



Reinforcement Learning: An Introduction

强化学习导论第二版习题解答

作者：吕昀璿

组织：UESTC

时间：March 24, 2020



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

目录

1	介绍	1
1.1	强化学习	1
1.2	例子	1
1.3	强化学习的要素	1
1.4	局限和范围	1
1.5	拓展例子：井字游戏	1
1.6	总结	2
1.7	强化学习的早期历史	2
	第一部分 表格解决方法	3
2	多臂赌博机	4
2.1	k 臂赌博机问题	4
2.2	动作值方法	4
2.3	10 臂试验	4
2.4	渐增实现	5
2.5	非平稳问题	5
2.6	乐观初始值	5
2.7	置信上限动作选择	6
2.8	梯度赌博机算法	6
2.9	关联搜索（上下文赌博机）	7
2.10	总结	7
3	有限马尔可夫决策过程	8
3.1	Agent-环境接口	8
3.2	目标和奖励	9
3.3	回报和 episode	9
3.4	回合和连续任务的统一符号	10
3.5	策略和值函数	10
3.6	最优策略和最优值函数	13
3.7	最优和近似	16
3.8	总结	16

4	动态规划	17
4.1	策略评估	17
4.2	策略改进	18
4.3	策略迭代	18
4.4	值迭代	19
4.5	异步动态规划	20
4.6	广义策略迭代	20
4.7	动态规划效率	20
4.8	总结	20
5	蒙特卡罗方法	21
5.1	蒙特卡洛预测	21
5.2	动作值的蒙特卡洛估计	21
5.3	蒙特卡洛控制	22
5.4	无探索起点的蒙特卡洛控制	22
5.5	重要性采样的 off-policy 预测	22
5.6	渐增实现	23
5.7	Off-policy 蒙特卡洛控制	24
5.8	* 折扣的重要性采样	25
5.9	* 每决策的重要性采样	25
5.10	总结	25
6	时间差分学习	26
6.1	TD 预测	26
6.2	TD 预测方法的优势	27
6.3	TD(0) 的最优性	28
6.4	Sarsa: on-policy TD 控制	29
6.5	Q-learning: off-policy TD 控制	29
6.6	期望 Sarsa	30
6.7	最大化偏差与 Double Learning	30
6.8	游戏、后期状态和其他特殊情况	30
6.9	总结	30
7	n 步自举	31
7.1	n 步 TD 预测	31
7.2	n 步 Sarsa	32
7.3	n 步 off-policy 学习	32
7.4	* 控制变量的每决策方法	32
7.5	无重要性采样的 off-policy 学习: n 步树备份算法	34

7.6	* 一种统一算法: $Q(\sigma)$	35
7.7	总结	35
8	表格法进行规划和学习	36
8.1	模型和规划	36
8.2	Dyna: 综合规划, 行动和学习	36
8.3	当模型错误时	36
8.4	优先扫描	37
8.5	期望与采样更新	37
8.6	轨迹采样	37
8.7	实时动态规划	38
8.8	决策时规划	38
8.9	启发式搜索	38
8.10	展开算法	38
8.11	蒙特卡洛树搜索	38
8.12	本章总结	38
8.13	第一部分总结: 维度	38

第一章 介绍

1.1 强化学习

1.2 例子

1.3 强化学习的要素

1.4 局限和范围

1.5 拓展例子：井字游戏

🔥 **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？ □

解 当与自身对战时：

- 比起固定的对手，与自身对战将学习不同的策略，因为在这种情况下，对手也会有所变化。
- 由于对手也在不断变化，因此可能无法学习最佳策略。
- 可能卡在循环中。因为与自身博弈，自身策略和对手策略都在优化。
- 策略可以保持静态，因为就平均而言，通过每次迭代它们处于平局。

🔥 **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？ □

解 我们可以将状态标记为对称的唯一状态，这样我们的搜索空间更小，这样我们就可以更好地估计最佳玩法。

如果我们面对的对手在比赛时没有考虑对称性，那么我们也不应将状态标记为相同。因为对手也是环境的一部分，而环境给出的这些状态并不一致。

🔥 **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？ □

解 贪婪的玩家不会探索，因此通常会比非贪婪的玩家表现更差。

如果贪婪的玩家对状态的价值有一个完美的估计，那它将更好。


🔥 **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？ □

解 如果我们不从探索性动作中学习，那么所学到的状态概率将是随机的，因为我们不会更新在给定状态下采取给定动作时会发生的情况。

如果我们从探索性动作中进行学习，那么我们的极限概率应该是状态和动作选择的期望分布。

显然，由于玩家更好地理解正在玩的“游戏”，因此对概率密度的更全面的了解应该会带来更好的玩法。

 **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？ □

解 一种可能的方法是持有已保存的玩法库。例如，当在一组已知状态中，始终执行库中所对应的移动。这有点像国际象棋游戏，其中有很多“开场”位置被专家玩家认为是好的。这可以加快整个学习过程，或至少改善强化学习玩家的初期发挥。

由于井字游戏是如此简单，我们可以使用递归解决此问题，并计算所有可能的对手移动，并在每一步中选择能最大化我们获胜机会的移动。

1.6 总结

1.7 强化学习的早期历史


第一部分

表格解决方法

第二章 多臂赌博机

2.1 k 臂赌博机问题


2.2 动作值方法

 **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？ □

解 设动作集合中总共具有 n 个动作。在 ε -greedy 方法中，agent 有 ε 的概率机会从动作集合中随机选择，有 $1 - \varepsilon$ 的概率机会选择贪婪动作。已知 $\varepsilon = 0.5$ 和 $n = 2$ ，那么选择贪婪动作的概率为：

$$\frac{1}{n} \times \varepsilon + (1 - \varepsilon) = \frac{1}{2} \times 0.5 + (1 - 0.5) = 0.75$$


2.3 10 臂试验

 **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？ □

解 根据题意列出每一步的动作值，已选择的动作，和选择该动作的原因如下：

时间步	动作值	已选择的动作	选择原因	原因说明				
1	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	0	0	0	0	1	贪婪或随机	所有动作值相等，都为 0
0	0	0	0					
2	<table><tr><td>-1</td><td>0</td><td>0</td><td>0</td></tr></table>	-1	0	0	0	2	贪婪或随机	2、3、4 的动作值相等
-1	0	0	0					
3	<table><tr><td>-1</td><td>1</td><td>0</td><td>0</td></tr></table>	-1	1	0	0	2	贪婪	2 的动作值最大
-1	1	0	0					
4	<table><tr><td>-1</td><td>-1/2</td><td>0</td><td>0</td></tr></table>	-1	-1/2	0	0	2	随机	2 的动作值最小
-1	-1/2	0	0					
5	<table><tr><td>-1</td><td>1/3</td><td>0</td><td>0</td></tr></table>	-1	1/3	0	0	3	随机	2 的动作值最大
-1	1/3	0	0					

由表可知， ε 情况肯定在 A_4 和 A_5 发生，可能在 A_1 和 A_2 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。 □

解 $\varepsilon = 0.01$ 将有更好的表现，因为在两种情况下，当 $t \rightarrow \infty$ 时，我们都有 $Q_t \rightarrow q_*$ 。因此，在这种情况下，总奖励和选择最佳行动的可能性将比 $\varepsilon = 0.1$ 大 10 倍。

2.4 渐增实现

2.5 非平稳问题

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？□

解 推导过程与 (2.6) 类似：

$$\begin{aligned}
 Q_{n+1} &= Q_n + \alpha_n[R_n - Q_n] \\
 &= \alpha_n R_n + (1 - \alpha_n)Q_n \\
 &= \alpha_n R_n + (1 - \alpha_n)[\alpha_{n-1}R_{n-1} + (1 - \alpha_{n-1})Q_{n-1}] \\
 &= \alpha_n R_n + (1 - \alpha_n)\alpha_{n-1}R_{n-1} + (1 - \alpha_n)(1 - \alpha_{n-1})Q_{n-1} \\
 &= \alpha_n R_n + (1 - \alpha_n)\alpha_{n-1}R_{n-1} + (1 - \alpha_n)(1 - \alpha_{n-1})\alpha_{n-2}R_{n-2} + \cdots + \\
 &\quad (1 - \alpha_n)(1 - \alpha_{n-1})(1 - \alpha_{n-2}) \cdots (1 - \alpha_2)(1 - \alpha_1)Q_1 \\
 &= \left(\prod_{i=1}^n (1 - \alpha_i) \right) Q_1 + \sum_{i=1}^n \alpha_i R_i \prod_{k=i+1}^n (1 - \alpha_k)
 \end{aligned}$$

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??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。□

