

Reinforcement Learning

Jiabin Liu

2026 年 9 月 13 日

目录

1	Introduction	1
1.1	About this note	1
1.2	Other study materials	2
2	Basic Concepts	3
2.1	Markov Decision Process (MDP)	3
3	Bellman Equation	5
3.1	State Value	5
3.2	Derivation of the Bellman Equation	6
4	Policy Gradient	8
4.1	Algorithm	8
4.2	Metrics to define optimal policies	8
4.3	Gradients of the metrics	9
4.4	REINFORCE	10

1 Introduction

1.1 About this note

本笔记主要参考赵世钰老师的《强化学习的数学原理》课程¹，课程视频可以在 [Bilibili](#) 和 [YouTube](#) 上观看，其所有课件可以在 [GitHub 仓库](#) 中找到。

整个笔记按照图 1 所示进行组织，首先给出强化学习中的基本概念和公式，然后按顺序介绍强化学习的各种算法思想，对其进行数学分析，并给出相关代码实现。

总体分为四个部分：

① 基础理论

Basic Concepts → Bellman Equation → Bellman Optimality Equation

② 表格型方法

Dynamic Programming → Monte Carlo → Temporal-Difference

③ 函数逼近

Stochastic Approximation → Value Function Approximation

④ 策略优化

Policy Gradient → Actor-Critic

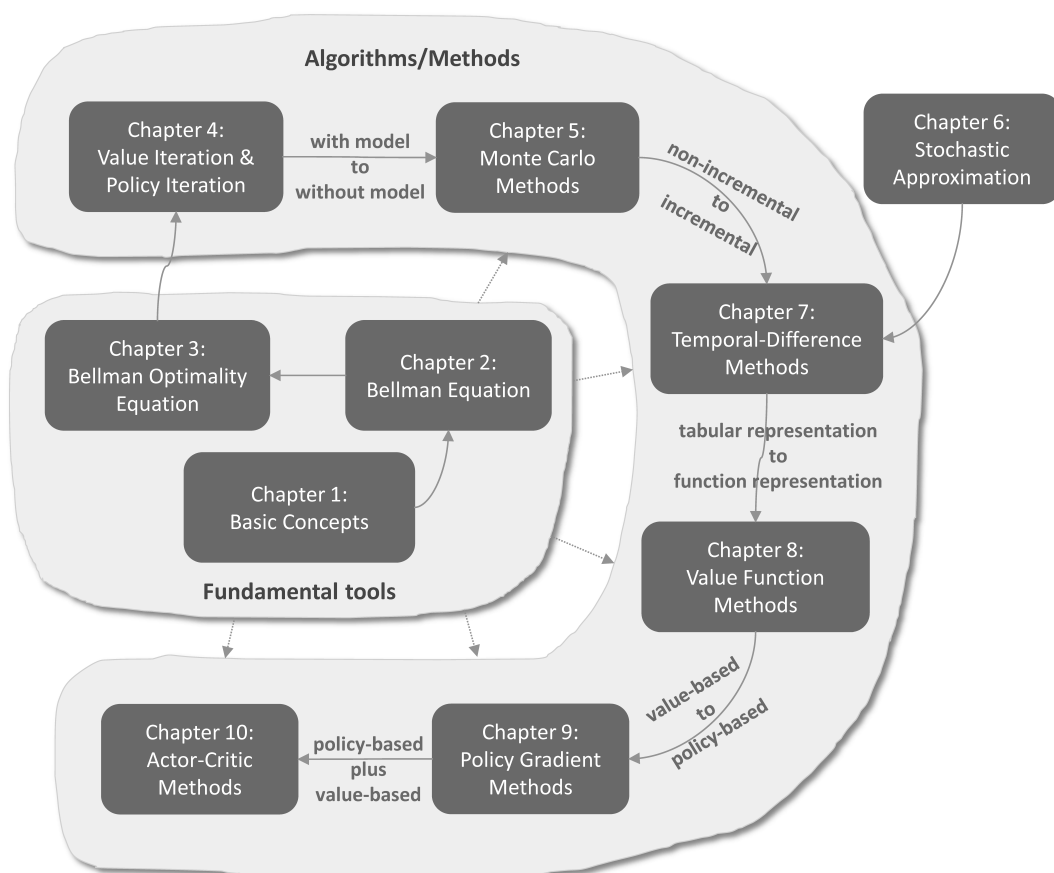


图 1 Roadmap of Reinforcement Learning (Source: [BookMap.png](#))

