强化学习笔记

卢奇
luqi.code@gmail.com

目录

Chapter 1

绪论

如何让机器学会下棋？监督学习的做法是给机器看大量棋谱，模仿人类高手的落子。但这有两个问题：一是棋谱覆盖不了所有可能的局面；二是下棋是**序列决策**——当前这步棋的好坏，可能要几十步后才能评判。

强化学习（Reinforcement Learning, RL）解决的正是这类问题：agent（智能体）通过与环境交互，在没有“标准答案”的情况下，逐渐学会做出最优决策。

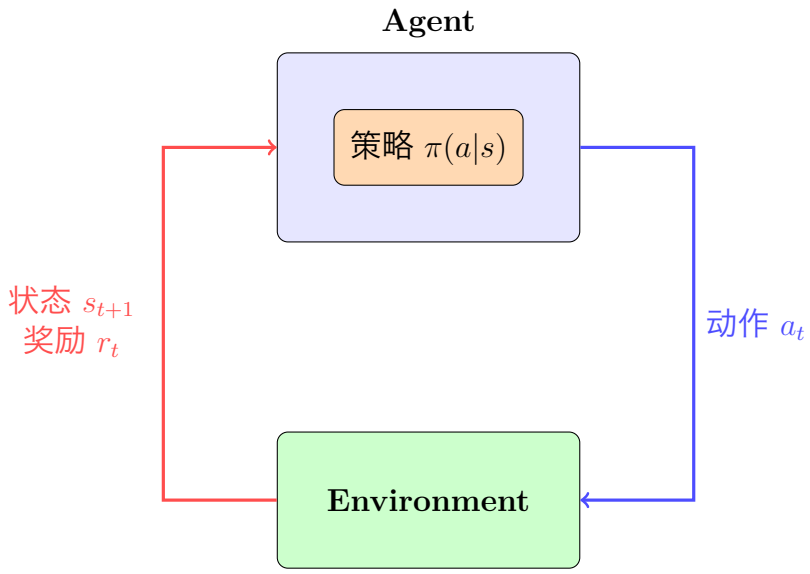
本章的目标是建立 RL 的数学框架, 核心结论是: RL 问题可以形式化为 Markov Decision Process（MDP），目标是找到使期望累积奖励最大的策略。

1.1 强化学习概述

1.1.1 什么是强化学习

强化学习是机器学习的一个重要分支，研究 agent 如何通过与环境的交互来学习最优决策策略。与监督学习和无监督学习相比，RL 有三个显著特点：

- **交互式学习**：Agent 通过执行动作、观察反馈来积累经验，没有“正确答案”的标签
- **延迟奖励**：动作的好坏可能要很久之后才能知道（想想下棋时的“弃子争先”）
- **序列决策**：当前决策影响未来状态，数据分布由策略自身决定



时刻 t：Agent 观察 s_t，执行 a_t；环境返回 r_t 和 s_{t+1}

图 1.1： Agent-Environment 交互循环。这是 RL 的基本范式：Agent 根据当前状态选择动作，环境根据动作返回奖励和新状态。这个循环不断重复，直到任务结束或无限持续。

1.1.2　与监督学习的本质区别

RL 与监督学习最关键的区别在于**数据分布**。监督学习假设数据独立同分布（i.i.d.），但 RL 中数据分布完全取决于当前策略——策略一变，采集到的数据就变了。

特性	监督学习	无监督学习	强化学习
反馈类型	标签（正确答案）	无反馈	奖励信号（标量）
数据分布	i.i.d.	i.i.d.	**由策略决定，非 i.i.d.**
目标	最小化预测误差	发现数据结构	最大化累积奖励
决策时序	单步独立决策	无决策	序列相关决策

表 1.1： 三种机器学习范式对比。RL 的核心难点在于数据分布由策略自身决定，导致 distribution shift 问题。

这个区别带来了实际训练中的诸多困难：策略变化导致数据分布变化，而数据分布变化又影响策略更新，形成复杂的动态过程。

关键点

RL 的三大核心挑战：

1. **探索与利用的权衡**（Exploration vs. Exploitation）：应该尝试新动作（探索）还是使用已知好动作（利用）？过度探索浪费资源，过度利用可能错过更好的策略。
2. **信用分配问题**（Credit Assignment）：最终奖励如何归因到之前的各个动作？在下棋中，哪一步导致了最终的胜利或失败？
3. **非稳态性**：数据分布随策略变化而变化，这让训练变得不稳定，也是 RL 比监督学习更难的根本原因。

1.1.3　应用场景

RL 在众多领域取得了突破性进展：

- **游戏 AI**：AlphaGo（围棋）击败世界冠军、OpenAI Five（Dota 2）、AlphaStar（星际争霸）
- **机器人控制**：机械臂精细操作、四足机器人运动、无人机控制
- **自动驾驶**：决策规划、路径规划
- **推荐系统**：优化长期用户满意度而非即时点击率
- **大语言模型对齐**：RLHF（Reinforcement Learning from Human Feedback）已成为 GPT、Claude 等模型的标准训练流程

值得注意的是，近年来 RL 最成功的应用可能不是游戏，而是 LLM 对齐——这也是本书第 五 章的重点内容。

1.2　Markov Decision Process (MDP)

有了直观理解后，我们来建立数学框架。MDP 是 RL 的标准形式化工具，它把 agent 与环境的交互过程描述为一个离散时间随机控制过程。

1.2.1　MDP 五元组定义

定义 1.1 (Markov Decision Process)**.** 一个 *MDP* 由五元组 $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ 定义：

- $\mathcal{S}$：**状态空间**（*State Space*），所有可能状态的集合
- $\mathcal{A}$：**动作空间**（*Action Space*），所有可能动作的集合
- $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$：**状态转移概率**，$P(s'|s, a)$ 表示在状态 s 执行动作 a 后转移到状态 s' 的概率
- $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$：**奖励函数**，$R(s, a)$ 表示在状态 s 执行动作 a 获得的即时奖励
- $\gamma \in [0, 1]$：**折扣因子**（*Discount Factor*），权衡即时奖励与未来奖励的相对重要性

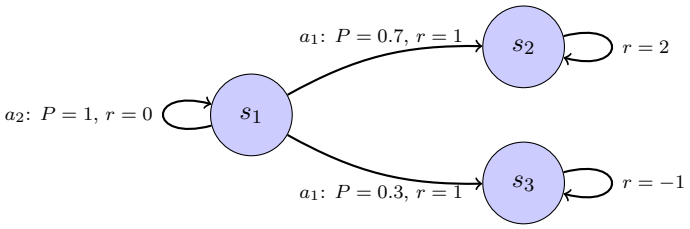


图 1.2: 一个简单的 MDP 示例。状态 s_1 有两个动作可选：a_1 以 0.7 概率转移到 s_2（高奖励），0.3 概率转移到 s_3（负奖励）；a_2 原地不动但无奖励。

注意

奖励函数的定义有多种形式：

- $R(s,a)$：只依赖于当前状态和动作
- $R(s,a,s')$：还依赖于下一状态
- $R(s)$：只依赖于状态

这些形式可以相互转化。例如，$R(s,a,s')$ 可以转化为 $R(s,a) = \sum_{s'} P(s'|s,a)R(s,a,s')$。本书默认使用 $R(s,a)$ 形式。

1.2.2　Markov 性质

MDP 的核心假设是 **Markov 性质**（Markov Property），也称为“无记忆性”：

定义 1.2 (Markov 性质)**.** 给定当前状态 s_t 和动作 a_t，下一状态 s_{t+1} 和奖励 r_t 的分布只依赖于 (s_t, a_t)，与历史状态和动作无关：

$$P(s_{t+1}, r_t|s_t, a_t) = P(s_{t+1}, r_t|s_0, a_0, s_1, a_1, \ldots, s_t, a_t) \tag{1.1}$$

直观理解：**当前状态包含了预测未来所需的所有信息。**历史轨迹可以被“压缩”为当前状态，状态是对历史的“充分统计量”。

关键点

如果真实环境不满足 Markov 性质怎么办？答案是扩展状态表示：

- **历史窗口**：Atari 游戏中用连续 4 帧图像作为状态，捕捉运动信息
- **RNN 隐状态**：用循环神经网络的隐状态编码历史
- **Transformer**：用 attention 机制处理变长历史

本质上是把“历史”编码进状态表示里，使得扩展后的状态满足 Markov 性质。

1.2.3　状态空间与动作空间的分类

根据状态空间和动作空间的性质，RL 问题可以分类为：

类型	状态空间	动作空间	典型例子
离散-离散	有限/可数	有限/可数	棋类游戏（棋盘布局 × 落子位置）
连续-离散	$\mathbb{R}^n$	有限/可数	Atari 游戏（像素图像 × 按键）
连续-连续	$\mathbb{R}^n$	$\mathbb{R}^m$	机器人控制（关节角度 × 力矩）

表 1.2: RL 问题的分类

1.3　轨迹与回报

1.3.1　轨迹的定义

Agent 与环境交互产生的状态-动作-奖励序列称为**轨迹**（Trajectory），也叫 episode 或 rollout。

定义 1.3 (轨迹). 轨迹 τ 是一个状态、动作、奖励的序列：

$$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_{T-1}, a_{T-1}, r_{T-1}, s_T) \tag{1.2}$$

其中 T 是轨迹的长度（*episode* 的终止时刻）。

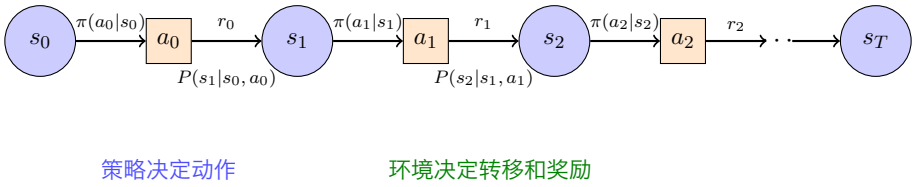


图 1.3: 轨迹的生成过程。蓝色圆圈表示状态，橙色方块表示动作。注意两种概率来源：策略 π 决定动作，环境 P 决定状态转移。

1.3.2　轨迹概率分解

轨迹的概率如何计算？

定理 1.1 (轨迹概率分解). 在策略 π 下，轨迹 τ 的概率为：

$$p(\tau|\pi) = p(s_0) \prod_{t=0}^{T-1} \pi(a_t|s_t) \cdot P(s_{t+1}|s_t, a_t) \tag{1.3}$$

证明. 利用条件概率的链式法则和 Markov 性质：

$$\begin{aligned} p(\tau|\pi) &= p(s_0, a_0, s_1, a_1, \ldots, s_T|\pi) && (1.4)\\ &= p(s_0) \cdot p(a_0|s_0) \cdot p(s_1|s_0, a_0) \cdot p(a_1|s_1) \cdot p(s_2|s_1, a_1) \cdots && (1.5)\\ &= p(s_0) \prod_{t=0}^{T-1} \underbrace{p(a_t|s_t)}_{\pi(a_t|s_t)} \cdot \underbrace{p(s_{t+1}|s_t, a_t)}_{P(s_{t+1}|s_t, a_t)} && (1.6) \end{aligned}$$

其中第二步使用了 Markov 性质：$p(s_{t+1}|s_0, a_0, \ldots, s_t, a_t) = p(s_{t+1}|s_t, a_t)$。 □

重要

这个分解非常重要！观察公式 (1.3)：

- $p(s_0)$：初始状态分布，由环境决定
- $\pi(a_t|s_t)$：策略，是我们要学习/优化的对象
- $P(s_{t+1}|s_t, a_t)$：环境动力学，由环境决定

关键观察：$p(s_0)$ 和 $P(s_{t+1}|s_t, a_t)$ 与策略参数 θ 无关。因此，当对 $\log p(\tau|\pi_\theta)$ 求关于 θ 的梯度时，环境动力学项会消失！这是 Policy Gradient 定理的核心（见第 三 章）。

1.3.3　Return（回报）的定义

定义 1.4 (回报 / Reward-to-go). 从时刻 t 开始的**回报**（*Return*）或 ***Reward-to-go*** 定义为未来奖励的折扣累积和：

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \tag{1.7}$$

其中 $\gamma \in [0, 1]$ 是折扣因子。

回报 G_t 满足一个简单但重要的递推关系：

$$G_t = r_t + \gamma G_{t+1} \tag{1.8}$$

Bellman 方程正是基于这个递推结构——下一章将详细展开。

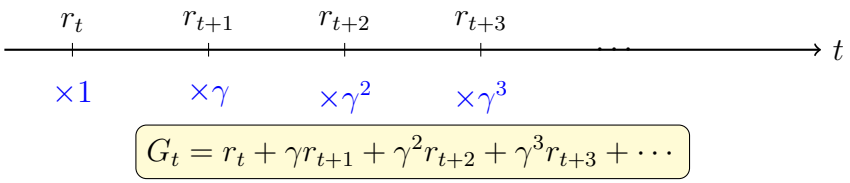


图 1.4: 回报 G_t 的计算：未来奖励按 γ^k 折扣后求和。γ 越小，远期奖励的权重越低。

1.3.4　Episodic vs Continuing Tasks

根据任务是否有终止状态，RL 问题分为两类：

定义 1.5 (Episodic 与 Continuing Tasks)**.**

Episodic Task 存在终止状态 s_T，轨迹有限长度。例如：棋类游戏（胜负终局）、机器人完成特定任务、一局游戏

Continuing Task 没有终止状态，$T \to \infty$。例如：股票交易、持续运行的控制系统、服务器调度

对于 Continuing Task，必须使用 $\gamma < 1$ 以确保回报有界。当 $|r_t| \le R_{\max}$ 时：

$$|G_t| \le \sum_{k=0}^{\infty} \gamma^k R_{\max} = \frac{R_{\max}}{1-\gamma} \tag{1.9}$$

1.3.5　折扣因子的多重意义

折扣因子 γ 是一个看似简单但含义丰富的超参数：

- $\gamma = 0$：Agent 只关心即时奖励，完全忽略未来（“短视”）
- $\gamma = 1$：Agent 平等对待所有未来奖励（仅适用于 Episodic Task）
- $0 < \gamma < 1$：权衡即时与未来奖励，γ 越大越“远视”

关键点

折扣因子的多重理解：

1. **数学角度**：确保无限和收敛，G_t 有界
2. **经济学角度**：时间价值——“现在的 1 元比未来的 1 元更值钱”
3. **不确定性角度**：未来越远，预测越不准确，应降低权重
4. **实践角度**：γ 定义了“有效时间尺度”。$1/(1-\gamma)$ 步之外的奖励贡献会指数衰减到可以忽略。例如 $\gamma = 0.99$ 时，有效视野约 100 步。

1.4　策略与价值函数

1.4.1　策略的定义

定义 1.6 (策略)**. 策略**（*Policy*）π 是从状态到动作的映射，定义了 *agent* 在每个状态下如何选择动作。策略分为两类：

- **确定性策略**（*Deterministic Policy*）：$\pi : \mathcal{S} \to \mathcal{A}$，即 $a = \pi(s)$
- **随机策略**（*Stochastic Policy*）：$\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$，即 $\pi(a|s)$ 表示在状态 s 选择动作 a 的概率

随机策略满足归一化条件：对于所有 $s \in \mathcal{S}$，

$$\sum_{a \in \mathcal{A}} \pi(a|s) = 1 \tag{1.10}$$

为什么需要随机策略？确定性策略不是更简单吗？

注意

随机策略的三个优势：

1. **探索**：随机性有助于探索未知动作，避免陷入局部最优
2. **对抗性**：在博弈场景中，确定性策略容易被对手预测（想想“石头剪刀布”）
3. **优化友好**：随机策略的梯度更容易计算（确定性策略的梯度涉及 argmax，不可微）

确定性策略可以看作随机策略的特例（概率集中在单一动作上）。

1.4.2　状态价值函数 $V^\pi(s)$

定义 1.7 (状态价值函数)**.** 策略 π 的**状态价值函数**（*State Value Function*）定义为从状态 s 出发，遵循策略 π 的期望回报：

$$V^\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid S_t = s \right] \tag{1.11}$$

其中期望是对策略 π 下产生的所有可能轨迹取的。

$V^\pi(s)$ 回答的问题是：**在状态 s，如果从现在开始一直遵循策略 π，能获得的期望总奖励是多少？**

1.4.3　动作价值函数 $Q^\pi(s, a)$

定义 1.8 (动作价值函数)**.** 策略 π 的**动作价值函数**（*Action Value Function*）定义为在状态 s 执行动作 a，之后遵循策略 π 的期望回报：

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid S_t = s, A_t = a \right] \tag{1.12}$$

$Q^\pi(s, a)$ 比 $V^\pi(s)$ 提供了更细粒度的信息：它允许我们单独评估每个动作的价值，而不是只知道“按 π 走的整体期望”。这使得我们可以比较不同动作的好坏，从而改进策略。

1.4.4　V 与 Q 的关系

两个价值函数之间存在简单而重要的联系：

定理 1.2 (V 与 Q 的关系)**.**

$$V^\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s) Q^\pi(s,a) = \mathbb{E}_{a\sim\pi(\cdot|s)}\left[Q^\pi(s,a)\right] \tag{1.13}$$

证明. 由全期望公式：

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi\left[G_t \mid S_t = s\right] && (1.14)\\ &= \sum_{a\in\mathcal{A}} p(A_t = a|S_t = s)\cdot \mathbb{E}_\pi\left[G_t \mid S_t = s, A_t = a\right] && (1.15)\\ &= \sum_{a\in\mathcal{A}} \pi(a|s)\cdot Q^\pi(s,a) && (1.16) \end{aligned}$$

□

直观理解：状态价值 $V^\pi(s)$ 就是所有动作价值 $Q^\pi(s,a)$ 按策略概率 $\pi(a|s)$ 加权的平均。

1.4.5　优势函数 $A^\pi(s,a)$

定义 1.9 (优势函数)**. 优势函数**（*Advantage Function*）定义为动作价值与状态价值之差：

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s) \tag{1.17}$$

优势函数衡量的是：在状态 s 执行动作 a 相对于“平均水平”（即按策略采样动作）好多少或差多少。

定理 1.3 (优势函数的关键性质)**.** 优势函数在策略分布下的期望为零：

$$\mathbb{E}_{a\sim\pi(\cdot|s)}\left[A^\pi(s,a)\right] = 0 \tag{1.18}$$

证明. 由公式 (1.13)：

$$\begin{aligned} \mathbb{E}_{a\sim\pi(\cdot|s)}\left[A^\pi(s,a)\right] &= \mathbb{E}_{a\sim\pi}\left[Q^\pi(s,a) - V^\pi(s)\right] && (1.19)\\ &= \mathbb{E}_{a\sim\pi}\left[Q^\pi(s,a)\right] - V^\pi(s) && (1.20)\\ &= V^\pi(s) - V^\pi(s) = 0 && (1.21) \end{aligned}$$

□

关键点

优势函数的直观意义：

- $A(s,a) > 0$：动作 a 优于平均水平
- $A(s,a) < 0$：动作 a 劣于平均水平
- $A(s,a) = 0$：动作 a 与平均水平持平

优势函数在 Policy Gradient 中的作用将在第 三 章详细讨论。

1.4.6　最优策略与最优价值函数

定义 1.10 (最优价值函数)**.** **最优状态价值函数** $V^*(s)$ 和**最优动作价值函数** $Q^*(s,a)$ 定义为所有策略中能达到的最大价值：

$$V^*(s) = \max_\pi V^\pi(s) \tag{1.22}$$

$$Q^*(s,a) = \max_\pi Q^\pi(s,a) \tag{1.23}$$

定义 1.11 (最优策略)**.** 策略 π^* 称为**最优策略**（*Optimal Policy*），如果对于所有状态 $s \in \mathcal{S}$：

$$V^{\pi^*}(s) = V^*(s) \tag{1.24}$$

定理 1.4 (最优策略的存在性)**.** 对于有限 *MDP*（有限状态空间和动作空间），至少存在一个确定性最优策略。

最优策略可以由最优动作价值函数直接导出：

定理 1.5 (从 Q^* 导出最优策略)**.** 给定 Q^*，最优策略为：

$$\pi^*(s) = \operatorname*{argmax}_{a \in \mathcal{A}} Q^*(s,a) \tag{1.25}$$

这个结论是 Q-Learning 等 Value-Based 方法的理论基础：**只要学到 Q^*，就能直接得到最优策略。**

1.5　RL 问题的形式化目标

综合以上定义，强化学习的目标可以形式化为：

定义 1.12 (RL 目标)**.** 找到最优策略 π^*，使得期望回报最大化：

$$\pi^* = \operatorname*{argmax}_\pi J(\pi) \tag{1.26}$$

其中 $J(\pi)$ 是策略的性能指标，常见定义包括：

1. **从固定初始状态出发**：$J(\pi) = V^\pi(s_0)$
2. **初始状态有分布**：$J(\pi) = \mathbb{E}_{s_0 \sim p(s_0)}\left[V^\pi(s_0)\right]$
3. **轨迹期望**：$J(\pi) = \mathbb{E}_{\tau \sim \pi}\left[\sum_{t=0}^{T} \gamma^t r_t\right]$

这三种定义在大多数情况下是等价的。后续章节将介绍解决这一优化问题的不同方法：

- **Value-Based 方法**（第 二 章）：学习 V^* 或 Q^*，再由 argmax 导出 π^*
- **Policy-Based 方法**（第 三 章）：直接优化参数化策略 π_θ
- **Actor-Critic 方法**（第 三 章）：同时学习策略（Actor）和价值函数（Critic）

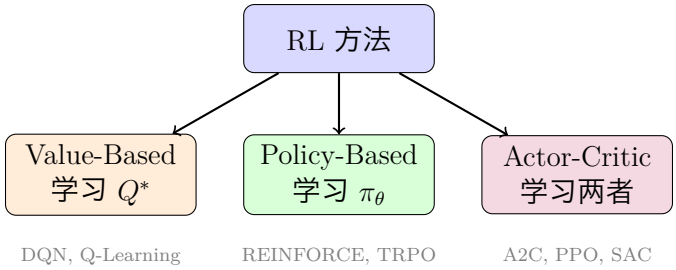


图 1.5: RL 方法的三大类别。Value-Based 学习价值函数，Policy-Based 直接学习策略，Actor-Critic 结合两者。

本章小结

关键点

回顾开篇问题：如何让机器学会序列决策？答案是将问题形式化为 MDP，目标是最大化期望累积奖励。

本章建立的核心概念：

1. **MDP 五元组** $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ 是 RL 的标准形式化框架
2. **Markov 性质**：当前状态包含预测未来所需的所有信息
3. **轨迹概率分解**：$p(\tau|\pi) = p(s_0)\prod_t \pi(a_t|s_t)P(s_{t+1}|s_t, a_t)$，策略与环境动力学分离
4. **回报的递推**：$G_t = r_t + \gamma G_{t+1}$，这是 Bellman 方程的起点
5. **价值函数**：V^π 和 Q^π 衡量策略的长期价值
6. **优势函数**：$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$，期望为零
7. **RL 目标**：$\max_\pi J(\pi)$，找到使期望回报最大的策略

下一章将介绍 Bellman 方程——利用回报的递推结构来高效计算价值函数，以及基于价值函数的 Q-Learning、DQN 等算法。

Chapter 2

基于价值的强化学习

2.1 引言：从价值到决策

2.1.1 核心问题

在第一章中，我们定义了价值函数 $V^\pi(s)$ 和 $Q^\pi(s,a)$，它们衡量了给定策略 π 下状态或状态-动作对的“好坏”。现在自然的问题是：

如何计算价值函数？如何从价值函数导出最优策略？

这两个问题的答案构成了 Value-Based RL 的核心。本章将介绍三类方法：

1. **动态规划**（Dynamic Programming）：当环境模型已知时，精确求解
2. **无模型估计**（MC 与 TD）：当环境模型未知时，从采样中学习
3. **深度 Q 网络**（DQN）：处理大规模或连续状态空间

2.1.2 为什么要学习价值函数？

一个核心观察是：**如果我们知道最优动作价值函数 $Q^*(s,a)$，就能直接导出最优策略**：

$$\pi^*(s) = \arg\max_a Q^*(s,a) \tag{2.1}$$

这是 Value-Based 方法的理论基础——不直接学习策略，而是通过学习价值函数间接得到策略。

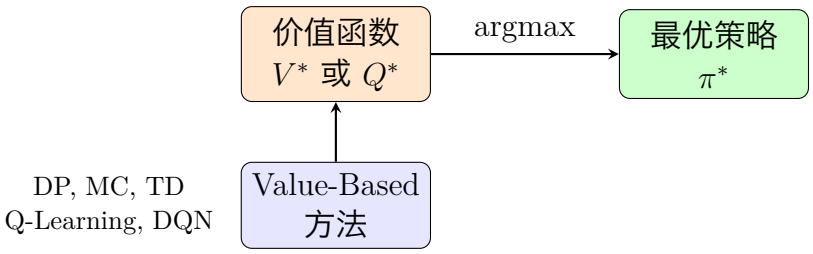


图 2.1: Value-Based RL 的核心思路：学习价值函数，通过 argmax 导出策略

2.2 Bellman 方程

Bellman 方程是 RL 的核心方程，它揭示了价值函数的递推结构。理解 Bellman 方程是掌握所有 Value-Based 方法的基础。

2.2.1　直觉：一步分解

Bellman 方程的核心形式是：

$$V^\pi(s) = \mathbb{E}\left[r + \gamma V^\pi(s')\right] \tag{2.2}$$

用文字说：**当前状态的价值 = 即时奖励 + 折扣后的下一状态价值。**
这个递推关系使得我们可以通过迭代来计算价值函数，而不需要枚举所有可能的轨迹。

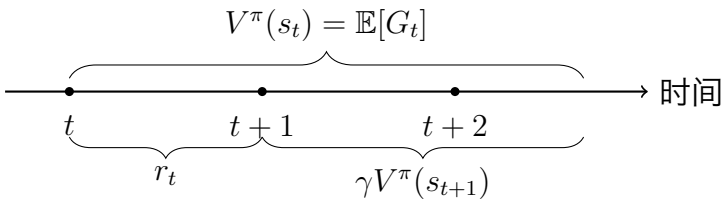


图 2.2: Bellman 方程的直觉：将无限长的回报分解为“一步奖励 + 剩余价值”

2.2.2　Bellman Expectation Equation 的详细推导

我们首先推导策略 π 下的 Bellman Expectation Equation。

定理 2.1 (Bellman Expectation Equation for V^π)**.** 对于任意策略 π，状态价值函数满足：

$$V^\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s) \left[R(s,a) + \gamma \sum_{s'\in\mathcal{S}} P(s'|s,a) V^\pi(s') \right] \tag{2.3}$$