解 见 `exercise-programming/exercise2.5.py`。

2.6 乐观初始值

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？

解 在步骤 10 之后的某个时刻，agent 将找到最优值。然后它将贪婪地选择此值。小步长参数（相对于初始值 5 较小）意味着最优值的估计值将朝着其真实值缓慢收敛。

该真实值可能小于 5。这意味着，由于步长较小，其中一个次优动作的值仍接近 5。因此，在某个时刻，agent 又开始次优动作。

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

$$\beta_n \doteq \alpha / \bar{o}_n, \quad (2.1)$$

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：

$$\bar{o}_n \doteq \bar{o}_{n-1} + \alpha(1 - \bar{o}_{n-1}), \quad \text{对于 } n \geq 0, \quad \bar{o}_0 \doteq 0. \quad (2.2)$$

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。

解 考虑练习 2.4 的答案。由于 $\beta_1 = 1$ ，因此对于 $k > 1$ ， Q_k 与 Q_1 无关。现在有迹象表明，随着我们往前看，剩余总和中的权重会降低。即

$$\omega_i = \beta_i \prod_{k=i+1}^n (1 - \beta_k)$$

对于固定的 n 随 i 而增加。为此，观测到

$$\frac{\omega_{i+1}}{\omega_i} = \frac{\beta_{i+1}}{\beta_i(1 - \beta_{i+1})} = \frac{1}{1 - \alpha} > 1$$

如果假定 $\alpha < 1$ 时。如果 $\alpha = 1$ ，那么对于 $\forall t$ ， $\beta_t = 1$ 。

2.7 置信上限动作选择

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。□

解 在前 10 个步中，agent 会循环执行所有动作，因为当 $N_t(a) = 0$ 时， a 被认为是最大值。然后，在第 11 步，agent 通常会贪婪地选择。agent 将继续贪婪地选择，直到 $\ln t$ 超过 $N_t(a)$ 进行其他动作之一为止，在这种情况下，agent 将开始再次探索，从而减少了奖励。

请注意，从长远来看， $N_t = O(t)$ 和 $\ln(t)/t \rightarrow 0$ 。因此，该 agent 是“渐近贪婪的”。

2.8 梯度赌博机算法


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。□

解 令这两个动作分别用 0 和 1 表示。现在

$$\Pr\{A_t = 1\} = \frac{e^{H_t(1)}}{e^{H_t(0)} + e^{H_t(1)}} = \frac{1}{1 + e^{-x}}$$

其中 $x = H_t(1) - H_t(0)$ 是 1 相对于 0 的相对偏好。


2.9 关联搜索（上下文赌博机）

 **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 A），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？ □

解 假定奖励平稳。应该选择具有最高期望奖励的动作。对于第一种情形，动作 1 和 2 的期望值都为 0.5，即 $0.5 \times (0.1 + 0.9) = 0.5$ ， $0.5 \times (0.2 + 0.8) = 0.5$ 。因此这两个动作可以随机进行选择。

针对第二种情形，应该对每种颜色分别运行正常的赌博机方法。在每种情况下确定最佳动作的期望奖励为 0.55，即 $0.5 \times 0.2 + 0.5 \times 0.9 = 0.55$ 。

2.10 总结

 **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。 □

解 见 `exercise-programming/exercise2.11.py`。

第三章 有限马尔可夫决策过程

3.1 Agent-环境接口

- 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。 □

解 (1) 网格迷宫：状态为格子编号，动作为东西南北移动，奖励为到达出口；

(2) 棋类游戏：状态为棋子在棋盘上的位置，动作为棋子的移动，奖励为游戏结果；

(3) 自动驾驶：状态为周围环境、视觉、雷达等传感器信息，动作为转向、加速、刹车等，奖励为到达目的地、避免撞车；

(4) Atari 游戏：状态为屏幕像素输入，动作为键盘/鼠标移动，奖励为游戏增加分数。

- 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？ □

解 不足以。当我们没有足够的计算能力去定义状态和奖励时，例如围棋，必须通过深度学习框架来解决。还比如射击游戏，由于视线限制，agent 没有关于其他玩家的直接信息，但状态会受到队友和对手的影响，使 agent 无法确定先前的动作对当前情况的影响，这使得不是 MDP。

- 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？ □

解 这个问题是要求正确的线路来定义环境和 agent。正确的划分界限应该可以观察到 agent 的动作对状态的影响。


- 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。

解 表格如下：

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


3.2 目标和奖励

3.3 回报和 episode

 **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。 □

解

$$\sum_{s' \in \mathcal{S}^+} \sum_{r \in \mathcal{R}} p(s', r|s, a) = 1, \quad \text{for all } s \in \mathcal{S}^+, a \in \mathcal{A}(s) \text{ and } \mathcal{S}^+ \doteq \{\text{all states plus terminal state}\}$$

 **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？ □

解 对于回合任务，回报为：

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$


使用折扣后，回报为：

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$$

由于失败时奖励为-1，其他情况奖励设置为 0，所以每个时间的回报为：

$$G_t = -\gamma^{T-t-1}$$

该回报实际上与折扣的、连续的情况下的回报 $-\gamma^K$ 相同，其中 K 是失败之前的时间步长。

 **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？ □

解 如果您不使用 γ 来执行折扣，则无论 agent 花费多长时间，最大的回报始终为 +1。与

agent 进行沟通的正确方法是在逃离前的每个时间步加-1 的惩罚或增加折扣。

🔥 **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。 □

解

$$\begin{aligned} G_5 &= 0 \\ G_4 &= R_5 + \gamma G_5 = 2 + 0.5 \times 0 = 2 \\ G_3 &= R_4 + \gamma G_4 = 3 + 0.5 \times 2 = 4 \\ G_2 &= R_3 + \gamma G_3 = 6 + 0.5 \times 4 = 8 \\ G_1 &= R_2 + \gamma G_2 = 2 + 0.5 \times 8 = 6 \\ G_0 &= R_1 + \gamma G_1 = -1 + 0.5 \times 6 = 2 \end{aligned}$$

🔥 **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? □

解

$$\begin{aligned} G_1 &= 7 \times \sum_{k=0}^{\infty} \gamma^k R_{k+2} = 7 \times \frac{1}{1-\gamma} = 7 \times \frac{1}{1-0.9} = 70 \\ G_0 &= R_1 + \gamma \times 7 \times \sum_{k=1}^{\infty} \gamma^k R_{k+1} = 2 + 7 \times \frac{\gamma}{1-\gamma} = 2 + 7 \times \frac{0.9}{1-0.9} = 2 + 7 \times 90 = 65 \\ G_0 &= R_1 + \gamma G_1 = 2 + 0.9 \times 70 = 65 \end{aligned}$$

🔥 **练习 3.10** 证明 (3.10) 中的第二个等式。 □

解 因为奖励为不为 0 的常数, 以及 $\gamma < 1$, 则:

$$\begin{aligned} S_N &\doteq \sum_{k=0}^N \gamma^k \\ \gamma S_N - S_N &= \gamma^{N+1} - 1 \\ S_N &= \frac{1 - \gamma^{N+1}}{1 - \gamma} \\ G_t &\doteq \lim_{N \rightarrow \infty} S_N = \frac{1}{1 - \gamma} \end{aligned}$$

3.4 回合和连续任务的统一符号

3.5 策略和值函数

🔥 **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? □

解

$$\mathbb{E}_{\pi}[R_{t+1}|S_t = s] = \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a)r$$

🔥 **练习 3.12** 用 q_{π} 和 π 给出 v_{π} 的等式。 □

解

$$v_{\pi}(s) \doteq \sum_a \pi(a|s) q_{\pi}(s, a)$$

练习 3.13 用 v_{π} 和四个参数 p 给出 q_{π} 的等式。

解

$$q_{\pi}(s, a) \doteq \sum_{s', r} p(s', r|s, a) [r + \gamma v_{\pi}(s')]$$

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。) □

解 由题意知: $\pi(a|s) = \frac{1}{4}, p(s', r|s, a) = 1, r = 0, \gamma = 0.9$, 那么:

$$\begin{aligned} v_{\pi}(\text{center}) &\doteq \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) [r + \gamma v_{\pi}(s')] \\ &= \frac{1}{4} \times 0.9 \times [2.3 + 0.4 - 0.4 + 0.7] \\ &= \frac{1}{4} \times 0.9 \times 3 \\ &= 0.675 \\ &\approx 0.7 \end{aligned}$$

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? □

解 G_t 的定义如下:

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

添加常数 c 后:

$$\begin{aligned} G_t^* &\doteq (c + R_{t+1}) + \gamma(c + R_{t+2}) + \gamma^2(c + R_{t+3}) + \cdots \\ &= G_t + \sum_{k=0}^{\infty} \gamma^k c \\ &= G_t + \frac{c}{1-\gamma} \end{aligned}$$

$$v_{\pi}^*(s) = \mathbb{E}_{\pi}[G_t^*|S_t = s] = \mathbb{E}_{\pi}\left[G_t + \frac{c}{1-\gamma} \middle| S_t = s\right] = \mathbb{E}_{\pi}[G_t|S_t = s] + \frac{c}{1-\gamma}$$

