \\ &\quad + \gamma \bar{V}_{h-1}(S_{t+1}), \end{aligned} \tag{7.17}$$

对于 $t < h \leq T$ 。如果 $h < T$, 则递归以 $G_{h:h} \doteq Q_{h-1}(S_h, A_h)$ 结束; 如果 $h = T$, 则以 $G_{T-1:T} \doteq R_T$ 结束。然后, 我们对 n 步 Sarsa (7.11) 使用常规更新 (off-policy)。方框中提供了完整的算法。

Off-policy n -step $Q(\sigma)$ for estimating $Q \approx q_*$ or q_π

输入: 对于所有 $s \in \mathcal{S}, a \in \mathcal{A}$, 一个任意的行为策略 b 使得 $b(a|s) > 0$

对于所有 $s \in \mathcal{S}, a \in \mathcal{A}$, 任意初始化 $Q(s, a)$

将 π 初始化为相对于 Q 的贪婪, 或固定的给定策略

算法参数: 步长 $\alpha \in (0, 1]$, 小的 $\epsilon > 0$, 一个正整数 n

所有存储和访问操作都可以采用其索引 $\text{mod } n + 1$

对于每个回合循环:

初始化和存储 $S_0 \neq \text{terminal}$

选择并存储动作 $A_0 \sim b(\cdot|S_0)$

$T \leftarrow \infty$

对于 $t = 0, 1, 2, \dots$ 循环:

如果 $t < T$:

采取动作 A_t ; 观测并存储下一个奖励和状态 R_{t+1}, S_{t+1}

如果 S_{t+1} 是终止:

$T \leftarrow t + 1$

否则:

选择并存储动作 $A_{t+1} \sim b(\cdot|S_{t+1})$

选择并存储 σ_{t+1}

存储 $\frac{\pi(A_{t+1}, S_{t+1})}{b(A_{t+1}, S_{t+1})}$ 为 ρ_{t+1}

$\tau \leftarrow t + 1 - n$ (τ 是更新其估计的时间)

如果 $\tau \geq 0$:

如果 $t + 1 < T$:

$G \leftarrow Q(S_{t+1}, A_{t+1})$

从 $k = \min(t, T - 1)$ 到 $\tau + 1$ 循环:

如果 $k = T$:

$G \leftarrow R_T$

否则:

$\bar{V} \leftarrow \sum_a \pi(a|S_k)Q(S_k, a)$

$G \leftarrow R_k + \gamma(\sigma_k \rho_k + (1 - \sigma_k)\pi(S_k, A_k)(G - Q(S_k, A_k)) + \gamma \bar{V}$

$Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha[G - Q(S_\tau, A_\tau)]$

如果 π 正在被学习, 那么确保 $\pi(\cdot|S_\tau)$ 是关于 Q 的贪婪

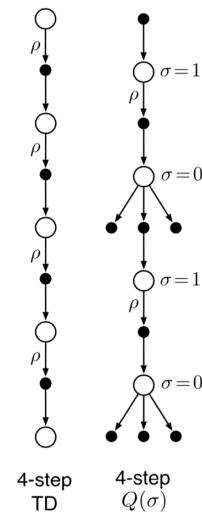
直到 $\tau = T - 1$

7.7 总结

在本章中，我们开发了一系列时间差分学习方法，它们介于前一章的一步式 TD 方法和前前一章的蒙特卡洛方法之间。涉及中间量 bootstrapping 的方法非常重要，因为它们通常会比两种极端方法都有更好的表现。

在本章中，我们的重点是 n 步方法，它们展望下 n 步的奖励，状态和动作。右边的两个 4 步备份图一起总结了大多数介绍的方法。所示状态值更新用于具有重要性采样的 n 步 TD，而动作值更新则用于 n 步 $Q(\sigma)$ ，从而概括了 Expected Sarsa 和 Q-learning。所有 n 步方法都需要在更新之前延迟 n 个时间步，因为只有这样才能知道所有必需的将来事件。另一个缺点是，与以前的方法相比，它们每个时间步涉及更多的计算。与单步方法相比， n 步方法还需要更多的内存来记录最近 n 个时间步的状态，动作，奖励，有时还包括其他变量。最终，在第 12 章中，我们将看到如何使用资格迹在最小的内存和计算复杂度的情况下实现多步 TD 方法，但是除一步法之外，总会有一些额外的计算。为了逃避单一时间步骤的专制，这样的代价是值得付出的。

尽管 n 步方法比使用资格迹的方法要复杂得多，但是它们有概念清晰的巨大好处。我们试图通过在 n 步情况下开发两种进行 off-policy 学习的方法来利用这一优势。一种基于重要性抽样，从概念上讲很简单，但方差可能很大。如果目标和行为策略非常不同，它可能需要一些新的算法思想，然后才能变得有效和实用。另一个基于树备份更新，是将 Q-learning 自然扩展到具有随机目标策略的多步骤情况。它不涉及重要性抽样，但是，如果目标和行为策略实质上不同，则即使 n 很大，bootstrapping 过程也可能仅跨越几个步骤。



7.8 书目与历史评论

n 步回报的概念归功于 Watkins (1989)，他还首先讨论了其减少误差的性质。在本书的第一版中对 n 步算法进行了探讨，将 n 步算法视为具有概念意义，但在实践中不可行。Cichosz (1995) 尤其是 van Seijen (2016) 的工作表明，它们实际上是完全实用的算法。鉴于此，以及它们的概念清晰性和简单性，我们选择在第二版中重点介绍它们。特别是，我们现在将对后视图和资格迹的所有讨论推迟到第 12 章。

7.1-7.2 本文是根据 Sutton (1988) 和 Singh and Sutton (1996) 的工作得出的随机行走示例的结果。在本章中，使用备份图描述这些算法和其他算法是全新的。

7.3-7.5 这些部分的发展基于 Precup, Sutton 和 Singh (2000), Precup, Sutton 和 Dasgupta (2001) 以及 Sutton, Mahmood, Precup 和 van Hasselt (2014) 的工作。

树备份算法是由于 Precup, Sutton 和 Singh (2000) 提出的，但是此处的介绍是新的。

7.6 $Q(\sigma)$ 算法对于本文来说是新的，但 De Asis, Hernandez-Garcia, Holland 和 Sutton (2017) 进一步探索了其密切相关的算法。



第八章 表格法进行规划和学习

在本章中，我们将对需要环境模型的强化学习方法（如动态规划和启发式搜索）以及无需模型即可使用的方法（如蒙特卡洛和时间差分方法）进行统一介绍。这些分别称为 **model-based** 和 **model-free** 的强化学习方法。**Model-based** 的方法主要依赖于规划作为其主要组成部分，而 **model-free** 的方法主要依赖于学习。尽管这两种方法之间确实存在差异，但也有很多相似之处。特别是，这两种方法的核心都是价值函数的计算。此外，所有方法都基于对未来事件的预测，计算备份值，然后将其用作近似值函数的更新目标。在本书的前面，我们提出了蒙特卡洛方法和时间差分方法作为不同的替代方法，然后说明了如何通过 n 步方法来统一它们。本章中的目标是将 **model-based** 的方法和 **model-free** 的方法进行类似的集成。在前面的章节中，我们已经确定了它们的区别，现在我们来探索它们可以混合在一起的程度。

8.1 模型和规划

所谓环境模型，我们表示 **agent** 可以用来预测环境将如何响应其动作的任何事物。在给定状态和动作的情况下，模型会生成结果的下一个状态和下一个奖励的预测。如果模型是随机的，则存在多个可能的下一状态和下一奖励，每一种都有一定的发生概率。有些模型描述了所有可能性及其概率；这些我们称为分布模型。其他模型仅产生一种可能性，并根据概率进行采样；这些我们称为样本模型。例如，考虑对一打骰子的总和进行建模。分布模型将产生所有可能的总和及其发生概率，而样本模型将产生根据该概率分布得出的单个总和。动态规划中假设的模型（即 MDP 的动态估计 $p(s', r|s, a)$ ）是一种分布模型。第5章中二十一点示例中使用的模型类型是样本模型。分布模型比样本模型更强大，因为它们始终可以用于生成样本。但是，在许多应用中，获得样本模型比分布模型要容易得多。掷一打骰子就是一个简单的例子。编写计算机程序来模拟掷骰子并返回总和是很容易的，但是要找出所有可能的总和及其概率，就变得更加困难且更容易出错。

模型可用于模仿或模拟经验。给定一个起始状态和一个动作，样本模型会产生一个可能的转移，而分布模型会生成所有可能的转移，这些转移都由它们的发生概率加权。给定开始状态和策略，样本模型可以产生整个回合，而分布模型可以产生所有可能的回合及其概率。无论哪种情况，我们都可以说该模型用于模拟环境并产生模拟经验。

规划一词在不同领域中以多种不同方式使用。我们使用该术语来指代任何将模型作为输入并产生或改进与建模环境进行交互的策略的计算过程：

$$\text{model} \xrightarrow{\text{planning}} \text{policy}$$

在人工智能中，根据我们的定义，有两种不同的规划方法。状态空间规划（包括我们在本书中采用的方法）主要被视为在状态空间中搜索最优策略或通向目标的最优路径。动

作会导致状态之间的转换，而值函数根据状态进行计算。在我们所谓的规划空间规划中，规划是对规划空间的搜索。运算将一个规划转换为另一个规划，并在规划范围内定义价值函数（如果有）。规划空间规划包括进化方法和“部分顺序计划”，这是人工智能中一种常见的规划，其中步骤的顺序未在规划的所有阶段完全确定。规划空间方法很难有效地应用于强化学习中重点关注的随机顺序决策问题，因此我们不再对其进行进一步考虑（例如，参见 Russell 和 Norvig, 2010 年）。

我们在本章中提出的统一观点是，所有状态空间规划方法都具有相同的结构，该结构也存在于本书介绍的学习方法中。需要本章的其余部分来发展这种观点，但是有两个基本思想：（1）所有状态空间规划方法都涉及计算价值函数，这是改进策略的关键中间步骤；（2）它们计算价值函数通过将更新或备份操作应用于模拟经验。这种通用结构可以用以下方式表示：

$$\text{model} \longrightarrow \text{simulated experience} \xrightarrow{\text{backups}} \text{values} \longrightarrow \text{policy}$$

动态规划方法显然适合这种结构：它们遍历状态空间，为每个状态生成可能的转移的分布。然后，每个分布都用于计算备份值（更新目标）以及更新状态的估算值。在本章中，我们认为其他各种状态空间规划方法也适用于此结构，其中各个方法的区别仅在于它们执行的更新类型，执行它们的顺序以及备份的信息保留的时间长短。

以这种方式看待规划方法强调了它们与我们在本书中描述的学习方法的关系。学习和规划方法的核心都是通过备份更新操作来评估价值函数。区别在于，规划使用模型产生的模拟经验，而学习方法则使用环境产生的真实经验。当然，这种差异会导致许多其他差异，例如，如何评估性能以及如何产生灵活的经验。但是，通用的结构意味着可以在规划和学习之间转移许多思想和算法。特别地，在许多情况下，学习算法可以代替规划方法的关键更新步骤。学习方法只需要经验作为输入，在许多情况下，它们可以应用于模拟经验，也可以应用于真实经验。下面的方框显示了基于一步式表格 Q-learning 和来自样本模型的随机样本的规划方法的简单示例。我们将这种方法称为随机样本的一步式表格 Q 规划，在与一步式表格 Q-learning 收敛至实际环境的最优策略相同的条件下（每个状态必须在步骤 1 中无限次选择一个动作对，并且 α 必须随时间适当减小），其收敛到模型的最优策略。

Random-sample one-step tabular Q-planning

永久循环：

1. 随机选择一个状态 $s \in \mathcal{S}$ 和一个动作 $a \in \mathcal{A}(s)$
2. 发送 S, A 到一个样本模型，并获取
一个下一奖励样本 R 和一个下一状态样本 S'
3. 应用一步表格 Q-learning 到 S, A, R, S' :

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

除了对规划和学习方法的统一看法外，本章的第二个主题是进行小规模渐进规划的好处。这使得规划可以在任何时间被中断或重定向，而几乎不会浪费计算量，这似乎是



有效地将规划与行动和学习模型相结合的关键要求。如果问题太大而无法精确解决，那么即使是纯规划问题，非常小步骤的规划也是最有效的方法。

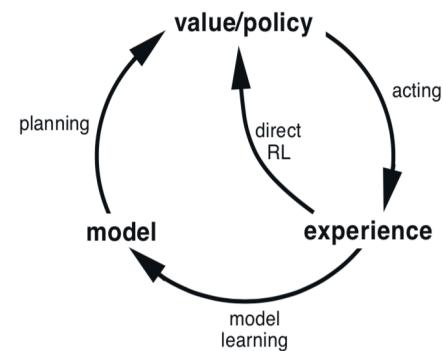
8.2 Dyna: 综合规划、行动和学习

在线进行规划时，在与环境互动的同时，会出现许多有趣的问题。从交互中获得的新信息可能会更改模型，从而与规划进行交互。可能需要以某种方式针对当前正在考虑或不久的将来的状态或决策来定制规划过程。如果决策和模型学习都是计算密集型过程，则可能需要在它们之间分配可用的计算资源。为了开始探讨这些问题，在本节中，我们介绍 Dyna-Q，这是一个简单的体系结构，集成了在线规划 agent 所需的主要功能。在 Dyna-Q 中，每个功能都以简单，几乎是平凡的形式出现。在随后的章节中，我们将详细介绍一些实现每种功能的替代方法以及它们之间的折衷方案。目前，我们仅试图说明想法并激发您的直觉。

在一个规划 agent 中，至少要有两个角色用于实际经验：可以用来改进模型（使其更准确地匹配实际环境），以及可以用来直接改善价值函数和策略，其使用我们已经在前面的章节中讨论过强化学习方法。前者称为模型学习，后者称为直接强化学习（direct RL）。经验，模型，价值和策略之间的可能关系在右图中总结。每个箭头表示影响和假定的改进之间的关系。注意经验如何通过模型直接或间接改善价值函数和策略。规划中涉及的是后者，有时也称为间接强化学习。

直接和间接方法都有优点和缺点。间接方法通常会充分利用有限的经验，从而通过较少的环境交互作用实现更好的策略。另一方面，直接方法要简单得多，并且不受模型设计中偏差的影响。有些人认为间接方法总是优于直接方法，而另一些人则认为直接方法对大多数人和动物的学习负责。心理学和人工智能方面的相关辩论关注认知相对于试错学习的相对重要性，以及考虑性规划相对于反应决策的相对重要性（有关心理学中某些问题的讨论，请参阅第 14 章）。我们的观点是，所有这些辩论中的替代方案之间的对比都被夸大了，认识到这两个方面之间的相似性而不是反对它们可以得到更多的见识。例如，在本书中，我们强调了动态规划和时间差分方法之间的深层相似性，即使其中一种是为规划而设计，而另一种是为无模型的学习而设计。

Dyna-Q 包括上图所示的所有过程，即规划、行动、模型学习和直接 RL，所有这些都连续发生。规划方法是第 161 页的随机样本单步表格 Q-planning 方法。直接 RL 方法是单步表格 Q-learning。模型学习方法也是基于表的，并且假定环境是确定性的。在每次转移 $S_t, A_t \rightarrow R_{t+1}, S_{t+1}$ 之后，该模型在其 S_t, A_t 的表项中记录了 R_{t+1}, S_{t+1} 将确定遵循的预测。因此，如果使用以前经历过的状态-动作对来查询模型，则它仅返回上次观察到的下一个状态和下一个奖励作为其预测。在规划过程中，Q-planning 算法仅从先前经



历过的状态-动作对中随机采样（在步骤 1 中），因此永远不会向模型查询没有信息的对。

Dyna agent 的整体体系结构如图8.1所示，其中 Dyna-Q 算法是其中一个示例。中间一栏代表 agent 与环境之间的基本交互，从而产生了真实经验的轨迹。图左侧的箭头表示直接强化学习，基于实际经验进行操作，以改善价值函数和策略。右边是基于模型的过程。该模型是从实际经验中学习，并产生模拟经验。我们使用搜索控制一词来指代为模型生成的模拟经验选择起始状态和动作的过程。最后，通过将强化学习方法应用于模拟经验就可以实现规划，就像模拟经验确实发生过一样。通常，与 Dyna-Q 中一样，相同的强化学习方法既可用于从实际经验中学习，也可以用于从模拟经验中进行规划。因此，强化学习方法是学习和规划的“最终共同路径”。学习和规划是深度集成的，因为它们几乎共享所有相同的机制，而只是在经验的来源上有所不同。

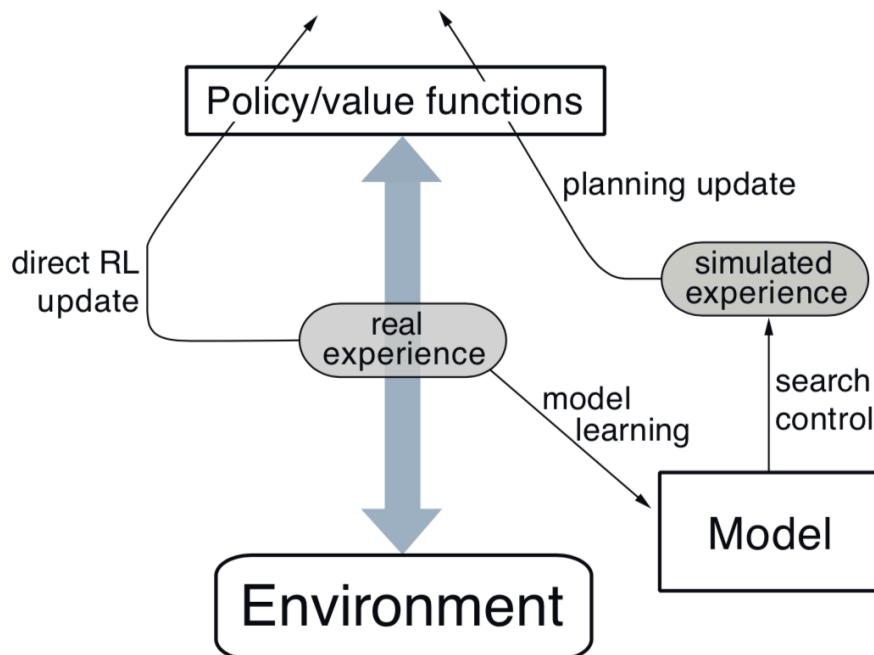


图 8.1: 通用的 Dyna 架构。在环境和策略之间来回传递的真实经验会以与环境模型所生成的模拟经验几乎相同的方式影响策略和价值函数。

从概念上讲，在 Dyna agent 中，规划、行动、模型学习和直接 RL 同时发生并行发生。但是，为了在串行计算机上实现具体性和可执行性，我们完全指定了它们在一个时间步中发生的顺序。在 Dyna-Q 中，行动、模型学习和直接 RL 流程几乎不需要计算，并且我们假设它们仅消耗一小部分时间。每个步骤中剩余的时间都可以用于规划过程，这本质上是计算密集型的。让我们假设在行动、模型学习和直接 RL 之后的每一步中都有时间来完成 Q-planning 算法的 n 次迭代（步骤 1-3）。在下面的方框中针对 Dyna-Q 的伪代码算法中， $Model(s, a)$ 表示状态-动作对 (s, a) 的内容（预测的下一个状态和奖励）。直接强化学习、模型学习和规划分别通过步骤 (d), (e) 和 (f) 实现。如果省略了 (e) 和 (f)，则剩下的算法将是一步式表格 Q-learning。