或写成紧凑的期望形式：

$$V^\pi(s) = \mathbb{E}_{a\sim\pi, s'\sim P}\left[R(s,a) + \gamma V^\pi(s') \mid S_t = s\right] \tag{2.4}$$

证明. 从状态价值函数的定义出发（回顾第一章 (1.11)）：

$$V^\pi(s) = \mathbb{E}_\pi\left[G_t \mid S_t = s\right] \tag{2.5}$$

$$= \mathbb{E}_\pi\left[r_t + \gamma G_{t+1} \mid S_t = s\right] \quad \text{（回报的递推关系 (1.8)）} \tag{2.6}$$

关键步骤：由于 G_{t+1} 只依赖于 S_{t+1} 之后的轨迹，我们可以条件分解：

$$V^\pi(s) = \mathbb{E}_\pi\left[r_t + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1}] \mid S_t = s\right] \tag{2.7}$$

$$= \mathbb{E}_\pi\left[r_t + \gamma V^\pi(S_{t+1}) \mid S_t = s\right] \tag{2.8}$$

其中最后一步使用了 $V^\pi(s') = \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']$ 的定义。
现在展开期望。由于给定 $S_t = s$：

- 动作 $A_t = a$ 的概率为 $\pi(a|s)$
- 给定 (s,a)，下一状态 $S_{t+1} = s'$ 的概率为 $P(s'|s,a)$
- 即时奖励 $r_t = R(s,a)$（假设确定性奖励，随机情况类似）

因此：

$$V^\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}} P(s'|s,a)\left[R(s,a) + \gamma V^\pi(s')\right] \tag{2.9}$$

$$= \sum_{a\in\mathcal{A}} \pi(a|s) \left[R(s,a) + \gamma \sum_{s'\in\mathcal{S}} P(s'|s,a) V^\pi(s')\right] \tag{2.10}$$

□

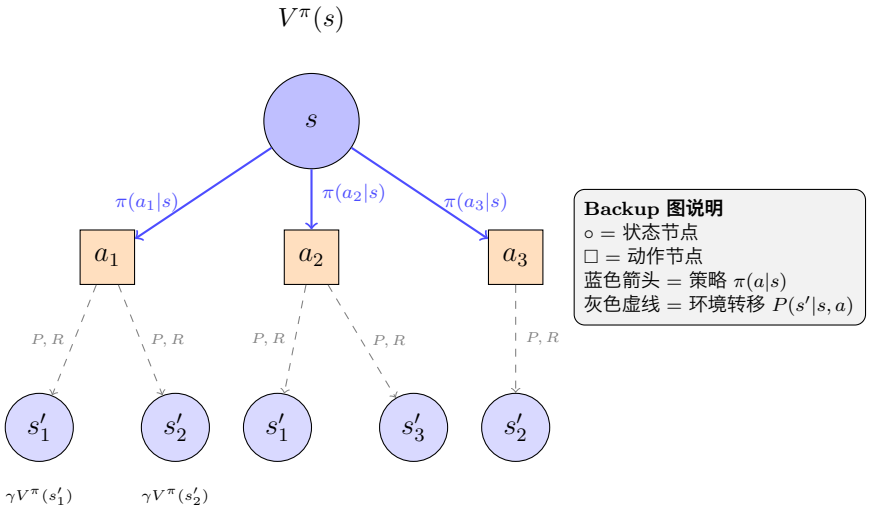


图 2.3: Bellman Expectation Equation 的 Backup 图。从状态 s 出发，先按策略 $\pi(a|s)$ 选择动作，再按环境转移概率 $P(s'|s,a)$ 到达下一状态。价值"回溯"地从叶子节点传播到根节点。

类似地，我们可以推导 Q^π 的 Bellman Expectation Equation：

定理 2.2 (Bellman Expectation Equation for Q^π)**.** 对于任意策略 π，动作价值函数满足：

$$Q^\pi(s,a) = R(s,a) + \gamma \sum_{s'\in\mathcal{S}} P(s'|s,a) \sum_{a'\in\mathcal{A}} \pi(a'|s') Q^\pi(s',a') \tag{2.11}$$

或写成期望形式：

$$Q^\pi(s,a) = R(s,a) + \gamma \mathbb{E}_{s'\sim P, a'\sim\pi}\left[Q^\pi(s',a')\right] \tag{2.12}$$

证明. 从动作价值函数的定义出发：

$$Q^\pi(s,a) = \mathbb{E}_\pi\left[G_t \mid S_t = s, A_t = a\right] \tag{2.13}$$

$$= \mathbb{E}_\pi\left[r_t + \gamma G_{t+1} \mid S_t = s, A_t = a\right] \tag{2.14}$$

$$= R(s,a) + \gamma \mathbb{E}_\pi\left[G_{t+1} \mid S_t = s, A_t = a\right] \tag{2.15}$$

关键步骤：由 Markov 性质，给定 (s,a) 后，G_{t+1} 只通过 S_{t+1} 影响。对 S_{t+1} 的分布求期望：

$$\mathbb{E}_\pi\left[G_{t+1} \mid S_t = s, A_t = a\right] = \sum_{s'} P(s'|s,a) \mathbb{E}_\pi\left[G_{t+1} \mid S_{t+1} = s'\right] \tag{2.16}$$

$$= \sum_{s'} P(s'|s,a) V^\pi(s') \tag{2.17}$$

利用第一章的 V-Q 关系 $V^\pi(s') = \sum_{a'} \pi(a'|s') Q^\pi(s', a')$，代入得：

$$Q^\pi(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \sum_{a'} \pi(a'|s') Q^\pi(s', a') \tag{2.18}$$

□

关键点

V^π 与 Q^π 的相互表示是推导 Bellman 方程的关键：

$$V^\pi(s) = \sum_a \pi(a|s) Q^\pi(s,a) \tag{2.19}$$

$$Q^\pi(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^\pi(s') \tag{2.20}$$

将 (2.20) 代入 (2.19) 得到 V^π 的 Bellman 方程；将 (2.19) 代入 (2.20) 得到 Q^π 的 Bellman 方程。

2.2.3 Bellman Optimality Equation 的详细推导

最优价值函数满足 Bellman Optimality Equation。与 Expectation 版本不同，这里对动作求**最大值**而非期望。

定理 2.3 (Bellman Optimality Equation for V^*)**.** 最优状态价值函数满足：

$$V^*(s) = \max_{a \in \mathcal{A}} \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^*(s') \right] \tag{2.21}$$

证明. 最优价值函数的定义是所有策略中最大的价值：

$$V^*(s) = \max_\pi V^\pi(s) \tag{2.22}$$

最优策略 π^* 是确定性的，在每个状态选择使 $Q^*(s,a)$ 最大的动作。因此：

$$V^*(s) = V^{\pi^*}(s) \tag{2.23}$$

$$= \max_a Q^*(s,a) \quad \text{（最优策略选择最大 Q 值的动作）} \tag{2.24}$$

由 (2.20) 对最优策略成立：

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s') \tag{2.25}$$

代入得：

$$V^*(s) = \max_a \left[R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s') \right] \tag{2.26}$$

□

定理 2.4 (Bellman Optimality Equation for Q^*)**.** 最优动作价值函数满足：

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) \max_{a' \in \mathcal{A}} Q^*(s', a') \tag{2.27}$$

证明.

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s') \tag{2.28}$$

$$= R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a') \tag{2.29}$$

其中最后一步使用了 $V^*(s') = \max_{a'} Q^*(s',a')$。 □

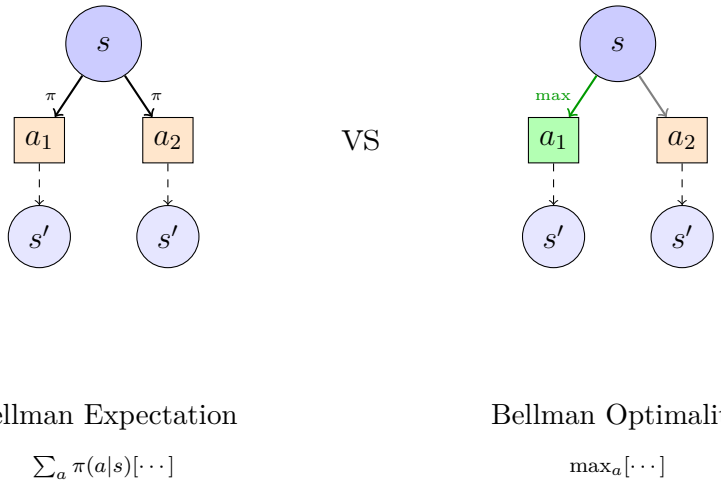


图 2.4: Bellman Expectation vs. Bellman Optimality。左图对动作求加权平均（按 π），右图对动作求最大值。

重要

Bellman Expectation vs. Bellman Optimality 的关键区别：

- **Bellman Expectation**：对动作求加权平均（按 $\pi(a|s)$），描述**给定策略**的价值
- **Bellman Optimality**：对动作求最大值（$\max_a$），描述**最优策略**的价值

Q-Learning 直接逼近 Q^*，因此使用 Bellman Optimality Equation 作为更新目标。

2.2.4　V^* 与 Q^* 的关系

最优价值函数之间的关系：

$$V^*(s) = \max_a Q^*(s,a) \tag{2.30}$$

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s') \tag{2.31}$$

由 Q^* 可以直接导出最优策略：

$$\pi^*(s) = \operatorname*{argmax}_a Q^*(s,a) \tag{2.32}$$

这是 Q-Learning 系列方法的理论基础：**只要学到 Q^*，就能得到最优策略。**

2.2.5　Bellman 算子与收缩性质

为了理解为什么基于 Bellman 方程的迭代方法能够收敛，我们引入 Bellman 算子的概念。

定义 2.1 (Bellman 算子)**.** 定义 *Bellman Expectation* 算子 $\mathcal{T}^\pi$ 和 *Bellman Optimality* 算子 $\mathcal{T}^*$:

$$(\mathcal{T}^\pi V)(s) = \sum_a \pi(a|s) \left[R(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s') \right] \tag{2.33}$$

$$(\mathcal{T}^* V)(s) = \max_a \left[R(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s') \right] \tag{2.34}$$

定理 2.5 (Bellman 算子的收缩性)**.** *Bellman* 算子是 γ-收缩映射。即对于任意两个价值函数 V_1, V_2:

$$\|\mathcal{T}^\pi V_1 - \mathcal{T}^\pi V_2\|_\infty \le \gamma \|V_1 - V_2\|_\infty \tag{2.35}$$

$\mathcal{T}^*$ 同样满足此性质。

证明. 对于任意状态 s:

$$|(\mathcal{T}^\pi V_1)(s) - (\mathcal{T}^\pi V_2)(s)| = \left| \gamma \sum_a \pi(a|s) \sum_{s'} P(s'|s,a)[V_1(s') - V_2(s')] \right| \tag{2.36}$$

$$\le \gamma \sum_a \pi(a|s) \sum_{s'} P(s'|s,a) |V_1(s') - V_2(s')| \tag{2.37}$$

$$\le \gamma \|V_1 - V_2\|_\infty \underbrace{\sum_a \pi(a|s) \sum_{s'} P(s'|s,a)}_{=1} \tag{2.38}$$

$$= \gamma \|V_1 - V_2\|_\infty \tag{2.39}$$

对所有 s 取最大，得到结论。 □

关键点

收缩性质的重要推论:

1. **唯一不动点**：Bellman 算子有唯一的不动点 V^π（或 V^*）
2. **迭代收敛**：从任意初始 V_0 出发，$V_{k+1} = \mathcal{T} V_k$ 收敛到不动点
3. **收敛速度**：误差以 γ^k 的速度指数衰减

2.3　动态规划方法

当环境模型 $P(s'|s,a)$ 和 $R(s,a)$ 完全已知时，可以使用动态规划（Dynamic Programming, DP）方法精确求解最优策略。

2.3.1　Policy Evaluation（策略评估）

给定策略 π，如何计算其价值函数 V^π？

定义 2.2 (Policy Evaluation)**.** *Policy Evaluation* 是通过迭代 *Bellman Expectation* 方程来计算 V^π 的过程：

$$V_{k+1}(s) = \sum_a \pi(a|s) \left[R(s,a) + \gamma \sum_{s'} P(s'|s,a) V_k(s') \right] \tag{2.40}$$

从任意初始 V_0 开始，迭代直到收敛 $V_k \to V^\pi$。

定理 2.6 (Policy Evaluation 的收敛性)**.** 对于任意初始 V_0 和 $\gamma < 1$，*Policy Evaluation* 迭代收敛到唯一的 V^π。收敛速度为 $O(\gamma^k)$。

证明. 由定理 2.5，Bellman 算子 $\mathcal{T}^\pi$ 是 γ-收缩映射。由 Banach 不动点定理：

1. 存在唯一不动点 V^π 使得 $\mathcal{T}^\pi V^\pi = V^\pi$
2. 对任意 V_0，序列 $V_{k+1} = \mathcal{T}^\pi V_k$ 收敛到 V^π

误差界：$\|V_k - V^\pi\|_\infty \le \gamma^k \|V_0 - V^\pi\|_\infty$ □

算法 1: Policy Evaluation

输入: 策略 π，MDP $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$，收敛阈值 θ
输出: 价值函数 V^π

初始化 $V(s) = 0$ 对于所有 $s \in \mathcal{S}$;
repeat
  $\Delta \leftarrow 0$;
  foreach $s \in \mathcal{S}$ **do**
    $v \leftarrow V(s)$;
    $V(s) \leftarrow \sum_a \pi(a|s) \left[R(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s') \right]$;
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$;
until $\Delta < \theta$;
return V

2.3.2　Policy Improvement（策略改进）

给定 V^π，如何构造一个更好的策略？

定理 2.7 (Policy Improvement Theorem)**.** 设 π 和 π' 是两个策略。如果对于所有 $s \in \mathcal{S}$：

$$Q^\pi(s, \pi'(s)) \ge V^\pi(s) \tag{2.41}$$

则 π' 至少与 π 一样好：对于所有 s，$V^{\pi'}(s) \ge V^\pi(s)$。

证明. 从条件 (2.41) 出发，利用 Q^π 的定义展开：

$$V^\pi(s) \le Q^\pi(s, \pi'(s)) \tag{2.42}$$

$$= R(s, \pi'(s)) + \gamma \sum_{s'} P(s'|s, \pi'(s)) V^\pi(s') \tag{2.43}$$

$$\le R(s, \pi'(s)) + \gamma \sum_{s'} P(s'|s, \pi'(s)) Q^\pi(s', \pi'(s')) \quad \text{（再用条件）} \tag{2.44}$$

$$= R(s, \pi'(s)) + \gamma \sum_{s'} P(s'|s, \pi'(s)) \left[R(s', \pi'(s')) + \gamma \sum_{s''} P(s''|s', \pi'(s')) V^\pi(s'') \right] \tag{2.45}$$

继续展开，直到 episode 结束（或利用折扣因子使尾部趋近于零）：

$$V^\pi(s) \le \mathbb{E}_{\pi'} \left[r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \mid S_0 = s \right] \tag{2.46}$$

$$= V^{\pi'}(s) \tag{2.47}$$

□

注意

Policy Improvement Theorem 的直觉：如果在每个状态，按 π' 选择动作比按 π 选择更好（体现在 Q 值上），那么整体策略 π' 也更好。

贪心策略改进保证了条件 (2.41) 以等号或严格大于成立：

定义 2.3 (贪心策略改进). 给定 V^π，定义贪心改进策略：

$$\pi'(s) = \operatorname*{argmax}_a Q^\pi(s, a) = \operatorname*{argmax}_a \left[R(s, a) + \gamma \sum_{s'} P(s'|s, a) V^\pi(s') \right] \tag{2.48}$$

对于贪心改进：

$$Q^\pi(s, \pi'(s)) = \max_a Q^\pi(s, a) \ge Q^\pi(s, \pi(s)) = V^\pi(s) \tag{2.49}$$

因此满足 Policy Improvement Theorem 的条件。

2.3.3 Policy Iteration

交替进行 Policy Evaluation 和 Policy Improvement：

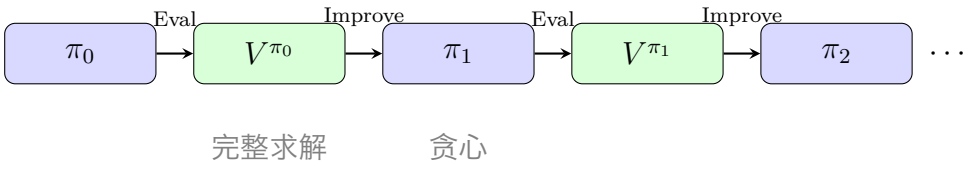


图 2.5: Policy Iteration 的流程：评估 → 改进 → 评估 → 改进 → ⋯

算法 2: Policy Iteration

输入: MDP $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$
输出: 最优策略 π^* 和最优价值 V^*

初始化 π 为任意策略;
repeat
  Policy Evaluation: 计算 V^π（使用算法 1）;
  policy-stable $\leftarrow$ true;
  foreach $s \in \mathcal{S}$ **do**
    old-action $\leftarrow \pi(s)$;
    $\pi(s) \leftarrow \text{argmax}_a \left[R(s,a) + \gamma \sum_{s'} P(s'|s,a)V^\pi(s')\right]$;
    if *old-action* $\neq \pi(s)$ **then**
      policy-stable $\leftarrow$ false;
until π 不再改变（*policy-stable* = *true*);
return π, V^π

定理 2.8 (Policy Iteration 的有限步收敛)**.** 对于有限 *MDP*，*Policy Iteration* 在有限步内收敛到最优策略 π^*。

证明.
1. 由 Policy Improvement Theorem，每次迭代 $V^{\pi_{k+1}}(s) \geq V^{\pi_k}(s)$ 对所有 s 成立
2. 有限 MDP 只有有限多个确定性策略（$|\mathcal{A}|^{|\mathcal{S}|}$ 个）
3. 价值函数严格单调递增（除非已是最优），因此不会循环
4. 结合上述，必在有限步内收敛

□

2.3.4 Value Iteration

Value Iteration 将 Policy Evaluation 和 Policy Improvement 合并为一步，直接迭代 Bellman Optimality 方程：

算法 3: Value Iteration

输入: MDP $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, 收敛阈值 θ
输出: 最优价值 V^* 和最优策略 π^*

初始化 $V(s) = 0$ 对于所有 s;
repeat
  $\Delta \leftarrow 0$;
  foreach $s \in \mathcal{S}$ **do**
    $v \leftarrow V(s)$;
    $V(s) \leftarrow \max_a \left[R(s,a) + \gamma \sum_{s'} P(s'|s,a)V(s')\right]$;
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$;
until $\Delta < \theta$;
$\pi^*(s) \leftarrow \text{argmax}_a \left[R(s,a) + \gamma \sum_{s'} P(s'|s,a)V(s')\right]$ 对于所有 s;
return V, π^*

关键点

Value Iteration vs. Policy Iteration 的对比：

	Policy Iteration	**Value Iteration**
Policy Evaluation	完整求解 V^π	仅一步 Bellman 更新
理论依据	Bellman Expectation	Bellman Optimality
每轮复杂度	高（需多次内循环）	低（单次遍历）
收敛轮数	少	多
总体效率	较低	通常更高

实践中，Value Iteration 通常更高效，因为不需要在每轮完整求解 V^π。

2.3.5　DP 方法的局限性

动态规划虽然能精确求解，但有明显局限：

1. **需要完整模型**：必须知道 $P(s'|s,a)$ 和 $R(s,a)$
2. **状态空间遍历**：每轮需要遍历所有状态，复杂度 $O(|\mathcal{S}|^2|\mathcal{A}|)$
3. **无法处理连续空间**：表格方法无法直接应用

这些局限促使了无模型方法（MC、TD）和函数逼近方法（DQN）的发展。

2.4　无模型方法：Monte Carlo vs Temporal Difference

当环境模型未知时，需要通过与环境交互的样本来估计价值函数。两种主要方法是 Monte Carlo (MC) 和 Temporal Difference (TD)。

2.4.1　核心问题

只有采样数据，没有环境模型 P 和 R，如何估计 $V^\pi(s)$？

回顾价值函数的定义：

$$V^\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] \tag{2.50}$$

这是一个期望，可以用样本均值来估计。

2.4.2　Monte Carlo 估计

MC 方法的核心思想：**用实际轨迹的回报 G_t 来估计期望。**

定义 2.4 (Monte Carlo 更新)**.** 对于在轨迹中访问过的状态 S_t：

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t - V(S_t)\right) \tag{2.51}$$

其中 $G_t = \sum_{k=0}^{T-t-1} \gamma^k r_{t+k}$ 是从 t 时刻开始到 *episode* 结束的实际回报，α 是学习率。

注意

MC 更新可以理解为增量式均值估计。如果 $\alpha = 1/N(s)$（$N(s)$ 是状态 s 的访问次数），则 $V(s)$ 正好是历史回报的均值。

MC 方法的特点：

- **必须等到 episode 结束**才能计算 G_t，仅适用于 episodic 任务
- **无偏估计**：$\mathbb{E}[G_t|S_t = s] = V^\pi(s)$，不依赖任何估计值
- **高方差**：G_t 累积了整条轨迹的随机性（动作选择、状态转移、奖励）
- **不使用 Bootstrap**：目标完全来自真实采样

2.4.3　Temporal Difference 估计

TD 方法的核心思想：**用“一步奖励 + 下一状态的估计价值”来代替完整回报。**

定义 2.5 (TD(0) 更新)**.** 对于每一步转移 (S_t, A_t, R_t, S_{t+1})：

$$V(S_t) \leftarrow V(S_t) + \alpha \left(\underbrace{r_t + \gamma V(S_{t+1})}_{TD\ target} - V(S_t) \right) \tag{2.52}$$

其中 TD 误差定义为：

$$\delta_t = r_t + \gamma V(S_{t+1}) - V(S_t) \tag{2.53}$$

TD 方法的特点：

- **每步都可更新**，适用于 continuing 任务和在线学习
- **有偏估计**：使用了 $V(S_{t+1})$ 的估计值（可能不准确）
- **低方差**：只使用单步奖励，不累积长期随机性
- **使用 Bootstrap**：用当前估计 $V(S_{t+1})$ 来更新 $V(S_t)$

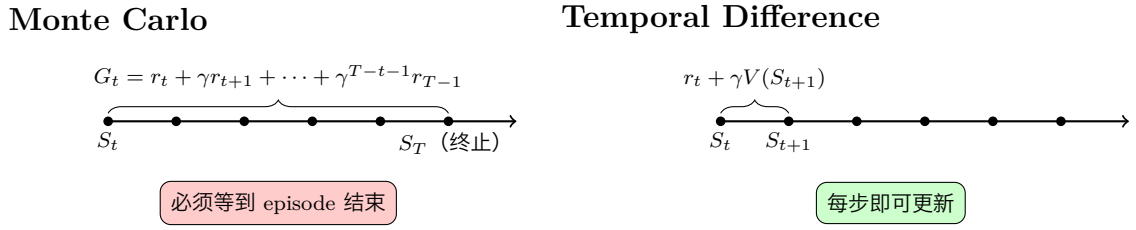


图 2.6: MC 与 TD 的更新范围对比。MC 需要完整轨迹，TD 只需单步转移。

2.4.4　偏差-方差权衡分析

MC 和 TD 代表了偏差-方差权衡的两个极端。

	Monte Carlo	TD(0)
目标	G_t（真实回报）	$r_t + \gamma V(S_{t+1})$
偏差	无偏	有偏（依赖 V 估计）
方差	高（累积轨迹随机性）	低（仅单步随机性）
Bootstrap	否	是
适用任务	仅 Episodic	Episodic 和 Continuing
数据效率	低（需完整轨迹）	高（每步更新）

表 2.1: MC 与 TD 的全面对比

关键点

为什么 TD 方差更小?