所以添加常数后不会影响状态的相对值, 另外, v_c 为:

$$v_c = \frac{c}{1-\gamma}$$

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。 □

解 回合任务中 G_t 为:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$$

加入常数 c 后:

$$\begin{aligned} G_t^* &= (c + R_{t+1}) + \gamma(c + R_{t+2}) + \gamma^2(c + R_{t+3}) + \cdots + \gamma^{T-t-1}(c + R_T) \\ &= G_t + \sum_{k=0}^{T-t-1} \gamma^k c \\ &= G_t + c \frac{1 - \gamma^{T-t}}{1 - \gamma} \end{aligned}$$

由上述等式可以看出, 回合任务中加入常数会产生影响。它会增加 v 的值。

例如假设我们有一个回合任务, 其只有一个状态 S 和两个动作 A_0, A_1 。采取 A_0 动作会导致从 S 状态进入终止状态, 此时得到的奖励为 +1。采取 A_1 动作会回到 S 状态, 奖励为零。在这种情况下 agent 应采取动作 A_0 进入终止状态以最大化奖励。

如果我们给每个奖励加 +1, 那么一直采取动作 A_1 获得的回报为 $\frac{1}{1-\gamma}$, 当选择的折扣因子小于 $\frac{1}{2}$, 那么获得的回报将大于 2。在这种情况下 agent 的最优策略是一直采取动作 A_1 。

练习 3.17 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。展示类似于 (3.14) 的方程序列, 但针对动作值。□

解

$$\begin{aligned} G_t &\doteq \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\ &= \sum_{s', r} p(s', r | s, a) \left[r + \gamma \sum_{a'} \pi(a' | s') q_\pi(s', a') \right] \end{aligned}$$

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。□

解

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[q_\pi(s, a)] \\ &= \sum_a \pi(a|s) q_\pi(s, a) \end{aligned}$$

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。 \square

解

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

3.6 最优策略和最优值函数

练习 3.20 画出或描述 golf 示例中的最优状态值函数。 \square

解 最佳状态值函数在绿地外时根据发球杆给出值，然后在绿地时根据推杆给出值。

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。 \square

解 应与始终使用推杆的策略的值相同，即图 (3.3) 中的 v_{putt} 图。

练习 3.22 考虑右图所示的连续 MDP。唯一要做出的决定是顶部状态，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？ \square

解 设顶部状态为 A，采取 left 和 right 动作后分别到达状态 B 和 C。由题意知： $R_{A, \text{left}, B} = +1, R_{B, \text{left}, A} = 0, R_{A, \text{right}, C} = 0, R_{C, \text{right}, A} = +2$ ，那么顶部状态的回报为：

$$\begin{aligned} G_{\pi_{\text{left}}} &= R_{A, \text{left}, B} + \gamma R_{B, \text{left}, A} + \gamma^2 R_{A, \text{left}, B} + \gamma^3 R_{B, \text{left}, A} + \gamma^4 R_{A, \text{left}, B} + \cdots \\ &= R_{A, \text{left}, B} + \gamma^2 R_{A, \text{left}, B} + \gamma^4 R_{A, \text{left}, B} + \cdots \\ &= \sum_{k=0}^{\infty} \gamma^{2k} \\ &= \frac{1}{1 - \gamma^2} \end{aligned}$$

$$\begin{aligned} G_{\pi_{\text{right}}} &= R_{A, \text{right}, C} + \gamma R_{C, \text{right}, A} + \gamma^2 R_{A, \text{right}, C} + \gamma^3 R_{C, \text{right}, A} + \gamma^4 R_{A, \text{right}, C} + \cdots \\ &= \gamma R_{C, \text{right}, A} + \gamma^3 R_{C, \text{right}, A} + \gamma^5 R_{C, \text{right}, A} + \cdots \\ &= \sum_{k=0}^{\infty} 2\gamma^{1+2k} \\ &= \frac{2\gamma}{1 - \gamma^2} \end{aligned}$$

根据上述每种策略的回报公式， $\gamma = 0.5$ 是临界点。如果 $\gamma > 0.5$ ，则 right 为最优动作。如果 $\gamma < 0.5$ ，则 left 为最优动作。如果 $\gamma = 0.5$ ，则两者均为最优动作。

练习 3.23 给出 recycling robot 中 q_* 的 Bellman 方程。 \square

解 q_* 的 Bellman 方程为：

$$\begin{aligned}
q_* &= \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') | S_t = s, A_t = a] \\
&= \sum_{s', r} p(s', r | s, a) [r + \gamma \max_{a'} q_*(s', a')]
\end{aligned} \tag{3.1}$$

为了简洁，将两个状态 high、low 和三个动作 search、wait、recharge 分别表示为 h, l, s, w, re ，那么 Bellman 方程为：


$$\begin{aligned}
q_*(h, s) &= p(h|h, s)[r_s + \gamma \max(q_*(h, s), q_*(h, w))] + \\
&\quad p(l|h, s)[r_s + \gamma \max(q_*(l, s), q_*(l, w), q_*(l, re))] \\
&= \alpha[r_s + \gamma \max(q_*(h, s), q_*(h, w))] + \\
&\quad (1 - \alpha)[r_s + \gamma \max(q_*(l, s), q_*(l, w), q_*(l, re))]
\end{aligned}$$

$$\begin{aligned}
q_*(l, s) &= p(h|l, s)[-3 + \gamma \max(q_*(h, s), q_*(h, w))] + \\
&\quad p(l|l, s)[r_s + \gamma \max(q_*(l, s), q_*(l, w), q_*(l, re))] \\
&= (1 - \beta)[-3 + \gamma \max(q_*(h, s), q_*(h, w))] + \\
&\quad \beta[r_s + \gamma \max(q_*(l, s), q_*(l, w), q_*(l, re))]
\end{aligned}$$

$$\begin{aligned}
q_*(h, w) &= p(h|h, w)[r_w + \gamma \max(q_*(h, s), q_*(h, w))] \\
&= r_w + \gamma \max(q_*(h, s), q_*(h, w))
\end{aligned}$$

$$\begin{aligned}
q_*(l, w) &= p(l|l, w)[r_w + \gamma \max(q_*(l, s), q_*(l, w), q_*(l, re))] \\
&= r_w + \gamma \max(q_*(l, s), q_*(l, w), q_*(l, re))
\end{aligned}$$

$$\begin{aligned}
q_*(l, re) &= p(h|l, re)[0 + \gamma \max(q_*(h, s), q_*(h, w))] \\
&= \gamma \max(q_*(h, s), q_*(h, w))
\end{aligned}$$

 **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。
解 v_* 的 Bellman 方程为：

$$v_*(s) = \max_a \mathbb{E}_{\pi_*}[G_t | S_t = s, A_t = a]$$

根据图 3.5 中的 v_* 和 π_* ，到达 A 后的最佳解决方案是移至 A 后快速返回 A。这需要 5 个时间步。所以我们将有

$$\begin{aligned}
G_t^* &= 10 + \gamma \times 0 + \gamma^2 \times 0 + \gamma^3 \times 0 + \gamma^4 \times 0 + \gamma^5 \times 10 + \gamma^6 \times 0 + \dots \\
&= 10 + \gamma^5 \times 10 + \dots \\
&= \sum_{k=0}^{\infty} 10\gamma^{5k} \\
&= \frac{10}{1 - \gamma^5}
\end{aligned} \tag{3.2}$$

$v_*(A) = G_t^*$, 状态 A 的理论值是 $10/(1 - \gamma^5)$ 。通过用 python 写一个小函数（循环 100 次就足够了）或使用计算器，我们得到的答案是 24.419428096993954。保留三位小数为 24.419。

 **练习 3.25** 给出用 q_* 表示的 v_* 方程。

□

解

$$v_*(s) = \max_a (q_*(s, a))$$

 **练习 3.26** 给出用 v_* 和四参数 p 表示的 q_* 方程。

□

解

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

 **练习 3.27** 给出用 q_* 表示的 π_* 方程。

□

解


$$\pi_*(s) = \arg \max_a q_*(s, a)$$

 **练习 3.28** 给出用 v_* 和四参数 p 表示的 π_* 方程。

□

解

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

 **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。

□

解

$$\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t | S_t = s] \\
&= \sum_a \pi(a | s) \sum_{s'} p(s' | s, a) [r(s, a) + \gamma v_\pi(s')]
\end{aligned}$$

$$\begin{aligned}
v_*(s) &= \max_a \mathbb{E}_{\pi_*}[G_t | S_t = s, A_t = a] \\
&= \max_a \sum_{s'} p(s' | s, a) [r(s, a) + \gamma v_*(s')]
\end{aligned}$$

$$\begin{aligned}
q_\pi(s, a) &= \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \\
&= \sum_{s'} p(s' | s, a) [r(s, a) + \gamma \sum_{a'} \pi(a' | s') q_\pi(s', a')]
\end{aligned}$$