Tabular Dyna-Q

对于所有 $s \in \mathcal{S}, a \in \mathcal{A}(s)$, 初始化 $Q(s, a)$ 和 $M(s, a)$

永久循环:

- (a) $S \leftarrow$ 当前 (非终止) 状态
- (b) $A \leftarrow \epsilon\text{-greedy}(S, Q)$
- (c) 采取动作 A , 观测结果的奖励 R 和状态 S'
- (d) $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$
- (e) $Model(S, A) \leftarrow R, S'$ (假定环境确定性)
- (f) 循环重复 n 次:
 - $S \leftarrow$ 随机先前观测到的状态
 - $A \leftarrow$ 先前在 S 中采取的随机动作
 - $R, S' \leftarrow Model(S, A)$
 - $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$

例 8.1: Dyna Maze 考虑一下图8.2中所示的简单迷宫。在 47 个状态中的每一个状态, 都有 up, down, right, 和 left 四个动作, 这些动作确定性地将 agent 带到相应的相邻状态, 除非移动受到障碍物或迷宫边缘的阻碍, 在这种情况下 agent 保持在原地。在所有转移中, 奖励为零, 但进入目标状态的奖励为 +1。达到目标状态 (**G**) 后, agent 返回起始状态 (**S**) 以开始新的回合。这是 $\gamma = 0.95$ 的折扣回合任务。

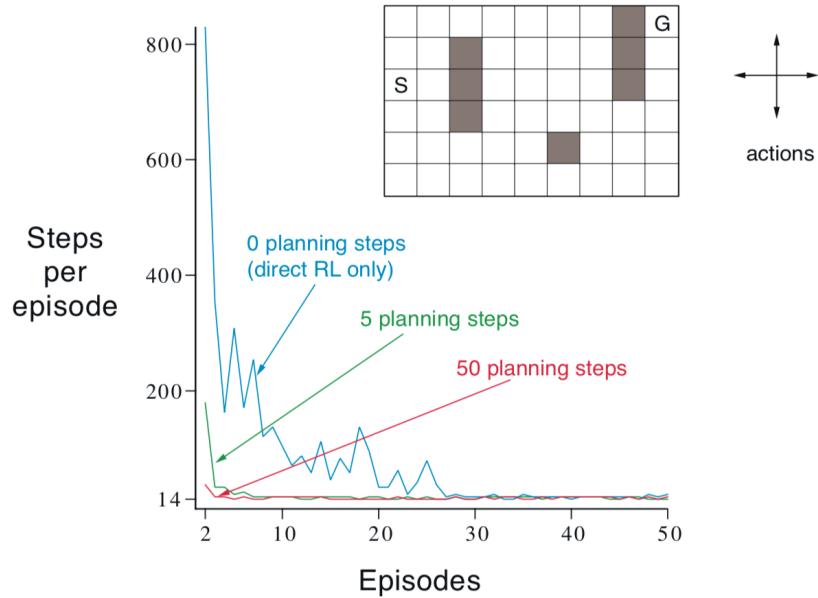


图 8.2: 一个简单迷宫 (插图) 和 Dyna-Q agent 的平均学习曲线, 每个实际步骤的规划步骤数 (n) 不同。任务是尽快从 **S** 行驶到 **G**。

图8.2的主要部分显示了将 Dyna-Q 应用于迷宫任务的实验的平均学习曲线。初始动作值为零, 步长参数为 $\alpha = 0.1$, 探索参数为 $\epsilon = 0.1$ 。当在动作中贪婪地选择时, 关联被随机打破。Agent 在每个实际步骤中执行的规划步骤数 n 有所不同。对于每个 n , 曲线

显示 agent 在每个回合中为达到目标所采取的步数，其平均重复了 30 次实验。在每次重复中，随机数生成器的初始种子在算法之间保持恒定。因此，对于所有 n 值，第一个回合都完全相同（约 1700 步），其数据未在图中显示。在第一个回合之后，对于所有 n 值，性能都有所提高，但是对于较大的值，性能提高得更快。回想一下， $n = 0$ agent 是非规划 agent，仅使用直接强化学习（一步式表格 Q-learning）。尽管实际上已针对该问题优化了参数值 (α 和 ε)，但这是迄今为止最慢的解决方案。非规划 agent 需要大约 25 个回合才能达到 ($\varepsilon-$) 最佳性能，而 $n = 5$ agent 需要大约 5 个回合，而 $n = 50$ agent 仅需要 3 个回合。

图8.3显示了规划 agent 为什么找到解决方案的速度比非规划 agent 快得多的原因。显示的是第二次回合中途由 $n = 0$ 和 $n = 50$ 的 agent 找到的策略。如果不进行规划 ($n = 0$)，则每个回合只会向策略添加一个额外的步，因此到目前为止，仅了解了一个步（最后一个）。通过规划，在第一个回合中只学习了一个步，但是在第二个回合中，已经制定了广泛的策略，到回合结束时，几乎可以回到起始状态。该策略是在 agent 仍在起始状态附近徘徊时由规划过程构建的。到第三回合结束时，将找到完全最优策略，并获得理想的性能。

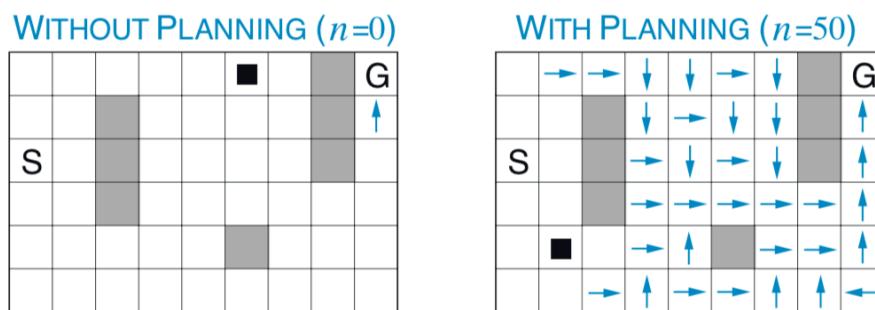


图 8.3: 规划和非规划 Dyna-Q agent 在第二回合中途发现的策略。箭头指示每种状态下的贪婪动作；如果状态没有显示箭头，则其所有动作值均相等。黑色正方形表示 agent 的位置。 ■

在 Dyna-Q 中，学习和规划是通过完全相同的算法来完成的，它们以学习的真实经验和规划的模拟经验为基础。由于规划是逐步进行的，因此将规划和行动混为一谈是微不足道的，两者都尽可能快地进行。Agent 总是反应迅速且深思熟虑，可以立即响应最新的感官信息，但始终在幕后进行规划。模型学习过程也在幕后进行。随着获得新信息，模型将更新以更好地匹配实际情况。随着模型的更改，正在进行的规划过程将逐渐计算出与新模型匹配的不同行为方式。

练习 8.1 非规划方法在图8.3中看起来特别糟糕，因为它是一种单步方法。使用多步自举的方法会更好。您是否认为第??章中的其中一种多步自举方法可以和 Dyna 方法一样好？解释为什么能或者为什么不能。

8.3 当模型错误时

在上一节介绍的迷宫示例中，模型的变化相对较小。该模型最初是空的，然后仅填写了完全正确的信息。总的来说，我们不能指望有这么幸运。模型可能是不正确的，因为

环境是随机的并且仅观察到有限数量的样本，或者因为模型是使用不完全泛化的函数逼近来学习的，或者仅仅是因为环境已更改且尚未观察到其新行为。当模型不正确时，规划过程可能会计算出次优策略。

在某些情况下，通过规划计算得出的次优策略会很快导致建模误差的发现和纠正。当模型在预测比实际可能获得的更大的奖励或更好的状态转移方面更乐观时，这往往会发生。规划中的策略试图利用这些机会，并发现这种情况并不存在。

例 8.2 : Blocking Maze 一个迷宫的例子说明了这种相对较小的建模错误和从中的恢复，如图8.4所示。最初，如图左上角所示，从起始到目标，在障碍物的右边有一条很短的路径。经过 1000 个时间步长后，短路径被“阻塞”，而更长的路径沿屏障的左侧打开，如图右上角所示。该图显示了即将描述的 Dyna-Q agent 和增强型 Dyna-Q+ agent 的平均累积奖励。该图的第一部分显示，两个 Dyna agent 都在 1000 个步内找到了短路径。当环境改变时，曲线图变得平坦，表示一段时间内 agent 没有获得任何奖励，因为他们在障碍物后面徘徊。但是，过了一段时间后，他们就能找到新的开口和新的最优行为。

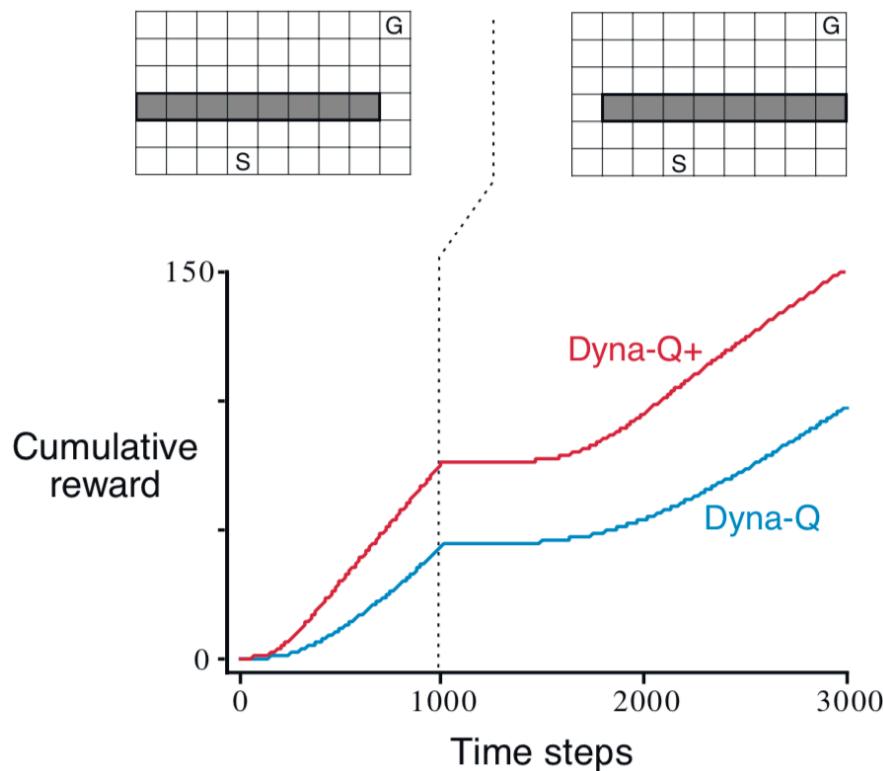


图 8.4: Dyna agent 在执行阻塞任务的平均性能。左边的环境用于前 1000 步，右边的环境用于剩余的步。Dyna-Q+ 是具有鼓励探索的探索奖励的 Dyna-Q。

当环境变化变得比以前更好时，困难增加，但是以前的正确策略并没有显示出这种改善。在这些情况下，建模错误可能很长一段时间都不会被检测到。 ■

例 8.3 : Shortcut Maze 由图8.5中所示的迷宫示例说明了由这种环境变化引起的问题。最初，最优路径是绕过障碍物的左侧（左上方）。但是，经过 3000 步后，会在右侧打开一条较短的路径，而不会打扰较长的路径（右上方）。该图显示常规 Dyna-Q agent 从未切换到快捷方式。实际上，它从未意识到它的存在。它的模型说没有捷径，所以规划得越多，

就越不可能朝右边走去发现它。即使采用 ϵ -greedy 捡策略, agent 也不大可能会采取如此多的探索性行动来发现捷径。

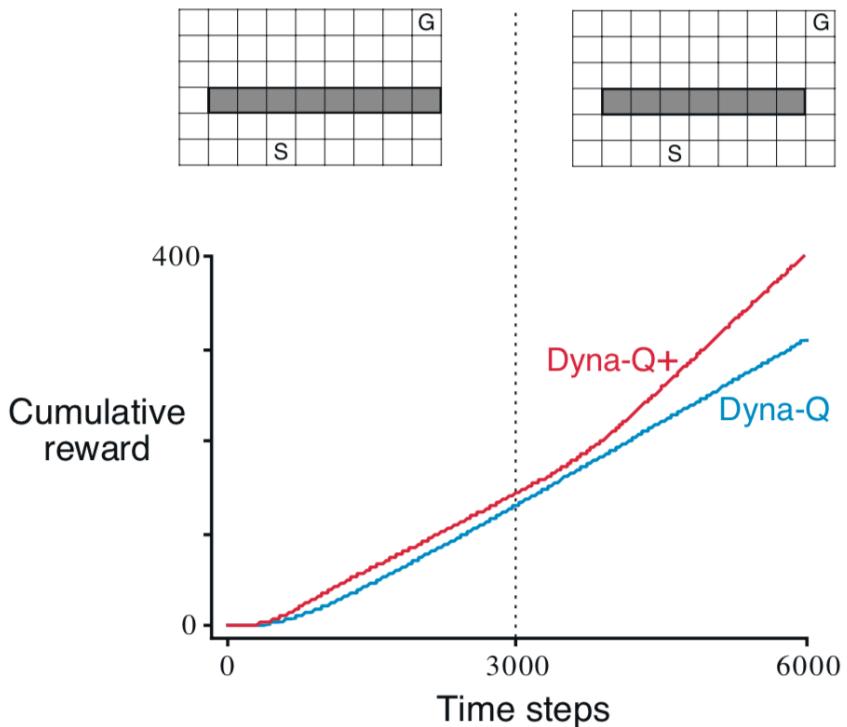


图 8.5: Dyna agent 在执行捷径任务的平均性能。左边的环境用于前 3000 步, 右边的环境用于剩余的步。 ■

这里的普遍问题 tan s 探索与利用之间的冲突的另一种形式。在规划环境中, 探索意味着尝试采取改进模型的工作, 而利用意味着在给定当前模型的情况下以最优方式表现。我们希望 agent 进行探索, 以发现环境中的变化, 但又不会导致性能大大降低。就像以前的探索/利用冲突一样, 可能没有完美和实用的解决方案, 但是简单的启发式方法通常是有用的。

确实解决了快捷迷宫的 Dyna-Q+ agent 使用了一种这样的启发式方法。该 agent 跟踪每个状态-动作对, 记录自从在与环境的实际交互中最后一次尝试该对以来已经历了多少时间步。经过的时间越长, 该对的动态变化和其模型不正确的概率就越大(我们可以假定)。为了鼓励测试长期未尝试动作的行为, 对涉及这些动作的模拟体验会给予特殊的“奖金奖励”。特别是, 如果一个转移的模型奖励为 r , 并且转移尚未在 τ 个时间步中尝试, 则完成计划更新就好像该转移产生了 $r + \kappa\sqrt{\tau}$ 的奖励, κ 是比较小的数。这鼓励 agent 继续测试所有可访问的状态转移, 甚至发现很长的动作序列以进行此类测试¹。当然, 所有这些测试都有其成本, 但是在许多情况下, 例如在捷径迷宫中, 这种测试好奇心的计算非常值得进行额外的探索。

练习 8.2 为什么具有探索奖金 Dyna-Q+ 的 Dyna agent 在阻塞和捷径实验的第一阶段以及

¹Dyna-Q+ agent 还在另外两个方面进行了更改。首先, 允许在上方方框的表格 Dyna-Q 算法的规划步骤(f)中考虑从未从某个状态尝试过的动作。其次, 此类动作的初始模型是它们将以零奖励返回到相同状态。

第二阶段都表现得更好? □

✎ **练习 8.3** 对图8.5的仔细检查表明, 在实验的第一部分中, Dyna-Q+ 和 Dyna-Q 之间的差异略小。这是什么原因呢? □

✎ **练习 8.4 (programming)** 上述探索奖励实际上改变了状态和动作的估计值。这有必要吗? 假设奖金 $\kappa\sqrt{\tau}$ 不是用于更新, 而是仅用于动作选择。也就是说, 假设选择的动作始终是 $Q(S_t, a) + \kappa\sqrt{\tau(S_t, a)}$ 最大的动作。进行一个网格世界实验, 测试并说明这种替代方法的优缺点。 □

✎ **练习 8.5** 如何修改第 164 页中所示的表格 Dyna-Q 算法以处理随机环境? 这种修改如何在本节所考虑的变化环境中表现不佳? 如何修改算法以处理随机环境和变化的环境? □

8.4 优先扫描

在前面各节中介绍的 Dyna agent 中, 模拟转移是从所有先前经历的对中统一随机选择的状态-动作对开始的。但是, 统一的选择通常不是最好的。如果模拟的转移和更新着重于特定的状态-行为对, 则规划将更加有效。例如, 请考虑在第一个迷宫任务的第二个回合中发生了什么 (图8.3)。在第二个回合开始时, 只有直接进入目标的状态 - 动作对才具有正值。所有其他对的值仍为零。这意味着在几乎所有转移上执行更新是没有意义的, 因为它们将 agent 从一个零值状态转移到另一个零值状态, 因此更新没有效果。只有沿转移到目标之前的状态或从目标之前的状态转移到目标的更新才会改变任何值。如果模拟转移是统一生成的, 那么在偶然发现其中一个有用的更新之前, 将会进行许多浪费的更新。随着规划的进行, 有用更新的区域不断扩大, 但是规划的效率仍然远远低于将精力集中在最擅长的地方。在我们现实目标中的大得多的问题中, 状态的数量如此之多, 以至于没有重点的搜索将极其无效率。

此示例表明, 可以通过从目标状态向后进行搜索可以有效的集中搜索。当然, 我们真的不想使用任何特定于“目标状态”的方法。我们需要适用于一般奖励函数的方法。目标状态只是一个特例, 方便激发直觉。通常, 我们不仅要从目标状态出发, 而且要从价值已更改的任何状态开始工作。假设给定的模型值最初是正确的, 就像在发现目标之前在迷宫示例中一样。现在假设 agent 发现了环境的变化并更改了其中一个状态的估计值, 增加或减少。通常, 这意味着还应该更改许多其他状态的值, 但是唯一有用的一次更新是直接导致其值已更改的一个状态的动作的值。如果这些动作的值被更新, 则先前状态的值可以依次改变。如果是这样, 则需要更新导致它们的动作, 然后它们的先前状态可能已更改。这样, 可以从值已更改的任意状态向后工作, 可以执行有用的更新或终止传播。可以将这一总体思想称为规划计算的后向聚焦。

随着有用更新的边缘向后传播, 它通常会迅速增长, 产生许多可以有效更新的状态-动作对。但并非所有这些都同样有用。某些状态的价值可能发生很大变化, 而其他状态则可能变化不大。这些变化很大的先前对也很有可能也发生很大变化。在随机环境中, 估计的转移概率的变化也会导致变化的大小以及需要更新对的紧迫性的变化。根据更新的紧急程度确定更新的优先级并按优先级顺序执行更新是很自然的。这就是优先扫

描的思想。每个状态-动作对都会保留一个队列，其估计值在更新时将有很大的变化，并按变化的大小排列优先级。更新队列中的最顶对时，将计算出其前对中每个对的影响。如果影响大于某个较小的阈值，那么该对将以新的优先级插入队列中（如果队列中存在该对的先前条目，则插入将导致仅较高优先级的条目保留在队列中）。这样，变化的影响被有效地向后传播，直到静止。在下一页的方框中提供了确定性环境的完整算法。