MC 目标 $G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$ 包含了从 t 到终止的所有奖励。每个 r_k 都是随机变量（取决于策略选择和环境转移），方差叠加:

$$\mathrm{Var}(G_t) = \sum_{k=0}^{T-t-1} \gamma^{2k} \mathrm{Var}(r_{t+k}) + \text{协方差项} \tag{2.54}$$

TD 目标 $r_t + \gamma V(S_{t+1})$ 只有 r_t 和 S_{t+1} 是随机的。虽然 $V(S_{t+1})$ 可能不准确（有偏），但它是一个相对平滑的函数估计，不会累积随机性。

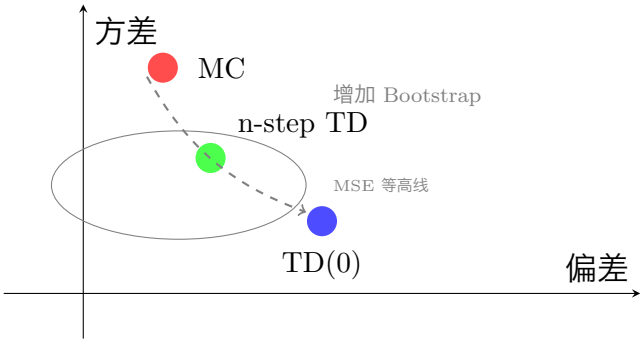


图 2.7: 偏差-方差权衡示意图。MC 无偏但高方差，TD 有偏但低方差。n-step TD 在两者之间取得平衡。

2.4.5　n-step TD 与 TD(λ)

n-step TD 是 MC 和 TD(0) 的中间形式，提供了在偏差-方差之间灵活权衡的方法。

定义 2.6 (n-step Return)**.**

$$G_t^{(n)} = r_t + \gamma r_{t+1} + \cdots + \gamma^{n-1} r_{t+n-1} + \gamma^n V(S_{t+n}) \tag{2.55}$$

- $n = 1$: $G_t^{(1)} = r_t + \gamma V(S_{t+1})$，即 TD(0)
- $n = \infty$: $G_t^{(\infty)} = r_t + \gamma r_{t+1} + \cdots$，即 Monte Carlo

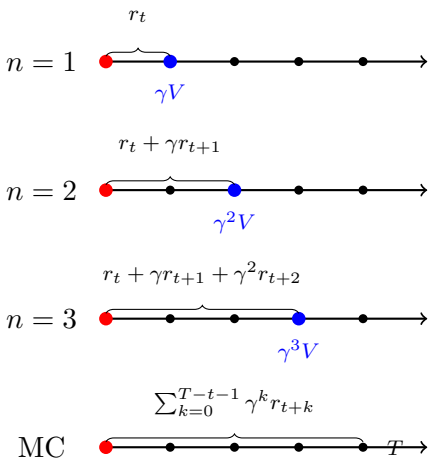


图 2.8: 不同 n 值下的 n-step return。红点为当前状态，蓝点为 Bootstrap 位置。

2.4.5.1　TD(λ)：加权平均

TD(λ) 方法将所有 n-step return 加权平均，而不是选择单一的 n：

定义 2.7 (λ-return)**.**

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^{(n)} \tag{2.56}$$

其中 $\lambda \in [0,1]$ 控制权重的衰减速度。

权重分布的直觉：

- $G_t^{(1)}$ 的权重：$(1-\lambda)$
- $G_t^{(2)}$ 的权重：$(1-\lambda)\lambda$
- $G_t^{(n)}$ 的权重：$(1-\lambda)\lambda^{n-1}$
- 权重之和为 1

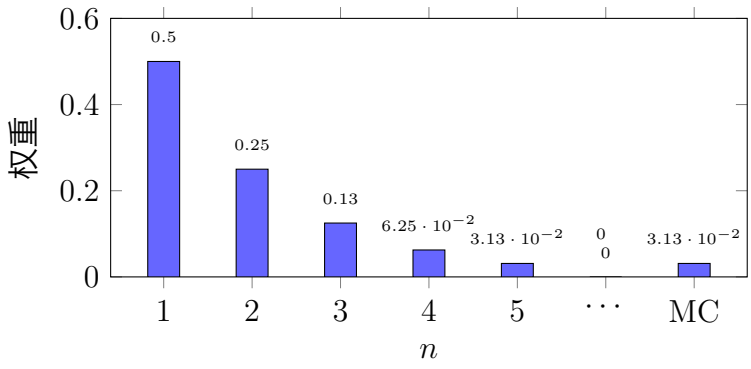


图 2.9: λ-return 的权重分布（$\lambda = 0.5$）。短期 return 权重大，长期 return 权重指数衰减。

特殊情况：

- $\lambda = 0$：$G_t^\lambda = G_t^{(1)}$，等价于 TD(0)
- $\lambda = 1$：$G_t^\lambda = G_t^{(\infty)}$，等价于 Monte Carlo

2.4.5.2 Eligibility Traces

直接计算 G_t^λ 需要等到 episode 结束。Eligibility Traces 提供了一种在线计算的方式。

定义 2.8 (Eligibility Trace). 为每个状态维护一个 *eligibility trace* $e_t(s)$:

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) + 1 & \text{if } s = S_t \\ \gamma\lambda e_{t-1}(s) & \text{otherwise} \end{cases} \tag{2.57}$$

初始时 $e_0(s) = 0$ 对所有 s。

TD(λ) 的在线更新:

$$V(s) \leftarrow V(s) + \alpha\delta_t e_t(s), \quad \forall s \tag{2.58}$$

其中 $\delta_t = r_t + \gamma V(S_{t+1}) - V(S_t)$ 是 TD 误差。

关键点

Eligibility Trace 的直觉:

- $e_t(s)$ 表示状态 s 对当前 TD 误差的“责任”
- 刚访问过的状态责任大，随时间指数衰减
- 当 $\delta_t \neq 0$ 时，所有有责任的状态都被更新

这实现了“信用分配”：奖励信号向过去的状态传播。

2.5 Q-Learning 与 SARSA

前面讨论的 MC 和 TD 方法是对 V^π 的估计。为了找到最优策略，我们需要估计动作价值函数 Q，这样才能通过 argmax 导出策略。

2.5.1 On-policy vs Off-policy

定义 2.9 (On-policy 与 Off-policy).

- ***On-policy***: 评估和改进的是同一个策略
  - 行为策略（收集数据）= 目标策略（被评估/改进）
  - 例子: *SARSA*
- ***Off-policy***: 用行为策略收集数据，评估/改进不同的目标策略
  - 行为策略（收集数据）$\neq$ 目标策略（被评估/改进）
  - 例子: *Q-Learning*

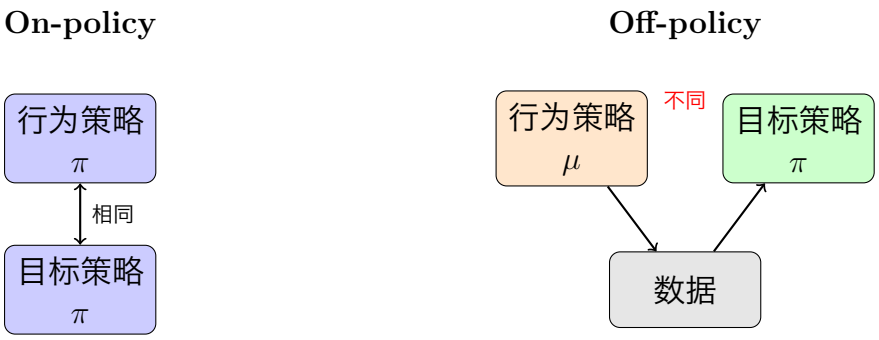


图 2.10: On-policy vs Off-policy。Off-policy 方法可以从任意行为策略的数据中学习目标策略。

Off-policy 的优势：

- 可以从历史数据（Experience Replay）中学习
- 可以从人类演示或其他策略的数据中学习
- 可以同时学习多个目标策略

2.5.2　SARSA 算法

SARSA 是一种 On-policy TD 控制方法,名字来自更新所需的五元组 $(S_t, A_t, R_t, S_{t+1}, A_{t+1})$。

定义 2.10 (SARSA 更新).

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(r_t + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right) \tag{2.59}$$

其中 A_{t+1} 是按当前策略（如 ϵ-*greedy*）实际采样的动作。

算法 4: SARSA

输入: 学习率 α，探索率 ϵ，折扣因子 γ

初始化 $Q(s,a)$ 为任意值，$Q(\text{terminal}, \cdot) = 0$;
foreach *episode* **do**
　初始化状态 S;
　用 ϵ-greedy 选择动作 A;
　repeat
　　执行 A，观察 R, S';
　　用 ϵ-greedy 选择 A';
　　$Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma Q(S', A') - Q(S,A)\right)$;
　　$S \leftarrow S'$, $A \leftarrow A'$;
　until S 是终止状态;

SARSA 学习的是当前策略（包括探索）的价值函数 Q^π，因此是 On-policy 的。

2.5.3　Q-Learning 算法

Q-Learning 是一种 Off-policy TD 控制方法，直接逼近最优 Q^*，而非当前策略的 Q^π。

定义 2.11 (Q-Learning 更新).

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(r_t + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t)\right) \tag{2.60}$$

关键区别:SARSA 使用 $Q(S_{t+1}, A_{t+1})$(实际采样的动作),Q-Learning 使用 $\max_{a'} Q(S_{t+1}, a')$(最优动作)。

定理 2.9 (Q-Learning 收敛性)**.** 在满足以下条件时，*Q-Learning* 收敛到 Q^*:

1. 所有状态-动作对被无限次访问
2. 学习率满足 *Robbins-Monro* 条件：$\sum_t \alpha_t = \infty$，$\sum_t \alpha_t^2 < \infty$

证明思路. Q-Learning 的更新可以写成随机近似（Stochastic Approximation）形式:

$$Q_{t+1}(s,a) = (1-\alpha_t)Q_t(s,a) + \alpha_t \left(r + \gamma \max_{a'} Q_t(s',a')\right) \tag{2.61}$$

这是对 Bellman Optimality 算子的随机逼近。由于 Bellman Optimality 算子是 γ-收缩的（定理 2.5），在 Robbins-Monro 条件下，随机近似收敛到不动点 Q^*。 □

算法 5: Q-Learning

输入: 学习率 α，探索率 ϵ，折扣因子 γ

初始化 $Q(s,a)$ 为任意值，$Q(\text{terminal}, \cdot) = 0$;
foreach *episode* **do**
  初始化状态 S;
  repeat
    用 ϵ-greedy 选择动作：$A = \begin{cases} \text{argmax}_a\, Q(S,a) & \text{以概率} 1-\epsilon \\ \text{随机动作} & \text{以概率} \epsilon \end{cases}$;
    执行 A，观察 R, S';
    $Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma \max_{a'} Q(S',a') - Q(S,A)\right)$;
    $S \leftarrow S'$;
  until S 是终止状态;

2.5.4　ϵ-greedy 探索策略

为了保证充分探索，常用 ϵ-greedy 策略:

定义 2.12 (ϵ-greedy)**.**

$$\pi(a|s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}|} & \text{if } a = \text{argmax}_{a'} Q(s,a') \\ \frac{\epsilon}{|\mathcal{A}|} & \text{otherwise} \end{cases} \tag{2.62}$$

- $\epsilon = 0$：纯贪心（Greedy），无探索
- $\epsilon = 1$：纯随机，无利用
- 通常 ϵ 随训练逐渐衰减（如 $\epsilon_t = \epsilon_0 / t$）

2.5.5　Cliff Walking 示例：Q-Learning vs SARSA

Cliff Walking 是一个经典的 Grid World 环境，清晰展示了 Q-Learning 和 SARSA 的行为差异。

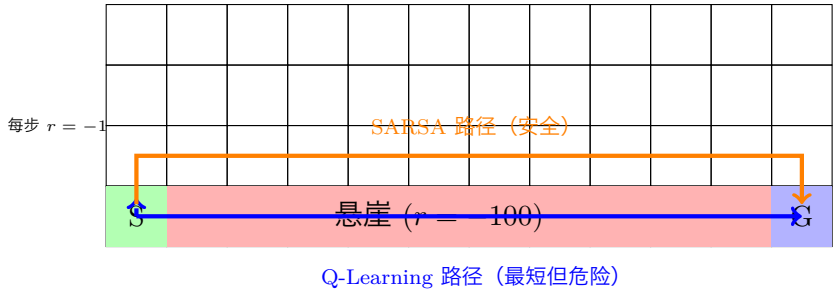


图 2.11: Cliff Walking 环境。S 是起点，G 是终点，红色区域是悬崖（掉下去 $r = -100$ 并回到 S）。Q-Learning 学习最优路径（沿悬崖边），SARSA 学习安全路径（远离悬崖）。

关键点

Q-Learning vs SARSA 在 Cliff Walking 中的行为：

- **Q-Learning**：学习最优路径（假设执行时不探索），即沿悬崖边走
  - 但训练时因 ϵ-greedy 探索，经常掉下悬崖
  - 学到的 Q 值反映“如果完美执行”的价值
- **SARSA**：学习考虑探索的保守路径，远离悬崖边缘
  - 训练时更安全，掉崖次数少
  - 学到的 Q 值反映“带探索执行”的价值

根本原因：SARSA 的目标包含实际执行的探索动作 A_{t+1}，Q-Learning 的目标总是选 max。

	Q-Learning	**SARSA**
类型	Off-policy	On-policy
TD target	$r + \gamma \max_{a'} Q(s', a')$	$r + \gamma Q(s', a')$
学习目标	Q^*（最优策略）	Q^π（当前策略）
行为	更激进/乐观	更保守
样本效率	高（可复用数据）	低（只用当前策略数据）
适合场景	安全代价低、需最优解	安全代价高、需稳定训练

表 2.2: Q-Learning 与 SARSA 的全面对比

2.6　Deep Q-Network (DQN)

表格 Q-Learning 无法处理大状态空间（如图像输入）或连续状态空间。DQN 使用神经网络逼近 Q^*，是深度强化学习的开创性工作。

2.6.1　函数逼近的动机

表格方法的局限：

- **状态空间爆炸**：Atari 游戏的像素空间约 $256^{210\times160\times3} \approx 10^{90000}$
- **无法泛化**：没见过的状态无法处理

- **连续状态**：无法用表格表示机器人关节角度等连续变量

解决方案：用参数化函数 $Q(s,a;\theta)$（如神经网络）逼近 Q^*。

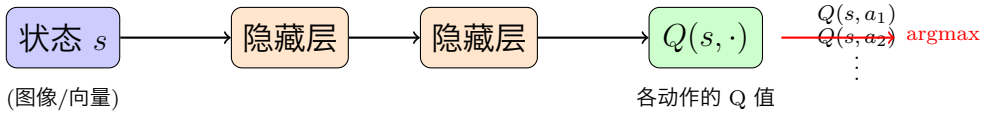


图 2.12: DQN 网络结构示意图。输入状态，输出各动作的 Q 值。选择 Q 值最大的动作。

2.6.2　DQN 损失函数

将 Q-Learning 更新转化为回归问题：

定义 2.13 (DQN 损失函数)**.**

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\left(\underbrace{r+\gamma\max_{a'}Q(s',a';\theta^-)}_{TD\ target\ y} - Q(s,a;\theta)\right)^2\right] \tag{2.63}$$

其中 $\mathcal{D}$ 是 *Replay Buffer*，θ^- 是 *Target Network* 的参数。

梯度计算：

$$\nabla_\theta\mathcal{L}(\theta) = \mathbb{E}\left[-2(y-Q(s,a;\theta))\nabla_\theta Q(s,a;\theta)\right] \tag{2.64}$$

注意：y 关于 θ 的梯度被忽略（因为用 θ^- 计算）。这是一种半梯度（semi-gradient）方法。

2.6.3　Experience Replay

定义 2.14 (Experience Replay)**.** 将转移 (s_t,a_t,r_t,s_{t+1}) 存入 *Replay Buffer* $\mathcal{D}$，训练时从 $\mathcal{D}$ 中均匀随机采样 *mini-batch*。

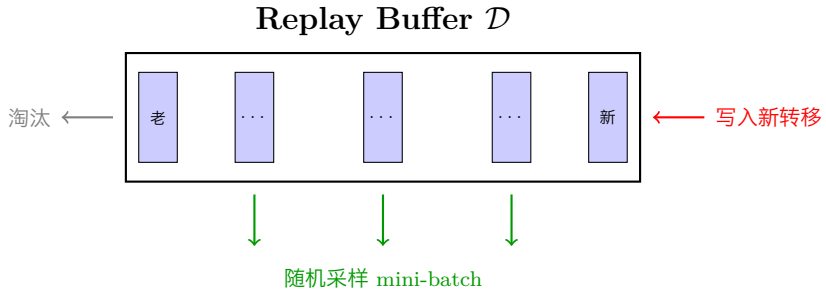


图 2.13: Experience Replay 机制。新转移写入 Buffer，旧转移被淘汰（FIFO），训练时随机采样。

Experience Replay 的好处：

1. **打破样本相关性**：连续收集的样本高度相关（同一轨迹），随机采样提供更独立的样本，符合 i.i.d. 假设
2. **提高数据效率**：每个样本可被多次使用，而非用一次就丢弃
3. **稳定数据分布**：Buffer 中的数据分布变化缓慢，训练更稳定

2.6.4　Target Network

定义 2.15 (Target Network). 使用一组旧参数 θ^- 计算 *TD target*，定期更新 $\theta^- \leftarrow \theta$（如每 C 步）。

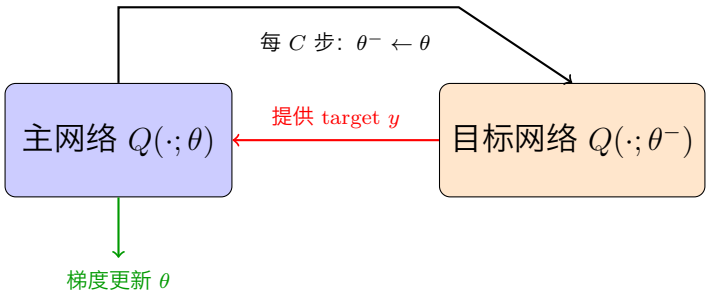


图 2.14: Target Network 机制。目标网络参数 θ^- 延迟更新，提供稳定的 TD target。

Target Network 的作用：

- 避免目标“追着当前估计跑”导致的震荡或发散
- TD target 在一段时间内保持固定，类似于监督学习的固定标签
- 虽然引入了一定的偏差（使用旧参数），但显著提高了稳定性

注意

另一种变体是**软更新**（Soft Update）：

$$\theta^- \leftarrow \tau\theta + (1-\tau)\theta^- \tag{2.65}$$

其中 $\tau \ll 1$（如 $\tau = 0.005$）。这在 TD3、SAC 等算法中广泛使用。

2.6.5　DQN 完整算法

算法 6: Deep Q-Network (DQN)

输入: Replay Buffer 容量 N，Mini-batch 大小 B，Target 更新频率 C，探索衰减参数

初始化 Replay Buffer $\mathcal{D}$，容量 N;
初始化 Q 网络参数 θ（随机）;
初始化 Target 网络参数 $\theta^- \leftarrow \theta$;
foreach *episode* **do**
  初始化状态 s_1（预处理，如堆叠 4 帧）;
  for $t = 1, 2, \ldots, T$ **do**
    以概率 ϵ 随机选择动作 a_t，否则 $a_t = \operatorname{argmax}_a Q(s_t, a; \theta)$;
    执行 a_t，观察 r_t, s_{t+1};
    存储转移 (s_t, a_t, r_t, s_{t+1}) 到 $\mathcal{D}$;
    从 $\mathcal{D}$ 随机采样 mini-batch：$\{(s_j, a_j, r_j, s'_j)\}_{j=1}^{B}$;
    计算 TD target：$y_j = \begin{cases} r_j & \text{if } s'_j \text{ 是终止状态} \\ r_j + \gamma \max_{a'} Q(s'_j, a'; \theta^-) & \text{otherwise} \end{cases}$;
    梯度下降：$\theta \leftarrow \theta - \alpha \nabla_\theta \frac{1}{B} \sum_j (y_j - Q(s_j, a_j; \theta))^2$;
    每 C 步：$\theta^- \leftarrow \theta$;
    衰减 ϵ;

重要

DQN 的两个关键技巧解决了深度 RL 的稳定性问题：

1. **Experience Replay**：解决样本相关性问题，提高数据效率
2. **Target Network**：解决目标不稳定问题，避免震荡发散

这两个技巧至今仍是 Value-Based 深度 RL 的标准做法。

2.6.6　DQN 变体

原始 DQN 存在一些问题，后续工作提出了多种改进。

2.6.6.1　Double DQN

原始 DQN 使用同一个网络选择动作和评估价值，导致**过估计**（Overestimation）问题。

定义 2.16 (过估计问题). max 操作天然倾向于选择被高估的动作：

$$\mathbb{E}\left[\max_a Q(s,a)\right] \geq \max_a \mathbb{E}[Q(s,a)] \tag{2.66}$$

当 Q 有估计噪声时，max 会放大正向误差。

定义 2.17 (Double DQN). 解耦动作选择和价值评估：

$$y = r + \gamma Q\left(s', \operatorname*{argmax}_{a'} Q(s', a'; \theta); \theta^-\right) \tag{2.67}$$

- 用**当前网络** θ 选择动作：$a^* = \operatorname{argmax}_{a'} Q(s', a'; \theta)$
- 用 ***Target* 网络** θ^- 评估价值：$Q(s', a^*; \theta^-)$

2.6.6.2　Dueling DQN

将 Q 值分解为状态价值和动作优势：

定义 2.18 (Dueling DQN)**.**

$$Q(s,a;\theta) = V(s;\theta_v) + \left(A(s,a;\theta_a) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s,a';\theta_a) \right) \tag{2.68}$$

其中 V 和 A 共享前几层特征，最后分叉。减去均值是为了可辨识性（*identifiability*）。

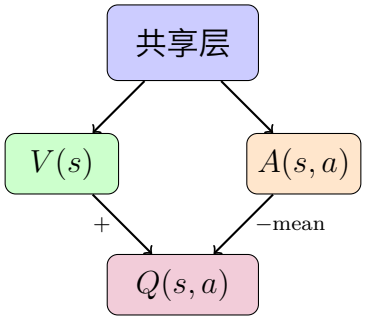


图 2.15: Dueling DQN 架构。分别估计 $V(s)$ 和 $A(s,a)$，再组合成 $Q(s,a)$。

Dueling DQN 的优势：

- 在某些状态下，所有动作的价值相近（如远离障碍物），分解结构允许网络直接学习状态价值
- 更高效地学习哪些状态是“好的”，独立于具体动作

2.6.6.3　其他改进

- **Prioritized Experience Replay**：优先采样 TD 误差大的样本
- **Multi-step Learning**：使用 n-step return 作为 target
- **Distributional RL**：学习回报的分布而非期望
- **Noisy Networks**：用网络参数噪声替代 ϵ-greedy 探索

Rainbow 将上述所有改进组合，在 Atari 游戏上取得了 SOTA 性能。

2.7　本章小结

关键点

本章核心内容：

1. **Bellman 方程**是 Value-Based RL 的理论基础
   - Bellman Expectation：描述给定策略的价值
   - Bellman Optimality：描述最优策略的价值
   - Bellman 算子是 γ-收缩映射，保证迭代收敛
2. **动态规划**（模型已知时）
   - Policy Evaluation：迭代求解 V^π
   - Policy Iteration：评估 $\rightarrow$ 改进 $\rightarrow$ 评估 $\rightarrow \cdots$
   - Value Iteration：直接迭代 Bellman Optimality 方程
3. **MC vs TD**（模型未知时）
   - MC：无偏高方差，需完整轨迹
   - TD：有偏低方差，每步更新
   - n-step TD 和 TD(λ) 在两者之间权衡
4. **Q-Learning vs SARSA**
   - Q-Learning：Off-policy，学习 Q^*，更激进
   - SARSA：On-policy，学习 Q^π，更保守
5. **DQN**：深度 Value-Based RL
   - Experience Replay：打破样本相关性
   - Target Network：稳定 TD target
   - Double DQN、Dueling DQN 等改进

注意

Value-Based 方法的局限性：

- 只能处理离散动作空间（argmax 需要枚举）
- 策略是隐式的，无法直接表示随机策略
- 对于高维动作空间，max 操作计算量大

这些局限促使了 Policy-Based 方法的发展，我们将在下一章详细介绍。

Chapter 3

基于策略的强化学习

3.1 引言：直接优化策略

3.1.1 核心问题

在第二章中，我们介绍了 Value-Based 方法：先学习 Q^*，再通过 argmax 导出策略。这种方法在离散动作空间中效果很好，但遇到以下问题时会遇到困难：

> **如果动作空间是连续的（如机器人关节角度），如何计算 $\text{argmax}_a Q(s,a)$？**
>
> **如果最优策略是随机的（如石头剪刀布），如何用确定性策略表示？**

Policy-Based 方法提供了一种更直接的思路：**直接参数化策略 $\pi_\theta(a|s)$，通过梯度上升最大化期望回报。**

3.1.2 Value-Based 方法的局限性

尽管 Q-Learning 和 DQN 取得了很大成功，但存在以下局限：

1. **连续动作空间困难**：$\max_a Q(s,a)$ 需要枚举或优化所有动作，在连续空间中无法直接计算
2. **函数逼近不稳定**（Deadly Triad）：当同时使用函数逼近、Bootstrapping 和 Off-policy 学习时，算法可能发散
3. **优化目标间接**：Value-Based 方法最小化 TD 误差，而非直接优化期望回报 $J(\pi)$
4. **只能学习确定性策略**：argmax 输出确定动作，但在某些环境中随机策略更优（如博弈、部分可观测环境）

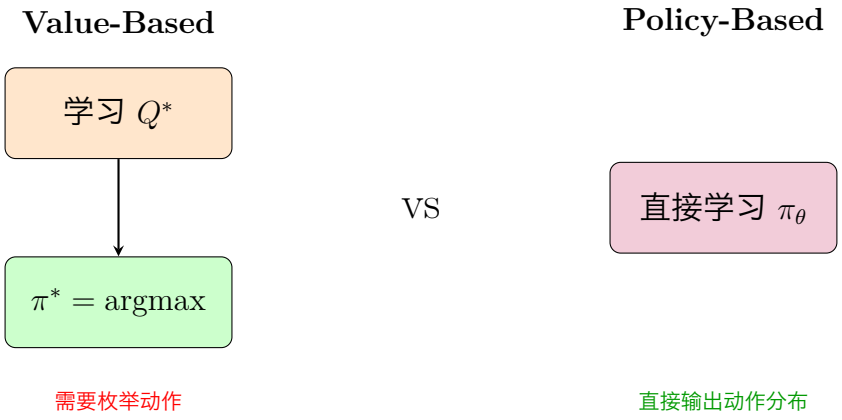


图 3.1： Value-Based vs. Policy-Based。前者间接导出策略，后者直接参数化策略。

3.1.3　参数化策略

Policy-Based 方法直接参数化策略 $\pi_\theta(a|s)$，通过梯度上升最大化期望回报：

$$J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[G_0] = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\gamma^t r_t\right] \tag{3.1}$$

策略参数化的常见形式：

- **离散动作空间**：Softmax 输出 Categorical 分布

$$\pi_\theta(a|s) = \frac{\exp(f_\theta(s,a))}{\sum_{a'}\exp(f_\theta(s,a'))} \tag{3.2}$$

- **连续动作空间**：输出高斯分布的参数 $(\mu_\theta(s), \sigma_\theta(s))$

$$\pi_\theta(a|s) = \mathcal{N}(a \mid \mu_\theta(s), \sigma_\theta^2(s)) \tag{3.3}$$

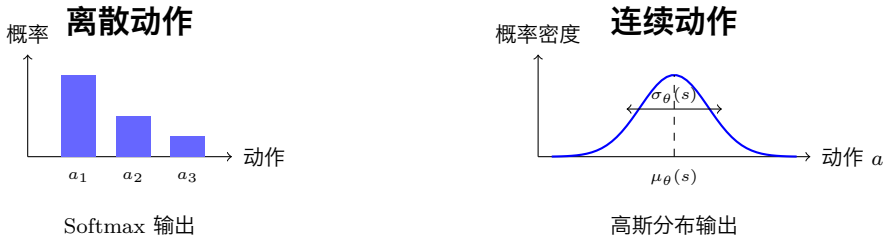


图 3.2: 策略参数化：离散动作用 Softmax，连续动作用高斯分布

关键点

Policy-Based 方法的优势：

1. **处理连续动作**：直接输出动作分布，无需 argmax
2. **学习随机策略**：可以输出动作的概率分布
3. **直接优化目标**：梯度上升直接最大化 $J(\theta)$
4. **更好的收敛性质**：策略参数的小变化导致策略的小变化（光滑）

3.2　Policy Gradient 定理

Policy Gradient 定理是 Policy-Based RL 的理论基础，它给出了目标函数 $J(\theta)$ 关于参数 θ 的梯度表达式。

3.2.1　目标函数定义

定义 3.1 (策略性能指标).

$$J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)] = \int p(\tau|\theta)R(\tau)d\tau \tag{3.4}$$