¹ Zhao, S. *Mathematical Foundations of Reinforcement Learning*. Springer Press, 2025

1.2 Other study materials

除了赵世钰老师的课程之外，还有一些我认为比较好的学习资源：

书籍

- [Sutton & Barto, *Reinforcement Learning: An Introduction*](#)
强化学习领域的“圣经”，内容全面，适合系统学习。

GitHub 项目

- [WalkingLabs, *Hands-On Modern RL*](#)
从经典强化学习到现代 LLM 与 Agentic RL 的开源教程，
强调理论推导、代码实现与实验相结合。
- [CleanRL, *Clean Implementation of RL Algorithms*](#)
高质量的深度强化学习单文件实现，代码简洁且易于阅读，
适合学习 DQN、PPO、SAC 等算法的完整工程实现。

经典课程

- 李宏毅 — National Taiwan University
 - [概述增强式学习（1-5 节）](#)
深入浅出讲解强化学习的基本概念，适合初学者建立直观认识。
 - [深度强化学习, 2018](#)
深度强化学习课程，涵盖 PPO、DQN、Actor-Critic 等基础算法，
适合掌握基础算法后进一步学习。
- [David Silver, *Reinforcement Learning*](#)
强化学习领域的奠基级公开课，基于 Sutton 的经典教材。

进阶课程

- [CS285, *Deep Reinforcement Learning*](#) — Sergey Levine, UC Berkeley
深度强化学习进阶课程，涵盖 Imitation Learning、Model-based RL、Offline RL 等内容，
建议掌握 DQN、Policy Gradient、Actor-Critic 等基础算法后学习。

2 Basic Concepts

2.1 Markov Decision Process (MDP)

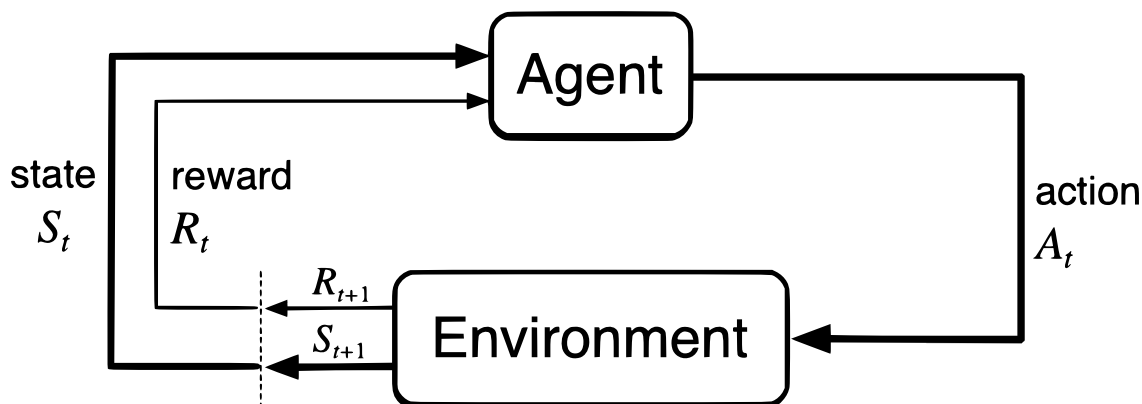


图 2 The agent–environment interaction in a Markov decision process

(Source: Sutton & Barto)

马尔可夫决策过程 (MDP) 是强化学习问题的数学框架。强化学习的基本交互过程如图 2 所示：智能体根据当前状态 s_t 选择动作 a_t ，环境据此反馈奖励 r_{t+1} 并转移到新的状态 s_{t+1} ，如此循环往复，从而生成一段轨迹 (trajectory)，如 $\{s_0, a_0, r_1, s_1, a_1, r_2, \dots\}$ 。整个交互过程构成了一个 MDP。

定义 2.1.1 (Markov Decision Process)

马尔可夫决策过程可以表示为五元组

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma) \quad (1)$$

其中：

- $\mathcal{S}$ ：状态空间；
- $\mathcal{A}$ ：动作空间；
- $P(s' | s, a)$ ：状态转移概率；
- $R(s, a)$ ：奖励函数；
- $\gamma \in [0, 1)$ ：折扣因子。