Prioritized sweeping for a deterministic environment

对于所有 s, a ，初始化 $Q(s, a)$, $Model(s, a)$ ，以及 $PQueue$ 为空

永久循环：

- (a) $S \leftarrow$ 当前（非终止）状态
- (b) $A \leftarrow policy(S, Q)$
- (c) 采取动作 A ；观测产生的奖励 R 和状态 S'
- (d) $Model(S, A) \leftarrow R, S'$
- (e) $P \leftarrow |R + \gamma \max_a Q(S', a) - Q(S, A)|$
- (f) 如果 $P > \theta$ ，那么以优先级 P 插入 S, A 到 $PQueue$
- (g) 循环重复 n 次，当 $PQueue$ 不为空：

$S, A \leftarrow first(PQueue)$

$R, S' \leftarrow Model(S, A)$

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

对于所有 \bar{S}, \bar{A} 循环，其被预测会导致 S ：

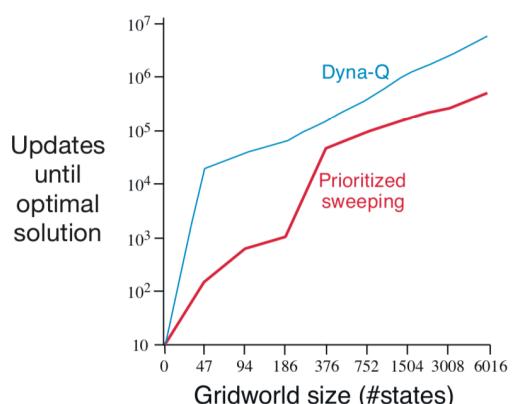
$\bar{R} \leftarrow \bar{S}, \bar{A}, S$ 的预测奖励

$P \leftarrow |\bar{R} + \gamma \max_a Q(S, a) - Q(\bar{S}, \bar{A})|$

如果 $P > \theta$ ，那么以优先级 P 插入 \bar{S}, \bar{A} 到 $PQueue$

例 8.4 : Prioritized Sweeping on Mazes

已发现优先扫描可以极大地提高迷宫任务中找到最优解的速度，通常提高 5 到 10 倍。右侧显示了一个典型示例。这些数据用于一系列与图 8.1 所示结构完全相同的迷宫任务，不同之处在于它们的网格分辨率不同。与未优先处理的 Dyna-Q 相比，优先扫描具有决定性的优势。两种系统中每次环境交互最多进行 $n = 5$ 次更新。改编自 Peng 和 Williams (1993)。 ■



将优先扫描扩展到随机环境非常简单。通过保持每个状态-动作对经历的次数以及下一个状态是什么的计数来维护该模型。然后很自然地不使用我们到目前为止一直在使用的样本更新来更新每对，而是使用期望更新来更新每个对，考虑所有可能的下一状态及

其发生的概率。

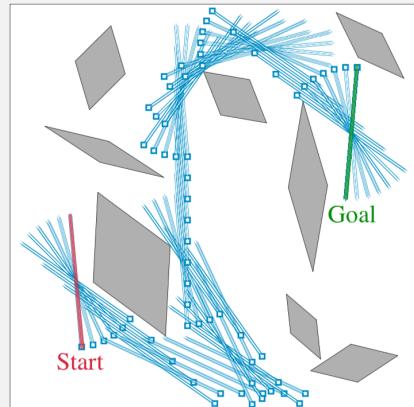
优先扫描只是分配计算以提高规划效率的一种方法，可能不是最佳方法。优先扫描的局限性之一是它使用期望更新，这在随机环境中可能会在低概率转移上浪费大量计算量。如下一节所示，尽管采样引入了方差，但样本更新在许多情况下可以通过较少的计算接近真实值函数。样本更新之所以能取胜，是因为它们将整个备份计算分解为较小的部分（与各个转移相对应的部分），从而使它可以更狭窄地集中在影响最大的部分上。这种想法在 van Seijen 和 Sutton (2013) 提出的“小型备份”中被带入了逻辑上的限制。这些是沿单个转移的更新（例如样本更新），但是基于不带采样的转移概率（如期望更新）。通过选择执行小更新的顺序，可以大大提高规划效率，使其超越优先扫描的规划效率。

我们在本章中建议，所有状态空间规划都可以看作是值更新的序列，仅在更新类型，期望或采样，大或小以及更新的顺序上有所不同。在本节中，我们强调了向后聚焦，但这只是一种策略。例如，另一种方法是根据当前策略下经常访问的状态到达状态的容易程度来关注它们，这可以称为前向聚焦。Peng 和 Williams (1993) 以及 Barto, Bradtke 和 Singh (1995) 探索了前向聚焦的版本，并且在接下来的几节中介绍的方法将其变为一种极端形式。

例 8.5

Prioritized Sweeping for Rod Maneuvering

这项任务的目标是在有限的矩形工作空间内，将一根杆子绕着一些摆放好的障碍物，以最少的步数移动到一个目标位置。杆可以沿其长轴平移或垂直于该轴平移，也可以绕其中心沿任一方向旋转。每个移动的距离约为工作空间的 $1/20$ ，旋转增量为 10 度。平移是确定性的，并量化为 20×20 个位置之一。右侧显示了从开始到目标的障碍和通过优先扫描找到的最短的解决方案。这个问题是确定性的，但是有四个动作和 14,400 个潜在状态（其中一些由于障碍而无法达到）。这个问题可能太大了，无法通过不优先的方法解决。该图转载自 Moore 和 Atkeson (1993)。



8.5 期望与采样更新

前面几节中的示例给出了一些将学习和规划方法相结合的可能性的想法。在本章的其余部分，我们将从期望和采样更新的相对优势入手，分析其中涉及的一些组件思想。

本书的大部分内容涉及各种不同的价值函数更新，并且我们考虑了很多种类。目前集中在一步更新上，它们主要沿三个维度变化。前两个维度是它们是否更新状态值或动作值，以及是否估计最优策略或任意给定策略的值。这两个维度会产生四类更新，以

近似四个值函数 q_* , v_* , q_π 和 v_π 。另一个两维度是更新是否是期望更新，其考虑所有可能的事件，或采样更新，其考虑可能发生的单个样本。这三个两维度导致八种情况，其中七种与特定算法相对应，如右图所示（第八种情况似乎与任何有用的更新都不对应）。这些单步更新中的任何一种都可以用于规划方法。前面讨论的 Dyna-Q agent 使用 q_* 采样更新，但是它们也可以使用 q_* 期望更新，或者使用 q_π 预期更新或采样更新。Dyna-AC 系统使用 v_π 采样更新以及学习策略结构（如第 13 章中所述）。对于随机问题，始终使用一种期望更新来完成优先扫描。

当我们在第 6 章介绍了一步式采样更新时，我们将其作为期望更新的替代。在没有分布模型的情况下，不可能进行期望更新，但是可以使用来自环境或样本模型的样本转移来完成样本更新。从这种观点出发，隐含的是，如果可能的话，期望更新要比采样更新好。但是，是吗？期望更新肯定会产生更好的估计，因为它们不受抽样误差的破坏，但是它们也需要更多的计算，并且计算通常是规划中的限制资源。为了正确评估规划中期望更新和采样更新的相对优势，我们必须控制它们的不同计算需求。

具体而言，请考虑期望和样本更新以逼近 q_* ，以及离散状态和动作的特殊情况，近似值函数 Q 的表格查找表示以及以估计动态形式的模型 $\hat{p}(s', r | s, a)$ 。状态-动作对 s, a 的期望更新为：

$$Q(s, a) \leftarrow \sum_{s', r} \hat{p}(s', r | s, a) [r + \gamma \max_{a'} Q(s', a')]. \quad (8.1)$$

给定采样的下一个状态和奖励， S' 和 R （来自模型），针对 s, a 的对应采样更新是类似 Q-learning 的更新：

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[R + \gamma \max_{a'} Q(S', a') - Q(s, a) \right]. \quad (8.2)$$

其中， α 是常规的正步长参数。

这些期望更新和采样更新之间的差异在环境是随机的程度上是重要的，特别是在给定一个状态和一个动作的情况下，许多可能的下一状态可能以各种概率发生。如果只有一个下一个状态是可能的，则上面给出的期望和采样更新是相同的（取 $\alpha = 1$ ）。如果存在许多可能的下一状态，那么可能会有显著的差异。支持期望更新的原因是它精确的计算，

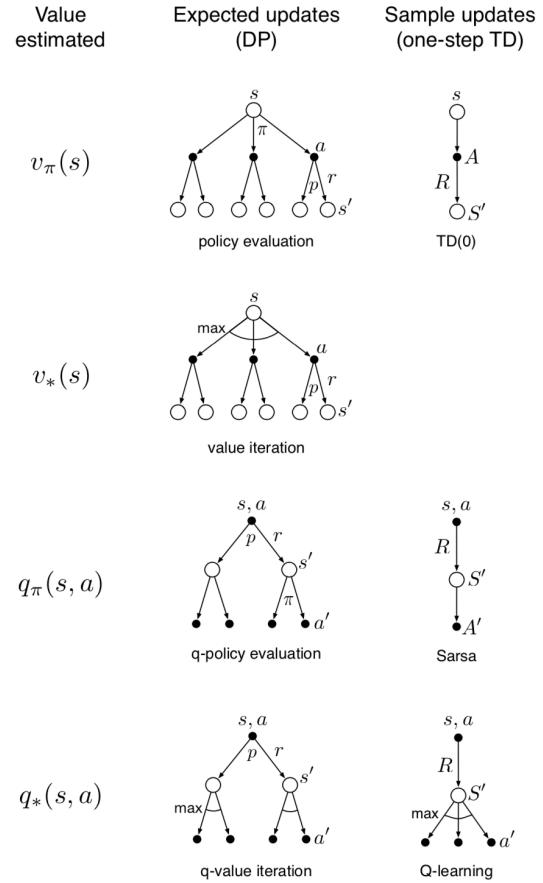


图 8.6：本书中考虑的所有一步更新的备份图。

从而导致新的 $Q(s, a)$ 的正确性仅受后继状态下 $Q(s', a')$ 的正确性限制。此外，采样更新还受到抽样误差的影响。另一方面，样本更新在计算上更便宜，因为它只考虑一个下状态，而不考虑所有可能的下一个状态。在实践中，更新操作所需的计算通常由评估 Q 的状态-动作对的数量决定。对于特定的起始对 s, a ，令 b 是分支因子（即 $\hat{p}(s', r|s, a) > 0$ 的可能下一状态 s' 的数量）。然后，此对的期望更新大约需要样本更新的 b 倍的计算量。

如果有足够的时间来完成期望更新，则由于没有采样误差，因此得出的估计值通常要好于 b 个样本更新的估计值。但是，如果没有足够的时间来完成期望更新，则样本更新始终是可取的，因为它们至少会在少于 b 个更新的情况下对价值估算进行一些改进。在许多状态-动作对的大问题中，我们经常处于后一种情况。由于有如此多的状态-动作对，因此所有这些动作的期望更新将花费很长时间。在此之前，在许多状态-动作对中进行一些样本更新可能要比在几对中进行期望更新要好得多。给定一个单位的计算能力，是将其更好地用于一些期望更新，还是将其用于 b 倍样本更新？

图8.7显示了分析结果，提出了对该问题的答案。它显示了期望更新和各种分支因子 b 的样本更新的计算时间的函数的估计误差。考虑的情况是，所有 b 个后继状态都是同等可能的，并且初始估计中的误差为 1。假定下一个状态的值是正确的，因此期望更新将在完成后将误差减少到零。在这种情况下，样本更新根据 $\sqrt{\frac{b-1}{bt}}$ 来减小误差，其中 t 是已执行的样本更新的数量（假定样本平均值，即 $\alpha = 1/t$ ）。关键的观察结果是，对于中等大小的 b 来说，错误会随着 b 更新的一小部分而急剧下降。对于这些情况，许多状态-动作对的值可能会大大提高，达到期望更新效果的百分之几以内，同时单个状态-动作对可能只会进行期望更新。

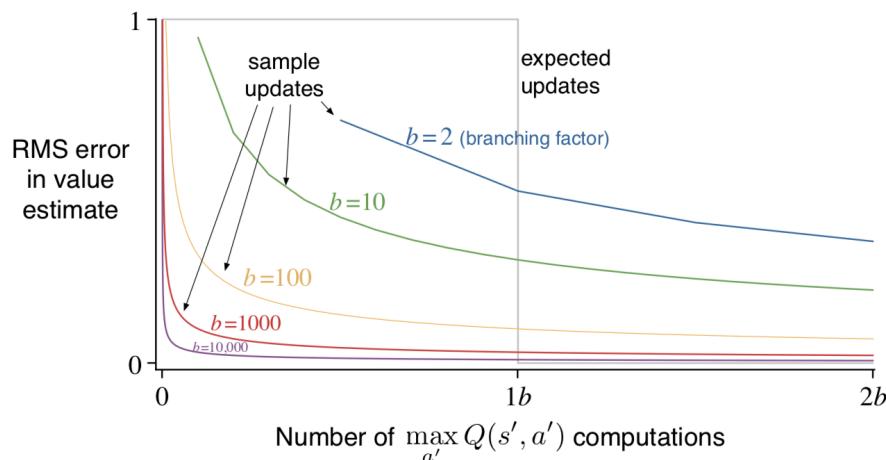


图 8.7：期望更新和样本更新的效率比较。

图8.7中所示的样本更新的优势可能低估了实际效果。在实际问题中，后继状态的值将是自身更新的估计。通过使估计更快更准确，样本更新将具有第二个优势，即从后继状态备份的值将更加准确。这些结果表明，对于具有较大随机分支因子和太多状态无法准确解决的问题，样本更新可能会优于期望更新。

 **练习 8.6** 上面的分析假设所有 b 个可能的下一状态都同样可能发生。相反，假设分布高度偏斜，则某些 b 个状态比大多数状态更可能发生。这会增强还是削弱样本更新超过期望更新的理由？支持您的答案。□

8.6 轨迹采样

在本节中，我们比较两种分布更新的方法。来自动态规划的经典方法是在整个状态（或状态-动作）空间执行扫描，每次扫描更新一次每个状态（或状态-动作对）。这对于大型任务是有问题的，因为即使一次扫描也可能没有时间来完成。在许多任务中，绝大多数状态都是无关紧要的，因为仅在非常差的策略或很小的概率下才可以访问它们。穷举的扫描隐含地将相等的时间分配给状态空间的所有部分，而不是集中在需要的地方。正如我们在第4章中讨论的那样，穷举扫描和对它们暗示的所有状态的同等对待并不是动态规划的必要属性。原则上，更新可以以任何喜欢的方式分布（为了确保收敛，必须无限次地访问所有状态或状态-动作对；尽管在下面的第 8.7 节中讨论了例外情况），但是在练习经常使用穷举扫描。

第二种方法是根据某种分布从状态或状态-动作空间采样。可以像 Dyna-Q agent 中那样均匀采样，但这会遇到与穷举扫描相同的一些问题。更具吸引力的是根据 *on-policy* 分布来分布更新，即根据遵循当前策略时观察到的分布。这种分布的一个优点是易于生成；只需遵循当前策略，即可与模型进行交互。在回合任务中，人们以一种起始状态（或根据起始状态分布）开始并进行模拟，直到达到终端状态为止。在一项连续的任务中，任何地方都可以开始，并且一直在模拟。在这两种情况下，模型都将给出样本状态转移和奖励，而当前策略将给出样本动作。换句话说，一个模拟显式的单个轨迹并在沿途遇到的状态或状态-动作对上执行更新。我们称这种产生经验和更新的方式为轨迹采样。

除了通过轨迹采样之外，很难想象有任何有效的方法可以根据 *on-policy* 的分布来分配更新。如果可以清楚地表示 *on-policy* 的分布，那么就可以扫描所有状态，并根据 *on-policy* 的分布对每个状态的更新进行加权，但这又使我们失去了穷举扫描的所有计算成本。可能可以从分布中采样并更新单个状态-动作对，但是即使可以有效地做到这一点，在模拟轨迹方面这将提供什么好处？即使以明确的形式知道 *on-policy* 的分布也不太可能。每当策略更改时，分布就会更改，计算分布需要进行与完整策略评估相当的计算。考虑到其他可能性，轨迹采样看起来既高效又优雅。

在 *on-policy* 分布更新是否很好？从直觉上看，这似乎是一个不错的选择，至少比均匀分布好。例如，如果您正在学习下棋，那么您将学习在真实游戏中可能出现的位置，而不是棋子的随机位置。后者可能是有效状态，但能够准确评估它们是评估真实游戏中位置的另一项技能。我们还将在第二部分中看到，使用函数逼近时，*on-policy* 分布具有显著优势。无论是否使用函数逼近，人们都可以期望以策略为中心来显著提高规划速度。

专注于 *on-policy* 分布可能是有益的，因为它会导致空间中大量的、无趣的部分被忽略，或者可能是有害的，因为它会导致空间的相同旧部分不断地更新。我们进行了一个小实验，以实证评估效果。为了隔离更新分布的影响，我们完全按照（8.1）的定义使用了一步式期望表格更新。在均匀情况下，我们循环浏览所有状态-动作对，到位进行更新，在 *on-policy* 情况下，我们模拟回合，所有回合都始于同一状态，更新在当前 ϵ -greedy 策略 ($\epsilon=0.1$) 发生的每个状态-动作对。这些任务是未折扣的回合任务，如下随机产生。从每个 $|\mathcal{S}|$ 状态中，可能有两个动作，每个动作都导致 b 个下个状态中的一个，它们都具

有相同的可能性，并且每个状态-动作对都有 b 个状态的不同随机选择。对于所有状态-动作对，分支因子 b 均相同。此外，在所有转移中，都有 0.1% 的可能性转移到终止状态，从而结束该回合。从均值 0 和方差 1 的高斯分布中选择每个转移的期望回报。在规划过程中的任何时候，都可以停止并详尽地计算 $v_{\tilde{\pi}}(s_0)$ ，即贪婪策略 $\tilde{\pi}$ 下起始状态的真实值，给定当前的动作值函数 Q ，作为 agent 在其贪婪地行动的新回合中做得有多好的指示（同时始终假设模型是正确的）。

右图的上半部分显示了 200 个样本任务的平均结果，这些任务具有 1000 个状态，分支因子分别为 1、3 和 10。绘制的策略质量与完成的期望更新次数成函数关系。在所有情况下，根据 on-policy 的分布进行采样都会使最初的规划速度更快，而从长远来看，这会延迟规划。在较小的分支因子下，效果更强，更快的规划的初始阶段更长。在其他实验中，我们发现随着状态数量的增加，这些影响也变得越来越强。例如，图的下半部分显示了对于具有 10,000 个状态的任务，分支因子为 1 的结果。在这种情况下，注重 on-policy 的优势是巨大而持久的。

所有这些结果都是有意义的。在短期内，根据 on-policy 的分布进行采样有助于将精力集中在起始状态的后代附近。如果状态很多且分支因子较小，则此影响将很大且持续时间长。从长远来看，专注于 on-policy 的分布可

能有害，因为常见的状态都已经具有正确的值。对它们进行采样是没有用的，而对其他状态进行采样实际上可能会执行一些有用的工作。这大概就是为什么穷举而未集中的方法从长远来看会更好的原因，至少对于小问题而言。这些结果不是结论性的，因为它们仅适用于以特定的随机方式生成的问题，但是它们确实表明，根据 on-policy 的分布进行采样对于大型问题（尤其是其中子集很小的问题）可能是一个巨大的优势，特别是在 on-policy 分布下访问状态-动作空间的小子集问题。

- ✍ 练习 8.7 图8.8中的某些图似乎在其早期部分呈扇形，特别是上半部分图中 $b = 1$ 和均匀分布。你认为这是为什么？显示的数据的哪些方面支持您的假设？ □
- ✍ 练习 8.8 (programming) 重复实验，其结果如图8.8的下半部分，然后尝试相同的实验，但 $b = 3$ 。讨论结果的含义。 □