其中 $R(\tau) = G_0 = \sum_{t=0}^{T}\gamma^t r_t$ 是轨迹的总回报。

问题：如何计算 $\nabla_\theta J(\theta)$?
直接求导会遇到困难：$R(\tau)$ 依赖于环境动力学 $P(s'|s,a)$，而我们通常不知道 P。

3.2.2 Log-Derivative Trick

计算 $\nabla_\theta J(\theta)$ 的关键技巧是 **Log-Derivative Trick**（也称为 REINFORCE trick 或 Score Function）：

引理 3.1 (Log-Derivative Trick)**.** 对于任意概率分布 $p(x|\theta)$：

$$\nabla_\theta p(x|\theta) = p(x|\theta)\nabla_\theta \log p(x|\theta) \tag{3.5}$$

证明. 由对数的求导法则（链式法则）：

$$\nabla_\theta \log p(x|\theta) = \frac{\nabla_\theta p(x|\theta)}{p(x|\theta)} \tag{3.6}$$

两边同乘 $p(x|\theta)$：

$$p(x|\theta)\nabla_\theta \log p(x|\theta) = \nabla_\theta p(x|\theta) \tag{3.7}$$

□

注意

Log-Derivative Trick 的妙处：将对 $p(x|\theta)$ 的求导转化为对 $\log p(x|\theta)$ 的求导，而后者往往更容易计算，特别是当 p 是乘积形式时。

3.2.3 Policy Gradient 定理的完整推导

定理 3.2 (Policy Gradient Theorem)**.**

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\cdot G_t\right] \tag{3.8}$$

其中 $G_t = \sum_{k=0}^{T-t}\gamma^k r_{t+k}$ 是从时刻 t 开始的 *reward-to-go*。

证明. **Step 1**：应用 Log-Derivative Trick
从目标函数的定义出发：

$$\nabla_\theta J(\theta) = \nabla_\theta \int p(\tau|\theta)R(\tau)d\tau \tag{3.9}$$

$$= \int \nabla_\theta p(\tau|\theta)R(\tau)d\tau \quad \text{（交换积分和求导）} \tag{3.10}$$

$$= \int p(\tau|\theta)\nabla_\theta \log p(\tau|\theta)R(\tau)d\tau \quad \text{（Log-Derivative Trick）} \tag{3.11}$$

$$= \mathbb{E}_{\tau\sim\pi_\theta}\left[\nabla_\theta \log p(\tau|\theta)\cdot R(\tau)\right] \tag{3.12}$$

Step 2：展开 $\nabla_\theta \log p(\tau|\theta)$
回顾轨迹概率分解（第一章公式 (1.3)）：

$$p(\tau|\theta) = p(s_0)\prod_{t=0}^{T-1}\pi_\theta(a_t|s_t)P(s_{t+1}|s_t,a_t) \tag{3.13}$$

取对数：

$$\log p(\tau|\theta) = \log p(s_0) + \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t) + \sum_{t=0}^{T-1} \log P(s_{t+1}|s_t, a_t) \tag{3.14}$$

对 θ 求梯度：

$$\nabla_\theta \log p(\tau|\theta) = \underbrace{\nabla_\theta \log p(s_0)}_{=0} + \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) + \underbrace{\sum_{t=0}^{T-1} \nabla_\theta \log P(s_{t+1}|s_t, a_t)}_{=0} \tag{3.15}$$

重要

关键观察：$p(s_0)$ 是环境的初始状态分布，$P(s_{t+1}|s_t, a_t)$ 是环境的动力学模型，它们都与策略参数 θ 无关，因此梯度为零！
最终：

$$\nabla_\theta \log p(\tau|\theta) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \tag{3.16}$$

这意味着：**即使不知道环境动力学 P，我们也能计算 Policy Gradient**——这是 Policy Gradient 方法能够 Model-Free 的根本原因。

Step 3：代入得到基本形式
将 (3.16) 代入：

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta} \left[\left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \right) \cdot R(\tau) \right] \tag{3.17}$$

Step 4：引入 Reward-to-go（因果性）
展开 (3.17)：

$$\nabla_\theta J(\theta) = \mathbb{E} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot \sum_{t'=0}^{T} r_{t'} \right] \tag{3.18}$$

考虑交叉项 $\nabla_\theta \log \pi_\theta(a_t|s_t) \cdot r_{t'}$，当 $t' < t$ 时：

- $r_{t'}$ 是在时刻 t' 获得的奖励，发生在动作 a_t 之前
- a_t 的选择不可能影响过去的奖励 $r_{t'}$
- 因此 $\mathbb{E}[\nabla_\theta \log \pi_\theta(a_t|s_t) \cdot r_{t'}] = 0$（由引理 3.3）

只有 $t' \geq t$ 的奖励才与 a_t 相关，因此可以用 reward-to-go $G_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k}$ 替代 $R(\tau)$：

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot G_t \right] \tag{3.19}$$

□

引理 3.3 (Score Function 的期望为零). 对于任意策略 π_θ：

$$\mathbb{E}_{a\sim\pi_\theta(\cdot|s)} [\nabla_\theta \log \pi_\theta(a|s)] = 0 \tag{3.20}$$

证明.

$$\mathbb{E}_{a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s)\right] = \sum_a \pi_\theta(a|s) \cdot \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} \tag{3.21}$$

$$= \sum_a \nabla_\theta \pi_\theta(a|s) \tag{3.22}$$

$$= \nabla_\theta \sum_a \pi_\theta(a|s) \tag{3.23}$$

$$= \nabla_\theta 1 = 0 \tag{3.24}$$

□

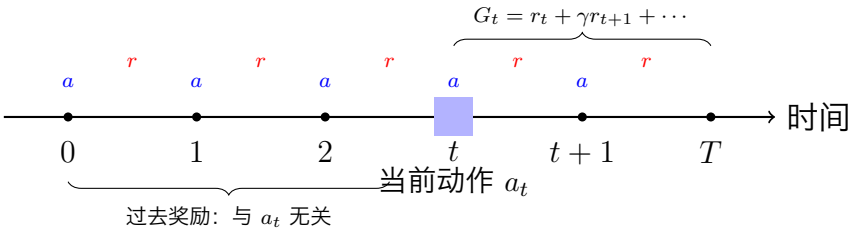


图 3.3: 因果性：动作 a_t 只影响未来奖励，不影响过去。因此 Policy Gradient 只需要 reward-to-go G_t。

关键点

Policy Gradient 定理的直观理解：

- $\nabla_\theta \log \pi_\theta(a_t|s_t)$ 是“增加动作 a_t 概率”的方向
- G_t 是该动作之后获得的累积奖励
- 如果 $G_t > 0$：沿梯度方向更新，增加 a_t 的概率
- 如果 $G_t < 0$：反向更新，减少 a_t 的概率

简言之：**好的动作更可能被选择，坏的动作更少被选择。**

3.3 REINFORCE 算法

REINFORCE 是最简单的 Policy Gradient 算法，直接使用蒙特卡洛采样来估计梯度。

算法 7: REINFORCE

输入: 学习率 α，初始策略参数 θ

foreach *episode* **do**
  采样轨迹 $\tau = (s_0, a_0, r_0, \ldots, s_T)$ 按策略 π_θ;
  foreach $t = 0, 1, \ldots, T$ **do**
    计算 $G_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k}$;
  更新参数：$\theta \leftarrow \theta + \alpha \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot G_t$;

3.3.1　无偏性

定理 3.4 (REINFORCE 是无偏估计). *REINFORCE* 的梯度估计：

$$\hat{g} = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot G_t \tag{3.25}$$

是 $\nabla_\theta J(\theta)$ 的无偏估计，即 $\mathbb{E}[\hat{g}] = \nabla_\theta J(\theta)$。

证明. 这是 Policy Gradient 定理（定理 3.2）的直接推论。轨迹 τ 按 π_θ 采样，G_t 是真实回报，期望就是 $\nabla_\theta J(\theta)$。 □

3.3.2　高方差问题

尽管 REINFORCE 无偏，但方差很大：

- G_t 累积了从 t 到终止的所有随机性（环境随机 + 策略随机）
- 轨迹越长，方差越大
- 奖励稀疏时，大部分轨迹的 $G_t \approx 0$，偶尔出现大的 G_t
- G_t 包含了很多与具体动作 a_t 无关的噪声

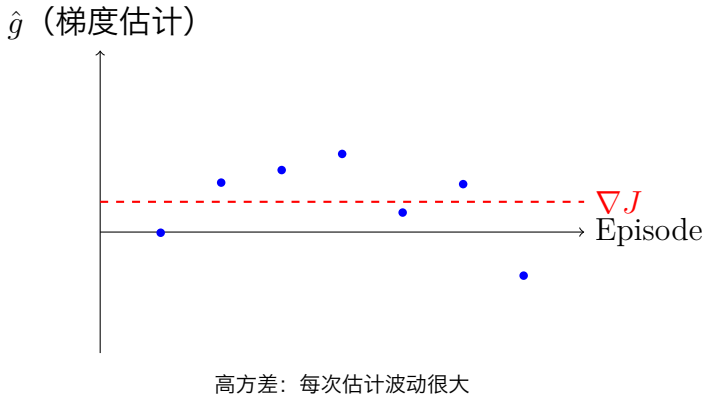


图 3.4: REINFORCE 的高方差问题示意图。虽然期望正确，但单次估计波动大。

3.4　Baseline 与方差降低

3.4.1　Baseline 技巧

一个巧妙的技巧是：从 G_t 中减去一个 baseline $b(s_t)$，可以降低方差而不引入偏差。

定理 3.5 (Baseline 不改变期望). 对于任意只依赖于状态 s（不依赖于动作 a）的函数 $b(s)$：

$$\mathbb{E}_{a\sim\pi_\theta(\cdot|s)}\left[\nabla_\theta \log \pi_\theta(a|s) \cdot b(s)\right] = 0 \tag{3.26}$$

证明. 由于 $b(s)$ 不依赖于 a，可以提出期望外：

$$\mathbb{E}_{a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s)\cdot b(s)\right] = b(s)\cdot \mathbb{E}_{a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s)\right] \tag{3.27}$$

$$= b(s)\cdot \sum_a \pi_\theta(a|s)\cdot \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} \tag{3.28}$$

$$= b(s)\cdot \sum_a \nabla_\theta \pi_\theta(a|s) \tag{3.29}$$

$$= b(s)\cdot \nabla_\theta \underbrace{\sum_a \pi_\theta(a|s)}_{=1} \tag{3.30}$$

$$= b(s)\cdot \nabla_\theta 1 = 0 \tag{3.31}$$

□

因此，Policy Gradient 可以写成：

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\sum_t \nabla_\theta \log \pi_\theta(a_t|s_t)\cdot (G_t - b(s_t))\right] \tag{3.32}$$

减去 baseline 不改变期望，但可以显著降低方差！

3.4.2　为什么 Baseline 能降低方差？

直觉解释：

- G_t 可能总是正的（如奖励都是正数），导致所有动作概率都被增加
- 减去 $b(s)$（如平均回报），使得 $G_t - b(s_t)$ 有正有负
- 好于平均的动作被增强，差于平均的动作被削弱

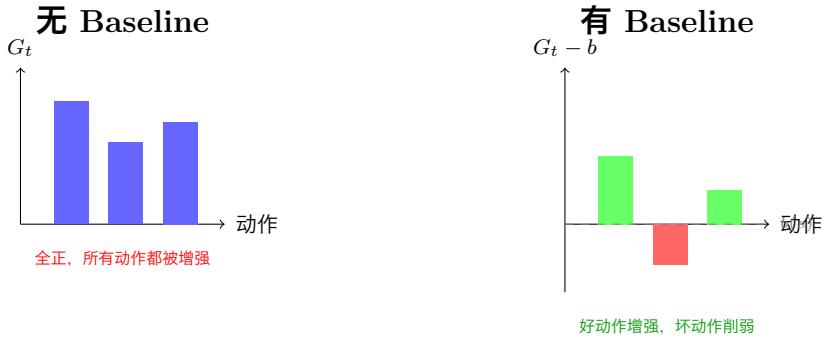


图 3.5: Baseline 的效果：将“绝对好坏”变为“相对好坏”，提供更清晰的学习信号。

3.4.3　最优 Baseline

定理 3.6 (最优 Baseline). 在不改变期望的前提下，使方差最小的 *baseline* 是状态价值函数：

$$b^*(s) = V^\pi(s) \tag{3.33}$$

证明思路. 方差 $\mathrm{Var}[\hat{g}]$ 关于 $b(s)$ 是二次函数，对 $b(s)$ 求导并令其为零：

$$\frac{\partial \mathrm{Var}[\hat{g}]}{\partial b(s)} = 0 \implies b^*(s) = \frac{\mathbb{E}[(\nabla \log \pi)^2 \cdot G]}{\mathbb{E}[(\nabla \log \pi)^2]} \tag{3.34}$$

在一定近似下（$(\nabla \log \pi)^2$ 与 G 独立），$b^*(s) \approx \mathbb{E}[G|s] = V^\pi(s)$。 □

当 $b(s) = V^\pi(s)$ 时，$G_t - V^\pi(s_t)$ 的期望正是 advantage 函数！

3.5 Advantage Function 与 Actor-Critic

3.5.1 Advantage 的定义与直觉

回顾第一章的 Advantage 函数定义：

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s) \tag{3.35}$$

当使用 $V^\pi(s)$ 作为 baseline 时：

$$\begin{aligned} \mathbb{E}_\pi\left[G_t - V^\pi(s_t) \mid s_t, a_t\right] &= \mathbb{E}_\pi[G_t|s_t,a_t] - V^\pi(s_t) && (3.36)\\ &= Q^\pi(s_t,a_t) - V^\pi(s_t) && (3.37)\\ &= A^\pi(s_t,a_t) && (3.38) \end{aligned}$$

因此，Policy Gradient with Advantage：

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\sum_t \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot A^\pi(s_t,a_t)\right] \tag{3.39}$$

关键点

Advantage $A(s,a)$ 的直觉：

- $V(s)$：在状态 s 的“平均”表现
- $Q(s,a)$：在状态 s 选择动作 a 的表现
- $A(s,a) = Q(s,a) - V(s)$：动作 a 比平均好多少

$A > 0$：这个动作好于平均，应该增加其概率 $A < 0$：这个动作差于平均，应该减少其概率

3.5.2 Advantage 的估计方法

实践中，需要估计 A^π。常见估计方法：

1. **Monte Carlo 估计**：
$$\hat{A}_t^{\mathrm{MC}} = G_t - \hat{V}(s_t) \tag{3.40}$$
无偏但高方差（G_t 累积了整条轨迹的随机性）。
2. **TD 估计**（1-step）：
$$\hat{A}_t^{\mathrm{TD}} = r_t + \gamma \hat{V}(s_{t+1}) - \hat{V}(s_t) = \delta_t \tag{3.41}$$
低方差但有偏（依赖于 $\hat{V}$ 的准确性）。

3. **n-step 估计**：介于两者之间

$$\hat{A}_t^{(n)} = \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n \hat{V}(s_{t+n}) - \hat{V}(s_t) \tag{3.42}$$

4. **GAE**（下一节介绍）：通过 λ 参数灵活权衡偏差和方差。

3.5.3　Actor-Critic 架构

为了估计 $\hat{V}(s)$，我们引入一个 **Critic** 网络。Actor-Critic 方法同时学习：

- **Actor**：策略网络 $\pi_\theta(a|s)$，输出动作分布
- **Critic**：价值网络 $\hat{V}_\phi(s)$，估计状态价值

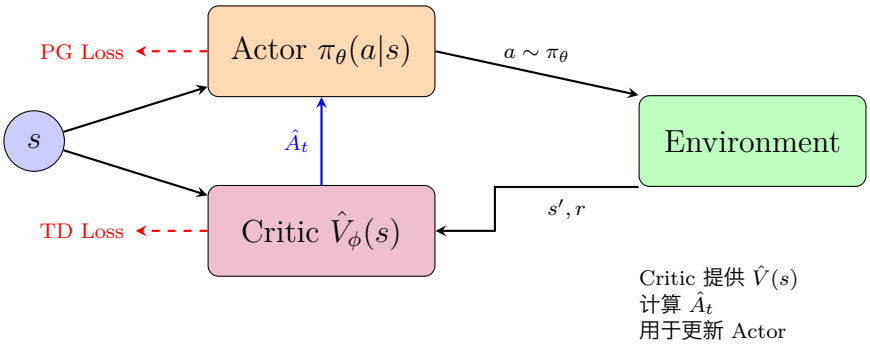


图 3.6: Actor-Critic 架构。Critic 估计 $V(s)$，提供 advantage 信号来更新 Actor。

3.5.4　A2C 算法

A2C（Advantage Actor-Critic）的核心更新规则：

Actor 更新（Policy Gradient with Advantage）：

$$\theta \leftarrow \theta + \alpha_\theta \sum_t \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot \hat{A}_t \tag{3.43}$$

Critic 更新（Value Function Regression）：

方式一：使用 MC target

$$\phi \leftarrow \phi - \alpha_\phi \nabla_\phi \sum_t \left(\hat{V}_\phi(s_t) - G_t \right)^2 \tag{3.44}$$

方式二：使用 TD target（更常用）

$$\phi \leftarrow \phi - \alpha_\phi \nabla_\phi \sum_t \left(\hat{V}_\phi(s_t) - (r_t + \gamma \hat{V}_\phi(s_{t+1})) \right)^2 \tag{3.45}$$

算法 8: Advantage Actor-Critic (A2C)

输入: Actor 参数 θ，Critic 参数 ϕ，学习率 $\alpha_\theta, \alpha_\phi$

foreach *episode* **do**
- 采样轨迹 $(s_0, a_0, r_0, \ldots, s_T)$ 按 π_θ;
- **foreach** $t = 0, \ldots, T-1$ **do**
  - 计算 TD 残差：$\delta_t = r_t + \gamma \hat{V}_\phi(s_{t+1}) - \hat{V}_\phi(s_t)$;
  - 或计算 MC advantage：$\hat{A}_t = G_t - \hat{V}_\phi(s_t)$;
- 更新 Actor：$\theta \leftarrow \theta + \alpha_\theta \sum_t \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot \hat{A}_t$;
- 更新 Critic：$\phi \leftarrow \phi - \alpha_\phi \nabla_\phi \sum_t (\hat{V}_\phi(s_t) - \text{target})^2$;

关键点

为什么需要 Critic?

- 提供 $\hat{V}(s)$ 来计算 advantage $\hat{A}_t$
- 比纯 MC（使用 G_t）方差更小
- 可每步更新，不用等 episode 结束
- Critic 的估计虽有偏差，但整体降低了梯度估计的方差

3.6 Generalized Advantage Estimation (GAE)

GAE 提供了一种在偏差和方差之间灵活权衡的 advantage 估计方法，是现代 Policy Gradient 算法（如 PPO）的核心组件。

3.6.1 从 n-step Advantage 到 GAE

回顾 n-step advantage 估计:

$$\hat{A}_t^{(n)} = \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n \hat{V}(s_{t+n}) - \hat{V}(s_t) \tag{3.46}$$

- $n = 1$：TD advantage，$\hat{A}_t^{(1)} = \delta_t = r_t + \gamma \hat{V}(s_{t+1}) - \hat{V}(s_t)$（低方差，高偏差）
- $n = \infty$：MC advantage，$\hat{A}_t^{(\infty)} = G_t - \hat{V}(s_t)$（高方差，低偏差）

自然的问题：能否组合不同 n 的估计，取得更好的权衡？答案是 **GAE**——通过对所有 n-step advantage 进行指数加权平均，用一个参数 λ 灵活控制偏差-方差的平衡点。

3.6.2 GAE 的定义与推导

定义 3.2 (Generalized Advantage Estimation)**.**

$$\hat{A}_t^{GAE(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l} \tag{3.47}$$

其中 $\delta_t = r_t + \gamma \hat{V}(s_{t+1}) - \hat{V}(s_t)$ 是 *TD* 残差，$\lambda \in [0, 1]$ 是衰减参数。

定理 3.7 (GAE 等价于 n-step Advantage 的加权和)**.**

$$\hat{A}_t^{GAE} = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}\hat{A}_t^{(n)} \tag{3.48}$$

证明. 首先，注意到 n-step advantage 可以写成 TD 残差的和：

$$\hat{A}_t^{(n)} = \sum_{k=0}^{n-1}\gamma^k\delta_{t+k} \tag{3.49}$$

这可以通过展开验证：

$$\sum_{k=0}^{n-1}\gamma^k\delta_{t+k} = \sum_{k=0}^{n-1}\gamma^k\left(r_{t+k}+\gamma\hat{V}(s_{t+k+1})-\hat{V}(s_{t+k})\right) \tag{3.50}$$

$$= \sum_{k=0}^{n-1}\gamma^k r_{t+k} + \sum_{k=0}^{n-1}\gamma^{k+1}\hat{V}(s_{t+k+1}) - \sum_{k=0}^{n-1}\gamma^k\hat{V}(s_{t+k}) \tag{3.51}$$

后两项是 telescoping sum：

$$= \sum_{k=0}^{n-1}\gamma^k r_{t+k} + \gamma^n\hat{V}(s_{t+n}) - \hat{V}(s_t) \tag{3.52}$$

$$= \hat{A}_t^{(n)} \tag{3.53}$$

现在计算 GAE 的加权和形式：

$$(1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}\hat{A}_t^{(n)} = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}\sum_{k=0}^{n-1}\gamma^k\delta_{t+k} \tag{3.54}$$

交换求和顺序。对于固定的 k，它出现在 $n > k$ 的所有项中：

$$= (1-\lambda)\sum_{k=0}^{\infty}\gamma^k\delta_{t+k}\sum_{n=k+1}^{\infty}\lambda^{n-1} \tag{3.55}$$

$$= (1-\lambda)\sum_{k=0}^{\infty}\gamma^k\delta_{t+k}\cdot\frac{\lambda^k}{1-\lambda} \tag{3.56}$$

$$= \sum_{k=0}^{\infty}(\gamma\lambda)^k\delta_{t+k} \tag{3.57}$$

$$= \hat{A}_t^{\text{GAE}} \tag{3.58}$$

□

3.6.3　λ 参数的偏差-方差权衡

λ 值	等价形式	偏差	方差
$\lambda = 0$	δ_t（TD）	高（依赖 $\hat{V}$）	低
$\lambda = 1$	$G_t - \hat{V}(s_t)$（MC）	低	高
$\lambda \in (0,1)$	加权平均	中等	中等

表 3.1: GAE 参数 λ 的效果

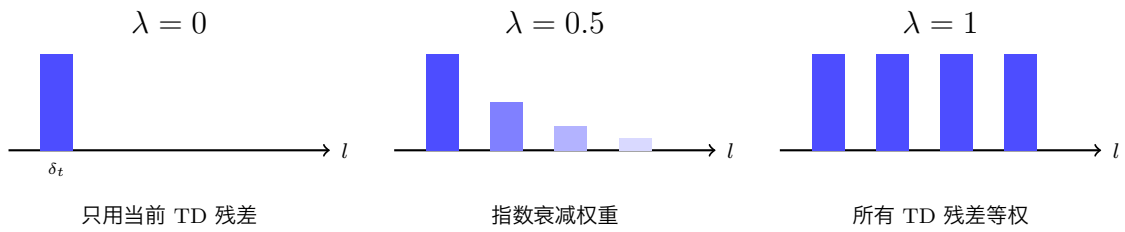


图 3.7: GAE 中 $(\gamma\lambda)^l$ 权重随 l 的变化。λ 越小，越依赖近期的 TD 残差。

实践中，$\lambda = 0.95$ 或 $\lambda = 0.97$ 是常用的选择。

关键点

GAE 的直觉理解：

- $\delta_t = r_t + \gamma\hat{V}(s_{t+1}) - \hat{V}(s_t)$ 是“一步后用 Critic 估计剩余价值”的 advantage
- GAE 把多步 δ 加权求和，$(\gamma\lambda)^l$ 让远处的 δ 权重指数衰减
- λ 越小，越依赖 Critic 估计（偏差大但方差小）
- λ 越大，越依赖实际回报（偏差小但方差大）

GAE 与第二章的 TD(λ) 有类似的思想，都是通过 λ 在 TD 和 MC 之间权衡。

3.6.4　GAE 的实际计算

GAE 可以通过递推高效计算：

$$\hat{A}_t^{\text{GAE}} = \delta_t + \gamma\lambda\hat{A}_{t+1}^{\text{GAE}} \tag{3.59}$$

边界条件：$\hat{A}_T^{\text{GAE}} = 0$（episode 结束后）。
从后往前计算，复杂度为 $O(T)$。

3.7　重要性采样与 Off-Policy Policy Gradient

3.7.1　On-Policy 的问题

Policy Gradient 是 on-policy 的：每次更新 θ 后，旧数据的分布 $p(\tau|\theta_{\text{old}})$ 就与新策略 $p(\tau|\theta)$ 不同了。这导致：

- 数据只能用一次，样本效率低
- 每次更新都需要重新采样

重要性采样（Importance Sampling, IS）允许我们复用旧数据。

3.7.2　重要性采样原理

定义 3.3 (重要性采样). 用分布 $q(x)$ 的样本估计 $p(x)$ 下的期望：

$$\mathbb{E}_{x\sim p}[f(x)] = \mathbb{E}_{x\sim q}\left[\frac{p(x)}{q(x)}f(x)\right] \tag{3.60}$$

其中 $\rho(x) = \frac{p(x)}{q(x)}$ 称为**重要性权重**（*Importance Weight*）。

证明.

$$\mathbb{E}_{x\sim q}\left[\frac{p(x)}{q(x)}f(x)\right]=\int q(x)\cdot\frac{p(x)}{q(x)}f(x)dx=\int p(x)f(x)dx=\mathbb{E}_{x\sim p}[f(x)] \tag{3.61}$$

□

3.7.3　应用到 Policy Gradient

用旧策略 π_{old} 的样本估计新策略 π_θ 下的期望:

$$J(\theta)=\mathbb{E}_{\tau\sim\pi_{\text{old}}}\left[\frac{p(\tau|\theta)}{p(\tau|\theta_{\text{old}})}R(\tau)\right] \tag{3.62}$$

轨迹的重要性权重:

$$\frac{p(\tau|\theta)}{p(\tau|\theta_{\text{old}})}=\frac{\cancel{p(s_0)}\prod_t\pi_\theta(a_t|s_t)\cancel{P(s_{t+1}|s_t,a_t)}}{\cancel{p(s_0)}\prod_t\pi_{\text{old}}(a_t|s_t)\cancel{P(s_{t+1}|s_t,a_t)}} \tag{3.63}$$

$$=\prod_{t=0}^{T-1}\frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)} \tag{3.64}$$

环境动力学 P 和初始分布 $p(s_0)$ 消掉了！这再次体现了 Policy Gradient 的 model-free 特性。

对于单步 advantage 估计，重要性权重简化为:

$$\rho_t(\theta)=\frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)} \tag{3.65}$$

定理 3.8 (Off-policy Policy Gradient)**.**

$$\nabla_\theta J(\theta)=\mathbb{E}_{(s,a)\sim\pi_{old}}\left[\rho_t(\theta)\nabla_\theta\log\pi_\theta(a_t|s_t)\hat{A}_t\right] \tag{3.66}$$

这使得可以用旧数据多次更新策略!