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

3.7 最优和近似

3.8 总结



第四章 动态规划

4.1 策略评估

🔴 **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? □

🟢 **解** 已知用 v_π 表示 q_π 为:

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

根据题意, 已知 $p(s', r|s, a) = 1, \gamma = 1$, 根据公式可得:

$$q_\pi(11, \text{down}) = -1 + v_\pi(T) = -1 + 0 = -1$$

$$q_\pi(7, \text{down}) = -1 + v_\pi(11) = -1 - 14 = -15$$

🔴 **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? □

🟢 **解** 已知 v_π 的 Bellman 方程为:

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


由题意知 $\pi(a|s) = 1/4 = 0.25, p(s', r|s, a) = 1, \gamma = 1$, 那么 $v_\pi(15)$ 的值为:

$$\begin{aligned} v_\pi(15) &= 0.25 \times [-1 + v_\pi(12)] + 0.25 \times [-1 + v_\pi(13)] + 0.25 \times [-1 + v_\pi(14)] \\ &\quad + 0.25 \times [-1 + v_\pi(15)] \\ &= -1 + 0.25 \times [v_\pi(12) + v_\pi(13) + v_\pi(14)] + 0.25 \times v_\pi(15) \\ &= -1 + 0.25 \times [-22 - 20 - 14] + 0.25 \times v_\pi(15) \\ &= -15 + 0.25 \times v_\pi(15) \end{aligned}$$

解得: $v_\pi(15) = -15/0.75 = -20$

改变动态不会导致整个游戏的重新计算: 状态 15 与状态 13 完全相同。因此, 它们

的状态值都相同且仍为-20。


 **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？ □

解


$$\begin{aligned}
 q_\pi(s, a) &\doteq \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \\
 &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\
 &= \mathbb{E}_\pi[R_{t+1} + \gamma \sum_{a'} q_\pi(S_{t+1}, a') | S_t = s, A_t = a] \\
 &= \sum_{s', r} p(s', r | s, a) \left[r + \gamma \sum_{a'} \pi(a' | s') q_\pi(s', a') \right] \\
 q_{k+1}(s, a) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma \sum_{a'} q_\pi(S_{t+1}, a') | S_t = s, A_t = a] \\
 &= \sum_{s', r} p(s', r | s, a) \left[r + \gamma \sum_{a'} \pi(a' | s') q_k(s', a') \right]
 \end{aligned}$$

4.2 策略改进

4.3 策略迭代

 **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。 □

解 将第 3 步策略改进中如果 $old-action \neq \pi(s)$ 改为，如果 $old-action \notin \{a_i\}$ ，其为所有 $\pi(s)$ 下的等价最优解。

 **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。 □

解

Policy Iteration for estimating $q \approx q_*$

1. 初始化

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

2. 策略评估

循环：

$\Delta \leftarrow 0$

对于每个 $s \in \mathcal{S}$ 和 $a \in \mathcal{A}$ 循环：

$q = Q(s, a)$

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

$$\Delta \leftarrow \max(\Delta, |q - Q(s, a)|)$$

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 $s \in \mathcal{S}$ 和 $a \in \mathcal{A}$


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

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

如果 $old-action \notin \{a_i\}$, 其为 $\pi(s)$ 下的等价最优解集合,

那么 $policy-stable \leftarrow false$


如果 $policy-stable$, 那么停止并返回 $Q \approx q_*$ 和 $\pi \approx \pi_*$; 否则转到 2

 **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。 □

解 第 3 步改变: 我们只会在不探索策略的情况下, 判断 $policy-stable$ 为 $false$ 。


第 2 步改变: θ 的设置不应超过任何 ε -soft 的限制。

第 1 步改变: π 应该很好地定义为 ε -soft 方法。应该给予 ϵ 。

 **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。 □

解 见 `exercise-programming/exercise4.7.py`。

4.4 值迭代


 **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? □

解 赌徒的问题具有最优策略的奇特形式, 因为资产为 50, 您可能以 0.4 的概率突然获胜。因此, 当资产为 50 时, 最好的策略将其全押。

可以将 51 的资本当作 50 加 1。当然, 当我们拥有 51 时, 我们可以全部下注, 但是最好的策略是看看我们是否可以从这额外的 1 美元中赚取更多。如果此带来的回报为正数, 我们可以说我们会多余的钱, 然后再次下注直至 75, 直到突然获胜的机会来临。这意

味着，我们有从 75 中获胜的额外机会。相反，如果我们在 51 的资产中首先下注 50，我们的获胜机会仅为 0.4，而且我们会失去达到 75 的机会。相反，如果我们输了赌注，我们必须尽力以 1 美元达到 25，这是一个更糟糕的情况。

结论：最优策略表明了更多的获胜机会，并保证了赌徒在输时会更好。

 **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？ □

解 当 $ph < 0.5$ 时，结果是稳定的。

见 exercise-programming/exercise4.9.py。

 **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？ □

解

$$\begin{aligned} q_{k+1}(s, a) &\doteq \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_k(s', a') | S_t = s, A_t = a] \\ &= \sum_{s', r} p(s', r | s, a) [r + \gamma \max_{a'} q_k(s', a')] \end{aligned}$$

4.5 异步动态规划

4.6 广义策略迭代

4.7 动态规划效率

4.8 总结

第五章 蒙特卡罗方法

5.1 蒙特卡罗预测

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？ □

解 这是由于这种策略，玩家直到满足 20 或 21 时才会停止。这表明玩家将面临命中失败的风险，这将导致在 20 和 21 点之前的低值部分。然而，在 20 和 21 点时，玩家会停下来，并且有很高的获胜机会，尤其是当发牌者将停在 17 点或更高的时候。

左图最后一行下降，是因为如果庄家显示王牌，当它计数为 11 时，它比玩家有很高的概率得到更高的分数。因此，发牌者的 A 值包含了庄家使其可用或不可用的赢率。其他的纸牌没有这样的条件，因此 A 很特殊并产生了差距。

在上图中，最前面的值较高，因为 A 表示在上图中用作 1 和 11 的对偶值。它使玩家更好，并且与最左边的行下降的情况类似。

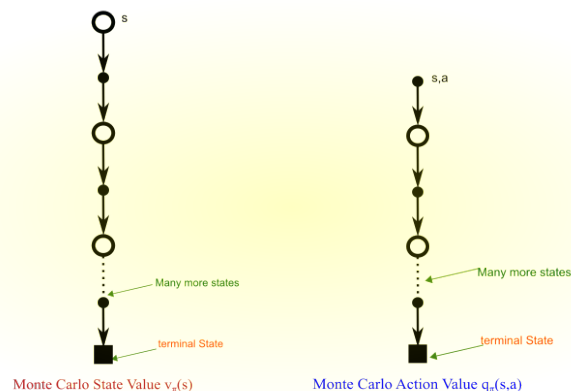
练习 5.2 假设在 21 点任务中使用了每次访问 MC 而不是首次访问 MC。您期望结果会有很大不同吗？为什么或者为什么不？ □

解 不会。在任何回合中，21 点都不包含两个重复的状态，因此首次访问和每次访问方法本质上是相同的。


5.2 动作值的蒙特卡罗估计

练习 5.3 蒙特卡罗估计 q_π 的备份图是什么？ □

解



5.3 蒙特卡洛控制


 **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。 □

解 蒙特卡洛 ES 的伪代码应该以以下方式更新：

$$\begin{aligned}
 Q_n(S_t, A_t) &= \frac{1}{n} \sum_{i=1}^n G_i(S_t, A_t) \\
 &= \frac{1}{n} \left(G_n(S_t, A_t) + \sum_{i=1}^{n-1} G_i(S_t, A_t) \right) \\
 &= \frac{1}{n} \left(G_n(S_t, A_t) + (n-1) \frac{1}{n-1} \sum_{i=1}^{n-1} G_i(S_t, A_t) \right) \\
 &= \frac{1}{n} (G_n(S_t, A_t) + (n-1)Q_{n-1}(S_t, A_t)) \\
 &= \frac{1}{n} (G_n(S_t, A_t) + nQ_{n-1}(S_t, A_t) - Q_{n-1}(S_t, A_t)) \\
 &= Q_{n-1}(S_t, A_t) + \frac{1}{n} (G_n(S_t, A_t) - Q_{n-1}(S_t, A_t))
 \end{aligned}$$