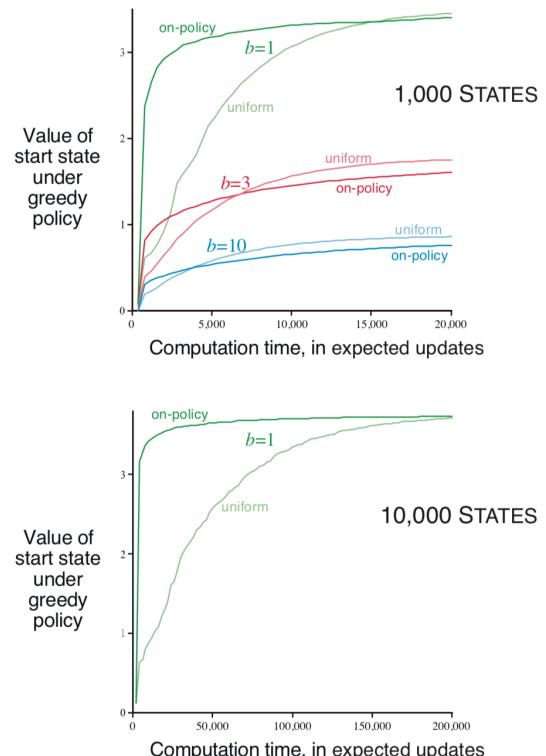


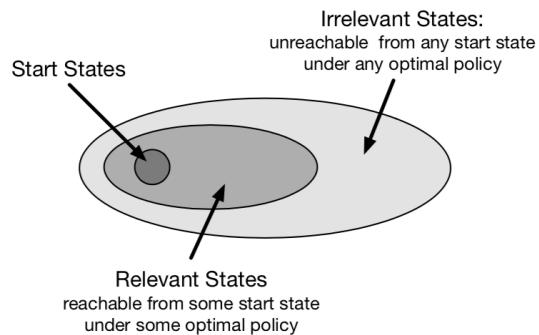
图 8.8：在状态空间中均匀分布和集中在模拟的 on-policy 轨迹的分布更新的相对效率，每个轨迹都从同一状态开始。结果是针对两种大小和各种分支因子 b 的随机生成任务。

8.7 实时动态规划

实时动态规划 (Real-time dynamic programming, RTDP) 是动态规划 (DP) 的值迭代算法的 on-policy 进行轨迹采样的版本。因为 RTDP 与常规的基于扫描的策略迭代密切相关, 所以 RTDP 以特别清晰的方式说明了 on-policy 轨迹采样可以提供的一些优势。RTDP 借助 (4.10) 定义的期望表格值迭代更新来更新在实际或模拟轨迹中访问的状态的值。基本上, 该算法产生了如图8.8所示的 on-policy 结果。

RTDP 与常规 DP 之间的紧密联系使得可以通过改编现有理论来得出一些理论结果。RTDP 是第4.5节中描述的异步 DP 算法的示例。异步 DP 算法没有按照状态集的系统扫描来组织; 它们使用其他状态的任意值以任何顺序更新状态值。在 RTDP 中, 更新顺序由在真实或模拟轨迹中访问的顺序状态决定。

如果轨迹只能从一组指定的起始状态开始, 并且您对给定策略的预测问题感兴趣, 那么 on-policy 的轨迹采样可使算法完全跳过从任何起始状态给定策略无法到达的状态: 这些状态与预测问题无关。对于控制问题, 其目标是找到最优策略而不是评估给定策略, 那么很可能存在某些最优策略无法从任何开始状态到达的状态, 因此对于这些无关紧要的状态无需指定最优动作。所需要的是最优的局部策略, 这意味着对于相关状态而言是最优的策略, 但是对于不相关的状态, 它可以指定任意动作或者甚至是不确定的。



但是, 使用诸如 Sarsa (第6.4节) 之类的 on-policy 的轨迹采样控制方法来找到这种最优的局部策略, 通常需要无限次访问所有的状态-动作对, 即使那些将被证明是无关紧要的对也是如此。例如, 这可以通过使用探索开始来完成 (第5.3节)。对于 RTDP 也是如此: 对于具有探索起点的回合任务, RTDP 是一种异步值迭代算法, 可以收敛到折扣有限 MDP (以及某些条件下的无折扣情况) 的最优策略。与预测问题的情况不同, 如果收敛到最优策略很重要, 则通常无法停止更新任何状态或状态-动作对。

对于 RTDP 而言, 最有趣的结果是, 对于满足合理条件的某些类型的问题, 可以保证 RTDP 能够找到对相关状态最优的策略, 而不必无限访问每个状态, 甚至根本不需要访问某些状态。确实, 在某些问题中, 只需要访问一小部分状态。这对于具有非常大的状态集的问题可能是一个很大的优势, 在这种情况下, 即使单次扫描也可能不可行。

该结果成立的任务是 MDP 的无折扣回合任务, 具有吸收的目标状态并产生零奖励的 MDP, 如3.4节所述。在真实或模拟轨迹的每个步骤中, RTDP 都会选择一个贪婪的动作 (随机打破关系), 并将期望值迭代更新操作应用于当前状态。它也可以在每个步骤中更新其他状态的任意集合的值; 例如, 它可以从当前状态更新在有限范围前瞻搜索中访问的状态值。

对于这些问题, 每个回合开始于从一组起始状态中随机选择的状态并结束于目标状

态时，RTDP 以概率 1 收敛到对所有相关状态均最优的策略：1) 每个目标状态的初始值均为零；2) 存在至少一个保证从任何起始状态以概率 1 到达目标状态的策略；3) 从非目标状态转移的所有奖励严格为负；以及 4) 所有初始值等于或大于其最优值（只需将所有状态的初始值设置为零即可满足）。Barto, Bradtke 和 Singh (1995) 通过将异步 DP 的结果与启发式搜索算法的结果相结合，证明了这一结果，该启发式搜索算法由于 Korf (1990) 而被称为学习实时 A^* 。

具有这些属性的任务是随机最优路径问题的例子，通常以代价最小化而不是我们在此处的奖励最大化来说明。在我们的版本中，使负回报最大化就等于使从起始状态到目标状态的路径代价最小化。这类任务的示例是最小时间控制任务，其中达到目标所需的每个时间步都会产生 -1 的奖励，或诸如第 3.5 节中的高尔夫示例（其目标是以最少的击球数击入球洞）之类的问题。

例 8.6： RTDP on the Racetrack 练习 5.12 (第 111 页) 的赛道问题是随机的最优路径问题。将 RTDP 与常规 DP 值迭代算法在一个示例赛道问题上进行比较，说明了 on-policy 轨迹采样的一些优势。

回想一下练习中的内容，agent 必须学习如何在如图 5.5 所示的转弯处驾驶汽车，并尽快越过终点线，同时留在跑道上。起始状态是起始线上的所有零速状态。目标状态是通过从轨道内部越过终点线可以在一个时间步中达到的所有状态。与练习 5.12 不同，这里没有限制车速，因此状态设置可能是无限的。但是，通过任何策略从起始状态集中达到的状态集是有限的，可以认为是问题的状态集。每个回合均以随机选择的起始状态开始，并在汽车越过终点线时结束。直到汽车越过终点线，每一步的奖励都是 -1。如果汽车撞到赛道边界，则将其移回随机起始状态，然后回合继续。

类似于图 5.5 左侧的小赛道的赛道具有 9,115 个状态可以通过任何策略从起始状态到达，其中只有 599 个是相关的，这意味着它们可以通过某种最优策略从某个起始状态到达（通过对 10^7 个回合执行最优动作时访问的状态进行计数，可以估算相关状态的数量）。

下表比较了通过常规 DP 和 RTDP 解决此任务的情况。这些结果是 25 次运行的平均值，每次运行均以不同的随机数种子开始。在这种情况下，传统的 DP 是使用状态集的穷尽扫描进行值迭代，其中每次都更新一个状态的值，这意味着每个状态的更新都使用其他状态的最新值（这是值迭代的 Gauss-Seidel 版本，其寻找速度大约是 Jacobi 版本的两倍，请参阅第 4.8 节）。没有特别注意更新的顺序；其他顺序可能会产生更快的收敛。两种方法每次运行的初始值都为零。当一次扫描中状态值的最大变化小于 10^{-4} 时，DP 被认为收敛，而当超过 20 个回合的终点线的平均时间似乎稳定在一个渐近步数时，RTDP 被认为收敛。此版本的 RTDP 仅在每个步骤上更新当前状态的值。

	DP	RTDP
收敛的平均计算	28 次扫描	4000 个回合
收敛的平均更新数	252,784	127,600
每回合更新的平均数	—	31.9
≤ 100 次更新的状态百分比	—	98.45
≤ 10 次更新的状态百分比	—	80.51
0 次更新的状态百分比	—	3.18

两种方法产生的策略平均要跨越 14 到 15 步才能越过终点线，但是 RTDP 仅需要 DP 进行的更新的一半左右。这是 RTDP 对 on-policy 轨迹进行采样的结果。尽管在每次 DP 扫描中都会更新每个状态的值，但是 RTDP 会将更新集中在更少的状态上。在平均运行中，RTDP 更新了 98.45% 的状态的值不超过 100 次；更新了 80.51% 的状态的值不超过 10 次；平均而言，大约 290 个状态的值完全没有更新。 ■

RTDP 的另一个优点是，当值函数接近最优值函数 v_* 时，agent 用于生成轨迹的策略也接近最优策略，因为它相对于当前值函数总是贪婪。这与常规值迭代中的情况相反。在实际中，当值函数在扫描中仅发生少量变化时，值迭代就会终止，这就是我们终止它以获取上表中的结果的方式。在这一点上，值函数非常接近 v_* ，贪婪策略也接近最优策略。但是，很可能在值迭代终止很久之前就对最新值函数贪婪的策略是最优的，或者几乎是最优的（从第4章回想一下，对于许多不同的值函数，而不仅仅是 v_* ，最优策略可能是贪婪的）。在值迭代收敛之前检查最优策略的出现不是常规 DP 算法的一部分，因此需要大量的额外计算。

在赛道示例中，通过在每次 DP 扫描之后运行许多测试回合，并根据该扫描的结果贪婪地选择动作，可以估算出 DP 计算中的最早点，在该点上近似最优评估函数足够好，因此相应的贪婪策略几乎是最优的。对于此跑道，在进行 15 次值迭代扫描后或在 136,725 个值迭代更新之后，出现了接近最优的策略。这大大少于 DP 收敛到 v_* 所需的 252,784 个更新，但仍大于 RTDP 所需的 127,600 个更新。

尽管这些模拟当然不是 RTDP 与基于常规扫描的值迭代的权威性比较，但它们说明了 on-policy 轨迹采样的一些优势。常规的值迭代不断更新所有状态的值，而 RTDP 则专注于与问题的目标相关的关系子集。随着学习的继续，这种专注变得越来越狭窄。因为 RTDP 的收敛定理适用于模拟，所以我们知道 RTDP 最终将只关注相关状态，即构成最优路径的状态。RTDP 实现了近乎最优的控制，计算量约为基于扫描的值迭代所需计算量的 50%。

8.8 决策时规划

规划至少可以以两种方式使用。我们到目前为止在本章中已经考虑过的一个以动态规划和 Dyna 为代表的方法是，在从模型（样本或分布模型）获得的模拟经验的基础上，使用规划逐步改进策略或价值函数。然后，选择动作就是比较从到目前为止我们已经考虑过的表格情况下从表中获得的当前状态的动作值，或者通过我们在下面的第二部分中考虑的近似方法来评估数学表达式。在为任何当前状态 S_t 选择一个动作之前，规划已经在改善表中条目或数学表达式方面起了一定的作用，需要为许多状态（包括 S_t ）选择动作。通过这种方式，规划并不专注于当前状态。我们称这种方式使用的规划是背景规划。

使用规划的另一种方式是在遇到每个新状态 S_t 之后开始并完成规划，这是一种计算，其输出是选择单个动作 A_t ；下一步，规划从 S_{t+1} 重新开始以生成 A_{t+1} ，依此类推。这种使用规划的最简单且几乎是退化的示例是，只有状态值可用时，才通过比较每个动作的模型预测的下一状态的值（或通过如第1章中的井字游戏示例来比较后继状态的值）

来选择一个动作。更一般而言，以这种方式使用的规划看起来比先行一步更深入，并且可以评估导致许多不同的预测状态和奖励轨迹的动作选择。与最初使用规划不同，此处规划着重于特定状态。我们称此为决策时规划。

可以通过自然和有趣的方式将规划的两种思维方式（使用模拟的经验逐步改善策略或价值函数，或使用模拟的经验为当前状态选择动作）融合在一起，但是它们倾向于分开研究。这是首先了解它们的好方法。现在让我们仔细研究决策时规划。

即使仅在决策时才进行规划，我们仍然可以像在8.1节中所做的那样，将其视为从模拟经验到更新和价值，最终到策略。仅仅是因为现在的值和策略特定于当前状态以及那里可用的动作选择，以至于如此，由规划过程创建的值和策略通常在用于选择当前动作后被丢弃。在许多应用中，这并不是很大的损失，因为有很多状态，而且我们不太可能长时间恢复到同一状态。通常，可能想要两者兼而有之：将规划重点放在当前状态上，并存储规划结果，以便将来返回同一状态时走得更远。决策时规划在不需要快速响应的应用中最有用。例如，在下棋程序中，每一步可能需要几秒钟或几分钟的计算时间，而强大的程序可能会在这段时间内规划数十步棋。另一方面，如果优先选择低延迟动作，那么通常最好在后台进行规划以计算策略，然后将该策略快速应用于每个新遇到的状态。

8.9 启发式搜索

人工智能中的经典状态空间规划方法是决策时规划方法，统称为启发式搜索。在启发式搜索中，对于遇到的每个状态，都会考虑一棵可能的连续树。近似值函数将应用于叶节点，然后在根处向当前状态备份。搜索树中的备份与本书中讨论的最大（用于 v_* 和 q_* ）的期望更新相同。备份在当前状态的状态-动作节点处停止。一旦计算出这些节点的备份值，就选择它们中的最优值作为当前动作，然后丢弃所有备份值。

在传统的启发式搜索中，没有通过更改近似值函数来保存备份值的方法。实际上，值函数通常是由人设计的，并且永远不会因搜索而改变。但是，自然可以考虑使用启发式搜索过程中计算出的备份值或本书中介绍的任何其他方法来随时间改进值函数。从某种意义上说，我们都采用这种方法。我们的贪婪， ε -greedy 和 UCB（第2.7节）动作选择方法与启发式搜索没有什么不同，尽管它的规模较小。例如，要计算给定模型和状态值函数的贪婪动作，我们必须从每个可能动作向前看到每个可能的下一个状态，考虑奖励和估计值，然后选择最好的动作。就像传统的启发式搜索一样，此过程会计算可能动作的备份值，但不会尝试保存它们。因此，启发式搜索可以看作是贪婪策略思想的延伸，超出一步。

更深一步搜索的目的是获得更好的动作选择。如果有一个完美的模型和一个不完美的动作值函数，那么事实上，更深入的搜索通常会产生更好的策略²。当然，如果搜索一直到回合的结束，那么不完美值函数的影响被消除，以这种方式确定的动作一定是最优的。如果搜索的深度 k 足够大，使得 γ^k 非常小，则相应的动作将接近最优。另一方面，搜索越深，需要的计算就越多，通常会导致响应时间变慢。Tesauro 的大师级西洋双

²对此有一些有趣的例外（例如，参见 Pearl，1984）。

陆棋 TD-Gammon 提供了一个很好的例子（第 16.1 节）。该系统使用 TD 学习通过许多自玩游戏来学习后继状态价值函数，并使用启发式搜索的形式进行移动。作为一种模型，TD-Gammon 使用了掷骰子概率的先验知识，并假设对手总是选择 TD-Gammon 认为最适合自己的动作。Tesauro 发现，启发式搜索越深入，TD-Gammon 所做的动作就越好，但是每次动作花费的时间也就越长。Backgammon 具有很大的分支系数，但必须在几秒钟内进行移动。只有选择性地向前搜索几个步骤才是可行的，但是即使如此，搜索仍可以显着改善动作选择。

我们不应忽视启发式搜索关注更新的最明显方式：关注当前状态。启发式搜索的大部分效果是由于其搜索树紧紧关注可能紧随当前状态的状态和动作。与下棋相比，下棋可能会花费更多的时间，但是在下棋时，考虑一下棋盘格以及您特定的棋盘格位置，您可能的下一个动作和后续位置是值得的。无论您如何选择动作，这些状态和动作都是更新的最高优先级，也是您最迫切希望近似值函数准确的地方。不仅您的计算应优先用于即将发生的事件，而且有限的内存资源也应如此。例如，在国际象棋中，存在太多可能的位置，无法为每个象棋存储不同的价值估计，但是基于启发式搜索的国际象棋程序可以轻松存储从单个位置向前看他们面对的数百万个位置的不同估计。将内存和计算资源集中在当前决策上可能是启发式搜索如此有效的原因。

可以以类似方式更改更新的分布，以专注于当前状态及其可能的后继者。作为一种极端情况，我们可能完全使用启发式搜索的方法来构建搜索树，然后从下至上执行单个的一步更新，如图8.9所示。如果以这种方式对更新进行排序并使用表格表示形式，则将实现与深度优先启发式搜索完全相同的整体更新。可以将这种状态空间搜索视为大量独立的单步更新的拼接。因此，在更深层的搜索中观察到的性能提高并不是由于使用了这样的多步更新。相反，这是由于更新关注和集中在当前状态下游的状态和动作。通过投入大量与候选动作特别相关的计算，决策时规划可以比依靠非专注的更新产生更好的决策。

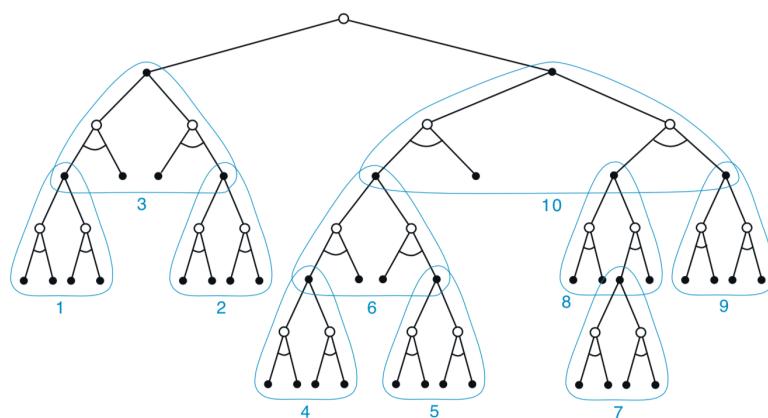


图 8.9：启发式搜索可以实现为一系列一步更新（此处以蓝色显示），将值从叶节点备份到根。此处显示的顺序用于选择性的深度优先搜索。

8.10 展开算法

展开算法 (rollout algorithms) 是基于蒙特卡洛控制的决策时规划算法，应用于所有从当前环境状态开始的模拟轨迹。他们通过平均许多模拟轨迹的回报来估算给定策略的动作值，这些轨迹从每个可能的动作开始，然后遵循给定策略。当动作值估计被认为足够准确时，执行具有最高估计值的动作（或动作之一），然后从所产生的下一个状态重新执行该过程。正如 Tesauro 和 Galperin (1997) 所解释的那样，他们曾尝试过用展开算法来玩 backgammon，“rollout”一词来自于通过展开玩法（即“rolling out”）来估计 backgammon 位置的价值，该位置在随机生成的掷骰子序列中多次到达游戏的结尾，其中两个玩家的移动均由某些固定策略决定。