3.7.4　严格的 Off-Policy 梯度与状态分布修正

上面的公式省略了一个重要细节。**严格的 off-policy policy gradient** 不仅需要修正动作概率，还需要修正**状态分布**:

$$\nabla_\theta J(\theta)=\mathbb{E}_{s\sim d_{\pi_{\text{old}}}}\left[\frac{d_{\pi_\theta}(s)}{d_{\pi_{\text{old}}}(s)}\cdot\mathbb{E}_{a\sim\pi_{\text{old}}(\cdot|s)}\left[\rho_t(\theta)\nabla_\theta\log\pi_\theta(a|s)\hat{A}(s,a)\right]\right] \tag{3.67}$$

其中 $d_\pi(s)$ 是策略 π 诱导的状态分布（也称为 discounted state visitation distribution）。

注意

为什么需要状态分布修正?

直观理解：用旧策略 π_{old} 采样时，不仅动作的分布变了，连访问到的状态分布也变了。例如:

- 新策略可能更倾向于进入某些状态
- 旧数据中这些状态的样本可能较少

因此严格的 off-policy 梯度需要同时修正这两个偏差。

PPO/TRPO 的隐式近似

然而，计算状态分布比 $\frac{d_{\pi_\theta}(s)}{d_{\pi_{\text{old}}}(s)}$ 非常困难——它依赖于整条轨迹的累积效应，无法像动作概率那样直接计算。

PPO/TRPO 的 **surrogate objective** 实际上做了一个关键近似：

$$\frac{d_{\pi_\theta}(s)}{d_{\pi_{\text{old}}}(s)} \approx 1 \tag{3.68}$$

即假设新旧策略诱导的状态分布相同。

重要

Trust Region 的作用：当 π_θ 与 π_{old} 足够接近时（KL 散度小），状态分布的差异也会很小。TRPO 的 KL 约束和 PPO 的 clip 机制正是为了保证这一点。

总结：PPO/TRPO 使用的 surrogate objective

$$L(\theta) = \mathbb{E}_{(s,a)\sim\pi_{\text{old}}}\left[\rho_t(\theta)\hat{A}_t\right] \tag{3.69}$$

实际上隐式地做了两件事：

1. 用 $\rho_t = \frac{\pi_\theta(a|s)}{\pi_{\text{old}}(a|s)}$ 修正动作概率偏差
2. 假设 $\frac{d_{\pi_\theta}(s)}{d_{\pi_{\text{old}}}(s)} \approx 1$，忽略状态分布偏差

这解释了为什么 PPO/TRPO 能直接使用 token-level 的 ρ_t 而不需要额外的状态分布修正——**前提是 trust region 约束成立**。

3.7.5　方差问题

重要性采样的问题：当 ρ_t 偏离 1 太多时，方差会急剧增大。

- 如果 π_θ 和 π_{old} 差异大，ρ_t 可能非常大或非常小
- 大的 ρ_t 导致梯度估计方差爆炸
- 需要限制策略更新幅度，保持 $\rho_t \approx 1$

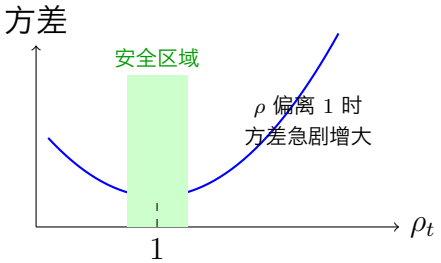


图 3.8: 重要性权重 ρ_t 偏离 1 时，方差增大。需要限制策略更新幅度。

3.8　Trust Region 方法：TRPO 与 PPO

3.8.1　动机：限制策略更新幅度

重要性采样允许复用旧数据，但如果 π_θ 与 π_{old} 差异太大，估计就不可靠。Trust Region 方法通过限制策略更新幅度来解决这个问题。

3.8.2　TRPO：KL 约束优化

TRPO（Trust Region Policy Optimization）通过 KL 散度约束限制策略更新：

定义 3.4 (TRPO 优化问题).

$$\max_{\theta} \quad L(\theta) = \mathbb{E}_{(s,a)\sim\pi_{old}}\left[\rho_t(\theta)\hat{A}_t\right] \tag{3.70}$$

$$s.t. \quad \bar{D}_{KL}(\pi_{old}\|\pi_\theta) \leq \delta \tag{3.71}$$

其中 $\bar{D}_{KL}$ 是状态分布上的平均 KL 散度。

TRPO 的特点：

- 理论上保证单调改进（在约束满足时）
- 需要计算 KL 散度的 Hessian（二阶优化，Conjugate Gradient）
- 实现复杂，计算开销大

3.8.3　PPO：简化的 Trust Region

PPO（Proximal Policy Optimization）通过更简单的方式近似 TRPO 的效果。

3.8.3.1　PPO-Clip

定义 3.5 (PPO-Clip 目标).

$$L^{CLIP}(\theta) = \mathbb{E}\left[\min\left(\rho_t\hat{A}_t,\ clip(\rho_t, 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right] \tag{3.72}$$

其中 $clip(x, a, b) = \max(a, \min(x, b))$，$\epsilon$ 通常取 0.1 或 0.2。

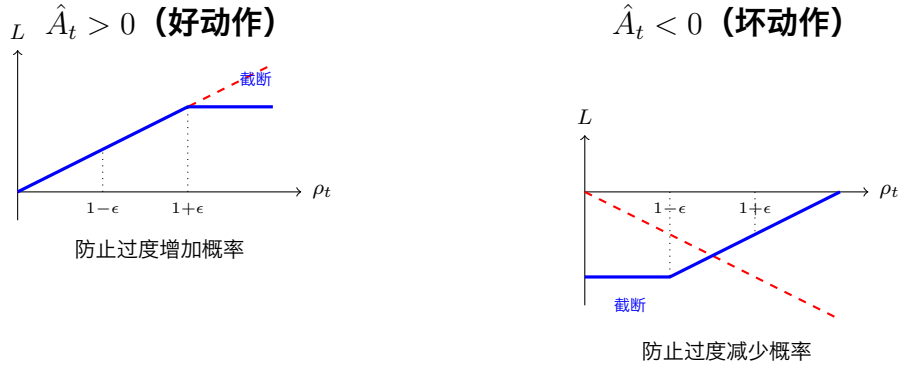


图 3.9: PPO-Clip 机制。当 ρ_t 偏离 $[1-\epsilon, 1+\epsilon]$ 时，目标被截断，防止策略变化过大。

定理 3.9 (PPO-Clip 的直觉).

- 当 $\hat{A}_t > 0$（好动作）：
  - 目标是增加 $\pi_\theta(a_t|s_t)$，即让 $\rho_t > 1$
  - 但当 $\rho_t > 1+\epsilon$ 时截断，防止过度增加
- 当 $\hat{A}_t < 0$（坏动作）：
  - 目标是减少 $\pi_\theta(a_t|s_t)$，即让 $\rho_t < 1$
  - 但当 $\rho_t < 1-\epsilon$ 时截断，防止过度减少

3.8.3.2　PPO-KL（可选）

PPO 的另一个变体使用 KL 惩罚项：

定义 3.6 (PPO-KL 目标)**.**

$$L^{KL}(\theta) = \mathbb{E}\left[\rho_t \hat{A}_t\right] - \beta \cdot KL(\pi_{old} \| \pi_\theta) \tag{3.73}$$

其中 β 是自适应调整的系数：

- 如果 KL 太大，增大 β
- 如果 KL 太小，减小 β

3.8.4　Entropy Bonus

为了鼓励探索，PPO 通常还会加入 entropy bonus：

$$L^{\text{total}}(\theta) = L^{\text{CLIP}}(\theta) + c_1 \cdot H(\pi_\theta) \tag{3.74}$$

其中 $H(\pi_\theta) = -\mathbb{E}[\log \pi_\theta(a|s)]$ 是策略的熵，c_1 是权重系数（通常 0.01）。

注意

Entropy bonus 的作用：

- 鼓励策略保持一定的随机性，避免过早收敛到确定性策略
- 促进探索，防止陷入局部最优
- 熵越大，策略越“均匀”，对各动作的概率分布越平坦

3.8.5　PPO 完整算法

算法 9: Proximal Policy Optimization (PPO)

输入: 初始参数 θ_0, ϕ_0，clip 参数 ϵ，每轮更新次数 K，GAE 参数 λ

for *iteration* $= 1, 2, \ldots$ **do**

- **数据收集**：用当前策略 π_θ 收集 N 条轨迹;
- 计算 GAE advantage $\hat{A}_t$（使用 $\hat{V}_\phi$）;
- 计算 return $\hat{R}_t = \hat{A}_t + \hat{V}_\phi(s_t)$（作为 Critic target）;
- 记录 $\pi_{\text{old}} = \pi_\theta$（固定，用于计算 ρ_t）;
- **策略更新**：**for** $k = 1, \ldots, K$ **do**
  - 对 mini-batch 数据计算：
    - $\rho_t = \frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)}$
    - $L^{\text{CLIP}} = \mathbb{E}[\min(\rho_t \hat{A}_t, \text{clip}(\rho_t, 1-\epsilon, 1+\epsilon)\hat{A}_t)]$
    - $L^{\text{Value}} = \mathbb{E}[(\hat{V}_\phi(s_t) - \hat{R}_t)^2]$
    - $L^{\text{Entropy}} = -\mathbb{E}[\log \pi_\theta(a_t|s_t)]$
  - 总目标：$L = L^{\text{CLIP}} - c_1 L^{\text{Value}} + c_2 L^{\text{Entropy}}$;
  - 梯度上升更新 θ，梯度下降更新 ϕ;

关键点

PPO 的成功原因：

1. **简单高效**：只需一阶优化，不需要计算 Hessian
2. **样本效率**：可多次复用同一批数据（K 次更新）
3. **稳定性**：clip 机制防止策略剧烈变化，避免“走太远”
4. **鲁棒性**：对超参数不敏感，适用于多种任务

PPO 是目前最常用的 Policy Gradient 算法，也是 RLHF（第五章）中的标准选择。

3.9　本章小结

关键点

本章核心内容：

1. **Policy Gradient 定理**
   - 给出了目标函数梯度的解析形式：$\nabla_\theta J = \mathbb{E}[\sum_t \nabla \log \pi \cdot G_t]$
   - Log-Derivative Trick 是推导的关键
   - 环境动力学与 θ 无关，实现了 Model-Free
2. **方差降低技术**
   - Baseline 技巧：减去 $b(s)$ 不改变期望但降低方差
   - 最优 baseline 是 $V^\pi(s)$
   - 使用 Advantage $A = Q - V$ 替代 G_t
3. **Actor-Critic 架构**
   - Actor（策略网络）+ Critic（价值网络）
   - Critic 提供 $\hat{V}(s)$ 来估计 advantage
4. **GAE**
   - $\hat{A}^{\text{GAE}} = \sum_l (\gamma\lambda)^l \delta_{t+l}$
   - λ 控制偏差-方差权衡（类似 TD(λ)）
5. **Trust Region 方法**
   - 重要性采样允许复用旧数据，但需要限制策略变化
   - TRPO：KL 约束优化，实现复杂
   - PPO：clip 机制，简单高效，是实践中的首选

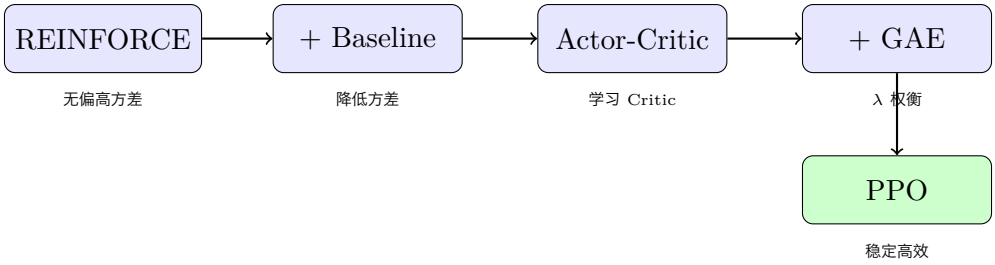


图 3.10: Policy Gradient 方法的演进：从 REINFORCE 到 PPO

注意

Policy-Based vs. Value-Based 的选择：

- **连续动作空间**：优先选择 Policy-Based（如 PPO、SAC）
- **离散动作空间**：两者都可以，DQN 可能更样本高效
- **需要随机策略**：使用 Policy-Based
- **LLM 对齐**：PPO 是 RLHF 的标准选择（第五章详述）

Chapter 4

基于模型的方法与多智能体学习

4.1 引言：样本效率的追求

4.1.1 核心问题

前三章介绍的 Model-Free 方法（Q-Learning、Policy Gradient、PPO）虽然强大，但有一个共同的缺陷：

> **样本效率极低**——训练一个 Atari 游戏 agent 需要数亿帧画面，相当于人类玩数百小时。而人类通常只需几分钟就能学会基本操作。为什么会有如此巨大的差距？

关键区别在于：**人类在脑中有一个世界模型。**当我们想象“如果我这样做会发生什么”时，我们在用这个模型进行**心智模拟**（mental simulation），而不需要真正尝试。

Model-Based RL 的核心思想正是：**学习或利用环境模型，通过规划（Planning）来提高样本效率。**

4.1.2 Model-Free vs Model-Based

根据是否使用环境模型，RL 方法分为两大类：

定义 4.1 (Model-Free 与 Model-Based)**.**

- ***Model-Free***：不学习或使用环境模型，直接从真实经验中学习价值函数或策略
- ***Model-Based***：学习或利用环境模型 $\hat{P}(s'|s,a)$, $\hat{R}(s,a)$，在模型中进行规划

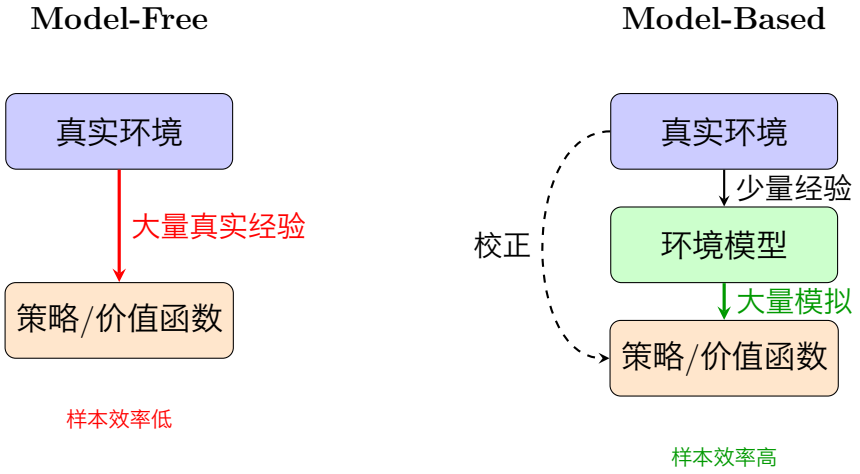


图 4.1: Model-Free vs Model-Based：后者用模型生成模拟经验，减少真实交互需求

	Model-Free	Model-Based
环境模型	不需要	需要（已知或学习）
样本效率	低	高
计算开销	低	高（规划）
模型误差	无	可能累积
适用场景	模型难以获取	模型已知或易学
典型算法	Q-Learning, PPO	Dyna, MCTS, MuZero

表 4.1: Model-Free 与 Model-Based 的对比

4.1.3 本章路线图

本章将介绍两个重要方向：

1. **Model-Based RL**（第4.2节-第4.5节）：如何利用环境模型进行规划
   - World Model 的定义与学习
   - Dyna 架构：结合直接学习与规划
   - MCTS：Decision-time Planning 的代表
   - AlphaGo/Zero：MCTS + 深度学习的里程碑
2. **Multi-Agent RL**（第4.6节-第4.7节）：多个 agent 的交互与博弈
   - 博弈论基础：Normal-form Game
   - Nash 均衡：稳定的策略组合
   - Self-Play：训练博弈 AI 的强大方法

AlphaGo/AlphaZero 是两者的完美结合：用 MCTS 进行规划，用 Self-Play 进行训练。

4.2 Model-Based RL 概述

4.2.1 World Model 的定义

定义 4.2 (World Model)**.** *World Model* 是对环境动力学的估计，包括：

- **状态转移模型**：$\hat{P}(s'|s,a) \approx P(s'|s,a)$
- **奖励模型**：$\hat{R}(s,a) \approx R(s,a)$

有了 *World Model*，*agent* 可以在“脑中”模拟动作的后果，而不需要真正执行。

World Model 的来源有两种：

1. **已知规则**：如棋类游戏的规则、物理引擎的方程
   - 优点：模型精确，无误差
   - 缺点：仅适用于规则完全已知的领域
2. **从数据学习**：用神经网络从交互经验中学习
   - 优点：适用于复杂环境
   - 缺点：模型存在误差

4.2.2　学习 World Model

学习 World Model 本质上是一个监督学习问题。给定经验数据 $\{(s_t, a_t, r_t, s_{t+1})\}$：

1. **确定性模型**：直接预测下一状态

$$\hat{s}_{t+1} = f_\theta(s_t, a_t), \quad L = \|s_{t+1} - \hat{s}_{t+1}\|^2$$

2. **概率模型**：预测状态分布

$$\hat{P}_\theta(s'|s, a), \quad L = -\log \hat{P}_\theta(s_{t+1}|s_t, a_t)$$

3. **隐空间模型**：在低维隐空间中预测

$$z_{t+1} = f_\theta(z_t, a_t), \quad z_t = \text{Encoder}(s_t)$$

注意

现代 World Model 方法（如 Dreamer、MuZero）通常在隐空间中进行预测，避免直接预测高维原始观测（如图像），大大降低了学习难度。

4.2.3　Model Bias 问题

定义 4.3 (Model Bias / Model Error)**.** 当学习的模型 $\hat{P}, \hat{R}$ 与真实环境 P, R 存在差异时，在模型中规划得到的策略在真实环境中可能表现不佳，这称为 ***Model Bias***。

Model Bias 的关键问题是**误差累积**（Error Compounding）：

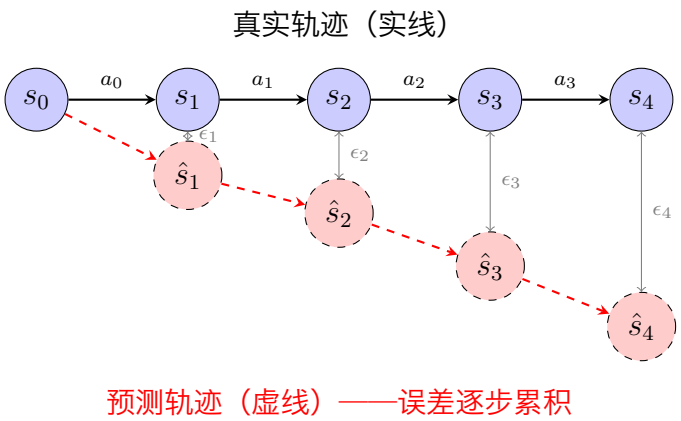


图 4.2: Model Error Compounding：每一步的预测误差会累积，导致长期预测严重偏离

定理 4.1 (误差累积上界)**.** 设单步模型误差为 $\epsilon = \max_{s,a} \|\hat{P}(\cdot|s,a) - P(\cdot|s,a)\|_1$，则 H 步规划的总变差距离上界为：

$$TV(\hat{P}^H, P^H) \leq H \cdot \epsilon$$

即误差随规划步数**线性累积**。

缓解 Model Bias 的策略：

1. **短期规划**：只用模型做短期预测（如 Dyna 中的 1-step）
2. **集成模型**：训练多个模型，用不确定性指导探索
3. **持续校正**：用真实数据不断更新模型
4. **隐空间规划**：在抽象空间中规划（如 MuZero）

4.3　Planning 方法

有了环境模型，下一步是利用模型进行**规划**（Planning）。根据规划时机，分为两类：

定义 4.4 (Background Planning vs Decision-time Planning)**.**

- ***Background Planning***：在与真实环境交互之外，利用模型生成模拟经验来训练策略略
- ***Decision-time Planning***：在需要做决策时，利用模型进行前向搜索，选择最优动作

图 4.3: 两种规划方式的对比

4.3.1　Dyna 架构

Dyna 是 Background Planning 的经典框架，由 Sutton 于 1991 年提出。其核心思想是：**每次真实交互后，用模型生成多次模拟经验来加速学习。**

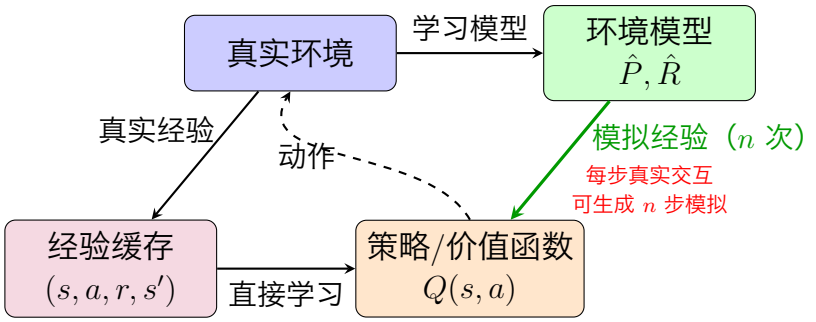


图 4.4: Dyna 架构：结合直接学习（从真实经验）与规划（从模拟经验）

算法 10: Dyna-Q

输入: 规划步数 n，学习率 α，探索率 ϵ

初始化 $Q(s,a) \leftarrow 0$，表格模型 $\text{Model}(s,a) \leftarrow \emptyset$;
foreach *episode* **do**
  初始化状态 s;
  while s 不是终止状态 **do**
    $a \leftarrow \epsilon\text{-greedy}(Q(s,\cdot))$;
    执行 a，观察 r, s';
    // 直接 RL 学习
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$;
    // 更新模型
    $\text{Model}(s,a) \leftarrow (r,s')$ /* 确定性模型 */
    // 规划：从模型中学习
    for $i = 1$ **to** n **do**
      随机选择之前访问过的状态-动作对 $(\tilde{s},\tilde{a})$;
      $(\tilde{r},\tilde{s}') \leftarrow \text{Model}(\tilde{s},\tilde{a})$;
      $Q(\tilde{s},\tilde{a}) \leftarrow Q(\tilde{s},\tilde{a}) + \alpha[\tilde{r} + \gamma \max_{a'} Q(\tilde{s}',a') - Q(\tilde{s},\tilde{a})]$;
    $s \leftarrow s'$;

关键点

Dyna 的核心优势：

1. **样本效率提升**：每次真实交互可产生 n 次模拟学习
2. **灵活的计算-样本权衡**：增大 n 可用更多计算换取更少真实交互
3. **渐进收敛**：当模型准确时，理论上与直接学习收敛到相同策略

4.3.2 Decision-time Planning

与 Background Planning 不同，Decision-time Planning 在每次决策时进行规划：

1. 从当前状态出发，用模型模拟多条可能的轨迹
2. 评估每条轨迹的回报
3. 选择最优的第一步动作
4. 执行后重新规划（不保存中间结果）

Decision-time Planning 的特点：

- **计算集中**：所有计算都为当前决策服务
- **动态精度**：可根据需要调整搜索深度和广度
- **无需训练**：可直接用于测试时

最著名的 Decision-time Planning 方法是 **Monte Carlo Tree Search (MCTS)**。

4.4　Monte Carlo Tree Search (MCTS)

MCTS 是一种基于树搜索的 Decision-time Planning 方法，广泛应用于棋类游戏和组合优化问题。

4.4.1　MCTS 的核心思想

MCTS 的目标是在有限的计算预算内，估计当前状态下各动作的价值。其核心思想是：**有选择性地扩展搜索树，把计算资源集中在最有希望的分支上。**

> 如何决定哪个分支“最有希望”？这需要平衡**利用**（选择已知好的分支）和**探索**（尝试不确定的分支）。

4.4.2　MCTS 四步流程

MCTS 的每次迭代包含四个步骤：

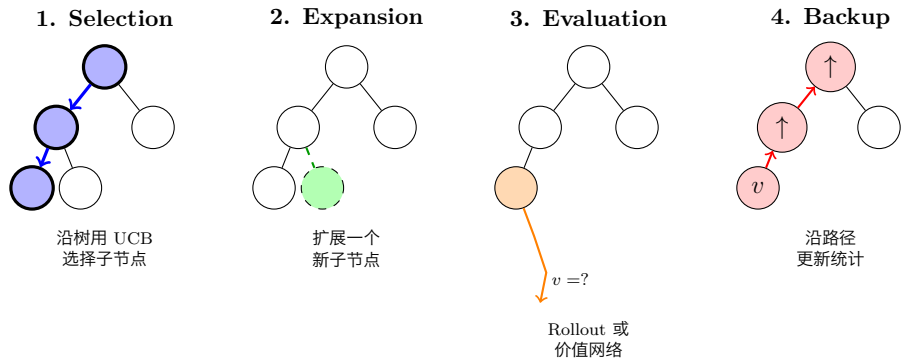


图 4.5: MCTS 的四个步骤

1. **Selection（选择）**：从根节点开始，使用**树策略**（如 UCB）递归选择子节点，直到到达叶节点（未完全扩展的节点）。
2. **Expansion（扩展）**：如果叶节点不是终止状态，根据可行动作扩展一个或多个新的子节点。
3. **Evaluation（评估）**：评估新扩展节点的价值。传统方法使用 **rollout**（随机模拟到终局）；现代方法使用**价值网络**直接估计。
4. **Backup（回溯）**：将评估值沿选择路径回传，更新路径上所有节点的访问次数 N 和价值估计 Q。

4.4.3　UCB 公式

Selection 阶段的核心是 **UCB（Upper Confidence Bound）**公式，它优雅地平衡了利用与探索：

定理 4.2 (UCB for Trees (UCT))**.** 在 *Selection* 阶段，选择最大化以下值的动作：

$$UCB(s,a) = \underbrace{Q(s,a)}_{\text{利用}} + \underbrace{c\sqrt{\frac{\ln N(s)}{N(s,a)}}}_{\text{探索}} \tag{4.1}$$