5.4 无探索起点的蒙特卡洛控制

5.5 重要性采样的 off-policy 预测


 **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？ □

解 对于首次访问估计，仅考虑状态的首次访问：

$$V(s) = G(S_0) = 10$$

对于每次访问估计，每个状态都被考虑：

$$\begin{aligned}
 V(s) &= \frac{1}{10} [G(S_0) + G(S_1) + \cdots + G(S_{10})] \\
 &= \frac{1}{10} (10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1) = 5.5
 \end{aligned}$$

 **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？ □

解

$$Q(s, a) \doteq \frac{\sum_{t \in \mathcal{T}(s, a)} \rho_{t+1:T(t)-1} G_t}{\sum_{t \in \mathcal{T}(s, a)} \rho_{t+1:T(t)-1}}$$

🔴 **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？ □

解 加权平均算法将需要几个回合来减少其偏差。特别是当 $\rho_{t:T(t)-1}$ 很大时，通过这些数据加权平均算法将发生偏移。当获得足够多的回合时，平均值开始稳定并减少偏差。

🔴 **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？ □

解 对于每次访问：

$$\begin{aligned} & \left[\left(\frac{1}{T-1} \sum_{k=1}^{T-1} \prod_{t=0}^k \frac{\pi(A_t, S_t)}{b(A_t, S_t)} G_t \right)^2 \right] \\ &= 0.5 \cdot \left(\frac{1}{0.5} \right)^2 \quad (\text{回合长度为 1}) \\ &+ \frac{1}{2} \left[0.5 \cdot 0.9 \cdot 0.5 \cdot 0.1 \cdot \left(\frac{1}{0.5} \frac{1}{0.5} \right)^2 + 0.5 \cdot 0.1 \left(\frac{1}{0.5} \right)^2 \right] \quad (\text{回合长度为 2}) \\ &+ \frac{1}{3} \left[0.5 \cdot 0.9 \cdot 0.5 \cdot 0.9 \cdot 0.5 \cdot 0.1 \left(\frac{1}{0.5} \frac{1}{0.5} \frac{1}{0.5} \right)^2 + 0.5 \cdot 0.9 \cdot 0.5 \cdot 0.1 \cdot \left(\frac{1}{0.5} \frac{1}{0.5} \right)^2 \right. \\ &\quad \left. + 0.5 \cdot 0.1 \left(\frac{1}{0.5} \right)^2 \right] \quad (\text{回合长度为 3}) \\ &+ \dots \\ &= 0.1 \sum_{k=1}^{\infty} \frac{1}{k} \sum_{l=0}^{k-1} 0.9^l \cdot 2^l \cdot 2 \\ &= 0.2 \sum_{k=1}^{\infty} \frac{1}{k} \sum_{l=0}^{k-1} 1.8^l = \infty \end{aligned}$$

5.6 渐增实现

🔴 **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。 □

解


$$\begin{aligned} V_n(S_t) &= \frac{1}{n} \sum_{i=1}^n G_i(t) \\ &= V_{n-1}(S_t) + \frac{1}{n} (G_n(t) - V_{n-1}(S_t)) \end{aligned}$$

🔴 **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。 □


解

$$\begin{aligned}
V_{n+1} &\doteq \frac{\sum_{k=1}^n W_k G_k}{\sum_{k=1}^n W_k} \\
&= \frac{W_n G_n + \sum_{k=1}^{n-1} W_k G_k}{\sum_{k=1}^{n-1} W_k G_k} \frac{\sum_{k=1}^{n-1} W_k G_k}{\sum_{k=1}^n W_k G_k} \\
&= \left[\frac{W_n G_n}{C_{n-1}} + V_n \right] \frac{C_{n-1}}{C_n} \\
&= \frac{W_n G_n}{C_n} + \frac{V_n C_{n-1}}{C_n} \\
&= V_n + \frac{W_n G_n}{C_n} + \frac{V_n C_{n-1}}{C_n} - V_n \\
&= V_n + \frac{W_n G_n}{C_n} + \frac{V_n C_{n-1} - V_n C_n}{C_n} \\
&= V_n + \frac{W_n G_n}{C_n} - \frac{V_n W_n}{C_n} \\
&= V_n + \frac{W_n}{C_n} [G_n - V_n]
\end{aligned}$$

5.7 Off-policy 蒙特卡洛控制

 **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？ \square


解 如果 $A_t = \pi(S_t)$ ，则仅允许 A_t 改变 W 。由于 π 是确定性的，因此可以肯定地说在更新 W 期间 $\pi(A_t|S_t) = 1$ 。

 **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 \times 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，以从每个起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）。 \square

解 见 [exercise-programming/exercise5.12.py](#)。

5.8 * 折扣的重要性采样


5.9 * 每决策的重要性采样

 **练习 5.13** 显示从 (5.12) 推导 (5.14) 的步骤。

解 t 之后, 任何重要性采样率都与 R_{t+1} 无关。这必须遵循:

$$\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y],$$

使用 5.13 代入 5.12, 确实会有 5.14。


 **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。

解 content...

5.10 总结

第六章 时间差分学习

6.1 TD 预测


 **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。 □

解 δ 定义: $\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$

TD 更新: $V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$

由此得:


$$\begin{aligned} G_t - V_t(S_t) &= R_{t+1} + \gamma G_{t+1} - V_t(S_t) + \gamma V_t(S_{t+1}) - \gamma V_t(S_{t+1}) \\ &= \delta_t + \gamma G_{t+1} - \gamma V_t(S_{t+1}) \\ &= \delta_t + \gamma G_{t+1} - \gamma V_t(S_{t+1}) + \gamma V_{t+1}(S_{t+1}) - \gamma V_{t+1}(S_{t+1}) \\ &= \delta_t + \gamma[G_{t+1} - V_{t+1}(S_{t+1})] + \gamma[V_{t+1}(S_{t+1}) - V_t(S_{t+1})] \\ &= \delta_t + \gamma[G_{t+1} - V_{t+1}(S_{t+1})] + \gamma\alpha[R_{t+2} + \gamma V_t(S_{t+2}) - V_t(S_{t+1})] \\ &= \delta_t + \gamma[\delta_{t+1} + \gamma[G_{t+2} - V_{t+2}(S_{t+2})] + \gamma\alpha[R_{t+3} + \gamma V_{t+1}(S_{t+3}) \\ &\quad - V_{t+1}(S_{t+2})]] + \gamma\alpha[R_{t+2} + \gamma V_t(S_{t+2}) - V_t(S_{t+1})] \\ &= \delta_t + \gamma\delta_{t+1} + \gamma^2[G_{t+2} - V_{t+2}(S_{t+2})] + \alpha\gamma^2[R_{t+3} + \gamma V_{t+1}(S_{t+3}) - V_{t+1}(S_{t+2})] \\ &\quad + \alpha\gamma[R_{t+2} + \gamma V_t(S_{t+2}) - V_t(S_{t+1})] \\ &\quad \vdots \\ &= \sum_{k=t}^{T-1} \gamma^{k-t} \delta_k + \alpha \sum_{k=t}^{T-2} \gamma^{k-t+1} [R_{k+2} + \gamma V_k(S_{k+2}) - V_k(S_{k+1})] \end{aligned}$$

 **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? □

解 所提到的方案对于 TD 非常有效, 因为高速公路部分的状态没有更改, 我们仍然可以对其状态值抱有坚定的信念。利用 TD 将加快我们对新状态的调整。然后, 每个更新的状态都会加速之前的状态。另一方面, 蒙特卡洛必须走到尽头, 并带来所有状态的平均改善, 尤其是那些高速路段, 这些路段根本不需要太多调整, 但仍会因蒙特卡洛方法而波动。

TD 更有效的另一种情况是，我们很难有终止状态。即使在我们不考虑状态值衰减过小的方法下，原始的蒙特卡洛方法仍然不够有效，甚至不实用。

6.2 TD 预测方法的优势

 **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？ □


解 它告诉我们第一个回合在左端终止状态结束。但是，我们没有关于中间转移如何的任何信息。因为这些奖励是 0 且 γ 是 1，并且非终止状态都初始化为 0.5，终止状态初始化为 0。因此，TD(0) 更新对不能直接导致终止的状态没有任何作用。即，对于 A、E 外的非终止状态：

$$\begin{aligned} V_1(S_t) &= V_0(S_t) + \alpha[0 + \gamma V_0(S_{t+1}) - V_0(S_t)] \\ &= V_0(S_t) \end{aligned}$$


对于状态 A：

$$\begin{aligned} V_1(A) &= V_0(A) + \alpha[0 + \gamma \cdot 0 - V_0(A)] \\ &= (1 - \alpha)V_0(A) \\ &= 0.9 \times 0.5 \\ &= 0.45 \end{aligned}$$