MDP 由三部分组成：

• M — Markov Property (马尔可夫性质)

马尔可夫性质又称“无记忆性”：未来只与当前状态 s_t 和动作 a_t 有关，与历史无关，即

$$p(s_{t+1} | a_t, s_t, \dots, a_0, s_0) = p(s_{t+1} | a_t, s_t) \quad (2)$$

$$p(r_{t+1} | a_t, s_t, \dots, a_0, s_0) = p(r_{t+1} | a_t, s_t) \quad (3)$$

也就是说，下一时刻的状态和奖励只取决于最近的状态 s_t 与动作 a_t ，更早的历史数据可以丢弃——状态 s_t 已包含决策所需的全部信息。这正是五元组中状态转移概率写作 $P(s' | s, a)$ 的原因。

• D — Policy (策略)

策略是智能体的行为准则：在状态 s 下选择动作 a 的概率为 $\pi(a | s)$ ，满足归一化条件 $\sum_a \pi(a | s) = 1$ 。若某一动作的概率为 1，则退化为确定性策略 (deterministic policy)。

• **P — Sets & Probability Distributions (状态集合与概率分布)**

状态集合 (Sets)

- **State 状态**: $s \in \mathcal{S}$ ，对环境的描述，是决策的依据。
- **Action 动作**: $a \in \mathcal{A}$ ，智能体可以采取的行为。
- **Reward 奖励**: $r \in \mathcal{R}$ ，环境给出的标量反馈，衡量动作的好坏。

概率分布 (Probability Distributions)

- **状态转移概率**: $p(s' | s, a)$ ，在状态 s 下采取动作 a 后，转移到新状态 s' 的概率。
- **奖励概率**: $p(r | s, a)$ ，在状态 s 下采取动作 a 后，获得奖励 r 的概率。

综上，一个完整的 MDP 可记为五元组 $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ ，其中 P 表示状态转移概率， R 表示奖励函数。

3 Bellman Equation

3.1 State Value

考虑如下多步轨迹：

$$S_t \xrightarrow{A_t} (R_{t+1}, S_{t+1}) \xrightarrow{A_{t+1}} (R_{t+2}, S_{t+2}) \xrightarrow{A_{t+2}} (R_{t+3}, \dots) \quad (4)$$

首先需要提出一个概念：**Discounted Return**，即从当前时刻 t 开始，未来所有奖励的折扣和：

定义 3.1.1 (Discounted Return)

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \quad (5)$$

- $\gamma \in [0, 1]$ 称为**折扣因子 (Discount Factor)**。
- G_t 也是一个随机变量，因为 $R_{t+1}, R_{t+2}, \dots$ 都是随机变量。
- 随着奖励距离当前时刻越来越远，其权重会按照 $1, \gamma, \gamma^2, \dots$ 逐渐衰减。

沿着一个 trajectory，其中每个状态 S_t 都对应一个折扣回报 G_t ，如图 3 所示。

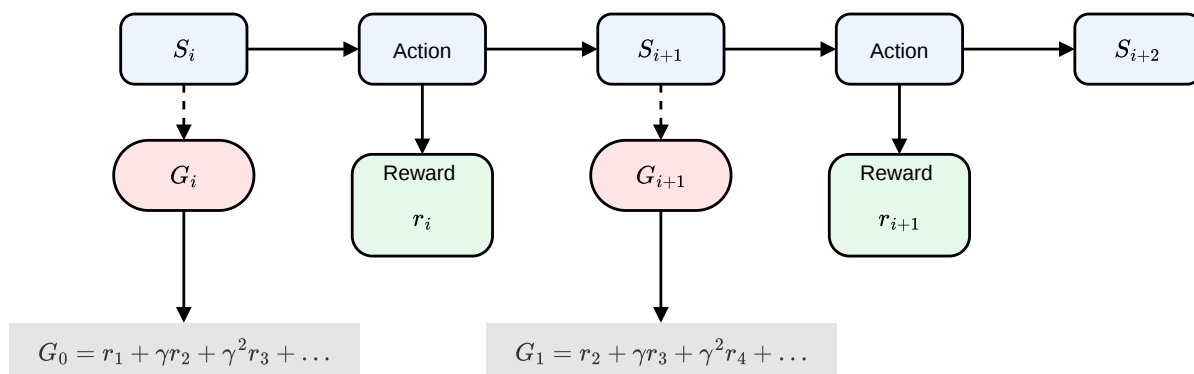


图 3 Discounted Return G_t in a trajectory

定义 3.1.2 (State Value)