其中：

- $Q(s,a)$：动作 a 的平均价值估计（从历史模拟中统计）
- $N(s)$：状态 s 的总访问次数
- $N(s,a)$：在状态 s 执行动作 a 的次数
- c：探索系数，控制探索-利用权衡

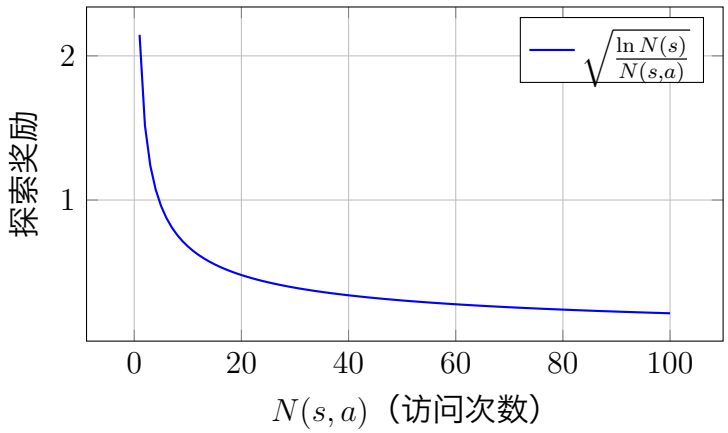


图 4.6: UCB 探索项随访问次数的变化

注意

UCB 的直觉理解：

- **利用项** $Q(s,a)$：选择历史表现好的动作
- **探索项**：选择访问次数少的动作（不确定性高）
- 随着访问次数增加，探索奖励逐渐减小，最终由利用主导
- c 越大，越倾向探索；c 越小，越倾向利用

4.4.4　MCTS 算法

算法 11: Monte Carlo Tree Search

输入: 当前状态 s_0，搜索预算 B（迭代次数），探索系数 c

输出: 最优动作 a^*

初始化根节点 root $= s_0$，$N(\text{root}) = 0$;

for $i = 1$ **to** B **do**

  // Selection

  node $\leftarrow$ root;

  while *node* 已完全扩展且不是终止状态 **do**

    $a \leftarrow \text{argmax}_a \, \text{UCB}(\text{node}, a)$;

    node $\leftarrow \text{child}(\text{node}, a)$;

  // Expansion

  if *node* 不是终止状态 **then**

    选择一个未扩展的动作 a;

    node $\leftarrow$ 扩展子节点 $\text{child}(\text{node}, a)$;

  // Evaluation

  $v \leftarrow \text{Evaluate}(\text{node})$ /* Rollout 或价值网络 */

  // Backup

  while *node* $\neq$ *null* **do**

    $N(\text{node}) \leftarrow N(\text{node}) + 1$;

    $Q(\text{node}) \leftarrow Q(\text{node}) + \frac{v - Q(\text{node})}{N(\text{node})}$;

    node $\leftarrow \text{parent}(\text{node})$;

return $\text{argmax}_a \, N(root, a)$ /* 选择访问次数最多的动作 */

重要

MCTS 最终选择动作的标准：

- 训练/搜索时：用 UCB（平衡探索-利用）
- 最终决策时：选择**访问次数最多**的动作（更稳健）

访问次数而非平均价值，因为高访问次数意味着高置信度。

4.5　AlphaGo 与 AlphaZero

AlphaGo 和 AlphaZero 是 MCTS + 深度学习 + Self-Play 的里程碑式成果。

4.5.1　围棋的挑战

围棋被认为是 AI 最难攻克的棋类游戏：

- **搜索空间巨大**：平均每步有 ~ 200 种合法走法，一局棋约 200 步，总状态数 $\sim 10^{170}$
- **局面评估困难**：不像国际象棋有明确的子力价值，围棋的局面优劣难以量化
- **长期规划**：需要考虑数十步后的战略影响

传统围棋 AI 使用穷举搜索 + 手工评估函数，水平仅达业余段位。

4.5.2　AlphaGo 架构（2016）

AlphaGo 在 2016 年以 4:1 击败世界冠军李世石，其架构包括：

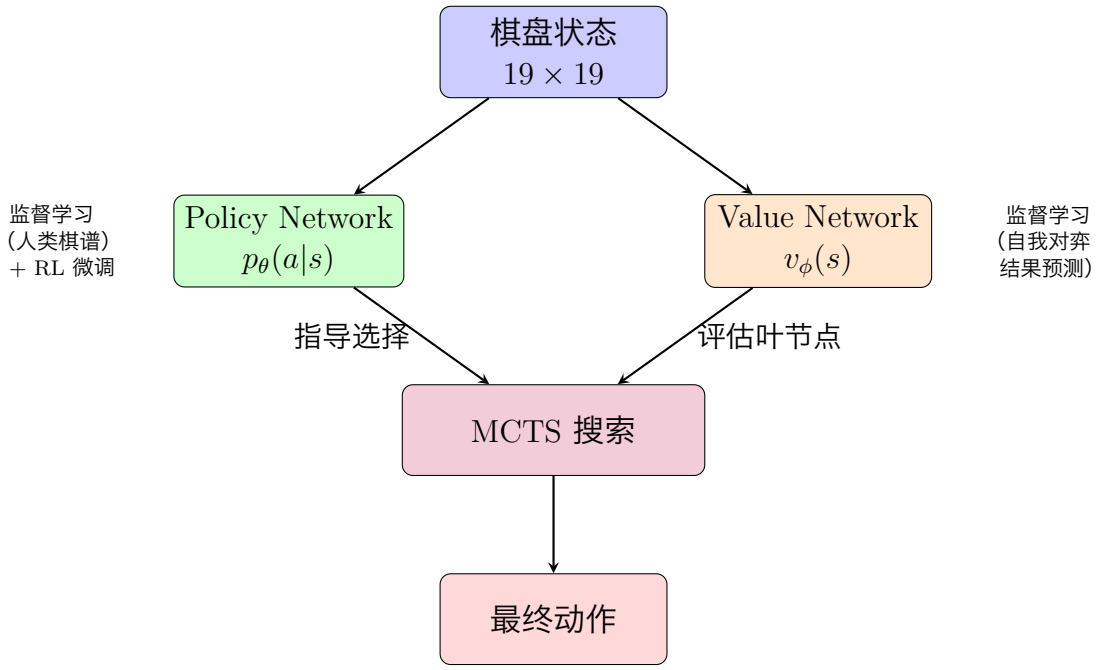


图 4.7: AlphaGo 架构

1. **Policy Network** $p_\theta(a|s)$：
   - 输入：棋盘状态（多通道特征）
   - 输出：每个位置的落子概率
   - 训练：先用人类棋谱监督学习，再用 Policy Gradient 自我对弈强化
2. **Value Network** $v_\phi(s)$：
   - 输入：棋盘状态
   - 输出：当前局面的胜率估计 $v_\phi(s) \approx \mathbb{E}[z|s]$，其中 $z \in \{-1, +1\}$
   - 训练：用自我对弈生成的 (s, z) 数据监督学习
3. **改进的 MCTS**：
   - Selection：使用 Policy Network 指导（PUCT 公式）

$$\text{UCB}(s,a) = Q(s,a) + c \cdot p_\theta(a|s) \cdot \frac{\sqrt{N(s)}}{1 + N(s,a)}$$

   - Evaluation：混合 Value Network 和 rollout

$$v = (1-\lambda) v_\phi(s) + \lambda z_{\text{rollout}}$$

4.5.3　AlphaZero 的简化与超越（2017）

AlphaZero 在 2017 年大幅简化了 AlphaGo 的设计，却取得了更强的性能：

	AlphaGo	AlphaZero
人类棋谱	需要（监督预训练）	**不需要**
网络结构	Policy + Value 分离	**统一网络**
Rollout	需要	**不需要**
特征工程	手工设计特征	**原始棋盘输入**
训练时间	数月	**数小时**
适用游戏	仅围棋	**围棋、国际象棋、将棋**

表 4.2: AlphaGo 与 AlphaZero 的对比

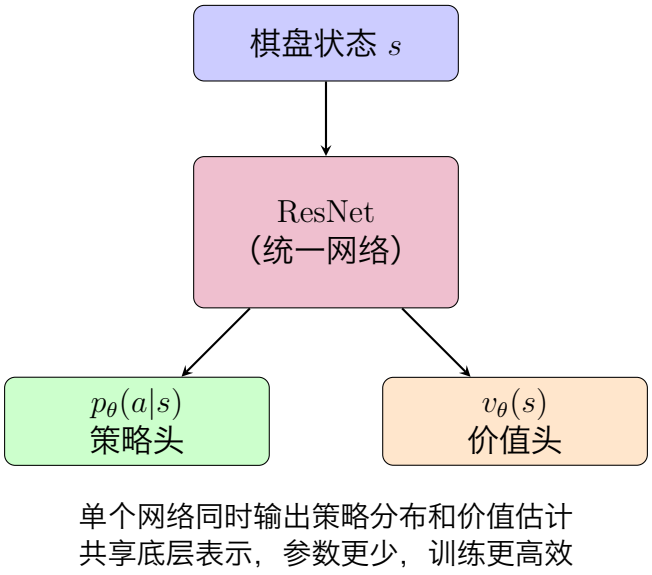


图 4.8: AlphaZero 的统一网络架构

4.5.4　AlphaZero 训练循环

AlphaZero 的训练是一个**自我增强**的循环：

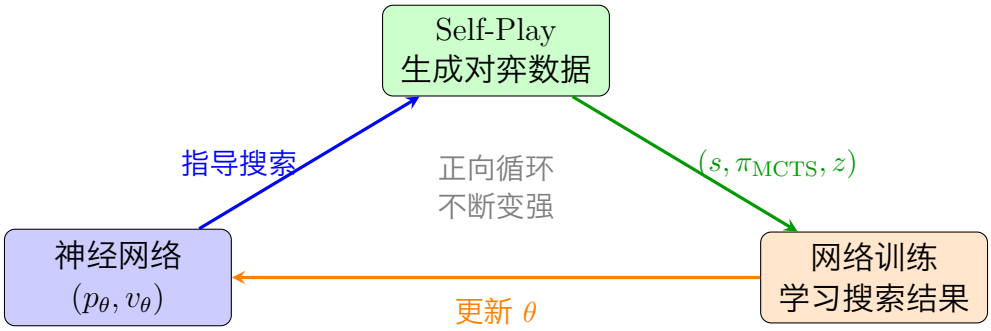


图 4.9: AlphaZero 的自我增强训练循环

算法 12: AlphaZero 训练

初始化网络参数 θ（随机初始化）;
repeat
  // Self-Play 生成数据
  for 每局对弈 **do**
    for 每步 t **do**
      用当前网络 + MCTS 搜索，得到 $\pi_{\text{MCTS}}(a|s_t)$;
      按 π_{MCTS} 采样动作 a_t;
      记录 (s_t, π_{MCTS});
    游戏结束，得到胜负 $z \in \{-1, +1\}$;
    将 $(s_t, \pi_{\text{MCTS}}, z)$ 加入训练数据;
  // 网络训练
  从训练数据中采样 batch;
  最小化损失:

$$L(\theta) = \underbrace{(z - v_\theta(s))^2}_{\text{价值损失}} - \underbrace{\pi_{\text{MCTS}}^\top \log p_\theta(s)}_{\text{策略损失}} + \underbrace{c\|\theta\|^2}_{\text{正则化}}$$

until 收敛;

关键点

AlphaZero 的核心洞察:

1. **MCTS 作为策略改进**：搜索产生的 π_{MCTS} 比原始网络 p_θ 更好
2. **网络学习搜索**：网络被训练去模仿 MCTS 的输出
3. **正向循环**：更好的网络 $\rightarrow$ 更好的搜索 $\rightarrow$ 更好的训练数据 $\rightarrow$ 更好的网络

这个循环不需要任何人类知识，完全从零开始（tabula rasa）学习。

4.6 Multi-Agent RL 基础

当环境中存在多个 agent 时，问题变得更加复杂。本节介绍 Multi-Agent RL 的基础概念。

4.6.1 从单 Agent 到多 Agent

> **核心问题**：当其他 agent 也在学习和改变策略时，环境对单个 agent 来说是**非稳态**的。这打破了 MDP 的基本假设。

定义 4.5 (Multi-Agent 环境的非稳态性). 在 *Multi-Agent* 环境中，状态转移和奖励不仅依赖于自己的动作，还依赖于其他 *agent* 的动作:

$$P(s'|s, a_1, a_2, \ldots, a_n), \quad R_i(s, a_1, a_2, \ldots, a_n)$$

当其他 *agent* 的策略 π_{-i} 在学习过程中变化时，从 *agent* i 的视角看，环境是非稳态的。

非稳态性的影响:

- 单 agent RL 的收敛性保证不再适用
- 最优响应策略随对手策略而变化
- 可能出现策略震荡，无法收敛

4.6.2 博弈论基础

Multi-Agent 问题可以用博弈论的语言来描述。

定义 4.6 (Normal-form Game)**.** 一个 n 人博弈由以下元素组成：

- **玩家集合**：$\mathcal{N} = \{1, 2, \ldots, n\}$
- **策略空间**：每个玩家 i 的策略集合 $\mathcal{A}_i$
- **效用函数**：$u_i : \mathcal{A}_1 \times \cdots \times \mathcal{A}_n \to \mathbb{R}$，表示每种策略组合下玩家 i 的收益

例 4.1 (囚徒困境)**.** 两个嫌犯被分开审讯，各自选择“合作”（沉默）或“背叛”（坦白）：

		玩家 *2* 合作	玩家 *2* 背叛
玩家 *1*	合作	$(-1, -1)$	$(-3, 0)$
	背叛	$(0, -3)$	$(-2, -2)$

分析：

- 无论对方如何选择，“背叛”对自己都更有利（支配策略）
- 结果：双方都背叛，各获 -2
- 但如果双方都合作，各获 -1（帕累托更优）

4.6.3 合作与竞争设定

Multi-Agent 场景主要分为两类：

1. **合作（Cooperative）**：所有 agent 共享奖励，最大化团队总收益
   - 例：多机器人协作搬运、多 agent 协调导航
   - 挑战：信用分配（Credit Assignment）——哪个 agent 贡献了多少？
   - 方法：集中训练、分布式执行（CTDE）
2. **竞争/零和（Competitive/Zero-sum）**：一方获益等于另一方损失
   - 例：棋类游戏、对抗博弈
   - 特点：$u_1 + u_2 = 0$
   - 目标：找到 Nash 均衡

4.6.4 Nash 均衡

定义 4.7 (Nash 均衡)**.** 策略组合 $(\pi_1^*, \pi_2^*, \ldots, \pi_n^*)$ 是 ***Nash* 均衡**，当且仅当没有任何玩家有动机单方面改变自己的策略：

$$\forall i, \forall \pi_i : \quad u_i(\pi_i^*, \pi_{-i}^*) \geq u_i(\pi_i, \pi_{-i}^*) \tag{4.2}$$

其中 π_{-i}^* 表示除玩家 i 外所有玩家的策略。

注意

Nash 均衡的含义：

- 每个玩家都在对其他玩家的策略做**最优响应**
- 是一种**稳定状态**：没有人有动机单方面偏离
- 不一定是全局最优（如囚徒困境中双方背叛是 Nash 均衡，但不是帕累托最优）

例 4.2 (石头剪刀布的 Nash 均衡). 石头剪刀布是一个零和博弈：

		玩家 *2* 石头	剪刀	布
玩家 *1*	石头	$(0,0)$	$(1,-1)$	$(-1,1)$
	剪刀	$(-1,1)$	$(0,0)$	$(1,-1)$
	布	$(1,-1)$	$(-1,1)$	$(0,0)$

Nash 均衡：双方都以 $(\frac{1}{3},\frac{1}{3},\frac{1}{3})$ 的概率随机选择。
任何确定性策略都可以被对手利用，只有**混合策略**（随机化）才能达到均衡。

定理 4.3 (Nash 均衡存在性). 每个有限博弈（有限玩家、有限策略）至少存在一个 *Nash* 均衡（可能是混合策略均衡）。

4.7　Self-Play 方法

Self-Play 是训练博弈 AI 的强大方法，也是 AlphaGo/AlphaZero 成功的关键之一。

4.7.1　Self-Play 的定义

定义 4.8 (Self-Play). *Agent* 与自己（或自己的历史版本）进行对弈，从对弈经验中学习改进策略。

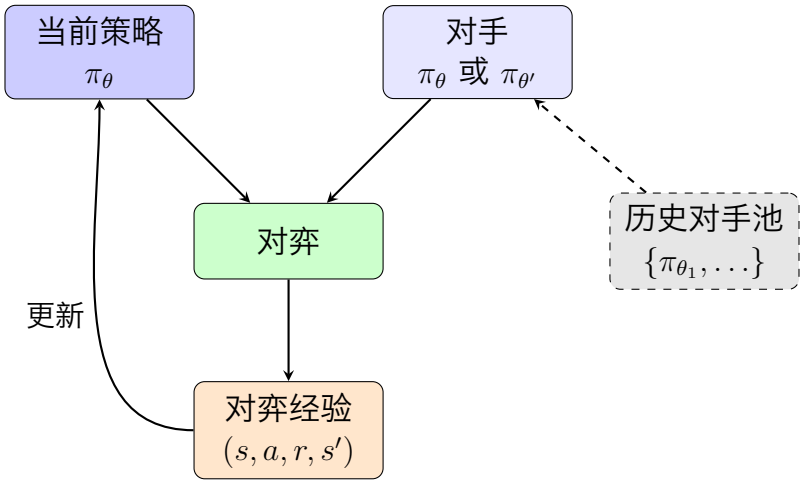


图 4.10: Self-Play 训练示意图

4.7.2　Self-Play 的优势

1. **无限数据**：可以生成任意多的对弈数据，不受人类对局数量限制
2. **自适应难度**：对手随自己一起变强，始终提供适当的挑战
   - 初期：对手弱，容易学习基本策略
   - 后期：对手强，推动学习高级策略
3. **发现新策略**：不受人类先验知识限制，可能发现人类未知的创新策略
   - AlphaGo 的“肩冲”等新招法震惊了职业棋手
4. **逼近 Nash 均衡**：在零和博弈中，Self-Play 在理论上收敛到 Nash 均衡

4.7.3　Self-Play 的挑战

1. **策略遗忘**：
   - 当策略更新后，可能“忘记”如何对付旧策略
   - 解决：维护历史对手池（Opponent Pool），随机抽取对手
2. **局部最优**：
   - 可能陷入“只擅长对付自己”的局部最优
   - 例：两个版本互相克制，形成循环
   - 解决：添加多样性奖励，或从对手池采样
3. **评估困难**：
   - 没有固定基准来衡量进步
   - 解决：用 Elo 评分系统或与固定对手对战

关键点

Self-Play 与 Nash 均衡的关系：

- 在两人零和博弈中，如果 Self-Play 收敛，则收敛到 Nash 均衡
- 直觉：Nash 均衡是“最优响应的不动点”，Self-Play 就是迭代求最优响应
- 但收敛不保证——可能出现策略循环

4.8　本章总结

关键点

本章核心内容：

1. **Model-Based RL** 利用环境模型提高样本效率
   - World Model = 状态转移 + 奖励模型
   - Model Bias：模型误差会累积，需要短期规划或持续校正
2. **Dyna 架构**结合直接学习与规划
   - 每次真实交互后，用模型生成 n 次模拟经验
   - 提供计算-样本的灵活权衡
3. **MCTS** 是 Decision-time Planning 的代表
   - 四步流程：Selection, Expansion, Evaluation, Backup
   - UCB 公式平衡探索与利用
4. **AlphaGo/AlphaZero** 展示了 MCTS + 深度学习 + Self-Play 的强大组合
   - AlphaZero 从零开始，无需人类知识
   - 核心循环：MCTS 改进策略 $\rightarrow$ 网络学习搜索 $\rightarrow$ 正向增强
5. **Multi-Agent RL** 面临非稳态性挑战
   - Nash 均衡：稳定的策略组合
   - Self-Play：训练博弈 AI 的有效方法

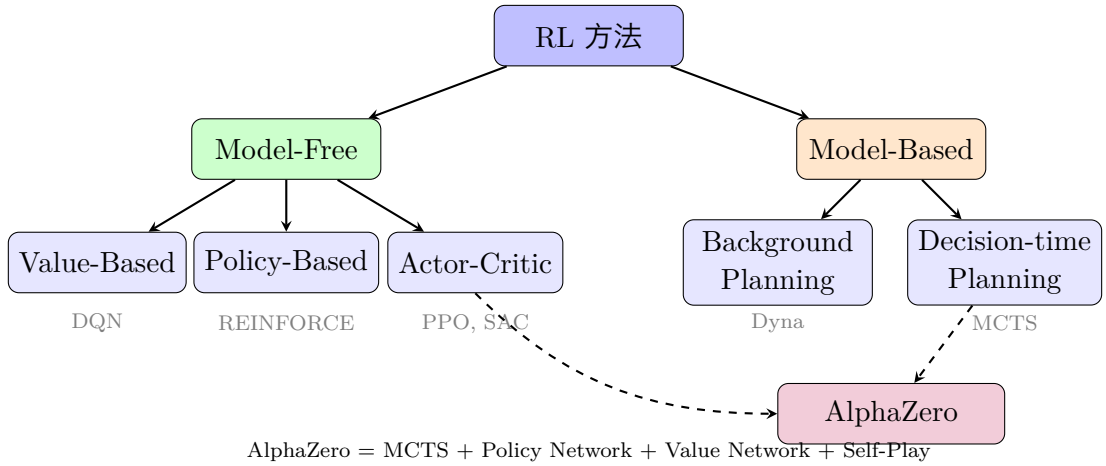


图 4.11: 本章内容在 RL 算法体系中的位置

Chapter 5

大语言模型与强化学习

5.1 引言：从预训练到对齐

5.1.1 核心问题

大语言模型（LLM）通过海量文本预训练，获得了强大的语言理解和生成能力。但预训练目标（预测下一个 token）与人类期望的行为之间存在鸿沟：

> **预训练的 LLM 只学会了“像人类一样说话”，但没有学会“按人类期望行事”。**
>
> 如何让 LLM 不仅流利，还能有帮助、诚实、无害？

这就是 **LLM 对齐**（Alignment）问题。而强化学习正是解决这一问题的核心技术。

5.1.2 为什么需要 RL？

监督学习（SFT）可以让模型模仿高质量回复，但存在局限：

1. **分布受限**：只能学习训练集中出现的回复方式
2. **无法表达偏好**：难以区分“好”和“更好”
3. **无法探索**：不会尝试新的回答策略

强化学习提供了不同的视角：

- 将 LLM 生成过程建模为 MDP
- 用人类偏好定义奖励函数
- 通过最大化奖励来优化策略

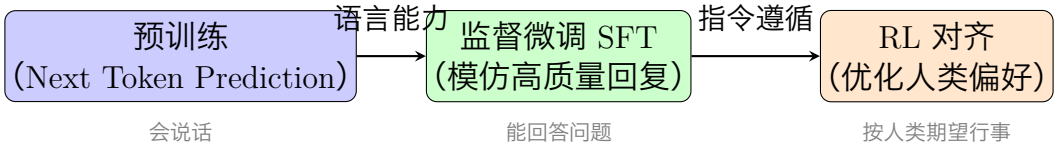


图 5.1: LLM 训练的三个阶段

5.1.3 本章路线图

本章将介绍 LLM 对齐中的 RL 方法：

1. **RL 建模**（第5.2节）：如何将 LLM 生成建模为 MDP
2. **RLHF**（第5.3节）：经典三阶段方法
3. **DPO**（第5.4节）：绕过 Reward Model 的简化方法
4. **GRPO**（第5.5节）：无需 Critic 的在线 RL
5. **KL 估计器**（第5.6节）：k1/k2/k3 的数学
6. **On-Policy Distillation**（第5.7节）：结合 RL 与知识蒸馏的高效方法
7. **PRM**（第5.8节）：过程奖励模型
8. **Long CoT RL**（第5.9节）：长序列 RL 的挑战与方法

5.2　LLM 对齐的 RL 建模

5.2.1　State/Action/Reward 定义

将 LLM 对齐问题建模为 RL 问题：

定义 5.1 (LLM 的 RL 建模)**.**

- ***State*** s_t：*prompt* x + 已生成的 *token* 序列 $y_{<t} = (y_1, \ldots, y_{t-1})$
- ***Action*** a_t：下一个 *token* y_t（词表大小 $|\mathcal{V}| \sim 100k$）
- ***Policy*** $\pi_\theta(a|s)$：*LLM* 本身，$\pi_\theta(y_t|x, y_{<t})$
- ***Trajectory*** τ：完整的生成序列 $y = (y_1, y_2, \ldots, y_T)$
- ***Reward*** r：通常只在序列结束时给出

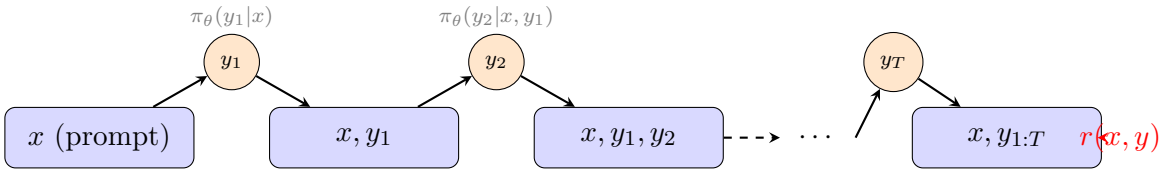


图 5.2: LLM 生成过程的 MDP 建模

LLM RL 的特点：

- **动作空间巨大**：词表通常有 10 万 + token
- **确定性状态转移**：下一状态 = 当前状态 + 新 token
- **Episode = 一次完整生成**：从 prompt 到 EOS
- **稀疏奖励**：只有序列结束时才有奖励信号

5.2.2　稀疏奖励问题

LLM 对齐的典型奖励结构：

$$r_t = \begin{cases} 0 & t < T \\ r_\phi(x, y) & t = T\text{(序列结束)} \end{cases} \tag{5.1}$$

稀疏奖励带来的挑战：

- **信用分配困难**：最终奖励如何归因到每个 token？
- **梯度信号弱**：大部分时刻没有学习信号
- **长序列尤其困难**：信号需要传播很远（数千 token）

注意

解决稀疏奖励的两种思路：

1. **序列级方法**：把整个序列当作一个 bandit，用序列奖励直接更新（如 REINFORCE）
2. **过程奖励**：训练 PRM 提供中间步骤的奖励信号

5.3　RLHF 三阶段

RLHF（Reinforcement Learning from Human Feedback）是 LLM 对齐的经典方法，由 OpenAI 在 InstructGPT 中系统化。