A 的估计值减少为 $\alpha V_0(A) = 0.05$ 。

 **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？ □

解 不会。对于 TD 和蒙特卡洛，足够小的 α 是收敛要求。因此，从长远来看，可以说较小的 α 对于 TD 和蒙特卡洛总是比较大的 α 更好，并且达到了它可以达到的最低误差。结果所示已触及每种方法性能的极限。因此，我们可以得出结论，没有任何固定 α 可以改善的魔术。另一方面，人们可能会希望更改不同状态的 α 或更改权重。

 **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？ □

解 状态 C 恰好已经初始化为其真实值。随着训练的开始，外部状态会发生更新（使各个状态更加准确），从而减少所有状态的误差。直到外部状态的误差传播到 C 为止，这种情况将一直发生。较高的 α 值使该效应更加明显，因为 C 的值估计在这些情况中更容易改变。

 **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? \square

解 一种方法是使用计算能力 (例如 DP) 来精确计算结果。由于我们没有策略, 因此只会导致长期而精确的状态评估。

由于 $V(s) = \mathbb{E}[\mathbb{I}\{\text{从 } s \text{ 到右端终止}\}] = P(\text{从 } s \text{ 到右端终止})$, 根据对称性意味着 $V(C) = 0.5$, 那么

$$\begin{aligned} V(E) &= \frac{1}{2} \times 1 + \frac{1}{2} \times V(D) \\ &= \frac{1}{2} + \frac{1}{4} [V(C) + V(E)] \end{aligned}$$

可得 $V(E) = \frac{5}{6}$, 进而可得 $V(D) = \frac{4}{6}$ 。我们可以用相同的方式计算其他状态的值。

然而, 另一种方法在于数学。考虑状态 E, 如果从 E 开始, 进入左端 $P_E(L)$ 的概率是多少, 进入右端 $P_E(R)$ 的概率是多少?

$$\begin{aligned} P_E(R) &= 1 - P_E(L) \\ &= 1 - P_E(D) * P_D(L) \\ &= 1 - P_E(D) * [P_D(C)P_C(L) + P_D(E) * P_E(L)] \end{aligned}$$

由于对称性, 我们直到 $P_C(L) = 0.5$, 列等式可解出。

6.3 TD(0) 的最优性

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)。 \square

解 令 G_t 为 b 产生的回合的回报。那么

$$\begin{aligned} v_{\pi}(s) &= \mathbb{E}[\rho_{t:T-1}G_t | S_t = s] \\ &= \mathbb{E}[\rho_{t:T-1}R_{t+1} + \gamma\rho_{t:T-1}G_{t+1} | S_t = s] \\ &= \rho_{t:t}\mathbb{E}[R_{t+1} | S_t = s] + \gamma\rho_{t:t}\mathbb{E}[\rho_{t+1:T-1}G_{t+1} | S_t = s] \\ &= \rho_{t:t}(\mathbb{E}[R_{t+1} | S_t = s] + \gamma\mathbb{E}[\rho_{t+1:T-1}G_{t+1} | S_t = s]) \\ &= \rho_{t:t}(r(s, A_t) + \gamma v_{\pi}(S_{t+1})) \end{aligned}$$

因此, off-policy 版本的 TD(0) 的更新 (通过采样近似) 为:

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

6.4 Sarsa: on-policy TD 控制

- 🔴 **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。 □

解

$$\begin{aligned}
 G_t - Q(S_t, A_t) &= R_{t+1} + \gamma G_{t+1} - Q(S_t, A_t) + \gamma Q(S_{t+1}, A_{t+1}) - \gamma Q(S_{t+1}, A_{t+1}) \\
 &= \delta_t + \gamma[G_{t+1} - Q(S_{t+1}, A_{t+1})] \\
 &= \delta_t + \gamma\delta_{t+1} + \gamma^2[G_{t+2} - Q(S_{t+2}, A_{t+2})] \\
 &= \sum_{k=t}^{T-1} \gamma^{k-t} \delta_k
 \end{aligned}$$

- 🔴 **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？ □

解 最优轨迹现在是 7 步，而不是 15 步。在此示例中，包括空动作没有帮助，因为风垂直吹，并且目标位置未与起始位置垂直分离。但是，它在其他风环境中可能很有用。

见 exercise-programming/exercise6.9.py。

- 🔴 **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。 □

解 见 exercise-programming/exercise6.10.py。

6.5 Q-learning: off-policy TD 控制

- 🔴 **练习 6.11** 为什么 Q-learning 被视为 off-policy 控制方法？ □

解 Q-learning 被视为 off-policy 的原因是它使用下一个状态 s 和贪婪动作 a 的 Q 值更新其当前 Q 值。换句话说，它通过假设遵循贪婪策略来估计状态-动作对的回报（未来奖励折扣总额），尽管实际上它遵循的不是贪婪策略。Sarsa 是 on-policy 的原因是它使用下一个状态 s 和当前策略的动作 a 的 Q 值更新其当前 Q 值。它通过假设继续遵循当前策略来估算状态-动作对的回报。


- 🔴 **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的动作选择和权重更新吗？ □

解 一般来说，它们似乎是相同的方法。但是偶尔情况会有所不同。在某些步骤中考虑 $S = S'$ 。在更新期间，Sarsa 将使用从 S' 获得的先前的贪婪动作 A' ，并前进到 $Q(S', A')$ 。

但是 Q-learning 将使用更新的 Q 重新选择贪婪 A^* 。由于 A' 可能与使 $Q(S', A^*)$ 最大化的新贪婪动作 A^* 的动作不同，因此这两种方法是不同的。

6.6 期望 Sarsa

6.7 最大化偏差与 Double Learning

 **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?

□

解

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q_2(S_{t+1}, a) - Q_1(S_t, A_t) \right]$$


$$\pi(a|s) = \begin{cases} 1 - \varepsilon, & \text{if } a = \arg \max_{a'} (Q_1(s, a') + Q_2(s, a')) \\ \frac{\varepsilon}{|\mathcal{A}(s)|}, & \text{otherwise} \end{cases} \quad (6.1)$$

6.8 游戏、后期状态和其他特殊情况

6.9 总结


第七章 n 步自举

7.1 n 步 TD 预测


 **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

解 已知 $\delta_t \doteq R_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t)$ ，则：

$$\begin{aligned}
 G_{t:t+n} - V_{t+n-1}(S_t) &= R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \\
 &\quad \gamma^n V_{t+n-1}(S_{t+n}) - V_{t+n-1}(S_t) \\
 &= R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \\
 &\quad \gamma^n V(S_{t+n}) - V(S_t) \\
 &\quad \text{(because we assume } V \text{ will not change)} \\
 &= \delta_t - \gamma V(S_{t+1}) + V(S_t) + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \\
 &\quad \gamma^n V(S_{t+n}) - V(S_t) \\
 &= \delta_t - \gamma V(S_{t+1}) + V(S_t) + \gamma [\delta_{t+1} - \gamma V(S_{t+2}) + V(S_{t+1})] \\
 &\quad + \cdots + \gamma^{n-1} [\delta_{t+n-1} - \gamma V(S_{t+n}) + V(S_{t+n-1})] \\
 &\quad + \gamma^n V(S_{t+n}) - V(S_t) \\
 &= \sum_{k=t}^{t+n-1} \left[\gamma^{k-t} \delta_k - \gamma^{k-t+1} V(S_{k+1}) + \gamma^{k-t} V(S_k) \right] \\
 &\quad + \gamma^n V(S_{t+n}) - V(S_t) \\
 &= \sum_{k=t}^{t+n-1} \gamma^{k-t} \delta_k - \sum_{k=t}^{t+n-2} \gamma^{k-t+1} V(S_{k+1}) + \sum_{k=t+1}^{t+n-1} \gamma^{k-t} V(S_k) \\
 &= \sum_{k=t}^{t+n-1} \gamma^{k-t} \delta_k - \sum_{k=t+1}^{t+n-1} \gamma^{k-t} V(S_k) + \sum_{k=t+1}^{t+n-1} \gamma^{k-t} V(S_k) \\
 &= \sum_{k=t}^{t+n-1} \gamma^{k-t} \delta_k
 \end{aligned}$$

 **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。 \square

解 见 `exercise-programming/exercise7.2.py`。

 **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? \square

解 因为使用的是 n 步回报, 所以我们希望避免随机行走过早完成。

较小的行走会把优势转移到较小的 n , 因为当 $n \geq \frac{|\mathcal{S}|-1}{2}$ ($|\mathcal{S}|$ 是状态数且为奇数) 时, 算法会使用终端奖励更新访问的所有状态。这意味着该算法仅更改步长 α 值的大小, 因为值不再被自举或备份。