State Value 定义为在状态 s 下，按照策略 π 进行决策所得到的折扣回报 G_t 的期望值，即

$$v_{\pi(s)} = \mathbb{E}[G_t \mid S_t = s] \quad (6)$$

- 它是关于状态 s 的函数。它是一个**条件期望 (Conditional Expectation)**，条件是系统从状态 s 出发。
- 它依赖于**策略 (Policy)**。对于不同的策略，状态价值 (State Value) 可能不同。
- 它表示一个状态的“价值” (Value)。状态价值越大，意味着从该状态出发能够获得的**累积奖励 (Cumulative Rewards)** 越大，因此对应的策略通常越好。

3.1.1 Discounted Return vs. State Value

- **Discounted Return**: 在状态 s 下，按照单个确定轨迹得到的 G_t 值。
- **State Value**: 在状态 s 下，如果存在多个可能的轨迹，则为这些轨迹对应 Discounted Return 的期望值。

3.2 Derivation of the Bellman Equation

3.2.1 Derivation

首先把折扣回报 G_t 拆成当前奖励与折扣的后续回报两部分：

$$G_t = R_{t+1} + \gamma G_{t+1} \quad (7)$$

代入 State Value 的定义，利用期望的线性性质：

$$\begin{aligned} v_{\pi(s)} &= \mathbb{E}[G_t \mid S_t = s] \\ &= \mathbb{E}[R_{t+1} \mid S_t = s] + \gamma \mathbb{E}[G_{t+1} \mid S_t = s] \end{aligned} \quad (8)$$

下面分别计算两项。

• 第一项：当前奖励的期望

从状态 s 出发，按策略 π 选择动作，当前奖励的期望为

$$\mathbb{E}[R_{t+1} \mid S_t = s] = \sum_a \pi(a \mid s) r(s, a) \quad (9)$$

其中 $r(s, a) = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$ 是采取动作 a 后的平均奖励（若奖励是确定的，则直接取奖励值）。

• 第二项：未来回报的期望

由马尔可夫性质， G_{t+1} 只与下一时刻的状态 S_{t+1} 有关，而与 S_t 无关，因此

$$\begin{aligned} \mathbb{E}[G_{t+1} \mid S_t = s] &= \sum_a \pi(a \mid s) \sum_{s'} p(s' \mid s, a) \mathbb{E}[G_{t+1} \mid S_{t+1} = s'] \\ &= \sum_a \pi(a \mid s) \sum_{s'} p(s' \mid s, a) v_{\pi(s')} \end{aligned} \quad (10)$$

其中最后一步用到了 State Value 的定义 $v_{\pi(s')} = \mathbb{E}[G_{t+1} \mid S_{t+1} = s']$ 。

3.2.2 Bellman Equation

将两项代入，合并即得：

定理 3.2.1 (Bellman 方程)

对任意状态 s ，状态价值 $v_{\pi(s)}$ 满足

$$v_{\pi(s)} = \sum_a \pi(a \mid s) \left[r(s, a) + \gamma \sum_{s'} p(s' \mid s, a) v_{\pi(s')} \right] \quad (11)$$

它是一个自洽方程 (self-consistent equation)：未知量 v_{π} 同时出现在等式两侧，状态 s 的价值被表示为后续状态价值的加权组合。这正是动态规划“由后往前递推”思想的体现。

3.2.3 Matrix-Vector Form (矩阵-向量形式)

将所有状态的 Bellman 方程写在一起，即得如下推论：

推论 3.2.1.1 (矩阵形式 (Matrix-Vector Form))

Bellman 方程可以写为紧凑的矩阵形式

$$v_{\pi} = r_{\pi} + \gamma P_{\pi} v_{\pi} \quad (12)$$

其中:

- $v_{\pi} = [v_{\pi(s_1)}, \dots, v_{\pi(s_n)}]^T$ 为状态价值向量;
- $r_{\pi} = [r_{\pi(s_1)}, \dots, r_{\pi(s_n)}]^T$ 为一步奖励期望向量, $r_{\pi(s)} = \sum_a \pi(a | s) r(s, a)$;
- P_{π} 为状态转移矩阵, $[P_{\pi}]_{i,j} = \sum_a \pi(a | s_i) p(s_j | s_i, a)$ 。