与第5章中描述的蒙特卡洛控制算法不同，展开算法的目标不是估计完整的最优动作值函数 q_* 或针对给定策略 π 的完整的动作值函数 q_π 。取而代之的是，它们仅针对每个当前状态以及给定的策略（通常称为展开策略）生成动作值的蒙特卡洛估计。作为决策时规划算法，展开算法会立即使用这些动作值估计值，然后将其丢弃。这使得展开算法相对容易实现，因为不需要为每个状态-动作对采样结果，也不需要在状态空间或状态-动作空间上近似函数。

展开算法会完成什么工作？第4.2节中描述的策略改进定理告诉我们，给定任何两个相同的策略 π 和 π' ，其除了对于某些状态 s , $\pi'(s) = a \neq \pi(s)$, 如果 $q_\pi(s, a) \geq v_\pi(s)$ ，则策略 π' 等于或优于 π 。此外，如果不等式严格，则 π' 实际上比 π 好。这适用于展开算法，其中 s 是当前状态，而 π 是展开策略。对每个动作 $a' \in \mathcal{A}(s)$ ，平均模拟轨迹的回报可得出 $q_\pi(s, a')$ 的估计值。然后，在 s 中选择一个将这些估计值最大化并随后遵循 π 的动作的策略是对 π 进行改进的策略的良好候选者。结果就像第4.3节中讨论的动态规划策略迭代算法的一个步骤（尽管它更像第4.5节中描述的异步值迭代的一个步骤，因为它仅改变当前状态的动作）。

换句话说，展开算法的目的是改进展开策略；而不是找到最优策略。经验表明，展开算法可以令人惊讶地有效。例如，Tesauro 和 Galperin (1997) 惊讶于展开方法所产生的 backgammon 游戏能力的显着提高。在某些应用中，即使展开策略是完全随机的，展开算法也可以产生良好的性能。但是，改进后的策略的性能取决于展开策略的属性以及由蒙特卡洛值估计产生的动作的排名。直觉表明，展开策略越好，价值估算越准确，则展开算法产生的策略可能越好（但请参见 Gelly 和 Silver, 2007）。

这涉及重要的权衡，因为更好的展开策略通常意味着需要更多的时间来模拟足够的轨迹以获得良好的价值估算。作为决策时规划方法，展开算法通常必须满足严格的时间限制。推出算法所需的计算时间取决于每个决策必须评估的动作数量，获得有用的样本回报所需的模拟轨迹中的时间步数，展开策略做出决策所需的时间，以及获得良好的蒙特卡洛动作值估计所需的模拟轨迹的数量。

尽管有多种方法可以缓解挑战，但平衡这些因素对于任何展开方法的应用都很重要。由于蒙特卡洛试验彼此独立，因此可以在单独的处理器上并行运行许多试验。另一种方法是截断没有完整回合的模拟轨迹，并通过存储的评估函数纠正截断的回报（这充分发挥

了我们在前面各章中所说的有关截断的回报和更新的所有内容)。正如 Tesauro 和 Galperin (1997) 所建议的那样, 还可以监视蒙特卡洛模拟, 并修剪那些不太可能证明是最好的候选动作, 或者那些其值与当前最优值足够接近以至于选择它们不会产生真正的差异的候选动作 (尽管 Tesauro 和 Galperin 指出这会使并行实现复杂化)。

通常, 我们不会将展开算法视为学习算法, 因为它们不维护价值或策略的长期记忆。但是, 这些算法利用了我们在本书中强调的强化学习的某些特性。作为蒙特卡洛控制的实例, 他们通过平均样本轨迹集合的回报来估算动作值, 在这种情况下, 模拟轨迹是与环境的样本模型相交互。这样, 它们就像强化学习算法一样, 通过轨迹采样可以避免进行穷尽的动态规划扫描, 并且可以通过依赖样本而不是期望更新来避免对分布模型的需求。最后, 展开算法通过对估计的动作值进行贪婪地行动来利用策略改进属性。

8.11 蒙特卡洛树搜索

蒙特卡洛树搜索 (Monte Carlo Tree Search, MCTS) 是决策时规划的最新且非常成功的示例。从根本上说, MCTS 是如上所述的展开算法, 但是通过添加一种用于累积从蒙特卡洛模拟获得的值估计的方法而得到增强, 以便将模拟连续地引向更高奖励的轨迹。MCTS 在很大程度上促进了计算机围棋的发展, 从 2005 年的业余爱好者水平提高到 2015 年的大师水平 (6 dan 或以上)。已经开发了许多基本算法变体, 包括我们在 16.6 节中讨论的一个变体, 它对于 2016 年 AlphaGo 程序战胜一名 18 届世界冠军围棋选手的惊人胜利至关重要。事实证明, MCTS 在各种各样的竞争环境中都是有效的, 包括一般的游戏玩法 (例如, 参见 Finnsson 和 Bjornsson, 2008; Genesereth 和 Thielscher, 2014), 但它并不局限于游戏; 如果有足够的简单的环境模型可以进行快速多步模拟, 则对单个 agent 顺序决策问题可能是有效的。

在遇到每个新状态后将执行 MCTS, 以选择该状态的 agent 动作; 再次执行以选择下一个状态的动作, 依此类推。与展开算法一样, 每次执行都是一个迭代过程, 它模拟从当前状态开始到终止状态 (或直到折扣使任何进一步的奖励对回报的贡献都可以忽略不计) 的许多轨迹。MCTS 的核心思想是通过扩展已从早期模拟获得了高度评估的轨迹的初始部分, 从当前状态开始连续关注多个模拟。MCTS 不必保留从一个动作选择到下一个动作选择的近似值函数或策略, 尽管在许多实现中, MCTS 保留了可能对下一次执行有用的所选动作值。

在大多数情况下, 模拟轨迹中的动作是使用简单策略生成的, 通常将其称为展开策略, 因为对于更简单的展开算法而言, 它就是如此。当展开策略和模型都不需要大量计算时, 可以在短时间内生成许多模拟轨迹。就像在任何表格蒙特卡洛方法中一样, 状态-动作对的值被估计为该对 (模拟) 回报的平均值。如图 8.10 所示, 仅对最有可能在几步之内到达状态-动作对子集的状态进行蒙特卡洛估计, 这形成了植根于当前状态的树。MCTS 通过添加表示基于模拟轨迹结果看似有希望的状态的节点来逐步扩展树。任何模拟的轨迹都将穿过树, 然后在某个叶节点处退出。在树的外部和叶节点处, 可以使用展开策略进行动作选择, 但是在树内部的状态下, 可能会更好。对于这些状态, 我们至少可以对某

些动作进行价值估算，因此我们可以使用一种知情策略（称为树策略）在其中进行权衡，以平衡探索和利用。例如，树策略可以使用 ε -greedy 或 UCB 选择规则选择动作（第2章）。

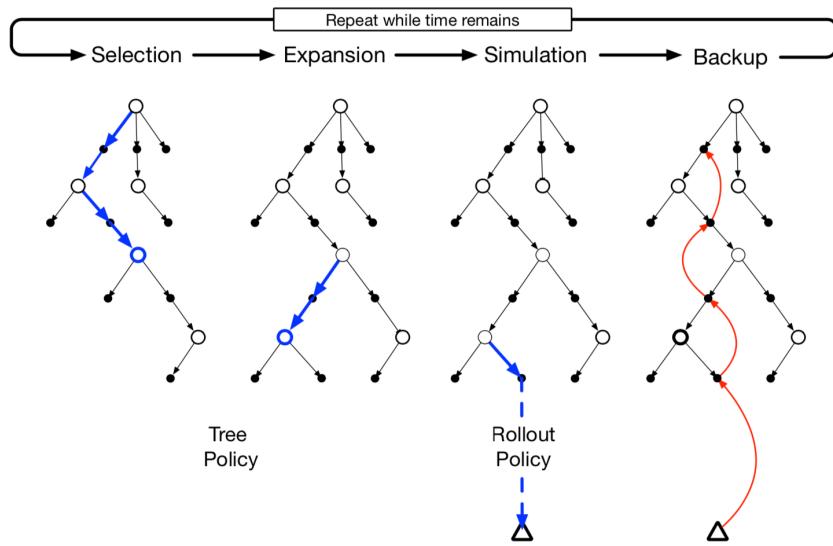


图 8.10: 蒙特卡洛树搜索。当环境更改到新状态时，MCTS 会在需要选择动作之前执行尽可能多的迭代，以增量方式构建其根节点代表当前状态的树。每次迭代均由四个操作组成：Selection、Expansion（尽管在某些迭代中可能会跳过）、Simulation 和 Backup，如文本中所述并由树中的粗体箭头所示。改编自 Chaslot, Bakkes, Szita 和 Spronck (2008)。

更详细地，MCTS 基本版本的每次迭代都包含以下四个步骤，如图8.10所示：

1. **Selection.** 从根节点开始，基于附加到树边缘的动作值的树策略会遍历树以选择叶节点。
2. **Expansion.** 在某些迭代中（取决于应用的详细信息），通过添加一个或多个子节点（通过未探索的动作从选择的节点到达）来从选择的叶节点扩展树。
3. **Simulation.** 从所选节点或其新添加的子节点之一（如果有）中，使用展开策略选择的动作来运行完整回合的模拟。结果是蒙特卡洛试验，其动作首先由树策略选择，而树外则由展开策略选择。
4. **Backup.** 模拟回合生成的回报被备份以更新或初始化附加在 MCTS 迭代中树策略所遍历的树的边缘的动作值。在树之外，不会为展开策略访问的状态和动作保存任何值。图8.10通过显示从模拟轨迹的终止状态直接到展开策略开始的树中的状态-动作节点的备份来说明这一点（尽管一般而言，整个模拟轨迹的全部回报都会备份到此状态-动作节点）。

MCTS 会继续执行这四个步骤，每次都从树的根节点开始，直到没有剩余时间或耗尽其他计算资源为止。然后，最后根据树中累积的统计信息的某种机制，从根节点（仍然代表环境的当前状态）中选择一个动作；例如，它可能是具有从根状态可用的所有动作中最大的动作值的动作，或者可能是具有最大访问计数以避免选择离群值的动作。这是 MCTS 实际选择的动作。在环境转移到新状态之后，MCTS 再次运行，有时从代表新状态的单个根节点的树开始，但通常从包含先前节点构造的树中剩余的该节点的任何后代的树开始执行 MCTS；所有其余节点以及与之关联的动作值都将被丢弃。

MCTS 最初在两人竞技游戏的程序中选择移步提出，例如 Go。对于游戏而言，每个模拟回合都是游戏的完整玩法，其中两个玩家都通过树和展开策略选择动作。16.6 节描述了 AlphaGo 程序中使用的 MCTS 的扩展，该扩展将 MCTS 的蒙特卡洛评估与由深度人工神经网络通过自我博弈强化学习所学习的动作值相结合。

将 MCTS 与我们在本书中描述的强化学习原则联系起来，可以对 MCTS 如何取得如此令人印象深刻的结果有所了解。从根本上说，MCTS 是基于蒙特卡洛控制的决策时间规划算法，该算法应用于从根状态开始的模拟；也就是说，这是上一节中介绍的一种展开算法。因此，它受益于在线、增量、基于样本的价值估计和策略改进。除此之外，它还保存了附加到树边缘的动作值估计值，并使用强化学习的样本更新来更新它们。这样可以有效地将蒙特卡罗试验集中在轨迹上，这些轨迹的初始段与先前模拟的高回报轨迹相同。此外，通过逐步扩展树，MCTS 有效地增长了一个查找表来存储部分动作值函数，并将内存分配给在高回报样本轨迹的初始段中访问的状态-动作对的估计值。因此，MCTS 避免了全局逼近动作值函数的问题，同时保留了使用以往经验指导探索的好处。

MCTS 在决策时规划方面取得的惊人成功已深深地影响着人工智能，许多研究人员正在研究对用于游戏和单个 agent 应用的基本程序的修改和扩展。

8.12 本章总结

规划需要环境模型。分布模型包括下一状态的概率和可能采取的动作的奖励；样本模型会产生单个转移，并根据这些概率生成奖励。动态规划需要一个分布模型，因为它使用期望更新，其中涉及对所有可能的下一个状态和奖励的期望计算。另一方面，需要一个样本模型来模拟与环境的交互，在此期间可以使用样本更新（如许多强化学习算法所使用的样本更新）。样本模型通常比分布模型更容易获得。

我们提出了一个观点，强调了规划最优行为和学习最优行为之间令人惊讶的密切关系。两者都涉及对相同值函数的估计，并且在两种情况下，很长一段时间都需要进行一系列小的备份操作，以递增方式更新估计值。通过简单地允许两者更新相同的估计值函数，就可以直接集成学习和规划过程。此外，任何学习方法都可以简单地通过将其应用于模拟（模型生成的）经验而非实际经验而转换为规划方法。在这种情况下，学习和规划变得更加相似。它们可能是在两个不同经验来源上运行的相同算法。

将增量规划方法与行动和模型学习相集成很简单。规划、行动和模型学习以循环的方式交互（如第 162 页的图所示），每个产生了其他方面需要改进的地方；它们之间不需要或禁止其他交互。最自然的方法是让所有进程异步并行地进行。如果进程必须共享计算资源，那么划分几乎可以任意进行—通过最方便，最高效的组织来处理当前任务。

在本章中，我们触及了状态空间规划方法之间的许多变化维度。一个维度是更新大小的变化。更新次数越少，规划方法就可以越递增。与 Dyna 一样，最小的更新是一步式样本更新。另一个重要方面是更新的分布，即搜索重点。优先扫描着重于向后转移其价值最近改变的状态的前身。On-policy 轨迹采样着重于 agent 在控制其环境时可能遇到的状态或状态-动作对。这可以允许计算跳过状态空间中与预测或控制问题无关的部分。实

时动态规划是值迭代的 on-policy 轨迹采样版本，它说明了该策略相对于传统的基于扫描的策略迭代具有一些优势。

规划还可以着重于相关状态的前向，例如在 agent 与环境交互过程中实际遇到的状态。最重要的形式是在决策时完成规划，即作为动作选择过程的一部分。在人工智能中研究的经典启发式搜索就是一个例子。其他示例包括展开算法和蒙特卡洛树搜索，它们得益于在线、增量、基于样本的价值估计和策略改进。

8.13 第一部分总结：维度

本章总结了本书的第一部分。在本部分中，我们试图将强化学习不是作为单个方法的集合来呈现，而是作为贯穿方法的连贯思想集来呈现。每种思想都可以视为方法变化的维度。这些维度的集合跨越了很多可能的方法空间。通过在维度层面上探索这个空间，我们希望获得最广泛、最持久的理解。在本节中，我们使用方法空间中的维度的概念来概括本书到目前为止开发的强化学习的观点。

到目前为止，我们在本书中探索的所有方法都具有三个共同的关键思想：首先，它们都试图估计价值函数；其次，它们都通过备份实际或可能的状态轨迹中的值来进行操作；第三，它们都遵循广义策略迭代（GPI）的通用策略，这意味着它们维护一个近似值函数和一个近似策略，并且不断地在彼此的基础上不断进行改进。这三个思想对于本书涵盖的主题至关重要。我们提议价值函数、备份值更新和 GPI 是强大的组织原则，潜在地与任何智能模型（无论是人工的还是自然的）相关。

图8.11显示了方法变化所依据的两个最重要的维度。这些维度与用于改进值函数的更新类型有关。水平维度是它们是样本更新（基于样本轨迹）还是期望更新（基于可能轨迹的分布）。期望更新需要一个分布模型，而样本更新则只需要一个样本模型，或者可以完全不使用模型根据实际经验来完成（另一维度的变化）。图8.11的垂直维度对应于更新的深度，即自举的程度。在空间的四个角中的三个角处是用于估计值的三种主要方法：动态规划、TD 和蒙特卡洛。沿着空间的左边缘是样本更新方法，范围从单步 TD 更新到全部回报的蒙特卡罗更新。在这两者之间是一个频谱，其中包括基于 n 步更新的方法（在第 12 章中，我们将其扩展到 n 步更新的混合，例如通过资格跟踪实现的 λ 更新）。

动态规划方法显示在空间的右上角，因为它们涉及一步的期望更新。右下角是期望更新的极端情况，其深度如此之深，以至于它们一直运行到终止状态（或者在一项连续的任务中，直到折扣将任何进一步奖励的贡献降低到可以忽略的水平）。这是穷举搜索的情况。沿着这个维度的中间方法包括启发式搜索和相关的方法，这些方法可以搜索并更新到有限的深度（可能有选择地）。也有一些沿水平方向的中间方法。其中包括混合期望更新和样本更新的方法，以及在单个更新中混合样本和分布的方法的可能性。正方形的内部代表所有此类中间方法的空间。

我们在本书中强调的第三个方面是 on-policy 方法与 off-policy 方法之间的区别。在前一种情况下，agent 为当前正在遵循的策略学习价值函数，而在后一种情况下，aagent 为不同的策略（通常是 agent 当前认为最佳的）学习价值函数。通常，由于需要进行探索，

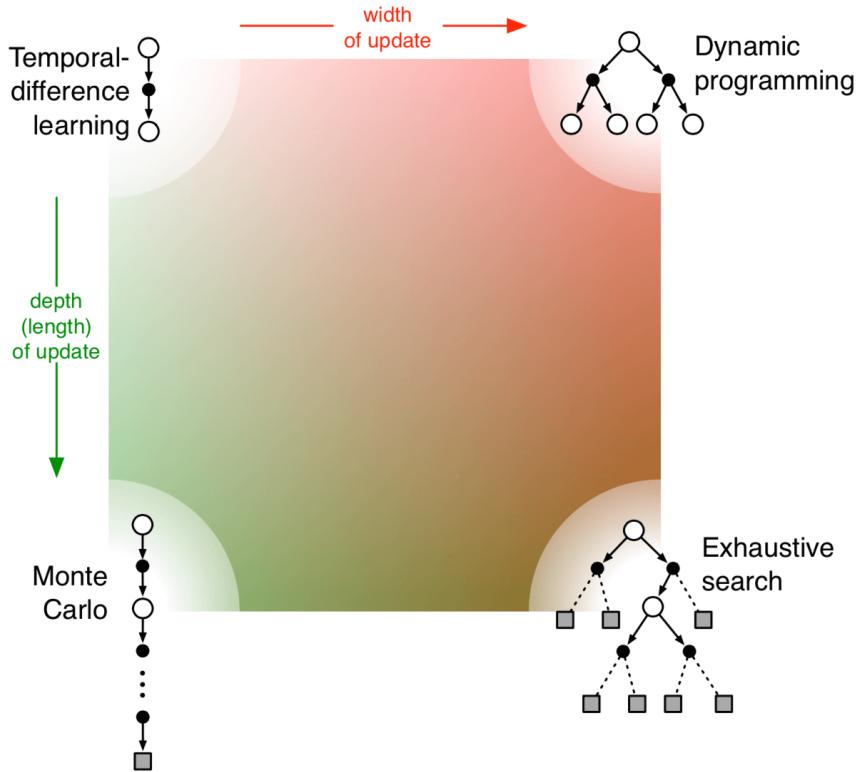


图 8.11: 纵览强化学习方法的空间，重点介绍本书第一部分中探讨的两个最重要的方面：更新的深度和宽度。

因此生成策略的行为与当前认为最好的行为有所不同。第三个维度可以可视化为垂直于图 8.11 中页面的平面。

除了刚才讨论的三个方面，我们在本书中还确定了其他一些方面：

Definition of return 任务是回合的还是连续的，折扣的还是未折扣的？

Action values vs. state values vs. afterstate values 应该估计什么样的价值？如果仅估计状态值，则需要模型或单独的策略（如 actor-critic 方法）来选择动作。