左边的 -1 奖励偏爱 n 的较小值, 因为在较长的回合中, n 的较大值将不得不通过最终奖励来更新许多状态, 现在从 0 变为 -1, 从而增加了方差。

7.2 n 步 Sarsa

 **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

$$G_{t:t+n} = Q_{t-1}(S_t, A_t) + \sum_{k=t}^{\min(t+n, T)-1} \gamma^{k-t} [R_{k+1} + \gamma Q_k(S_{k+1}, A_{k+1}) - Q_{k-1}(S_k, A_k)]. \quad (7.1)$$

\square

解 对于 $n \geq 1, 0 \leq t < T - n$, 有:

$$\begin{aligned} G_{t:t+n} &\doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}) \\ &= R_{t+1} + \gamma Q_t(S_{t+1}, A_{t+1}) - \cancel{\gamma Q_t(S_{t+1}, A_{t+1})} - Q_{t-1}(S_t, A_t) + \cancel{Q_{t-1}(S_t, A_t)} \\ &\quad + \gamma R_{t+2} + \gamma^2 Q_{t+1}(S_{t+2}, A_{t+2}) - \cancel{\gamma^2 Q_{t+1}(S_{t+2}, A_{t+2})} - \gamma Q_t(S_{t+1}, A_{t+1}) \\ &\quad + \cancel{\gamma Q_t(S_{t+1}, A_{t+1})} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}) \\ &\quad - \cancel{\gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n})} - \gamma^{n-1} Q_{t+n-2}(S_{t+n-1}, A_{t+n-1}) \\ &\quad + \cancel{\gamma^{n-1} Q_{t+n-2}(S_{t+n-1}, A_{t+n-1})} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}) \\ &= Q_{t-1}(S_t, A_t) + \sum_{k=t}^{\min(t+n, T)-1} \gamma^{k-t} [R_{k+1} + \gamma Q_k(S_{k+1}, A_{k+1}) - Q_{k-1}(S_k, A_k)] \end{aligned}$$

抵消总是发生在每个 R_k 的第二个 Q 项和下一个 R_{k+1} 的第四个 Q 项之间。作为第一个 R 和最后一个 R 的特殊情况: 划线部分必须拿出来, 是因为 R_{t-1} 不存在。最后一项 $\gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n})$ 与 R_{t+n} 的第二个 Q 项抵消。最终结果完成了证明。

7.3 n 步 off-policy 学习

7.4 * 控制变量的每决策方法

 **练习 7.5** 编写上述 off-policy 状态值预测算法的伪代码。 \square

解

```

if  $\tau \geq 0$ :
    For  $\tau \dots \min\{\tau + n, T - 1\}$ , indexed as  $j$ :
         $\rho_j \leftarrow \frac{\pi(A_j, S_j)}{b(A_j, S_j)}$ 
         $\rho \leftarrow \rho \cdot \rho_j$ 
         $G \leftarrow G + \rho R_{j+1} + (1 - \rho_j) \frac{\rho}{\rho_j} \gamma V(S_j)$ 
    If  $\tau + n < T$ :  $G \leftarrow G + \rho \gamma^n V(S_{\tau+n})$ 
     $V(S_\tau, A_\tau) \leftarrow V(S_\tau, A_\tau) + \alpha(G - V(S_\tau, A_\tau))$ 

```

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。 □

解

$$\begin{aligned}
 \mathbb{E}[G_{t:h}] &= \mathbb{E}[R_{t+1} + \gamma \rho_{t+1}(G_{t+1:h} - Q_{h-1}(S_{t+1}, A_{t+1})) + \gamma \bar{V}_{h-1}(S_{t+1})] \\
 &= \mathbb{E}[R_{t+1}] + \mathbb{E}[\gamma \rho_{t+1}(G_{t+1:h} - Q_{h-1}(S_{t+1}, A_{t+1}))] + \mathbb{E}[\gamma \bar{V}_{h-1}(S_{t+1})] \\
 &\quad (\text{因为 } \rho \text{ 与任何估计无关, } \mathbb{E}[\rho] = 1) \\
 &= R_{t+1} + \mathbb{E}[\gamma G_{t+1:h} - \gamma Q_{h-1}(S_{t+1}, A_{t+1})] + \gamma \bar{V}_{h-1}(S_{t+1}) \\
 &= R_{t+1} + \mathbb{E}[G_{t:h} - R_{t+1} - \gamma Q_{h-1}(S_{t+1}, A_{t+1})] + \gamma \bar{V}_{h-1}(S_{t+1}) \\
 &= \mathbb{E}[G_{t:h}] + \gamma [\mathbb{E}[-Q_{h-1}(S_{t+1}, A_{t+1})] + \bar{V}_{h-1}(S_{t+1})] \\
 &\quad (\text{由 7.8 的定义}) \\
 &= \mathbb{E}[G_{t:h}]
 \end{aligned}$$

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。 □

解 将水平线处的终止视为: $G_{h:h} \doteq \bar{V}_{h-1}(S_h)$, 并将非终止条件视为 $G_{h:h} \doteq Q_{h-1}(S_h, A_h)$ 。

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)。 □

解 已知: $\delta_t \doteq R_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t)$, 则:

$$\begin{aligned}
 G_{t:h} &\doteq \rho_t(R_{t+1} + \gamma G_{t+1:h}) + (1 - \rho_t)V(S_t) \\
 &= \rho_t(R_{t+1} + \gamma[\rho_{t+1}(R_{t+2} + \gamma G_{t+2:h}) + (1 - \rho_{t+1})V(S_{t+1})]) + (1 - \rho_t)V(S_t) \\
 &= \rho_t(R_{t+1} + \gamma V(S_{t+1}) - V(S_t) + \gamma[\rho_{t+1}(R_{t+2} + \gamma G_{t+2:h}) - \rho_{t+1}V(S_{t+1})]) \\
 &\quad + V(S_t) \\
 &= \rho_t(\delta_t + \gamma[\rho_{t+1}(R_{t+2} + \gamma G_{t+2:h}) - \rho_{t+1}V(S_{t+1})]) + V(S_t) \\
 &= \rho_t(\delta_t + \gamma[\rho_{t+1}(R_{t+2} + \gamma[\rho_{t+2}(\delta_{t+2} + \gamma[\rho_{t+3}(R_{t+4} + \gamma G_{t+4:h}) \\
 &\quad - \rho_{t+3}V(S_{t+3})]) + V(S_{t+2})]) - \rho_{t+1}V(S_{t+1})]) + V(S_t) \\
 &= \rho_t(\delta_t + \gamma[\rho_{t+1}(R_{t+2} + \gamma V(S_{t+2}) - V(S_{t+1}) + \gamma[\rho_{t+2}(\delta_{t+2} \\
 &\quad + \gamma[\rho_{t+3}(R_{t+4} + \gamma G_{t+4:h}) - \rho_{t+3}V(S_{t+3})])])]) + V(S_t) \\
 &= \rho_t(\delta_t + \gamma[\rho_{t+1}(\delta_{t+1} + \gamma[\rho_{t+2}(\delta_{t+2} + \gamma[\rho_{t+3}(R_{t+4} + \gamma G_{t+4:h}) - \rho_{t+3}V(S_{t+3})])])])
 \end{aligned}$$

$$\begin{aligned}
& +V(S_t) \\
& = \sum_{k=t}^{\min(T-1, h)} \left(\prod_{m=t}^k \rho_m \gamma^{k-t} \right) \delta_k + V(S_t)
\end{aligned}$$