当状态数有限时, 可以解析求解:

$$v_{\pi} = (\mathbb{I} - \gamma P_{\pi})^{-1} r_{\pi} \quad (13)$$

当状态数很多时矩阵求逆不可行, 需要用迭代法求解, 这将在后续章节介绍。

4 Policy Gradient

强化学习的目标无非就是在给定环境中得到一个最好的策略（对应最高的价值）。之前章节的核心算法是值函数方法（Value-based Method）：

1. 先进行 Policy Evaluation，即学习状态价值函数 $v_\pi(s)$ 或动作价值函数 $q_\pi(s, a)$ 。
2. 然后再进行 Policy Improvement，比如使用贪心策略或梯度上升法来更新策略参数。
3. 不断重复上述两个步骤，直到策略收敛。

学习价值函数网络（如 DQN）可以处理离散空间、高维图像状态的强化学习问题，但对于连续动作空间的强化学习问题，值函数方法就显得力不从心了。因为在连续动作空间中，动作的数量是无限的，无法像离散动作空间那样直接对每个动作进行价值评估， $\max_{a'} Q(s', a'; \theta^-)$ 无法计算。尽管可以通过对连续空间进行离散化或改变网络结构来“解决”这个问题，但实际效果并不理想。

那么，是否可以跳过值函数的计算，直接优化策略 $\pi(a|s)$ 呢？答案是肯定的，这就是策略梯度方法（Policy Gradient Method）的核心思想。

4.1 Algorithm

什么是优化策略？在强化学习中，策略是一个映射函数，它将状态映射到动作的概率分布。优化策略的目标是找到一个最优策略，使得在给定环境下，智能体能够最大化其累积奖励。

- 当策略 π 采用表格形式表示时，如果它能够最大化**每一个状态价值**，则该策略是最优策略。
- 当策略 π 采用函数形式表示时，如果它能够最大化某个**标量评价指标**（scalar metric），则该策略是最优策略。

我们使用第二种函数形式 $\pi_\theta(a|s)$ 来表示策略，其中 θ 是策略的参数。这样有更好的泛化能力，能够处理连续状态空间和连续动作空间的问题。接下来就可以使用经典的机器学习思想来优化策略参数 θ ，从而得到最优策略。

算法思路：

1. 定义一个标量评价指标 $J(\theta)$ ，它是策略 π_θ 的函数。
2. 计算该指标的梯度 $\nabla_\theta J(\theta)$ 。
3. 通过梯度上升法来优化策略参数 θ ：

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\theta_t) \quad (14)$$

4.2 Metrics to define optimal policies

评价指标 $J(\theta)$ 可以是状态价值函数的期望值，也可以是动作价值函数的期望值，或者其他与策略性能相关的指标。论文中常用的评价指标有两种：

- average state value:

$$\begin{aligned} J(\theta) &= \mathbb{E}_{s \sim d_\pi} [v_\pi(s)] \\ &= \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \right] \end{aligned} \quad (15)$$

表示在策略 π 下，智能体访问每个状态的概率分布为 d_π ，而 $v_\pi(s)$ 是状态 s 的价值函数。该指标衡量了在策略 π 下，智能体在所有可能状态中获得的平均价值。

- average one-step reward:

$$\begin{aligned}
J(\theta) &= \mathbb{E}_{s \sim d_\pi} [r_\pi(s)] \\
&= \lim_{n \rightarrow \infty} \frac{1}{n} \mathbb{E} \left[\sum_{k=1}^n R_{t+k} \right]
\end{aligned} \tag{16}$$

表示在策略 π 下，智能体访问每个状态的概率分布为 d_π ，而

$$r_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a | s) r(s, a) = \sum_{a \in \mathcal{A}} \sum_r \pi(a | s) \cdot r \cdot p(r | s, a) \tag{17}$$

是状态 s 下，智能体在采取动作 a 后获得的即时奖励的期望值。该指标衡量了在策略 π 下，智能体在所有可能状态中获得的平均即时奖励。

直觉上式 15 和式 16 关注的方面不同：式 15 关注的是长期累积奖励，而式 16 关注的是即时奖励。但是二者在数学上是**等价的**。因为长期累积奖励可以通过即时奖励的期望值来计算，而即时奖励的期望值也可以通过长期累积奖励来推导。因此，在实际应用中，可以根据具体问题的需求选择合适的评价指标。