5.3.1　RLHF 整体架构

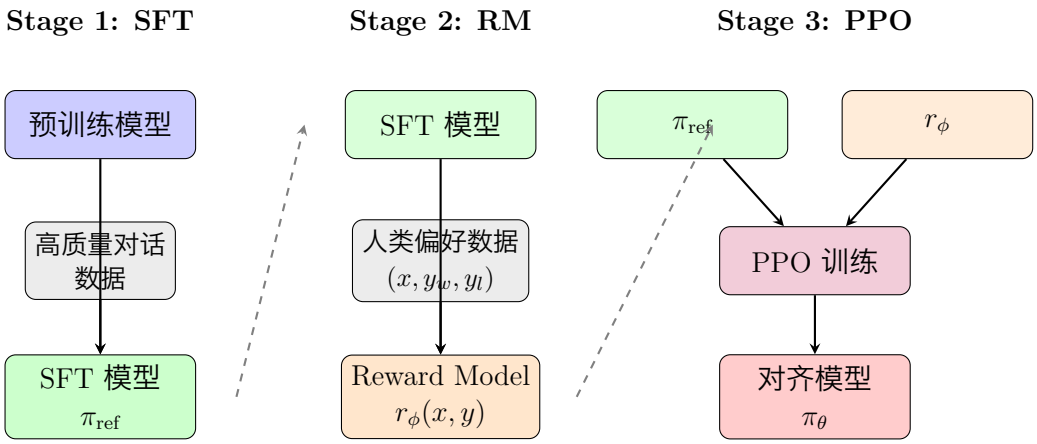


图 5.3: RLHF 三阶段架构

5.3.2　Stage 1: Supervised Fine-Tuning (SFT)

用高质量对话数据微调预训练模型：

$$L_{\text{SFT}}(\theta) = -\mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{SFT}}}\left[\log \pi_\theta(y|x)\right] = -\mathbb{E}\left[\sum_{t=1}^{T} \log \pi_\theta(y_t|x, y_{<t})\right] \tag{5.2}$$

SFT 的作用：

- 让模型学会“指令遵循”的基本格式
- 提供 RL 的起点（参考模型 π_{ref}）
- 过滤预训练中的低质量模式

5.3.3　Stage 2: Reward Model 训练

从人类偏好数据中学习 Reward Model。

定义 5.2 (偏好数据). 对于 *prompt* x，人类标注者比较两个回复，给出偏好：$y_w \succ y_l$（y_w 优于 y_l）。

5.3.3.1　Bradley-Terry 模型

定义 5.3 (Bradley-Terry 模型). 假设人类偏好遵循 *Bradley-Terry* 模型——偏好概率由"能力差"决定：

$$P(y_w \succ y_l|x) = \sigma(r(x, y_w) - r(x, y_l)) = \frac{1}{1 + e^{-(r(x,y_w)-r(x,y_l))}} \tag{5.3}$$

其中 $\sigma(z) = \frac{1}{1+e^{-z}}$ 是 *sigmoid* 函数，$r(x, y)$ 是回复的"得分"。

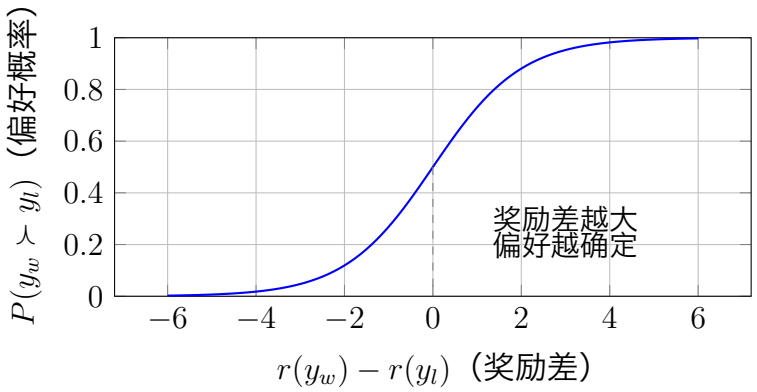


图 5.4: Bradley-Terry 模型：偏好概率由奖励差决定

Bradley-Terry 模型的直觉：

- 奖励差 $= 0$ 时，偏好概率 $= 0.5$（无法区分）
- 奖励差越大，偏好概率越接近 1（更确定）
- 模型假设偏好是基于"内在质量分数"的概率比较

5.3.3.2　Reward Model 训练

Reward Model 的训练目标是最大化偏好数据的似然：

$$L_{\text{RM}}(\phi) = -\mathbb{E}_{(x,y_w,y_l)} \left[\log \sigma(r_\phi(x, y_w) - r_\phi(x, y_l))\right] \tag{5.4}$$

这是一个**二分类问题**：给定 (y_w, y_l)，预测哪个更好。

注意

Reward Model 的架构选择：

- 通常用 SFT 模型初始化
- 去掉语言模型头，加上标量输出头
- 输入 (x, y)，输出标量 $r_\phi(x, y) \in \mathbb{R}$

5.3.4　Stage 3: PPO 微调

使用 Reward Model 提供奖励信号，用 PPO 优化策略。

定义 5.4 (RLHF 优化目标).

$$\max_{\theta} \mathbb{E}_{x\sim\mathcal{D},y\sim\pi_\theta(\cdot|x)}\left[r_\phi(x,y)\right] - \beta \cdot KL(\pi_\theta \| \pi_{ref}) \tag{5.5}$$

其中 $\beta > 0$ 是 KL 正则系数。

5.3.4.1　KL 正则的作用

KL 正则项 $\mathrm{KL}(\pi_\theta \| \pi_{\text{ref}})$ 至关重要：

1. **防止 Reward Hacking**：
   - Reward Model 是不完美的代理
   - 无约束优化会找到“欺骗” RM 的方式
   - 例如：生成特定模式获得高分，但实际质量差
2. **保持生成质量**：
   - SFT 模型已经有较好的语言能力
   - KL 约束防止偏离太远导致流利度下降
3. **稳定训练**：
   - 约束优化空间，避免策略崩溃
   - 提供正则化效果

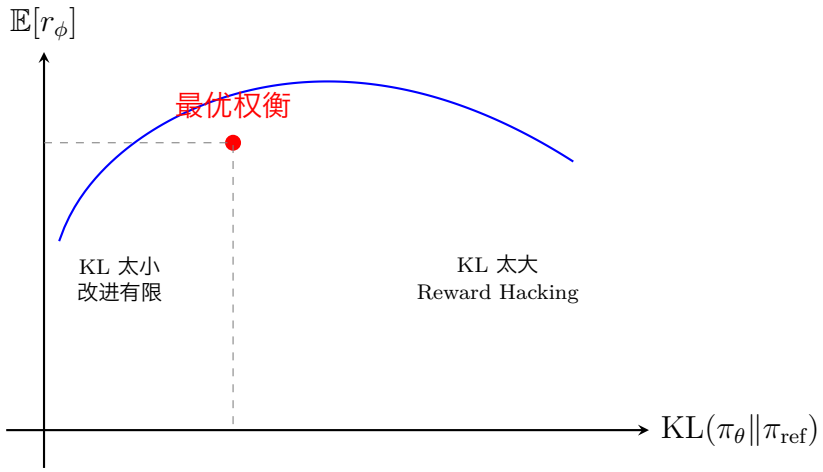


图 5.5: KL 正则的权衡：β 控制在 reward 和 KL 之间的平衡点

5.3.4.2　PPO 更新流程

RLHF 中 PPO 的具体步骤：

算法 13: RLHF with PPO

输入: SFT 模型 π_{ref}，Reward Model r_ϕ，KL 系数 β

初始化 $\pi_\theta \leftarrow \pi_{\text{ref}}$，Critic V_ψ;

foreach *iteration* **do**

  // 采样

  从 prompt 分布采样 $x \sim \mathcal{D}$;

  用当前策略生成回复 $y \sim \pi_\theta(\cdot|x)$;

  // 计算奖励

  计算 RM 奖励：$r^{\text{RM}} = r_\phi(x, y)$;

  计算 KL 惩罚：$r_t^{\text{KL}} = -\beta \log \frac{\pi_\theta(y_t|x,y_{<t})}{\pi_{\text{ref}}(y_t|x,y_{<t})}$;

  总奖励：$r_t = r_t^{\text{KL}} + \mathbb{I}_{t=T} \cdot r^{\text{RM}}$;

  // GAE 计算

  用 Critic V_ψ 计算 advantage $\hat{A}_t$;

  // PPO 更新

  用 PPO-Clip 目标更新 π_θ;

  用 TD 目标更新 V_ψ;

重要

RLHF 需要维护的模型：

1. π_θ：正在训练的策略（Active Model）
2. π_{ref}：参考模型（冻结）
3. r_ϕ：Reward Model（冻结）
4. V_ψ：Critic 网络

共 4 个大模型，显存开销巨大！这是 DPO、GRPO 等方法试图解决的问题。

5.4 Direct Preference Optimization (DPO)

DPO 是一种绕过 Reward Model 和 PPO 的简化方法，由 Rafailov et al. 2023 提出。

5.4.1 DPO 的动机

RLHF + PPO 的问题：

- **模型开销大**：需要维护 4 个模型
- **采样成本高**：大模型在线生成很贵
- **实现复杂**：PPO 超参敏感，需要精细调参
- **训练不稳定**：RL 训练容易崩溃

DPO 的核心问题：能否直接在偏好数据 (x, y_w, y_l) 上优化，像监督学习一样简单？

答案是可以的！关键洞察：KL 正则的 RL 问题有**闭式解**。

5.4.2　DPO Loss 公式

定义 5.5 (DPO Loss)**.**

$$L_{DPO}(\theta) = -\mathbb{E}_{(x,y_w,y_l)}\left[\log\sigma\left(\beta\left[\log\frac{\pi_\theta(y_w|x)}{\pi_{ref}(y_w|x)} - \log\frac{\pi_\theta(y_l|x)}{\pi_{ref}(y_l|x)}\right]\right)\right] \tag{5.6}$$

5.4.3　DPO 完整推导

定理 5.1 (DPO 等价性)**.** *DPO Loss* (5.6) 与 *RLHF* 目标 (5.5) 在最优解处等价。

证明. 推导分为 5 个关键步骤。

Step 1：RLHF 目标展开

RLHF 优化目标:

$$\max_\pi \mathbb{E}_{y\sim\pi}\left[r(x,y)\right] - \beta\cdot\mathrm{KL}(\pi\|\pi_{\text{ref}}) \tag{5.7}$$

展开 KL 散度:

$$= \mathbb{E}_{y\sim\pi}\left[r(x,y)\right] - \beta\mathbb{E}_{y\sim\pi}\left[\log\frac{\pi(y|x)}{\pi_{\text{ref}}(y|x)}\right] \tag{5.8}$$

$$= \mathbb{E}_{y\sim\pi}\left[r(x,y) - \beta\log\frac{\pi(y|x)}{\pi_{\text{ref}}(y|x)}\right] \tag{5.9}$$

Step 2：引入配分函数 $Z(x)$

为了让最优策略是合法的概率分布，定义配分函数:

$$Z(x) = \sum_y \pi_{\text{ref}}(y|x)\exp\left(\frac{r(x,y)}{\beta}\right) \tag{5.10}$$

$Z(x)$ 是归一化常数，只依赖于 x（不依赖于被优化的策略)。

Step 3：最优策略的闭式解

KL 正则 RL 问题有闭式解:

引理 5.2 (KL 正则 RL 的最优策略)**.** 目标 $\max_\pi \mathbb{E}_{y\sim\pi}[r(y)] - \beta\cdot KL(\pi\|\pi_{ref})$ 的最优解为:

$$\pi^*(y|x) = \frac{1}{Z(x)}\pi_{ref}(y|x)\exp\left(\frac{r(x,y)}{\beta}\right) \tag{5.11}$$

证明. 这是一个有约束优化问题（π 需要是概率分布)。用拉格朗日乘子法或变分推断可得。直觉：最优策略是参考策略按 $\exp(r/\beta)$ 重新加权。奖励越高，概率提升越多。 □

Step 4：从最优策略反解 reward

关键步骤：从 (5.11) 反解 reward。

取对数:

$$\log\pi^*(y|x) = \log\pi_{\text{ref}}(y|x) - \log Z(x) + \frac{r(x,y)}{\beta} \tag{5.12}$$

整理得：

$$r(x,y) = \beta \log \frac{\pi^*(y|x)}{\pi_{\text{ref}}(y|x)} + \beta \log Z(x) \tag{5.13}$$

关键点

核心洞察：reward 可以用策略的 log-ratio 表示！虽然有 $\log Z(x)$ 项，但它只依赖于 x，在 pairwise 比较中会消除。

Step 5：代入 Bradley-Terry 模型，$Z(x)$ 消除

将 (5.13) 代入 Bradley-Terry 模型 (5.3)：

$$P(y_w \succ y_l) = \sigma(r(x,y_w) - r(x,y_l)) \tag{5.14}$$

$$= \sigma\Bigg(\underbrace{\beta \log \frac{\pi^*(y_w|x)}{\pi_{\text{ref}}(y_w|x)} + \beta \log Z(x)}_{r(x,y_w)} - \underbrace{\beta \log \frac{\pi^*(y_l|x)}{\pi_{\text{ref}}(y_l|x)} - \beta \log Z(x)}_{r(x,y_l)} \Bigg) \tag{5.15}$$

$$= \sigma\left(\beta \left[\log \frac{\pi^*(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \log \frac{\pi^*(y_l|x)}{\pi_{\text{ref}}(y_l|x)} \right] \right) \tag{5.16}$$

$\beta \log Z(x)$ 项相消了！

最大化偏好数据的 log-likelihood，用 π_θ 代替 π^*，得到 DPO Loss：

$$L_{\text{DPO}}(\theta) = -\mathbb{E}\left[\log \sigma\left(\beta \left[\log \frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \log \frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)} \right] \right) \right] \tag{5.17}$$

□

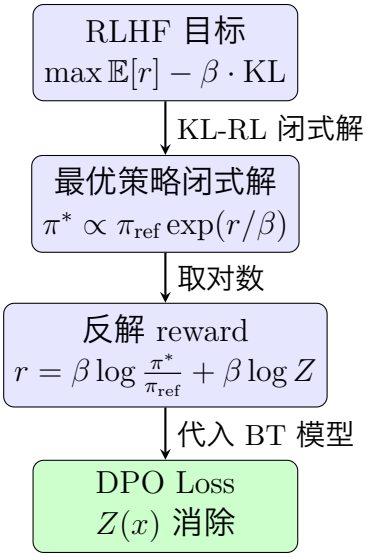


图 5.6: DPO 推导的逻辑链条

重要

DPO 的核心洞察：

1. KL 正则 RL 问题有闭式解，最优策略是参考策略的指数重加权
2. 可以从最优策略反解隐式 reward
3. 配分函数 $Z(x)$ 在 pairwise 比较中消除——这是 DPO 能 work 的关键
4. 最终形式只需要计算 log-probability，像监督学习一样简单

5.4.4　DPO 的直观理解

定义 **隐式奖励**：

$$\hat{r}_\theta(x, y) = \beta \log \frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)} \tag{5.18}$$

DPO Loss 可以写成：

$$L_{\text{DPO}} = -\mathbb{E}\left[\log \sigma(\hat{r}_\theta(x, y_w) - \hat{r}_\theta(x, y_l))\right] \tag{5.19}$$

直觉：

- $\hat{r}_\theta(x, y_w) > \hat{r}_\theta(x, y_l)$：$y_w$ 的隐式奖励更高，loss 变小
- 训练过程提高 y_w 相对于 π_{ref} 的概率，压低 y_l 的概率
- β 控制“相对于参考策略偏离多少”的尺度

5.4.5　DPO vs RLHF 对比

	RLHF + PPO	**DPO**
需要 Reward Model	是	否
需要 Critic 网络	是	否
训练方式	在线采样	离线训练
模型数量	4 个	2 个
实现复杂度	高	低
超参敏感性	高	低
探索能力	有	无
适用场景	复杂任务	简单对齐

表 5.1: RLHF + PPO 与 DPO 的对比

DPO 的局限：

- **无探索**：完全离线，只能在已有偏好数据的分布内优化
- **Pairwise 信号粗糙**：只知道谁更好，不知道好多少
- **难任务提升有限**：在数学、代码等需要探索的任务上效果不如 RL

5.5　GRPO：Group Relative Policy Optimization

GRPO 是 DeepSeek 提出的方法，介于 PPO 和 DPO 之间：保留在线探索能力，但不需要 Critic 网络。

5.5.1　GRPO 的动机

- **PPO 的问题**：需要 Critic 网络，显存开销大（额外一个大模型）
- **DPO 的问题**：完全离线，缺乏探索，难任务提升有限

GRPO 的思路：**用组内相对奖励代替 Critic**，实现“无 Critic 的在线 RL”。

5.5.2　组内标准化 Advantage

定义 5.6 (GRPO Advantage)**.** 对于 *prompt* x，采样一组回复 $\{y_1, \ldots, y_G\}$，计算各自的奖励 $\{R_1, \ldots, R_G\}$，然后：

$$\hat{A}_i = \frac{R_i - \bar{R}}{Std(R) + \epsilon} \tag{5.20}$$

其中 $\bar{R} = \frac{1}{G}\sum_i R_i$ 是组内均值，$Std(R)$ 是组内标准差。

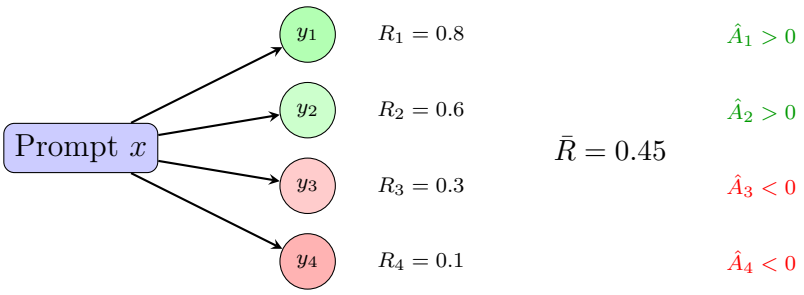


图 5.7: GRPO 组内标准化示意图

组内标准化的优势：

1. **无需 Critic**：用组内均值代替价值函数估计
2. **Baseline 效果**：均值减法降低方差
3. **尺度归一化**：标准差归一化使 advantage 尺度稳定
4. **相对比较**：focus 在同一 prompt 下的相对好坏

5.5.3　GRPO 目标函数

定义 5.7 (GRPO 目标)**.**

$$L_{GRPO} = \mathbb{E}_x \left[\frac{1}{G} \sum_{i=1}^{G} \sum_{t=1}^{|y_i|} \min\left(\rho_{i,t}\hat{A}_i,\ clip(\rho_{i,t}, 1-\epsilon, 1+\epsilon)\hat{A}_i \right) \right] \tag{5.21}$$

其中 $\rho_{i,t} = \frac{\pi_\theta(y_{i,t}|x, y_{i,<t})}{\pi_{\theta_{old}}(y_{i,t}|x, y_{i,<t})}$ 是 *importance sampling* 比率。

注意

GRPO 与 PPO 的关键区别：

- PPO：$\hat{A}_t$ 由 GAE 计算，需要 Critic
- GRPO：$\hat{A}_i$ 对整个序列恒定，由组内标准化得到

5.5.4 GRPO 实用技巧

1. **Clip-Higher**：上界可以更宽松（如 $1+0.28$ 而非 $1+0.2$），允许好回复更大幅度提升
2. **Dynamic Sampling**：过滤全对或全错的 prompt（方差为 0 时 advantage 无定义）
3. **长度惩罚**：防止生成过长的回复

$$R_i = r_\phi(x, y_i) - \lambda \cdot |y_i| \tag{5.22}$$

4. **KL as Loss**：KL 惩罚作为独立 loss 项，而非放入 reward

$$L = -L_{\text{GRPO}} + \lambda_{\text{KL}} \cdot \mathbb{E}[\text{KL}(\pi_\theta \| \pi_{\text{ref}})] \tag{5.23}$$

5.6 KL 散度估计：k1, k2, k3

KL 散度 $\text{KL}(\pi_\theta \| \pi_{\text{ref}})$ 无法精确计算（需要遍历所有可能的序列），需要蒙特卡洛估计。

5.6.1 KL 散度的定义

对于两个分布 p 和 q：

$$\text{KL}(p \| q) = \mathbb{E}_{x \sim p}\left[\log \frac{p(x)}{q(x)}\right] \tag{5.24}$$

在 LLM 场景中，$p = \pi_\theta$，$q = \pi_{\text{ref}}$，从 π_θ 采样。

5.6.2 k1 估计器：直接估计

定义 5.8 (k1 估计器). 定义比率 $r = \frac{\pi_{ref}(y|x)}{\pi_\theta(y|x)}$，则：

$$k_1 = -\log r = \log \frac{\pi_\theta(y|x)}{\pi_{ref}(y|x)} \tag{5.25}$$

性质：

- **无偏**：$\mathbb{E}_{y \sim \pi_\theta}[k_1] = \text{KL}(\pi_\theta \| \pi_{\text{ref}})$
- **高方差**：当 π_θ 和 π_{ref} 差异大时，方差很大

用法：通常放入 reward

$$r_t^{\text{RL}} = r_t^{\text{RM}} - \beta \cdot k_1^{(t)} \tag{5.26}$$

5.6.3　k2 估计器：平方形式

定义 5.9 (k2 估计器).

$$k_2 = \frac{1}{2}(\log r)^2 = \frac{1}{2}\left(\log \frac{\pi_{ref}(y|x)}{\pi_\theta(y|x)}\right)^2 \tag{5.27}$$

性质：

- **有偏**：$\mathbb{E}[k_2] \neq \mathrm{KL}$
- **梯度等价**：$\nabla_\theta \mathbb{E}[k_2] = \nabla_\theta \mathrm{KL}$
- **更平滑**：平方形式在 $r = 1$ 附近更平滑

用法：适合作为 loss 项（因为梯度正确）

5.6.4　k3 估计器：Control Variate

定义 5.10 (k3 估计器).

$$k_3 = (r-1) - \log r = \frac{\pi_{ref}(y|x)}{\pi_\theta(y|x)} - 1 - \log \frac{\pi_{ref}(y|x)}{\pi_\theta(y|x)} \tag{5.28}$$

定理 5.3 (k3 的性质). *$k3$ 是无偏且低方差的 KL 估计器。*

证明. 无偏性：

$$\mathbb{E}_{y\sim\pi_\theta}[k_3] = \mathbb{E}[(r-1)] - \mathbb{E}[\log r] \tag{5.29}$$

$$= \underbrace{\mathbb{E}\left[\frac{\pi_{\mathrm{ref}}}{\pi_\theta}\right] - 1}_{=0} + \mathbb{E}\left[\log \frac{\pi_\theta}{\pi_{\mathrm{ref}}}\right] \tag{5.30}$$

$$= \mathrm{KL}(\pi_\theta \| \pi_{\mathrm{ref}}) \tag{5.31}$$

其中 $\mathbb{E}_{y\sim\pi_\theta}\left[\frac{\pi_{\mathrm{ref}}(y)}{\pi_\theta(y)}\right] = \sum_y \pi_\theta(y) \cdot \frac{\pi_{\mathrm{ref}}(y)}{\pi_\theta(y)} = \sum_y \pi_{\mathrm{ref}}(y) = 1$。
低方差：$(r-1)$ 项是一个 control variate，期望为 0，与 $\log r$ 负相关，减少方差。 □

5.6.5　三种估计器对比

估计器	公式	偏差	方差	推荐用法
k1	$\log \frac{\pi_\theta}{\pi_{\mathrm{ref}}}$	无偏	高	KL in reward
k2	$\frac{1}{2}(\log \frac{\pi_{\mathrm{ref}}}{\pi_\theta})^2$	有偏	低	KL as loss
k3	$\frac{\pi_{\mathrm{ref}}}{\pi_\theta} - 1 - \log \frac{\pi_{\mathrm{ref}}}{\pi_\theta}$	无偏	低	KL as loss

表 5.2: KL 估计器对比

关键点

KL 的两种用法：

1. **KL in Reward**（经典 RLHF）：
   - Token reward 减去 $\beta \cdot k_1$
   - 然后用 PPO-Clip 更新
   - 优点：直接影响每步决策
2. **KL as Loss**（GRPO 等）：
   - 总 loss $= -L_{\text{RL}} + \lambda \cdot \mathbb{E}[k_3]$
   - 优点：更稳定，方差更低

5.7　On-Policy Distillation

On-Policy Distillation 是近年来 LLM 后训练的重要进展，由 GKD [1] 和 MiniLLM [2] 等工作奠定基础，并在 Qwen3 [3] 等前沿模型中得到成功应用。

5.7.1　动机：Off-policy 蒸馏的分布偏移问题

传统的知识蒸馏（SFT on 教师数据）是 **off-policy** 的：

学生从教师生成的数据学习，但推理时走到的状态可能与训练时完全不同。

这导致 **compounding errors**——学生在训练时没见过自己犯的错误，一旦偏离教师轨迹就会持续恶化。尤其在长序列推理任务中，这种分布偏移问题更为严重。

- **Off-policy SFT**：密集监督，但分布偏移
- **On-policy RL**：分布匹配，但奖励稀疏、效率低

自然的问题：能否结合两者优势——在学生自己的分布上获得密集监督？

5.7.2　三种后训练范式对比

方法	采样来源	奖励密度	特点
SFT（监督微调）	教师数据（Off-policy）	密集	分布受限
RL（强化学习）	自身采样（On-policy）	稀疏	搜索成本高
On-Policy Distillation	**自身采样**	**密集**	两者优势结合

表 5.3: 三种后训练范式对比

On-Policy Distillation 的核心思想：

- **学生自己采样**（On-policy）：避免分布偏移
- **教师逐 token 打分**（Dense reward）：提供密集监督信号

5.7.3　Reverse KL 损失

On-Policy Distillation 使用 **反向 KL 散度**作为损失函数：

$$L_{\text{OPD}} = D_{\text{KL}}(\pi_\theta \| \pi_{\text{teacher}}) = \mathbb{E}_{y\sim\pi_\theta}\left[\sum_t \log \frac{\pi_\theta(y_t|x, y_{<t})}{\pi_{\text{teacher}}(y_t|x, y_{<t})}\right] \tag{5.32}$$

注意

Forward KL vs Reverse KL：

- Forward KL $D_{\text{KL}}(\pi_{\text{teacher}}\|\pi_\theta)$：mode covering，学生试图覆盖教师的所有模式
- Reverse KL $D_{\text{KL}}(\pi_\theta\|\pi_{\text{teacher}})$：mode seeking，学生专注于教师的高概率区域

Reverse KL 更适合蒸馏场景——让学生在自己访问的状态上模仿教师，而非试图覆盖教师的全部行为。

5.7.4　KDRL：结合知识蒸馏与强化学习

KDRL [4]（Knowledge Distillation + Reinforcement Learning）将教师监督和奖励驱动的自我探索统一为联合优化：

$$\mathcal{J}_{\text{KDRL}}(\theta) = \underbrace{\mathcal{J}_{\text{GRPO}}(\theta)}_{\text{奖励驱动}} - \beta \cdot \underbrace{D_{\text{KL}}^{k_2}(\pi_\theta\|\pi_{\text{teacher}})}_{\text{教师监督}} \tag{5.33}$$

关键设计选择：

1. **KL 估计器选择**：k_2 优于 k_3，因为 k_2 提供无偏梯度估计
2. **系数退火**：从强模仿（β 大）逐渐过渡到奖励驱动（β 小）
3. **奖励引导 Masking**：只在低奖励样本上应用 KD，减少输出长度同时保持准确率

5.7.5　效率优势

关键点

On-Policy Distillation 的效率提升（以数学推理任务为例）：

- 相比 RL：训练速度快 7-10 倍，总计算效率提升 50-100 倍
- 相比离线蒸馏：计算成本降低 9-30 倍
- Qwen3 实验：蒸馏显著优于 RL，GPU 小时仅需 RL 的约 1/10

效率提升的根本原因：

> **教师提供的密集 token 级信号，比 RL 的稀疏序列级奖励包含更多信息。**
>
> RL 的大部分计算花在搜索（探索策略空间），蒸馏则直接利用教师的知识指导学生在关键“分叉点”做出正确选择。

5.7.6　应用场景

1. **小模型专业化**：用大模型（如 Qwen3-235B）蒸馏小模型（如 Qwen3-32B）

2. **持续学习**：新领域知识训练后，用蒸馏恢复遗忘的指令遵循能力
3. **行为恢复**：从早期 checkpoint 恢复特定能力

重要

Qwen3 的发现：

- 蒸馏不仅匹配 RL 性能，还能**扩展探索空间**
- 蒸馏后 pass@64 分数提升，而 RL 在 pass@64 上无改进
- 原因：蒸馏学习教师的完整分布，而非单一最优答案

5.8　Process Reward Model (PRM)

PRM 提供过程级监督，将稀疏的终局奖励变成密集的步级奖励。

5.8.1　ORM vs PRM

定义 5.11 (ORM 与 PRM).