Action selection/exploration 如何选择动作以确保在探索与利用之间进行适当的权衡？我们只考虑了最简单的方法： ϵ -greedy、值的乐观初始化、soft-max 和置信区间上限。

Synchronous vs. asynchronous 所有状态的更新是同时执行还是以某种顺序一个接一个地执行？

Real vs. simulated 应该根据真实经验还是模拟经验进行更新？如果两者兼而有之，则各取多少？

Location of updates 什么状态或状态-动作对应该更新？无模型的方法只能在实际遇到的状态和状态-动作对之间进行选择，但是基于模型的方法可以任意选择。这里有很多可能性。

Timing of updates 更新是应该作为选择动作的一部分还是仅在之后进行？

Memory for updates 更新的值应保留多长时间？是否应该永久保留它们，或者像在启发式搜索中一样，仅在计算动作选择时保留它们？

当然，这些维度既不详尽也不互斥。各个算法也有许多其他方式，并且许多算法位于多

个维度的多个位置。例如，Dyna 方法使用真实和模拟经验来创建相同的值函数。维护以不同方式或针对不同状态和动作表示计算的多个值函数也非常明智。但是，这些方面确实构成了一套连贯的思路，用于描述和探索各种可能的方法。

这里没有提到并且在本书的第一部分中没有提到的最重要的维度是函数逼近。函数逼近可以看作是一种正交的可能性谱，范围从极端的表格方法到状态聚合，各种线性方法，再到各种非线性方法。此维度在第二部分中进行了探讨。

8.14 书目与历史评论

8.1 多年来，这里提出的规划和学习的总体观点已逐步发展，部分原因是作者（Sutton, 1990, 1991a, 1991b; Barto, Bradtke 和 Singh, 1991, 1995; Sutton 和 Pinette, 1985; Sutton 和 Barto, 1981b）；它受到 Agre 和 Chapman (1990; Agre 1988), Bertsekas 和 Tsitsiklis (1989), Singh (1993) 等人的强烈影响。作者还受到潜在学习的心理学研究 (Tolman, 1932) 和思想本质的心理学观点的强烈影响（例如 Galanter 和 Gerstenhaber, 1956; Craik, 1943; Campbell, 1960; Dennett, 1978）。在本书的第三部分中，第 14.6 节将基于模型和无模型的方法与学习和行为的心理学理论联系起来，第 15.11 节讨论了关于大脑如何实现这些方法的想法。

8.2 我们用来描述不同种类的强化学习的 *direct* 和 *indirect* 术语来自于自适应控制文献（例如 Goodwin 和 Sin, 1984），它们被用来进行相同的区分。术语 *system identification* 在所谓的模型学习的自适应控制中使用（例如 Goodwin 和 Sin, 1984; Ljung 和 Söödstrom, 1983; Young, 1984）。Dyna 的体系结构归功于 Sutton (1990)，本节和下一部分的结果均基于那里报道的结果。Barto 和 Singh (1990) 在比较直接和间接强化学习方法时考虑了一些问题。Sutton, Szepesvari, Geramifard 和 Bowling (2008) 和 Parr, Li, Taylor, Painter-Wake Field 和 Littman (2008) 进行了将 Dyna 扩展到线性函数逼近的早期工作（第 9.4 节）。

8.3 有一些基于模型的强化学习的工作将探索性奖励和乐观初始化的思想推向了逻辑极限，其中所有不完全探索的选择都被假定为最大回报，并计算出最优路径来对其进行测试。Kearns 和 Singh (2002) 的 E3 算法以及 Brafman 和 Tennenholz (2003) 的 R-max 算法被保证是状态多项式和时间多项式的近似最优解。对于实际算法而言，这通常太慢，但在最坏的情况下可能是最好的。

8.4 Moore 和 Atkeson (1993) 以及 Peng 和 Williams (1993) 同时并独立地开发了优先扫描。第 170 页的方框中的结果归功于 Peng 和 Williams (1993)。第 171 页的方框中的结果归因于 Moore 和 Atkeson。随后该领域的主要工作包括 McMahan 和 Gordon (2005) 以及 van Seijen 和 Sutton (2013) 的工作。

8.5 Singh (1993) 的实验强烈影响了这一部分。

8.6-8.7 轨迹采样从一开始就暗含了强化学习的一部分，但是 Barto, Bradtke 和 Singh (1995) 在引入 RTDP 时最明确地强调了这一点。他们认识到，Korf (1990) 学习实时 A* (LRTA*) 算法是一种异步 DP 算法，适用于随机问题以及 Korf 关注的确定性问题。除了 LRTA* 外，RTDP 还包括在执行动作之间的时间间隔内更新许多状态的值的选项。Barto 等 (1995 年)

通过将 Korf (1990 年) 关于 LRTA* 的收敛性证明与 Bertsekas (1982 年) (也是 Bertsekas 和 Tsitsiklis, 1989 年) 的结果相结合, 证明了此处描述的收敛性结果, 从而确保了非折扣情况下随机最短路径问题的异步 DP 收敛。将模型学习与 RTDP 结合起来也称为“自适应 RTDP”, 这也是 Barto 等人 (1995) 提出和 Barto (2011) 讨论。

8.9 为了进一步了解启发式搜索, 鼓励读者查阅诸如 Russell 和 Norvig (2009) 和 Korf (1988) 的文本和综述。Peng 和 Williams (1993) 像本节所建议的那样, 探索了更新的前瞻性。

8.10 Abramson (1990) 的期望结果模型是一种适用于两人游戏的展开算法, 其中两个模拟玩家的游戏都是随机的。他认为, 即使是随机玩法, 它也是一种“强大的启发式”, 即“精确, 准确, 易于估算, 有效计算且与领域无关”。Tesauro 和 Galperin (1997) 展示了展开算法对改善 backgammon 程序的玩法的有效性, 它通过评估不同随机生成的骰子掷骰序列的位置而采用了“rollout”一词来评估 backgammon 的位置。Bertsekas, Tsitsiklis 和 Wu (1997) 研究了应用于组合优化问题的展开算法, Bertsekas (2013) 调查了它们在离散确定性优化问题中的使用, 并指出它们“常常出人意料地有效”。

8.11 MCTS 的中心思想由 Coulom (2006) 以及 Kocsis 和 Szepesvari (2006) 提出。他们建立在先前研究的基础之上, 这些研究使用的是蒙特卡洛规划算法, 这些作者对此进行了评论。Browne, Powley, Whitehouse, Lucas, Cowling, Rohlfsagen, Tavener, Perez, Samothrakis 和 Colton (2012) 是对 MCTS 方法及其应用的出色调查。David Silver 为本节中的思想和演示做出了贡献。

第二部分

近似解决方法

特别声明

在过去的 2019 年，**ElegantL^AT_EX** 系列模板均逐步上线 **GitHub**、**CTAN**、**Overleaf** 以及 **Gitee** 上。截止到 2019 年底，**ElegantNote**、**ElegantBook**、**ElegantPaper** 三个模板在 **GitHub** 上的收藏数达到了 194、333 和 220，从 2019 年 5 月开启捐赠之后收到了用户 33 笔合计超过 1500 元的捐赠，用户群人数也超过了 400 人。这些数字的背后，反映出 **ElegantL^AT_EX** 越来越受用户的喜爱，在此非常感谢大家。

但是，我想声明的是：

由于某些原因，**ElegantL^AT_EX** 项目 不再接受任何非我本人预约的提交。

我是一个理想主义者，关于这个模板，我有自己的想法。我所关心的是，我周围的人能方便使用 **L^AT_EX** 以及此模板，我自己会为自己的东西感到开心。如果维护模板让我不开心，那我就不会再维护了。诚然这个模板并不是完美的，但是相比 2.x 好很多了，这些改进离不开大家的反馈、**ChinaT_EX** 和逐鹿人的鼓励以及支援人员的帮助！

如果你无法认同我的想法，建议直接删除本模板。

Ethan Deng

February 10, 2020

第九章 ElegantL^AT_EX 系列模板介绍

ElegantL^AT_EX 项目组致力于打造一系列美观、优雅、简便的模板方便用户使用。目前由 **ElegantNote**, **ElegantBook**, **ElegantPaper** 组成，分别用于排版笔记，书籍和工作论文。强烈推荐使用最新正式版本！本文将介绍本模板的一些设置内容以及基本使用方法。如果您有其他问题，建议或者意见，欢迎在 GitHub 上给我们提交 issues 或者邮件联系我们。

我们的联系方式如下，建议加入用户 QQ 群提问，这样能更快获得准确的反馈，加群时请备注 L^AT_EX 或者 ElegantL^AT_EX 相关内容。

- 官网: <https://elegantlatex.org/>
- GitHub 网址: <https://github.com/ElegantLaTeX>
- CTAN 地址: <https://ctan.org/pkg/elegantbook>
- 文档 Wiki: <https://github.com/ElegantLaTeX/ElegantBook/wiki>
- 下载地址: 正式发行版, 最新版
- 微博: ElegantL^AT_EX
- 微信公众号: ElegantL^AT_EX
- 用户 QQ 群: 692108391
- 邮件: elegantlatex2e@gmail.com

9.1 ElegantBook 更新说明

此次更新主要有

1. 增加数学字体选项 `math`, 可选项为 `newtx` 和 `cm`。

重要提示：原先通过 `newtxmath` 宏包设置的数学字体改为 L^AT_EX 默认数学字体，如果需要保持原来的字体，需要显式声明数学字体 (`math=newtx`)；

2. 新增中文字体选项 `chinesefont`, 可选项为 `ctexfont`、`founder` 和 `nofont`。
3. 将封面作者信息设置为可选，并且增加自定义信息命令 `\bioinfo`;
4. 在说明文档中增加版本历史，新增 `\datechange` 命令和 `change` 环境;
5. 增加汉化章节选项 `scheme`, 可选项为汉化 `chinese`;
6. 由于 `\lvert` 问题已经修复，重新调整 `ctex` 宏包和 `amsmath` 宏包位置。
7. 修改页眉设置，去除了 `\lastpage` 以避免 page anchor 问题，加入 `\frontmatter`。
8. 修改参考文献选项 `cite`, 可选项为数字 `number`、作者-年份 `authoryear` 以及上标 `super`。
9. 新增参考文献样式选项 `bibstyle`, 并将英文模式下参考文献样式 `apalike` 设置为默认值，中文仍然使用 `gbt7714` 宏包设置。



笔记 如果你之前使用了本模板，在使用新版本时，需要删除文档中的 `\hypersetup{pageanchor=true}`，并且在 `\maketitle` 和 `\tableofcontents` 之间添加 `\frontmatter`。

2.x 版本的用户请仔细查看[跨版本转换](#)。

9.2 模板安装与更新

你可以通过免安装的方式使用本模板，包括在线使用和本地（文件夹内）使用两种方式，也可以通过 TeX 发行版安装使用。

9.2.1 在线使用模板

我们把三套模板全部上传到 Overleaf 上了，网络便利的用户可以直接通过 Overleaf 在线使用我们的模板。使用 Overleaf 的好处是无需安装 TeX Live 2019，可以随时随地访问自己的文件。查找模板，请在 Overleaf 模板库里面搜索 `elegantlatex` 即可，你也可以直接访问[搜索结果](#)。选择适当的模板之后，将其 `Open as Template`，即可把模板存到自己账户下，然后可以自由编辑以及与别人一起协作。更多关于 Overleaf 的介绍和使用，请参考 Overleaf 的[官方文档](#)。

注 Overleaf 上，中文需要使用 XeLaTeX 进行编译，英文建议使用 pdfLaTeX 编译。

9.2.2 本地免安装使用

免安装使用方法如下，从 GitHub 或者 CTAN 下载最新（正式）版文件，严格意义上只需要类文件 `elegantbook.cls`。然后将模板文件放在你的工作目录下即可使用。这样使用的好处是，无需安装，简便；缺点是，当模板更新之后，你需要手动替换 `cls` 文件。

9.2.3 发行版安装使用

如果你是 TeX Live 2019 用户，我们推荐你直接进行安装和更新。你可以通过 TeX Live 2019 自带的 `tlshell`¹ 进行安装。安装非常简单，步骤如下，搜索并打开 `tlshell`，然后通过 `File -> Load Default Repository` 加载远程仓库，如果你不想使用默认的仓库，你可以通过 `Options` 下的菜单设置远程仓库。设置好仓库之后，等待仓库加载完毕，你可以在下面的搜索栏搜索 `elegantbook`，然后选择进行安装与更新。

9.2.4 更新问题

如果使用 `tlshell` 无法更新模板，请使用命令行全部更新全部宏包或者使用免安装的方法使用本模板。

通过命令行（管理员权限）输入下面的命令对 `tlmgr` 自身和全部宏包进行更新。

```
tlmgr update --self  
tlmgr update --all
```

更多的内容请参考 [How do I update my TeX distribution?](#)

¹也叫 TeX Live Manager

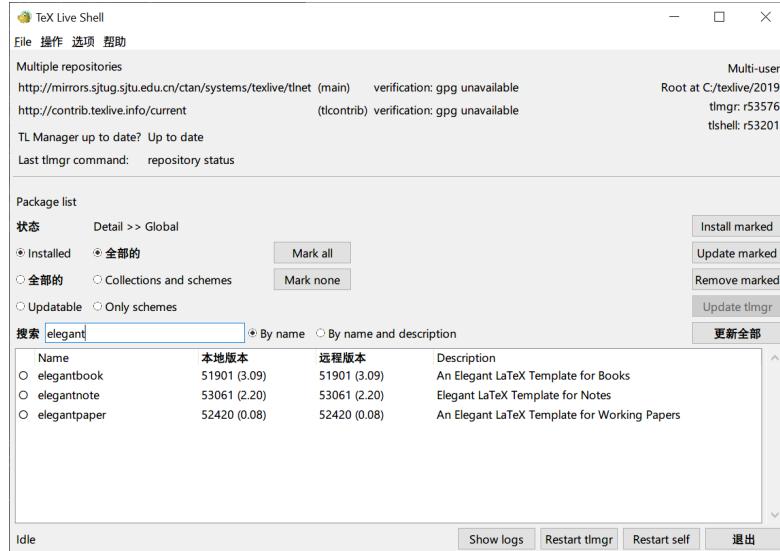


图 9.1: 使用 TeX Live Shell 安装 ElegantBook 模板

9.2.5 其他发行版本

如果你是 TeX Live 2018 的用户，由于 2018 难以更新到 2019，建议卸载 2018 重装 2019。如果嫌麻烦，你可以手动安装模板，将 `elegantbook.cls` 复制到你的 TeX Live 目录下，默认安装目录为 `C:\texlive\2019\texmf-dist\tex\latex\simplebook`，然后通过命令行（管理员权限），运行 `texhash` 即可。

由于宏包版本问题，本模板不支持 CTeX 套装。更多关于 TeX Live 2019 的安装使用以及 CTeX 与 TeX Live 的兼容、系统路径问题，请参考官方文档以及 [一份简短的安装 L^AT_EX 的介绍](#)。

9.3 用户作品计划

ElegantL^AT_EX 系列模板从创立至今已经有 9 年了，我们的模板也受到了很多用户的喜爱，在此，为了促进模板用户之间的交流，了解用户需求，完善本模板，我们将建立一个区域专门展示用户的文档，包括但不限于 GitHub 和官网等。如果你愿意将自己的作品展示出来，请邮件或者其他方式联系我们。如果自己代码已经传到 GitHub 或者 Gitee 等网站，可以提供对应网址。[用户文档中心](#)目前有下面一些作品：

1. 唐绍东：微积分笔记
2. 曲豆豆：超甜微积分习题集
3. 王世强：《化工数值计算与 MATLAB》复习指南
4. 李晨迪：Fluid Mechanics Notes
5. 肖明顺：地球物理勘查 常用规范汇编
6. 白衣卿相：期末复习笔记-拓扑学摘要

9.4 关于提交

出于某些因素的考虑, Elegant^{La}T_EX 项目自 2019 年 5 月 20 日开始, 不再接受任何非作者预约性质的提交 (pull request)! 如果你想改进模板, 你可以给我们提交 issues, 或者可以在遵循协议 (LPPL-1.3c) 的情况下, 克隆到自己仓库下进行修改。

9.5 协作人员招募

招募 Elegant^{La}T_EX 的协作人员 (志愿者), 没有报酬。工作内容: 翻译 Elegant^{La}T_EX 系列模板文档, 维护模板的维基, 如果有公众号文稿写作经历的话, 也可以帮忙写微信稿。本公告长期有效。目前 Elegant^{La}T_EX 共有 4 名协作人员, 在此感谢他们无私的奉献!

- 官方文档翻译: [YPY](#);
- GitHub 维基维护: [Ingo Zinngo](#)、[追寻原风景](#);
- QQ 群管理、FAQ 整理: [Sikouhjw](#).

另外, 也感谢 [Xiangdong Zeng](#)、逐鹿人等人帮忙群管理。

9.6 致谢

2019 年 5 月 20 日, ElegantBook 模板在 GitHub 上的收藏数 (star) 达到了 100²。在此特别感谢 China^{La}T_EX 以及 [La](#)T_EX 工作室对于本系列模板的大力宣传与推广。[La](#)T_EX 工作室网站上有很多精彩的帖子和精致的模板, 欢迎大家去挖掘里面的宝藏, 这也是国内最全面的 [La](#)T_EX 相关的网站。

如果你喜欢我们的模板, 你可以在 GitHub 上收藏我们的模板。

9.7 捐赠

如果您非常喜爱我们的模板或者我, 你还可以选择捐赠³以表达您对我们模板和我的支持。本模板自 3.08 版本发布了捐赠信息之后, 收到了超过 1500 元的捐赠 (四舍五入就是一个亿), 非常感谢!