4.3 Gradients of the metrics

指标 $J(\theta)$ 对 θ 的依赖来自两个层次：策略 $\pi_\theta(a | s)$ 直接依赖于 θ ，而状态分布 d_π 由策略**隐式**决定——策略一变，状态的访问分布也随之改变。后者是平稳方程 $d_\pi = d_\pi P_\pi$ 的解，对 θ 的依赖难以显式表达，直接对 $J(\theta)$ 求导会遇到“如何求 $\nabla_\theta d_\pi$ ”的困难。

能否避开 $\nabla_\theta d_\pi$ ，只对策略本身求导？答案是可以，关键是下面的对数导数技巧（log-derivative trick）。由链式法则：

$$\nabla_\theta \ln \pi_\theta(a | s) = \frac{\nabla_\theta \pi_\theta(a | s)}{\pi_\theta(a | s)} \tag{18}$$

即 $\nabla_\theta \pi_\theta(a | s) = \pi_\theta(a | s) \nabla_\theta \ln \pi_\theta(a | s)$ 。这个等式看似平凡，却是整个策略梯度方法的基石：

- 它要求概率 $\pi_\theta(a | s) > 0$ 恒成立。离散动作空间常用 softmax 输出层：

$$\pi_\theta(a | s) = \frac{\exp(h(s, a, \theta))}{\sum_{a'} \exp(h(s, a', \theta))} \tag{19}$$

其中 $h(s, a, \theta)$ 为网络给出的 logit；连续动作空间可用高斯策略（均值、方差由网络输出）。两者都保证概率恒正。

- 由概率归一化 $\sum_a \pi_\theta(a | s) = 1$ 对 θ 求导可得

$$\sum_a \nabla_\theta \pi_\theta(a | s) = 0 \tag{20}$$

即“把概率质量从动作 a 移走多少，就必须给其他动作补回多少”——推导中常用它消去多余项。

- 它把“对概率的梯度”改写为“概率与对数概率梯度之积”，从而可以把 $\pi_\theta \nabla_\theta \ln \pi_\theta$ 放进对动作的期望中，得到便于用采样近似的形式。

在此基础上，策略梯度定理给出了 $J(\theta)$ 梯度的简洁表达式：

定理 4.3.1 (Policy Gradient Theorem (策略梯度定理))

对平均状态价值指标 式 15 与平均一步奖励指标 式 16, 当 $\pi_\theta(a | s)$ 对 θ 可微时, 梯度 $\nabla_\theta J(\theta)$ 可统一写成

$$\nabla_\theta J(\theta) = \sum_s d_\pi(s) \sum_a \pi_\theta(a | s) \nabla_\theta \ln \pi_\theta(a | s) q_\pi(s, a) \quad (21)$$

或等价的期望形式:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim d_\pi, a \sim \pi_\theta} [\nabla_\theta \ln \pi_\theta(a | s) q_\pi(s, a)] \quad (22)$$

其中 $q_\pi(s, a)$ 为与所选指标对应的动作价值函数。不同指标下, 梯度表达式的差别只在于 q_π 的取值形式与一个与 θ 无关的常数因子 (折扣情形为 $\frac{1}{1-\gamma}$), 不影响梯度上升的方向。

这个定理的关键意义在于, 等号右侧不再出现 $\nabla_\theta d_\pi$ ——状态分布对 θ 的隐式依赖在推导中被消去了。推导思想是: 令 θ 产生一个小扰动, 把指标的一阶变化展开, 再借助平稳方程 $d_\pi = d_\pi P_\pi$ 与概率归一化恒等式, 把状态分布的变化全部吸收进动作价值 $q_\pi(s, a)$ (细节从略)。

直观理解: 对每个状态-动作对 (s, a) , $\nabla_\theta \ln \pi_\theta(a | s)$ 给出参数空间中增大 $\pi_\theta(a | s)$ 的方向, 梯度上升按动作价值 $q_\pi(s, a)$ 给该方向加权——价值高的动作被抬高, 价值低的动作被压低; 外层的权重 $d_\pi(s)$ 则让常访问的状态优先被优化。整体效果就是把概率质量从“差动作”移向“好动作”。