🔴 **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。 □

🟢 **解** 已知期望 Sarsa 的 TD 误差为 $\delta_t = R_{t+1} + \gamma \sum_a \pi(a|S_{t+1})Q(S_{t+1}, a) - Q(S_t, A_t)$, 则:

$$\begin{aligned}
G_{t:h} &= R_{t+1} + \gamma \rho_{t+1}(G_{t+1:h} - Q(S_{t+1}, A_{t+1})) + \gamma \bar{V}_{h-1}(S_{t+1}) \\
&= \delta_t - \gamma \sum_a \pi(a|S_{t+1})Q(S_{t+1}, a) + Q(S_t, A_t) + \gamma \rho_{t+1}(G_{t+1:h} - Q(S_{t+1}, A_{t+1})) \\
&\quad + \gamma \bar{V}_{h-1}(S_{t+1}) \\
&= \delta_t - \gamma \sum_a \pi(a|S_{t+1})Q(S_{t+1}, a) + Q(S_t, A_t) + \gamma \rho_{t+1}(-Q(S_{t+1}, A_{t+1})) \\
&\quad + \gamma \bar{V}_{h-1}(S_{t+1}) + \gamma \rho_{t+1}G_{t+1:h} \\
&= \delta_t - \gamma \sum_a \pi(a|S_{t+1})Q(S_{t+1}, a) + Q(S_t, A_t) + \gamma \rho_{t+1}(-Q(S_{t+1}, A_{t+1})) \\
&\quad + \gamma \bar{V}_{h-1}(S_{t+1}) + \gamma \rho_{t+1}[R_{t+2} + \gamma \rho_{t+2}(G_{t+2:h} - Q(S_{t+2}, A_{t+2})) + \gamma \bar{V}_{h-1}(S_{t+2})] \\
&= \delta_t - \gamma \sum_a \pi(a|S_{t+1})Q(S_{t+1}, a) + Q(S_t, A_t) + \gamma \rho_{t+1}(-Q(S_{t+1}, A_{t+1})) \\
&\quad + \gamma \bar{V}_{h-1}(S_{t+1}) + \gamma \rho_{t+1}[\delta_{t+1} - \gamma \sum_a \pi(a|S_{t+2})Q(S_{t+2}, a) + Q(S_{t+1}, A_{t+1}) \\
&\quad + \gamma \rho_{t+2}(G_{t+2:h} - Q(S_{t+2}, A_{t+2})) + \gamma \bar{V}_{h-1}(S_{t+2})] \\
&= \sum_{k=t}^{h-1} \left[\gamma^{k-t} \left[\prod_{i=t+1}^k \rho_i \right] \left[\delta_k - \gamma \sum_a \pi(a|S_{k+1})Q(S_{k+1}, a) + Q(S_k, A_k) \right. \right. \\
&\quad \left. \left. - \gamma \rho_{k+1}Q(S_{k+1}, A_{k+1}) + \gamma \bar{V}_{h-1}(S_{k+1}) \right] \right] \\
&= \sum_{k=t}^{h-1} \left[\gamma^{k-t} \left[\prod_{i=t+1}^k \rho_i \right] [\delta_k] \right] + Q(S_t, A_t)
\end{aligned}$$

🔴 **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。 □

🟢 **解** 见 exercise-programming/exercise7.10.py。

7.5 无重要性采样的 off-policy 学习：n 步树备份算法

🔴 **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。 □

解

$$\begin{aligned}
G_{t:t+n} &\doteq R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1})Q(S_{t+1}, a) + \gamma \pi(A_{t+1}|S_{t+1})G_{t+1:t+n} \\
&= R_{t+1} + \gamma [\bar{V}_t(S_{t+1}) - \pi(A_{t+1}|S_{t+1})Q(S_{t+1}, A_{t+1})] + \gamma \pi(A_{t+1}|S_{t+1})G_{t+1:t+n} \\
&= \sum_{k=t}^{\min(T-1, t+n-1)} \left[[R_{k+1} + \gamma(\bar{V}_k(S_{k+1}) - \pi(A_{k+1}|S_{k+1})Q(S_{k+1}, A_{k+1}))] \prod_{i=t+1}^k \gamma \pi(A_i, S_i) \right] \\
&= \sum_{k=t}^{\min(T-1, t+n-1)} \left[[\delta_k + Q(S_k, A_k) - \gamma \pi(A_{k+1}|S_{k+1})Q(S_{k+1}, A_{k+1})] \prod_{i=t+1}^k \gamma \pi(A_i, S_i) \right] \\
&= \sum_{k=t}^{\min(T-1, t+n-1)} \left[\delta_k \prod_{i=t+1}^k \gamma \pi(A_i, S_i) \right] + Q(S_t, A_t)
\end{aligned}$$

7.6 * 一种统一算法: $Q(\sigma)$

7.7 总结

第八章 表格法进行规划和学习

8.1 模型和规划

8.2 Dyna: 综合规划，行动和学习

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 7 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。

解 由于初始动作值 Q 为零，对于 n 步自举来说，只有距离目标 n 步内的状态才可以更新为非零值，所以这种更新方式会导致其他状态的延迟更新，因此需要时间来传递更新的信息。

即使 n 足够大并且忽略计算和内存的成本，Dyna-Q 仍然更好。因为 Dyna-Q 的规划部分在计算上非常有效。它访问像数据结构这样的哈希表，并且仅执行两个乘法。除了其高效率外，循环本身不像 n 步方法那样局限于目标的 n 步之内。Dyna-Q 随机选择经历过的 S 和 A 加以利用。尽管需要改善这种随机性，但是 Dyna-Q 成功地使任何一对 (S_t, A_t) 都可以最佳地学习其 Q 值（如果该对确实是最优）。由于它具有一个 $\max_a Q(S', a)$ 项，并且随机选择，因此，如果在规划期间更改了任何 Q 值，则如果规划选择了它们，它将被转移到相同状态 S 内的任何相邻的 Q 。通过这种方法，Dyna-Q 通过随机选择减轻了 n 步方法的时间不规则性，通过哈希表结构降低了计算成本，并且乘法运算更少。这就是 Dyna-Q 的巨大优势。当然，它会有一些探索问题，这将在 Dyna-Q+ 中解决。

8.3 当模型错误时

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第二阶段都表现得更好？ □

解 在第一个阶段，两个示例中，Dyna-Q+ 都比 Dyna-Q 更快地找到了最优策略。这是因为 Dyna-Q+ 善于探索以找到最优策略。在第二阶段，这两种算法都可以在阻塞示例中找到新的最优策略，因为两者都可以处理环境恶化的情况。同样，我们可以看到 Dyna-Q+ 更快。在捷径示例中，Dyna-Q 无法找到新的最优策略，因为原始的 ϵ -greedy 方法很难在许多步骤中保持探索。Dyna-Q+ 强制进一步探索从而造成了巨大差距。

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？ □

解 这是因为 Dyna-Q+ 的更快收敛和探索也将付出一定的代价所造成的。这样的代价将迫使 Dyna-Q+ 进行比 Dyna-Q 更大的探索，即使它已经是最优的。这样的代价将逐渐消耗领先的差距，这就是为什么差异略小的原因。如果游戏继续进行而没有捷径，那么 Dyna-Q+

的累积奖励将低于 Dyna-Q，但最优策略应保持不变。

- 🔥 **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。 □

解 见 exercise-programming/exercise8.4.py。

- 🔥 **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？ □

解 在随机情况下，模型应维护在导致 S' 的任何给定对 (S, A) 中获得的每个奖励的可能性表，即 $P(S', r|S, A)$ 。然后在规划期间对奖励进行抽样。但是，如果环境在变化，则任务将更加艰巨。

第一个想法将削弱旧奖励出现的可能性。第二个是使用旧数据时衰减 α 。第三是使用数量有限的历史数据，旧奖励的数据将被删除。除此之外，我们应该始终保持恒定的 α ，因为我们不能期望策略能够收敛。

解决该问题的一种尝试将是跟随。首先，我们仅接受每对 (S, A) 有限数量的最新奖励数据和常数 α ，以确保我们可以从最新中学习。然后，维护并跟踪样本均值和奖励方差的变化。如果我们从统计方法中观察到环境是稳定的，那么我们将衰减 α ，增加数据大小的限制并加速收敛。另一方面，如果无法检测到稳定，则将始终保持最新的数据容量限制和常数 α 。

8.4 优先扫描

8.5 期望与采样更新

- 🔥 **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。 □


解 它将加强样本更新。考虑一个极端的情况，其中 $b = 1020$ ，但是在 99.99% 的时间内只有 10,000 个状态出现。期望更新将永远发生来计算真实值。但是，样本更新将达到几乎为 0 的 RMS 误差，就像在均匀情况下一样。样本更新的性能更好，因为从本质上讲，它可以忽略非常不可能的状态并强调重要而频繁的状态，而不会忽略任何一个在给定的计算上足够数量的样本状态。

8.6 轨迹采样

- 🔥 **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？ □

解 它呈扇形，是因为进行一次完整扫描需要花费大量的计算时间。每次更新都会带来适当的变化，尤其是在 $b = 1$ 的情况下，其中一个状态的变化已经完全转移到其他状态。当

b 较大时, 由于后续状态的部分变化的多个总和, 系统会更稳定, 但也将需要更长的时间来更新并使曲线的扇形部分更少。这就是为什么在 10,000 个状态的情况下扇形非常致密且可见, 而在 1,000 个状态的情况下, $b = 10$ 情况是最光滑的, 而 $b = 1$ 情况具有最大的扇形。实际上, 如果在计算时间轴 (x 轴) 上使用相同的比例, 则下半部分图形的扇形将更宽且更粗糙。

 **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。 □

解 见 `exercise-programming/exercise8.8.py`。

8.7 实时动态规划

8.8 决策时规划

8.9 启发式搜索

8.10 展开算法

8.11 蒙特卡洛树搜索

8.12 本章总结

8.13 第一部分总结：维度

第二部分

近似解决方法

第九章 基于近似的 on-policy 预测