微信



支付宝

² 截止 3.10 版本正式发布, star 数为 374。

³ 最好在捐赠时备注信息。

赞赏费用的使用解释权归 **ElegantL^AT_EX** 所有，并且不接受监督，请自愿理性打赏。10元以上的赞赏，我们将列入捐赠榜，谢谢各位金主！

表 9.1: ElegantL^AT_EX 系列模板捐赠榜

捐赠者	金额	时间	渠道	捐赠者	金额	时间	渠道
Lerh	10 RMB	2019/05/15	微信	越过地平线	10 RMB	2019/05/15	微信
银桑	20 RMB	2019/05/27	微信	* 空	10 RMB	2019/05/30	微信
latextudio.net	666 RMB	2019/06/05	支付宝	A*n	40 RMB	2019/06/15	微信
* 夏	22 RMB	2019/06/15	微信	* 倩	21 RMB	2019/06/15	微信
Cassis	11 RMB	2019/06/30	微信	* 君	10 RMB	2019/07/23	微信
P*u	50 RMB	2019/07/30	微信	* 萌	19 RMB	2019/08/28	微信
曲豆豆	10 RMB	2019/08/28	微信	李博	100 RMB	2019/10/06	微信
Njustsl	10 RMB	2019/10/11	微信	刘志阔	99.99 RMB	2019/10/15	支付宝
* 韬	16 RMB	2019/10/17	微信	赤霓	12 RMB	2019/10/17	支付宝
追寻原风景	10 RMB	2019/10/28	微信	郭德良	88 RMB	2019/11/03	微信
自强不息	20 RMB	2019/11/04	支付宝	读书之虫	20 RMB	2019/11/18	微信
* 等	10 RMB	2019/11/18	微信	* 哲	20 RMB	2019/11/18	微信
佚名	10 RMB	2019/11/24	微信	Jiye Qian	66 RMB	2019/12/04	微信
* 阳	20 RMB	2019/12/05	微信	Catcher	11 RMB	2019/12/08	支付宝
希尔波特门徒	10 RMB	2019/12/09	支付宝	* 伟	10 RMB	2019/12/09	微信
Simon	20 RMB	2019/12/11	支付宝	流殇、浅忆	66.60 RMB	2019/12/18	支付宝
羽	10 RMB	2019/12/20	支付宝	* 珂	15 RMB	2019/12/20	微信
随风	20 RMB	2019/12/27	支付宝	Ws	23.30 RMB	2019/12/28	微信
初八	100 RMB	2020/01/02	支付宝	p*e	20 RMB	2020/01/03	微信
Shunmx	100 RMB	2020/01/03	微信	hj	10 RMB	2020/01/03	微信
F*5	10 RMB	2020/01/03	微信	S*m	20.20 RMB	2020/01/03	微信
二代青雉	13 RMB	2020/01/14	支付宝	*?	66 RMB	2020/01/15	微信
Mr. Xiong	20 RMB	2020/01/17	微信	* 博	15 RMB	2020/01/18	微信
* 者	10 RMB	2020/02/02	微信	Jackie	88.80 RMB	2020/02/09	微信

另外，为了表示感谢，我们制作了捐赠纪念证，欢迎大家来信告知邮箱以及姓名（艺名），我们将通过邮件发送电子版纪念证。



第十章 ElegantBook 设置说明

本模板基于基础的 book 文类，所以 book 的选项对于本模板也是有效的（纸张无效，因为模板有设备选项）。默认编码为 UTF-8，推荐使用 TeX Live 编译。本文编写环境为 Win10 (64bit) + TeX Live 2019，支持 pdfTeX 以及 XeTeX 编译。中文请尽量使用 XeTeX 编译。

10.1 语言模式

本模板内含两套语言环境，改变语言环境会改变图表标题的引导词（图，表），文章结构词（比如目录，参考文献等），以及定理环境中的引导词（比如定理，引理等）。不同语言模式的启用如下：

```
\documentclass[cn]{elegantbook}  
\documentclass[lang=cn]{elegantbook}
```

注 只有中文环境 (`lang=cn`) 才可以输入中文。另外如果抄录环境 (`lstlisting`) 中有中文字符，请务必使用 XeTeX 编译。

10.2 设备选项

最早我们在 ElegantNote 模板中加入了设备选项 (`device`)，后来，我们觉得这个设备选项的设置可以应用到 ElegantBook 中¹，而且 Book 一般内容比较多，如果在 iPad 上看无需切边，放大，那用户的阅读体验将会得到巨大提升。你可以使用下面的选项将版面设置为 iPad 设备模式²

```
\documentclass[pad]{elegantbook} %or  
\documentclass[device=pad]{elegantbook}
```

10.3 颜色主题

本模板内置 5 组颜色主题，分别为 `green`³、`cyan`、`blue`（默认）、`gray`、`black`。另外还有一个自定义的选项 `nocolor`。调用颜色主题 `green` 的方法为

```
\documentclass[green]{elegantbook} %or  
\documentclass[color=green]{elegantbook}
```

¹不过因为 ElegantBook 模板封面图片的存在，在修改页面设计时，需要对图片进行裁剪。

²默认为 `normal` 模式，也即 A4 纸张大小。

³为原先默认主题。

表 10.1: ElegantBook 模板中的颜色主题

	green	cyan	blue	gray	black	主要使用的环境
structure						chapter section subsection
main						definition exercise problem
second						theorem lemma corollary
third						proposition

如果需要自定义颜色的话请选择 `nocolor` 选项或者使用 `color=none`, 然后在导言区定义 `structurecolor`、`main`、`second`、`third` 颜色, 具体方法如下:

```
\definecolor{structurecolor}{RGB}{0,0,0}
\definecolor{main}{RGB}{70,70,70}
\definecolor{second}{RGB}{115,45,2}
\definecolor{third}{RGB}{0,80,80}
```

10.4 封面

10.4.1 封面个性化

从 3.10 版本开始, 封面更加弹性化, 用户可以自行选择输出的内容, 包括 `\title` 在内的所有封面元素都可为空。目前封面的元素有

- 标题: `\title`
- 副标题: `\subtitle`
- 作者: `\author`
- 机构: `\institute`
- 日期: `\date`
- 版本: `\version`
- 箴言: `\extrainfo`
- 封面图: `\cover`
- 徽标: `\logo`

另外, 额外增加一个 `\bioinfo` 命令, 有两个选项, 分别是信息标题以及信息内容。比如需要显示 User Name: 111520, 则可以使用

```
\bioinfo{User Name}{111520}
```

10.4.2 封面图

本模板使用的封面图片来源于 [pixabay.com](#)⁴，图片完全免费，可用于任何场景。封面图片的尺寸为 1280×1024 ，更换图片的时候请严格按照封面图片尺寸进行裁剪。推荐一个免费的在线图片裁剪网站 [fotor.com](#)。用户 QQ 群内有一些合适尺寸的封面，欢迎取用。

10.4.3 徽标

本文用到的 Logo 比例为 1:1，也即正方形图片，在更换图片的时候请选择合适的图片进行替换。

10.4.4 自定义封面

另外，如果需要使用自定义的封面，比如 Adobe illustrator 或者其他软件制作的 A4 PDF 文档，请把 `\maketitle` 注释掉，然后借助 `pdfpages` 宏包将自制封面插入即可。如果使用 `titlepage` 环境，也是类似。如果需要 2.x 版本的封面，请参考 [etitlepage](#)。

10.5 章标题

本模板内置 2 套章标题显示风格，包含 `hang`（默认）与 `display` 两种风格，区别在于章标题单行显示（`hang`）与双行显示（`display`），本说明使用了 `hang`。调用方式为

```
\documentclass[hang]{elegantbook} %or  
\documentclass[titlestyle=hang]{elegantbook}
```

在章标题内，章节编号默认是以数字显示，也即第 1 章，第 2 章等等，如果想要把数字改为中文，可以使用

```
\documentclass[chinese]{elegantbook} %or  
\documentclass[scheme=chinese]{elegantbook}
```

10.6 数学环境简介

在我们这个模板中，我们定义了两种不同的定理模式 `mode`，包括简单模式（`simple`）和炫彩模式（`fancy`），默认为 `fancy` 模式，不同模式的选择为

```
\documentclass[simple]{elegantbook} %or  
\documentclass[mode=simple]{elegantbook}
```

本模板定义了四大类环境

- 定理类环境，包含标题和内容两部分，全部定理类环境的编号均以章节编号。根据格式的不同分为 3 种

⁴感谢 ChinaTeX 提供免费图源网站，另外还推荐 [pexels.com](#)。

- **definition** 环境，颜色为 `main`；
- **theorem、lemma、corollary** 环境，颜色为 `second`；
- **proposition** 环境，颜色为 `third`。
- 示例类环境，有 **example、problem、exercise** 环境（对应于例、例题、练习），自动编号，编号以章节为单位，其中 **exercise** 有提示符。
- 提示类环境，有 **note** 环境，特点是：无编号，有引导符。
- 结论类环境，有 **conclusion、assumption、property、remark、solution** 环境⁵，三者均以粗体的引导词为开头，和普通段落格式一致。

10.6.1 定理类环境的使用

由于本模板使用了 `tcolorbox` 宏包来定制定理类环境，所以和普通的定理环境的使用有些许区别，定理的使用方法如下：

```
\begin{theorem}{theorem name}{label}
The content of theorem.
\end{theorem}
```

第一个必选项 `theorem name` 是定理的名字，第二个必选项 `label` 是交叉引用时所用到的标签，交叉引用的方法为 `\ref{thm:label}`。请注意，交叉引用时必须加上前缀 `thm:`。

其他相同用法的定理类环境有：

表 10.2: 定理类环境

环境名	标签名	前缀	交叉引用
<code>definition</code>	<code>label</code>	<code>def</code>	<code>\ref{def:label}</code>
<code>theorem</code>	<code>label</code>	<code>thm</code>	<code>\ref{thm:label}</code>
<code>lemma</code>	<code>label</code>	<code>lem</code>	<code>\ref{lem:label}</code>
<code>corollary</code>	<code>label</code>	<code>cor</code>	<code>\ref{cor:label}</code>
<code>proposition</code>	<code>label</code>	<code>pro</code>	<code>\ref{pro:label}</code>

10.6.2 其他环境的使用

其他三种环境没有选项，可以直接使用，比如 `example` 环境的使用方法与效果：

```
\begin{example}
This is the content of example environment.
\end{example}
```

这几个都是同一类环境，区别在于

- 示例环境（`example`）、练习（`exercise`）与例题（`problem`）章节自动编号；
- 注意（`note`），练习（`exercise`）环境有提醒引导符；

⁵本模板还添加了一个 `result` 选项，用于隐藏 `solution` 和 `proof` 环境，默认为显示（`result=answer`），隐藏使用 `result=noanswer`。

- 结论（`conclusion`）等环境都是普通段落环境，引导词加粗。

10.7 装饰物

本模板为章节后和页面下方的装饰物（`base`）添加了隐藏选项，有 `show`（默认）和 `hide` 两个选项。

```
\documentclass[hide]{elegantbook} % or
\documentclass[base=hide]{elegantbook}
```

10.8 列表环境

本模板借助于 `tikz` 定制了 `itemize` 和 `enumerate` 环境，其中 `itemize` 环境修改了 3 层嵌套，而 `enumerate` 环境修改了 4 层嵌套（仅改变颜色）。示例如下

- | | |
|---|--|
| <ul style="list-style-type: none"> • first item of nesti; • second item of nesti; <ul style="list-style-type: none"> • first item of nestii; • second item of nestii; <ul style="list-style-type: none"> • first item of nestiii; • second item of nestiii. | <ul style="list-style-type: none"> 1. first item of nesti; 2. second item of nesti; <ul style="list-style-type: none"> (a). first item of nestii; (b). second item of nestii; <ul style="list-style-type: none"> I. first item of nestiii; II. second item of nestiii. |
|---|--|

10.9 参考文献

此模板使用了 `BIBTEX` 来生成参考文献，在中文示例中，使用了 `gbt7714` 宏包。参考文献示例：[1-3] 使用了中国一个大型的 P2P 平台（人人贷）的数据来检验男性投资者和女性投资者在投资表现上是否有显著差异。

你可以在谷歌学术、Mendeley、Endnote 中获得文献条目（`bib item`），然后把它们添加到 `reference.bib` 中。在文中引用的时候，引用它们的键值（`bib key`）即可。注意需要在编译的过程中添加 `BIBTEX` 编译。如果你想添加未引用的文献，可以使用

```
\nocite{EINAV2010,Havrylchyk2018} % or include some bibitems
\nocite{*} % include all the bibitems
```

本模板还添加了 `cite=numbers`、`cite=super` 和 `cite=authoryear` 三个参考文献选项，用于设置参考文献格式的设置，默认为 `numbers`。据我们所知，理工科类一般使用数字形式 `numbers` 或者上标形式 `super`，而文科类使用作者-年份 `authoryear` 比较多，所以我们将 `numbers` 作为默认格式。如果需要改为 `cite=super` 或者 `authoryear`，可以使用

```
\documentclass[cite=super]{elegantbook} % set super style ref style
\documentclass[super]{elegantbook}
\documentclass[cite=authoryear]{elegantbook} % set author year ref style
```

```
\documentclass[authoryear]{elegantbook}
```

为了方便文献样式修改，模板引入了 `bibstyle` 选项，默认为 `apalike`，更多的选择可以参考 **BIBTeX Bibliography Styles**。用法为

```
\documentclass[bibstyle=apalike]{elegantbook}
```

10.10 添加序章

如果你想在第一章前面添序章，不改变原本章节序号，可以在第一章内容前面使用

```
\chapter*{Introduction}  
\markboth{Introduction}{Introduction}  
The content of introduction.
```

10.11 目录选项与深度

本模板添加了一个目录选项 `toc`，可以设置目录为单栏（`onecol`）和双栏（`twocol`）显示，比如双栏显示可以使用

```
\documentclass[twocol]{elegantbook}  
\documentclass[toc=twocol]{elegantbook}
```

默认本模板目录深度为 1，你可以在导言区使用

```
\setcounter{tocdepth}{2}
```

将其修改为 2 级目录（章与节）显示。

10.12 章节摘要

模板新增了一个章节摘要环境（`introduction`），使用示例

```
\begin{introduction}  
  \item Definition of Theorem  
  \item Ask for help  
  \item Optimization Problem  
  \item Property of Cauchy Series  
  \item Angle of Corner  
\end{introduction}
```

效果如下：



内容提要

- | | |
|--|--|
| <input type="checkbox"/> Definition of Theorem
<input type="checkbox"/> Ask for help
<input type="checkbox"/> Optimization Problem | <input type="checkbox"/> Property of Cauchy Series
<input type="checkbox"/> Angle of Corner |
|--|--|

环境的标题文字可以通过这个环境的可选参数进行修改，修改方法为：

```
\begin{introduction}[Brief Introduction]
...
\end{introduction}
```

10.13 章后习题

前面我们介绍了例题和练习两个环境，这里我们再加一个，章后习题（`problemset`）环境，用于在每一章结尾，显示本章的练习。使用方法如下

```
\begin{problemset}
  \item exercise 1
  \item exercise 2
  \item exercise 3
\end{problemset}
```

效果如下：

第十章 习题

1. exercise 1
2. exercise 2
3. exercise 3
4. 测试数学公式

$$a^2 + b^2 = c_{2_i}(1, 2)[1, 23] \quad (10.1)$$

注 如果你想把 `problemset` 环境的标题改为其他文字，你可以类似于 `introduction` 环境修改 `problemset` 的可选参数。另外，目前这个环境会自动出现在目录中，但是不会出现在页眉页脚信息中（待解决）。

解 如果你想把 `problemset` 环境的标题改为其他文字，你可以类似于 `introduction` 环境修改 `problemset` 的可选参数。另外，目前这个环境会自动出现在目录中，但是不会出现在页眉页脚信息中（待解决）。

10.14 旁注

在 3.08 版本中，我们引入了旁注设置选项 `marginpar=margintrue` 以及测试命令 `\elegantpar`，但是由此带来一堆问题。我们决定在 3.09 版本中将其删除，并且，在旁注

命令得到大幅度优化之前,不会将此命令再次引入书籍模板中。对此造成各位用户的不方便,非常抱歉!不过我们保留了 `\marginpar` 这个选项,你可以使用 `\marginpar=margintrue` 获得保留右侧旁注的版面设计。然后使用系统自带的 `\marginpar` 或者 `\marginnote` 宏包的 `\marginnote` 命令。

注 在使用旁注的时候,需要注意的是,文本和公式可以直接在旁注中使用。

```
% text  
\marginpar{margin paragraph text}  
  
% equation  
\marginpar{  
  \begin{equation}  
    a^2 + b^2 = c^2  
  \end{equation}  
}
```

但是浮动体(表格、图片)需要注意,不能用浮动体环境,需要使用直接插图命令或者表格命令环境。然后使用 `\captionof` 为其设置标题。为了得到居中的图表,可以使用 `\centerline` 命令或者 `center` 环境。更多详情请参考: [Caption of Figure in Marginpar](#)。

```
% graph with centerline command  
\marginpar{  
  \centerline{  
    \includegraphics[width=0.2\textwidth]{logo.png}  
  }  
  \captionof{figure}{your figure caption}  
}  
  
% graph with center environment  
\marginpar{  
  \begin{center}  
    \includegraphics[width=0.2\textwidth]{logo.png}  
    \captionof{figure}{your figure caption}  
  \end{center}  
}
```

第十一章 字体选项

字体选项独立成章的原因是，我们希望本模板的用户关心模板使用的字体，知晓自己使用的字体以及遇到字体相关的问题能更加便捷地找到答案。

重要提示：从 3.10 版本更新之后，沿用至今的 `newtx` 系列字体被重新更改为 `cm` 字体。并且新增中文字体（`chinesefont`）选项。

11.1 数学字体选项

本模板定义了一个数学字体选项（`math`），可选项有三个：

1. `math=cm`（默认），使用 `LATEX` 默认数学字体（推荐，无需声明）；
2. `math=newtx`，使用 `newtxmath` 设置数学字体（潜在问题比较多）。
3. `math=mtpro2`，使用 `mtpro2` 宏包设置数学字体，要求用户已经成功安装此宏包。

11.2 使用 `newtx` 系列字体

如果需要使用原先版本的 `newtx` 系列字体，可以通过显示声明数学字体：

```
\documentclass[math=newtx]{elegantbook}
```

11.2.1 连字符

如果使用 `newtx` 系列字体宏包，需要注意下连字符的问题。

$$\int_{R^q} f(x, y) dy \text{.off} \quad (11.1)$$

的代码为

```
\begin{equation}
\int_{R^q} f(x, y) dy \text{.of } \kern0pt f
\end{equation}
```

11.2.2 宏包冲突

另外在 3.08 版本中，有用户反馈模板在和 `yhmath` 以及 `esvect` 等宏包搭配使用的时候会出现报错：

```
LaTeX Error:
Too many symbol fonts declared.
```

原因是在使用 `newtxmath` 宏包时，重新定义了数学字体用于大型操作符，达到了最多 16 个数学字体的上限，在调用其他宏包的时候，无法新增数学字体。为了减少调用非常用宏包，在此给出如何调用 `yhmath` 以及 `esvect` 宏包的方法。

请在 `elegantbook.cls` 内搜索 `yhmath` 或者 `esvect`，将你所需要的宏包加载语句取消注释即可。

```
%%% use yhmath pkg, uncomment following code
% \let\oldwidering\widering
% \let\widering\undefined
% \RequirePackage{yhmath}
% \let\widering\oldwidering

%%% use esvect pkg, uncomment following code
% \RequirePackage{esvect}
```

11.3 中文字体选项

模板从 3.10 版本提供中文字体选项 `chinesefont`，可选项有

1. `ctexfont`: 默认选项，使用 `ctex` 宏包根据系统自行选择字体，可能存在字体缺失的问题，更多内容参考 `ctex` 宏包[官方文档](#)¹。
2. `founder`: 方正字体选项，调用 `ctex` 宏包并且使用 `fontset=none` 选项，然后设置字体为方正四款免费字体，方正字体下载注意事项见后文。
3. `nofont`: 调用 `ctex` 宏包并且使用 `fontset=none` 选项，不设定中文字体，用户可以自行设置中文字体，具体见后文。

注 使用 `founder` 选项或者 `nofont` 时，必须使用 X_E^AT_EX 进行编译。

11.3.1 方正字体选项

由于使用 `ctex` 宏包默认调用系统已有的字体，部分系统字体缺失严重，因此，用户希望能够使用其它字体，我们推荐使用方正字体。方正的方正书宋、方正黑体、方正楷体、方正仿宋四款字体均可免费试用，且可用于商业用途。

用户可以自行从[方正字体官网](#)下载此四款字体，在下载的时候请**务必**注意选择 GBK 字符集

也可以使用 [L^AT_EX 工作室](#)提供的[方正字体](#)，提取码为：njy9 进行安装。安装时，Win 10 用户请右键选择为全部用户安装，否则会找不到字体。

11.3.2 其他中文字体

如果你想完全自定义字体²，你可以选择 `chinesefont=nofont`，然后在导言区设置

¹可以使用命令提示符，输入 `texdoc ctex` 调出本地 `ctex` 宏包文档

²这里仍然以方正字体为例。

The screenshot shows a web-based font purchase interface. At the top, there are three tabs: '全部字体订单' (All Font Orders), '待付款' (Pending Payment), and '已完成' (Completed). Below the tabs is a search bar with the placeholder '字体/订单号' (Font/Order Number) and a '搜索' (Search) button. A note below the search bar states: '如果订单中包含方正黑体、方正文宋、方正仿宋、方正楷体这四款字体，针对“商业发布”使用方式免费，其它字体仅用于“个人非商业”使用' (If the order contains FZHei-B01, FZShuSong-Z01, FZKai-Z03, and FZFangSong-Z02, the 'Commercial Release' usage method is free, while other fonts are only for 'Personal Non-commercial' use). The main area is a table with the following columns: 字体名称 (Font Name), 编码 (Encoding), 单价 (Unit Price), 实付价 (Actual Payment), 交易状态 (Transaction Status), and 操作 (Operations). The table has four rows, each representing a purchased font. The first three rows have a status of '免费' (Free) and '已完成' (Completed), while the fourth row has a status of '已支付' (Paid). A '下载字体' (Download Font) button is located in the '操作' column of the fourth row.