4.4 REINFORCE

策略梯度定理把梯度写成了期望形式, 但仍然无法直接计算, 因为其中有两个未知量: 状态分布 d_π (环境模型未知时不可得) 与动作价值 q_π (这正是值函数方法要学习的对象)。机器学习的标准对策是**随机梯度**: 不求期望, 而是采样一个样本、用样本梯度代替期望梯度。这一步要解决两个问题:

1. 状态与动作从哪里采? 沿轨迹真实交互即可——状态按 d_π 出现、动作按 π_θ 采样, 样本的分布恰好是期望里需要的 $d_\pi \times \pi_\theta$ 。
2. $q_\pi(s_t, a_t)$ 怎么估计? 从 (s_t, a_t) 出发实际走一段, 用得到的回报来估计。

用样本 (s_t, a_t) 与其价值估计 $q_t(s_t, a_t)$ 代入, 得到随机梯度上升的更新律:

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta \ln \pi_\theta(a_t | s_t) q_t(s_t, a_t) \quad (23)$$

其中 α 为学习率。 q_t 的不同估计方式, 导出不同的算法:

- **蒙特卡洛 (MC)**: 等一个 episode 结束, 用从 (s_t, a_t) 出发的真实折扣回报

$$G_t = \sum_{k=1}^{T-t} \gamma^{k-1} R_{t+k} \quad (24)$$

作为 $q_t(s_t, a_t)$ 。它是 $q_\pi(s_t, a_t)$ 的无偏估计, 由此得到 **REINFORCE** 算法。

- **时序差分 (TD)**: 用自举的方式估计 q_t , 引出 Actor-Critic 类方法, 将在后续章节介绍。

由于样本期望恰好等于真实梯度, 随机梯度上升“平均而言”就是梯度上升。REINFORCE 的具体流程:

1. 初始化策略参数 θ 。
2. 用当前策略 π_θ 从初始状态采样一条完整轨迹 $s_0, a_0, r_1, s_1, a_1, r_2, \dots, s_{T-1}, a_{T-1}, r_T$ 。
3. 对 $t = 0, 1, \dots, T-1$: 计算回报 G_t , 按上式更新参数。

4. 回到第 2 步，用更新后的策略重新采样，重复直至收敛。

REINFORCE 的几个要点：

- **On-policy、整条轨迹更新**：样本必须服从当前的 π_θ ，轨迹随策略更新而作废，因此每一轮都要用新策略重新采样；同时 MC 回报要等整条轨迹结束才能得到，无法像 TD 那样边走边学。
- **高方差，可加基线**： G_t 累积了整条轨迹的随机性，方差很大，需要较小的学习率。常用降方差手段是减去基线（baseline） $b(s_t)$ ：

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta \ln \pi_\theta(a_t | s_t) (G_t - b(s_t)) \quad (25)$$

只要 $b(s_t)$ 与动作无关，减基线就不改变更新的期望：由归一化恒等式 $\sum_a \nabla_\theta \pi_\theta(a | s) = 0$ 可知 $\mathbb{E}_{a \sim \pi_\theta} [\nabla_\theta \ln \pi_\theta(a | s) b(s)] = 0$ ；但合适的 $b(s)$ ，如状态价值 $v_\pi(s)$ ，能显著降低方差。此时 $G_t - b(s_t)$ 是对**优势函数** $A_\pi(s, a) = q_\pi(s, a) - v_\pi(s)$ 的蒙特卡洛估计。

- **探索与利用的自平衡**：利用 $\nabla \ln \pi = \nabla \frac{\pi}{\pi}$ ，更新律可改写为

$$\theta_{t+1} = \theta_t + \alpha \frac{G_t}{\pi_\theta(a_t | s_t)} \nabla_\theta \pi_\theta(a_t | s_t) \quad (26)$$

回报 G_t 越大，该动作的概率被抬得越多（利用）；概率 $\pi(a_t | s_t)$ 越小——动作越少被选——同样的回报带来越大的相对提升（探索）。探索与利用在更新中被自动平衡。

策略梯度方法直接在策略空间上做优化：不需要显式的策略改进操作，梯度上升本身就完成了改进；它天然支持连续动作空间，也便于利用神经网络等函数逼近。REINFORCE 是最早、最简单的策略梯度算法，它无偏但方差高。以 TD 或价值网络来估计 q_t 、减基线降方差，就构成了后续 Actor-Critic 等更先进方法的出发点。