字体名称	编码	单价	实付价	交易状态	操作
订单号: C20200204164821OW1F				2020-02-04 16:48:21	
方正仿宋_GBK • 免费商用	简繁扩展(GBK)	¥ 0.00		免费	已完成
方正黑体_GBK • 免费商用	简繁扩展(GBK)	¥ 0.00			
方正文宋_GBK • 免费商用	简繁扩展(GBK)	¥ 0.00			
方正楷体_GBK • 免费商用	简繁扩展(GBK)	¥ 0.00			

```
\setCJKmainfont [BoldFont={FZHei-B01},ItalicFont={FZKai-Z03}]{FZShuSong-Z01}
\setCJKsansfont [BoldFont={FZHei-B01},ItalicFont={FZHei-B01}]{FZHei-B01}
\setCJKmonofont [BoldFont={FZHei-B01},ItalicFont={FZHei-B01}]{FZFangSong-Z02}
\setCJKfamilyfont{zhsong}{FZShuSong-Z01}
\setCJKfamilyfont{zhhei}{FZHei-B01}
\setCJKfamilyfont{zhkai}{FZKai-Z03}
\setCJKfamilyfont{zhfs}{FZFangSong-Z02}
\newcommand*\songti{\CJKfamily{zhsong}}
\newcommand*\heiti{\CJKfamily{zhhei}}
\newcommand*\kaishu{\CJKfamily{zhkai}}
\newcommand*\fangsong{\CJKfamily{zhfs}}
```

第十二章 ElegantBook 写作示例

内容提要

- 积分定义 12.1
- 柯西列性质 12.1.1
- Fubini 定理 12.1
- 韦达定理
- 最优性原理 12.1

12.1 Lebesgue 积分

在前面各章做了必要的准备后，本章开始介绍新的积分。在 Lebesgue 测度理论的基础上建立了 Lebesgue 积分，其被积函数和积分域更一般，可以对有界函数和无界函数统一处理。正是由于 Lebesgue 积分的这些特点，使得 Lebesgue 积分比 Riemann 积分具有在更一般条件下的极限定理和累次积分交换积分顺序的定理，这使得 Lebesgue 积分不仅在理论上更完善，而且在计算上更灵活有效。

Lebesgue 积分有几种不同的定义方式。我们将采用逐步定义非负简单函数，非负可测函数和一般可测函数积分的方式。

由于现代数学的许多分支如概率论、泛函分析、调和分析等常常用到一般空间上的测度与积分理论，在本章最后一节将介绍一般的测度空间上的积分。

12.1.1 积分的定义

我们将通过三个步骤定义可测函数的积分。首先定义非负简单函数的积分。以下设 E 是 \mathcal{R}^n 中的可测集。

定义 12.1. 可积性

设 $f(x) = \sum_{i=1}^k a_i \chi_{A_i}(x)$ 是 E 上的非负简单函数，其中 $\{A_1, A_2, \dots, A_k\}$ 是 E 上的一个可测分割， a_1, a_2, \dots, a_k 是非负实数。定义 f 在 E 上的积分为 $\int_a^b f(x) dx$

$$\int_E f dx = \sum_{i=1}^k a_i m(A_i) \pi \alpha \beta \sigma \gamma \nu \xi \epsilon \varepsilon. \oint_a^b \oint_a^b \prod_{i=1}^n$$
 (12.1)

一般情况下 $0 \leq \int_E f dx \leq \infty$ 。若 $\int_E f dx < \infty$ ，则称 f 在 E 上可积。



一个自然的问题是，Lebesgue 积分与我们所熟悉的 Riemann 积分有什么联系和区别？在 4.4 在我们将详细讨论 Riemann 积分与 Lebesgue 积分的关系。这里只看一个简单的例子。设 $D(x)$ 是区间 $[0, 1]$ 上的 Dirichlet 函数。即 $D(x) = \chi_{Q_0}(x)$ ，其中 Q_0 表示 $[0, 1]$ 中

的有理数的全体。根据非负简单函数积分的定义, $D(x)$ 在 $[0, 1]$ 上的 Lebesgue 积分为

$$\int_0^1 D(x)dx = \int_0^1 \chi_{Q_0}(x)dx = m(Q_0) = 0 \quad (12.2)$$

即 $D(x)$ 在 $[0, 1]$ 上是 Lebesgue 可积的并且积分值为零。但 $D(x)$ 在 $[0, 1]$ 上不是 Riemann 可积的。

有界变差函数是与单调函数有密切联系的一类函数。有界变差函数可以表示为两个单调递增函数之差。与单调函数一样, 有界变差函数几乎处处可导。与单调函数不同, 有界变差函数类对线性运算是封闭的, 它们构成一线空间。练习题 12.1 是一个性质的证明。

练习 12.1 设 $f \notin L(\mathcal{R}^1)$, g 是 \mathcal{R}^1 上的有界可测函数。证明函数

$$I(t) = \int_{\mathcal{R}^1} f(x+t)g(x)dx \quad t \in \mathcal{R}^1 \quad (12.3)$$

是 \mathcal{R}^1 上的连续函数。

解 即 $D(x)$ 在 $[0, 1]$ 上是 Lebesgue 可积的并且积分值为零。但 $D(x)$ 在 $[0, 1]$ 上不是 Riemann 可积的。

证明 即 $D(x)$ 在 $[0, 1]$ 上是 Lebesgue 可积的并且积分值为零。但 $D(x)$ 在 $[0, 1]$ 上不是 Riemann 可积的。

定理 12.1. Fubini 定理

(1) 若 $f(x, y)$ 是 $\mathcal{R}^p \times \mathcal{R}^q$ 上的非负可测函数, 则对几乎处处的 $x \in \mathcal{R}^p$, $f(x, y)$ 作为 y 的函数是 \mathcal{R}^q 上的非负可测函数, $g(x) = \int_{\mathcal{R}^q} f(x, y)dy$ 是 \mathcal{R}^p 上的非负可测函数。并且

$$\int_{\mathcal{R}^p \times \mathcal{R}^q} f(x, y)dxdy = \int_{\mathcal{R}^p} \left(\int_{\mathcal{R}^q} f(x, y)dy \right) dx. \quad (12.4)$$

(2) 若 $f(x, y)$ 是 $\mathcal{R}^p \times \mathcal{R}^q$ 上的可积函数, 则对几乎处处的 $x \in \mathcal{R}^p$, $f(x, y)$ 作为 y 的函数是 \mathcal{R}^q 上的可积函数, 并且 $g(x) = \int_{\mathcal{R}^q} f(x, y)dy$ 是 \mathcal{R}^p 上的可积函数。而且 12.4 成立。



笔记 在本模板中, 引理 (lemma), 推论 (corollary) 的样式和定理 12.1 的样式一致, 包括颜色, 仅仅只有计数器的设置不一样。

我们说一个实变或者复变量的实值或者复值函数是在区间上平方可积的, 如果其绝对值的平方在该区间上的积分是有限的。所有在勒贝格积分意义下平方可积的可测函数构成一个希尔伯特空间, 也就是所谓的 L^2 空间, 几乎处处相等的函数归为同一等价类。形式上, L^2 是平方可积函数的空间和几乎处处为 0 的函数空间的商空间。

命题 12.1. 最优性原理

如果 u^* 在 $[s, T]$ 上为最优解, 则 u^* 在 $[s, T]$ 任意子区间都是最优解, 假设区间为 $[t_0, t_1]$ 的最优解为 u^* , 则 $u(t_0) = u^*(t_0)$, 即初始条件必须还是在 u^* 上。



我们知道最小二乘法可以用来处理一组数据, 可以从一组测定的数据中寻求变量之间的依赖关系, 这种函数关系称为经验公式。本课题将介绍最小二乘法的精确定义及如何寻求点与点之间近似成线性关系时的经验公式。假定实验测得变量之间的 n 个数据, 则在平面上, 可以得到 n 个点, 这种图形称为“散点图”, 从图中可以粗略看出这些点大致

散落在某直线近旁，我们认为其近似为一线性函数，下面介绍求解步骤。

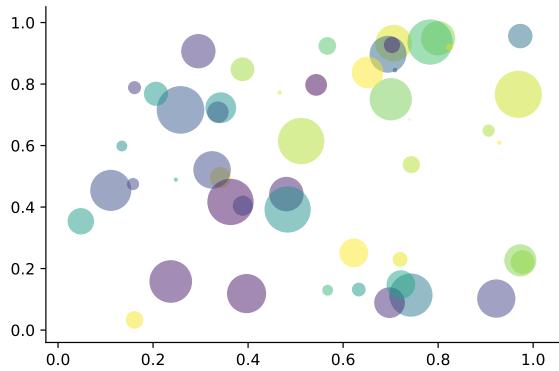


图 12.1：散点图示例 $\hat{y} = a + bx$

以最简单的一元线性模型来解释最小二乘法。什么是一元线性模型呢？监督学习中，如果预测的变量是离散的，我们称其为分类（如决策树，支持向量机等），如果预测的变量是连续的，我们称其为回归。回归分析中，如果只包括一个自变量和一个因变量，且二者的关系可用一条直线近似表示，这种回归分析称为一元线性回归分析。如果回归分析中包括两个或两个以上的自变量，且因变量和自变量之间是线性关系，则称为多元线性回归分析。对于二维空间线性是一条直线；对于三维空间线性是一个平面，对于多维空间线性是一个超平面。

性质 柯西列的性质

1. $\{x_k\}$ 是柯西列，则其子列 $\{x_k^i\}$ 也是柯西列。
2. $x_k \in \mathcal{R}^n$, $\rho(x, y)$ 是欧几里得空间，则柯西列收敛， (\mathcal{R}^n, ρ) 空间是完备的。

结论 回归分析 (regression analysis) 是确定两种或两种以上变量间相互依赖的定量关系的一种统计分析方法。运用十分广泛，回归分析按照涉及的变量的多少，分为一元回归和多元回归分析；按照因变量的多少，可分为简单回归分析和多重回归分析；按照自变量和因变量之间的关系类型，可分为线性回归分析和非线性回归分析。

第十二章 习题

1. 设 A 为数域 K 上的 n 级矩阵。证明：如果 K^n 中任意非零列向量都是 A 的特征向量，则 A 一定是数量矩阵。
2. 证明：不为零矩阵的幂零矩阵不能对角化。
3. 设 $A = (a_{ij})$ 是数域 K 上的一个 n 级上三角矩阵，证明：如果 $a_{11} = a_{22} = \dots = a_{nn}$ ，并且至少有一个 $a_{kl} \neq 0 (k < l)$ ，则 A 一定不能对角化。

第十三章 常见问题集

我们根据用户社区反馈整理了下面一些常见的问题，用户在遇到问题时，应当首先查阅本手册和本部分的常见的问题。

1. 有没有办法章节用“第一章，第一节，(一)”这种？

见前文介绍，可以使用 `scheme=chinese` 设置。

2. 3.07 版本的 `cls` 的 `natbib` 加了 `numbers` 编译完了没变化，群主设置了不可更改了？

之前在 3.07 版本中在引入 `gbt7714` 宏包时，加入了 `authoryear` 选项，这个使得 `natbib` 设置了 `numbers` 也无法生效。3.08 和 3.09 版本中，模板新增加了 `numbers`、`super` 和 `authoryear` 文献选项，你可以参考前文设置说明。

3. 大佬，我想把正文字体改为亮色，背景色改为黑灰色。

页面颜色可以使用 `\pagecolor` 命令设置，文本命令可以参考[这里](#)进行设置。

4. Package `ctex` Error: CTeX fontset ‘Mac’ is unavailable.

在 Mac 系统下，中文编译请使用 X_ET_EX。

5. ! LaTeX Error: Unknown option ‘`scheme=plain`’ for package ‘`ctex`’.

你用的 CTeX 套装吧？这个里面的 `ctex` 宏包已经是 10 年前的了，与本模板使用的 `ctex` 宏集有很大区别。不建议 CTeX 套装了，请卸载并安装 T_EX Live 2019。

6. 我该使用什么版本？

请务必使用[最新正式发行版](#)，发行版间不定期可能会有更新（修复 bug 或者改进之类），如果你在使用过程中没有遇到问题，不需要每次更新[最新版](#)，但是在发行版更新之后，请尽可能使用最新版（发行版）！最新发行版可以在 GitHub 或者 T_EX Live 2019 内获取。

7. 我该使用什么编辑器？

你可以使用 T_EX Live 2019 自带的编辑器 T_EXworks 或者使用 T_EXstudio，T_EXworks 的自动补全，你可以参考我们的总结 [T_EXworks 自动补全](#)。推荐使用 T_EX Live 2019 + T_EXstudio。我自己用 VS Code 和 Sublime Text，相关的配置说明，请参考 [L_AT_EX 编译环境配置：Visual Studio Code 配置简介](#) 和 [Sublime Text 搭建 L_AT_EX 编写环境](#)。

8. 您好，我们想用您的 ElegantBook 模板写一本书。关于机器学习的教材，希望获得您的授权，谢谢您的宝贵时间。

模板的使用修改都是自由的，你们声明模板来源以及模板地址（GitHub 地址）即可，其他未尽事宜按照开源协议 LPPL-1.3c。做好之后，如果方便的话，可以给我们一个链接，我把你们的教材放在 ElegantL_AT_EX 用户作品集里。

9. 请问交叉引用是什么？

本群和本模板适合有一定 **LATEX** 基础的用户使用，新手请先学习 **LATEX** 的基础，理解各种概念，否则你将寸步难行。

10. 定义等环境中无法使用加粗命令么？

是这样的，默认中文并没加粗命令，如果你想在定义等环境中使用加粗命令，请使用 `\heiti` 等字体命令，而不要使用 `\textbf`。或者，你可以将 `\textbf` 重新定义为 `\heiti`。英文模式不存在这个问题。

11. 代码高亮环境能用其他语言吗？

可以的，ElegantBook 模板用的是 `listings` 宏包，你可以在环境(`lstlisting`)之后加上语言（比如 Python 使用 `language=Python` 选项），全局语言修改请使用 `lset` 命令，更多信息请参考宏包文档。

12. 群主，什么时候出 Beamer 的模板（主题），ElegantSlide 或者 ElegantBeamer？

由于 Beamer 中有一个很优秀的主题 `Metropolis`。在找到非常好的创意之前不会发布正式的 Beamer 主题，如果你非常希望得到 Elegant**LATEX** “官方”的主题，请在用户 QQ 群内下载测试主题 `PreElegantSlide`。正式版制作计划在今年或者明年。



第十四章 版本更新历史

根据用户的反馈，我们不断修正和完善模板。截止到此次更新，ElegantBook 模板在 GitHub 上有将近 100 次提交，正式发行版本（release）有 16 次。

2020/02/10 更新：版本 3.10 正式发布

- ① 增加数学字体选项 `math`，可选项为 `newtx` 和 `cm`。
 重要提示：原先通过 `newtxmath` 宏包设置的数学字体改为 L^AT_EX 默认数学字体，如果需要保持原来的字体，需要显式声明数学字体 (`math=newtx`)；
- ② 新增中文字体选项 `chinesefont`，可选项为 `ctexfont`、`founder` 和 `nofont`。
- ③ 将封面作者信息设置为可选，并且增加自定义信息命令 `\bioinfo`；
- ④ 在说明文档中增加版本历史，新增 `\datechange` 命令和 `change` 环境；
- ⑤ 增加汉化章节选项 `scheme`，可选项为汉化 `chinese`；
- ⑥ 由于 `\lvert` 问题已经修复，重新调整 `ctex` 宏包和 `amsmath` 宏包位置。
- ⑦ 修改页眉设置，去除了 `\lastpage` 以避免 page anchor 问题，加入 `\frontmatter`。
- ⑧ 修改参考文献选项 `cite`，可选项为数字 `number`、作者-年份 `authoryear` 以及上标 `super`。
- ⑨ 新增参考文献样式选项 `bibstyle`，并将英文模式下参考文献样式 `apalike` 设置为默认值，中文仍然使用 `gbt7714` 宏包设置。

2019/08/18 更新：版本 3.09 正式发布

- ① 由于 `\elegantpar` 存在一些问题，暂时性删除 `\elegantpar` 命令，并提示用户改用 `\marginnote` 和 `\marginpar` 旁注/边注命令。
- ② 积分操作符统一更改为 `esint` 宏包设置；
- ③ 新增目录选项 `toc`，可选项为单栏 `onecol` 和双栏 `twocol`；
- ④ 手动增加参考文献选项 `cite`，可选项为上标形式 `super`；
- ⑤ 修正章节习题（`problemset`）环境。

参考文献

- [1] QUADRINI V. Financial Frictions in Macroeconomic Fluctuations[J]. FRB Richmond Economic Quarterly, 2011, 97(3):209-254.
- [2] CARLSTROM C T, FUERST T S. Agency Costs, Net Worth, and Business Fluctuations: A Computable General Equilibrium Analysis[J]. The American Economic Review, 1997:893-910.
- [3] LI Q, CHEN L, ZENG Y. The Mechanism and Effectiveness of Credit Scoring of P2P Lending Platform: Evidence from Renrendai.com[J]. China Finance Review International, 2018, 8(3):256-274.
- [4] 方军雄. 所有制、制度环境与信贷资金配置[J]. 经济研究, 2007(12):82-92.
- [5] 刘凤良, 章潇萌, 于泽. 高投资、结构失衡与价格指数二元分化[J]. 金融研究, 2017(02):54-69.
- [6] 吕捷, 王高望. CPI 与 PPI “背离”的结构性解释[J]. 经济研究, 2015, 50(04):136-149.

附录 A 基本数学工具

本附录包括了计量经济学中用到的一些基本数学，我们扼要论述了求和算子的各种性质，研究了线性和某些非线性方程的性质，并复习了比例和百分数。我们还介绍了一些在应用计量经济学中常见的特殊函数，包括二次函数和自然对数，前 4 节只要求基本的代数技巧，第 5 节则对微分学进行了简要回顾；虽然要理解本书的大部分内容，微积分并非必需，但在一些章末附录和第 3 篇某些高深专题中，我们还是用到了微积分。

A.1 求和算子与描述统计量

求和算子是用以表达多个数求和运算的一个缩略符号，它在统计学和计量经济学分析中扮演着重要作用。如果 $\{x_i : i = 1, 2, \dots, n\}$ 表示 n 个数的一个序列，那么我们就把这 n 个数的和写为：

$$\sum_{i=1}^n x_i \equiv x_1 + x_2 + \cdots + x_n \quad (\text{A.1})$$