- ***ORM (Outcome Reward Model)***：只看最终结果
  - 输入：(x, y)
  - 输出：最终答案的正确性分数
- ***PRM (Process Reward Model)***：评估每个中间步骤
  - 输入：$(x, y_{\le t})$
  - 输出：到第 t 步为止的正确性分数

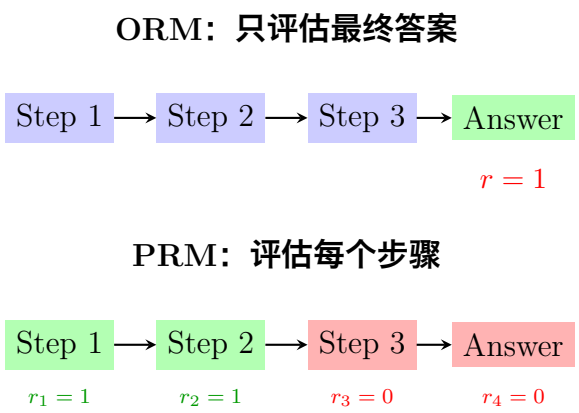


图 5.8: ORM vs PRM：PRM 提供 dense reward 信号

5.8.2　PRM 的优势

1. **信用分配更清晰**：每步都有反馈，知道哪一步出错
2. **长链推理收敛更快**：dense reward 比 sparse reward 更容易学习
3. **可以做早停**：如果中间步骤得分持续下降，可以截断重采样
4. **支持 Best-of-N**：用累积 PRM 分数选择最佳推理路径

5.8.3　PRM 训练

PRM 的训练数据来源：

- **人工标注**：专家标注每个推理步骤的正确性
- **自动标注**：用最终答案正确性反推中间步骤
- **MCTS 探索**：搜索发现正确/错误的分支点

PRM 用于 RL 的奖励：

$$r_t = \text{PRM}(x, y_{\le t}) - \text{PRM}(x, y_{\le t-1}) \tag{5.34}$$

即每步的“边际贡献”。

5.9　Long CoT RL

长序列 Chain-of-Thought (Long CoT) 的 RL 训练面临独特挑战。随着 o1、DeepSeek-R1 等推理模型的出现，这成为重要的研究方向。

5.9.1　长序列 RL 的挑战

1. **方差爆炸**：Token 级 importance sampling 权重累积后方差指数增长

$$\prod_{t=1}^{T} \frac{\pi_\theta(y_t|s_t)}{\pi_{\text{old}}(y_t|s_t)} \approx e^{\sum_t \delta_t} \tag{5.35}$$

   当 T 很大时（数千 token），这个乘积的方差会爆炸。

2. **策略偏移**：长序列使 π_θ 和 π_{old} 偏离更大
3. **稀疏奖励更难**：只有最终答案有反馈，信号传播数千步

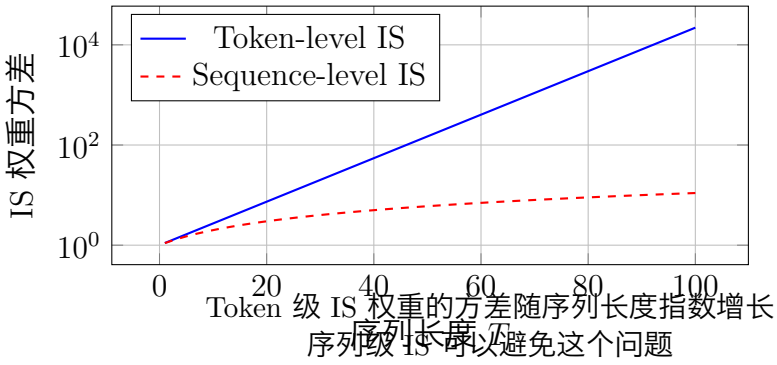


图 5.9: Long CoT RL 的方差爆炸问题

5.9.2　GSPO：序列级 IS

GSPO（Group Sequence Policy Optimization）使用序列级重要性采样：

定义 5.12 (GSPO 序列级 IS)**.** 定义长度归一化的序列级 *IS* 权重：

$$w_i(\theta) = \left(\frac{\pi_\theta(y_i|x)}{\pi_{old}(y_i|x)}\right)^{1/|y_i|} = \exp\left(\frac{1}{|y_i|}\sum_t \log \frac{\pi_\theta(y_{i,t}|s_{i,t})}{\pi_{old}(y_{i,t}|s_{i,t})}\right) \tag{5.36}$$

长度归一化后再 *clip*，所有 *token* 共用同一个权重。

GSPO 的优势：

- 避免 token 级权重累乘的方差爆炸
- 长度归一化使不同长度序列可比
- 保持 PPO-Clip 的稳定性

5.9.3 CISPO：Clipped IS-weight

CISPO（Clipped Importance Sampling Policy Optimization）采用另一种策略：

定义 5.13 (CISPO).

- 保持 *GRPO* 的组内标准化
- 回到 *token* 级 *REINFORCE*
- *Clip IS* 权重（而非 *loss*）：

$$\hat{\rho}_{i,t} = clip\left(\frac{\pi_\theta(y_{i,t}|s_{i,t})}{\pi_{old}(y_{i,t}|s_{i,t})}, 1-\epsilon, 1+\epsilon\right) \tag{5.37}$$

5.9.4 Kimi k1.5 技术要点

Kimi k1.5 的长 CoT RL 配方：

- **超长上下文**：128k context 直接 RL
- **Mirror Descent 更新**：更保守的策略更新
- **部分 Rollout**：不需要完整生成，截断后用价值估计
- **异步 Train/Infer**：分离训练和推理以提高效率
- **重复检测 + 早停**：检测到循环生成就截断
- **长度惩罚**：鼓励简洁的推理

Long2Short RL（长到短蒸馏）：

- 长 CoT 模型作 teacher
- 训练短 CoT student
- 奖励 = 正确性 + token 数惩罚
- 目标：又对又短

5.9.5 DeepSeek-V3.2 改进

DeepSeek-V3.2 对 GRPO 的改进（核心思想：让 off-policy 训练尽量接近 on-policy 的行为）：

1. **Unbiased KL Estimate**：用 IS 比率修正 k3 的 off-policy 偏差

$$\hat{\mathrm{KL}} = \mathbb{E}_{\pi_{\text{old}}}\left[\frac{\pi_\theta}{\pi_{\text{old}}} \cdot k_3\right] \tag{5.38}$$

2. **Off-Policy Sequence Masking**：负 advantage 且 KL 偏离过大的序列被 mask
3. **Keep Routing**（MoE 专用）：保持采样时的 expert routing 路径
4. **Keep Sampling Mask**：保持 top-p/top-k 的 truncation mask

5.10　Token-level vs Sequence-level 目标

5.10.1　一阶近似理论

Token-level 目标（如 REINFORCE、GRPO）是 sequence-level 目标的一阶近似。

设 $\frac{\pi_\theta(y_t|s_t)}{\pi_{\theta_{\text{old}}}(y_t|s_t)} = 1 + \delta_t$，其中 δ_t 是小量，则：

$$\prod_t (1 + \delta_t) \approx 1 + \sum_t \delta_t \quad \text{（一阶 Taylor 展开）} \tag{5.39}$$

因此，当策略变化较小时：

$$\nabla \mathcal{J}^{\text{seq}} \approx \nabla \mathcal{J}^{\text{token}} \tag{5.40}$$

成立条件：$\pi_\theta \approx \pi_{\theta_{\text{old}}}$，即每次更新步长足够小。

5.10.2　Training-Inference Discrepancy

实际中，采样分布 μ 可能与训练时的 $\pi_{\theta_{\text{old}}}$ 不同：

$$\frac{\pi_\theta(y)}{\mu(y)} = \underbrace{\frac{\pi_{\theta_{\text{old}}}(y)}{\mu(y)}}_{\text{train-infer gap}} \times \underbrace{\frac{\pi_\theta(y)}{\pi_{\theta_{\text{old}}}(y)}}_{\text{policy staleness}} \tag{5.41}$$

两个因素都会导致一阶近似失效：

- **Train-infer gap**：训练和推理的采样策略不同（如 temperature、top-p）
- **Policy staleness**：异步训练中，数据来自旧策略

5.11　本章总结

关键点

本章核心内容：

1. **LLM 对齐的 RL 建模**：State = prompt + 已生成 tokens，Action = 下一个 token
2. **RLHF 三阶段**：
   - Stage 1 (SFT)：监督微调，学习指令遵循
   - Stage 2 (RM)：从偏好数据训练 Reward Model
   - Stage 3 (PPO)：用 RM 提供奖励，PPO 优化
3. **DPO**：绕过 RM，利用 KL-RL 闭式解直接优化偏好
4. **GRPO**：用组内相对奖励代替 Critic，实现无 Critic 的在线 RL
5. **KL 估计器**：k1（无偏高方差）、k2（有偏低方差）、k3（无偏低方差）
6. **PRM**：过程奖励模型，提供 dense reward 信号
7. **Long CoT RL**：GSPO/CISPO 解决长序列方差爆炸问题

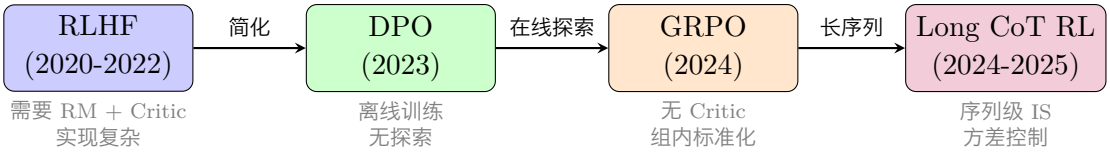


图 5.10: LLM 对齐 RL 方法的演进

Chapter 6

总结与展望

> "学习了这么多算法，如何在实际问题中选择合适的方法？各种方法之间有什么内在联系？强化学习的未来在哪里？"

本章是全书的总结与升华。前五章分别介绍了 RL 的基础概念、Value-Based 方法、Policy-Based 方法、Model-Based 方法与多智能体学习，以及 LLM 时代的 RL 应用。这些内容看似独立，实则存在深刻的内在联系。本章将：

1. 构建 RL 算法的完整知识体系，揭示各方法之间的关联
2. 提供算法选择的决策指南，帮助读者应对实际问题
3. 给出核心公式的速查手册，便于快速回顾
4. 展望强化学习的未来发展方向

6.1 知识体系回顾

6.1.1 从问题到解法：RL 的思维框架

强化学习的核心任务是**寻找最优策略**。让我们回顾这一目标是如何通过不同路径实现的：

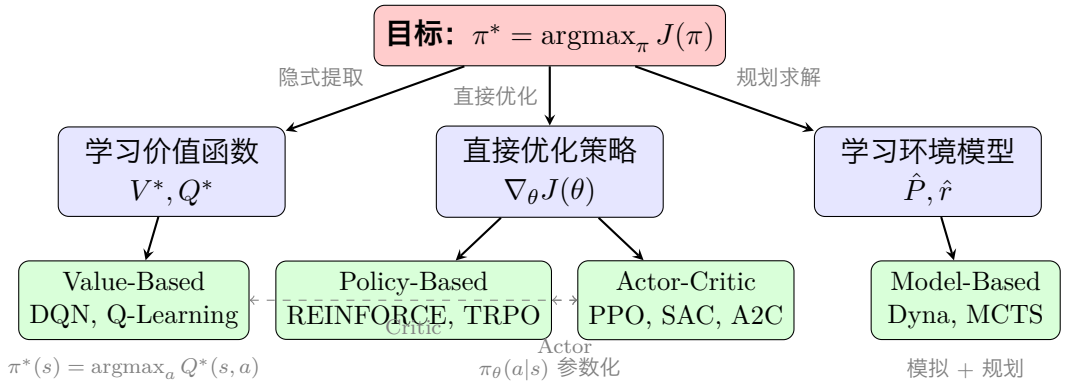


图 6.1: RL 算法的三条主要路径及其关联

关键点

三条路径的核心思想：

1. **Value-Based**：“如果我知道每个状态-动作对的价值，选择最优动作就是 argmax”
2. **Policy-Based**：“直接参数化策略，沿梯度方向改进”
3. **Model-Based**：“如果我有环境模型，可以通过模拟和规划找到最优策略”

Actor-Critic 是 Value-Based 和 Policy-Based 的融合，用 Critic（价值网络）指导 Actor（策略网络）的更新。

6.1.2　核心概念关系图

全书涉及的核心概念可以组织为如下的关系网络：

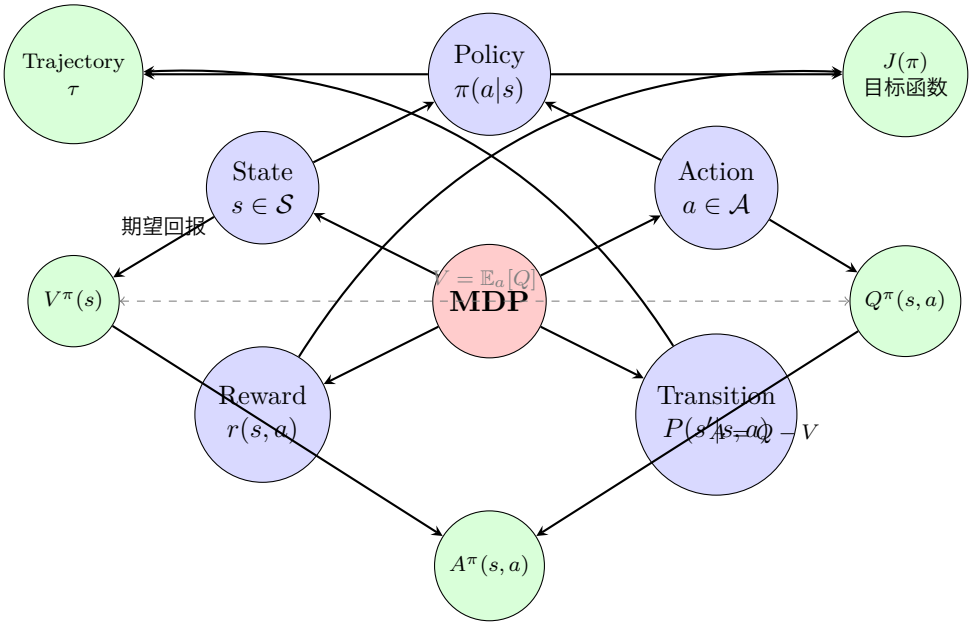


图 6.2: RL 核心概念关系网络

6.2　RL 算法全景图

6.2.1　多维度算法分类

RL 算法可以从多个维度进行分类，每个维度反映了不同的设计选择：

分类维度	类别 A	类别 B
环境模型	**Model-Free**: 不使用模型, 直接从交互学习	**Model-Based**: 学习或利用环境模型规划
学习目标	**Value-Based**:学习 V^* 或 Q^*, 隐式导出策略	**Policy-Based**: 直接优化参数化策略 π_θ
数据来源	**On-policy**: 只用当前策略产生的数据	**Off-policy**: 可用任意策略产生的数据
动作空间	**离散**: 有限动作集合, 可枚举	**连续**: 无限动作空间, 需参数化
更新频率	**Episode-based**: 完成完整轨迹后更新	**Step-based**: 每步或每批次更新

表 6.1: RL 算法的多维度分类

6.2.2　算法分类树

综合以上维度, RL 算法形成如下的分类体系:

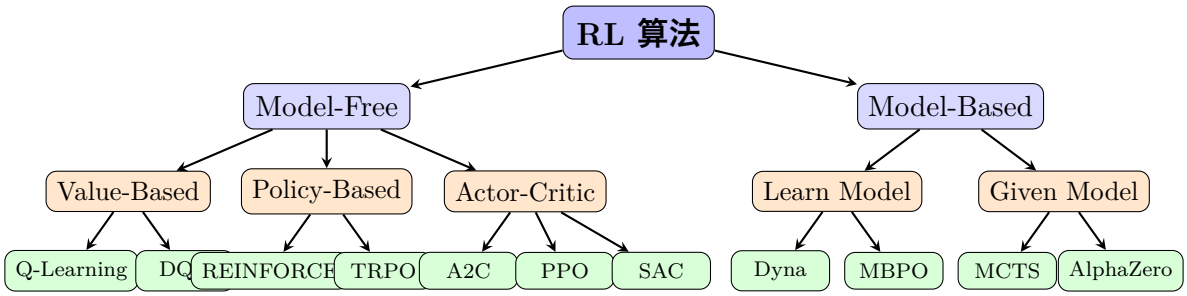


图 6.3: RL 算法分类树（含代表性算法）

6.2.3　算法特性对比

下表从多个关键维度对比主要算法的特性:

算法	类型	On/Off	离散	连续	样本效率	稳定性	复杂度
Q-Learning	Value	Off	✓	×	中	高	低
DQN	Value	Off	✓	×	中	中	中
REINFORCE	Policy	On	✓	✓	低	低	低
A2C	AC	On	✓	✓	低	中	中
PPO	AC	On	✓	✓	低	高	中
SAC	AC	Off	×	✓	高	高	中
TD3	AC	Off	×	✓	高	高	中
Dyna	MB	混合	✓	✓	高	中	高
MCTS	MB	–	✓	×	–	高	高
AlphaZero	MB+AC	–	✓	×	–	高	极高
LLM 对齐方法							
RLHF/PPO	AC	On	×	✓	低	高	极高
DPO	–	Off	×	✓	高	高	低
GRPO	Policy	On	×	✓	中	高	中

表 6.2: 主要 RL 算法特性对比（AC = Actor-Critic, MB = Model-Based）

注意

关于“样本效率”的理解：

- **Off-policy** 方法通常比 On-policy 更高效，因为可以重用历史数据
- **Model-Based** 方法可以通过模拟生成数据，大幅提升效率
- **LLM 场景**中，生成一个 response 的成本极高，所以 DPO 的“高效率”（离线训练）非常有价值

6.2.4　算法选择决策树

面对实际问题，可以按照以下决策流程选择合适的算法：

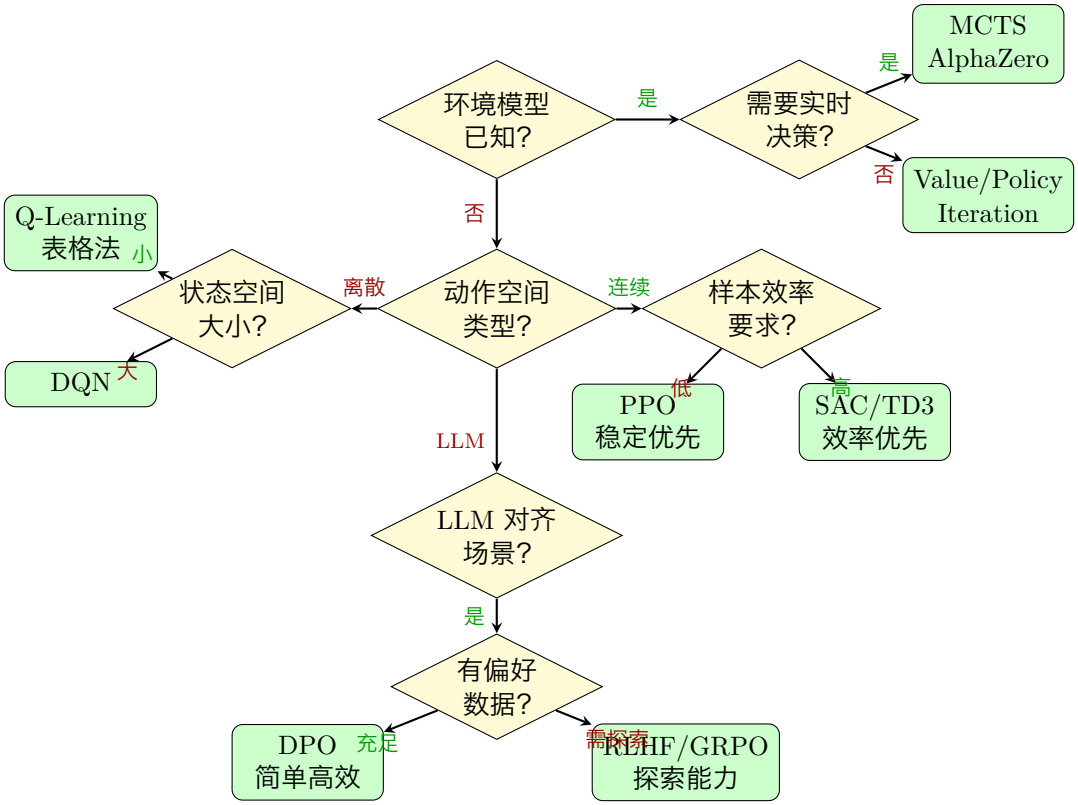


图 6.4: 算法选择决策树

关键点

算法选择的核心考量：

1. **环境模型**：有模型用规划，无模型用学习
2. **动作空间**：离散用 DQN 系列，连续用 Actor-Critic 系列
3. **样本效率**：低要求用 On-policy（PPO），高要求用 Off-policy（SAC）
4. **LLM 场景**：有充足偏好数据用 DPO，需要探索用 GRPO

6.3　核心公式速查

本节汇总全书的核心公式，便于快速查阅。

6.3.1 Value-Based 核心公式

公式名称	表达式
状态价值函数	$V^\pi(s) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t r_t \mid s_0 = s\right]$
动作价值函数	$Q^\pi(s,a) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t r_t \mid s_0 = s, a_0 = a\right]$
Bellman Expectation (V)	$V^\pi(s) = \sum_a \pi(a\|s)\left[r(s,a) + \gamma\sum_{s'} P(s'\|s,a)V^\pi(s')\right]$
Bellman Expectation (Q)	$Q^\pi(s,a) = r(s,a) + \gamma\sum_{s'} P(s'\|s,a)\sum_{a'}\pi(a'\|s')Q^\pi(s',a')$
Bellman Optimality (V)	$V^*(s) = \max_a\left[r(s,a) + \gamma\sum_{s'} P(s'\|s,a)V^*(s')\right]$
Bellman Optimality (Q)	$Q^*(s,a) = r(s,a) + \gamma\sum_{s'} P(s'\|s,a)\max_{a'} Q^*(s',a')$
TD 误差	$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$
Q-Learning 更新	$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma\max_{a'} Q(s',a') - Q(s,a)\right]$
DQN Loss	$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\left(r + \gamma\max_{a'} Q_{\theta^-}(s',a') - Q_\theta(s,a)\right)^2\right]$

表 6.3: Value-Based 方法核心公式

6.3.2 Policy-Based 核心公式

公式名称	表达式
目标函数	$J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)] = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\gamma^t r_t\right]$
Policy Gradient 定理	$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T}\nabla_\theta\log\pi_\theta(a_t\|s_t)\cdot\Psi_t\right]$
REINFORCE (Ψ_t)	$\Psi_t = \sum_{t'=t}^{T}\gamma^{t'-t}r_{t'}$　(Reward-to-go)
带 Baseline	$\Psi_t = \sum_{t'=t}^{T}\gamma^{t'-t}r_{t'} - b(s_t)$，常取 $b(s_t) = V(s_t)$
Advantage Function	$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$
GAE(λ)	$\hat{A}_t^{\text{GAE}} = \sum_{l=0}^{\infty}(\gamma\lambda)^l\delta_{t+l}$
重要性采样比	$\rho_t = \frac{\pi_\theta(a_t\|s_t)}{\pi_{\theta_{\text{old}}}(a_t\|s_t)}$
PPO-Clip 目标	$L^{\text{CLIP}}(\theta) = \mathbb{E}_t\left[\min\left(\rho_t\hat{A}_t, \text{clip}(\rho_t, 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]$

表 6.4: Policy-Based 方法核心公式

6.3.3 LLM-RL 核心公式

公式名称	表达式
RLHF 目标	$\max_\pi \mathbb{E}_{x\sim\mathcal{D},y\sim\pi(\cdot\|x)}\left[r_\phi(x,y)\right]-\beta\cdot D_{\mathrm{KL}}(\pi\|\pi_{\mathrm{ref}})$
Bradley-Terry 模型	$P(y_w\succ y_l\|x)=\sigma(r(x,y_w)-r(x,y_l))$
Reward Model Loss	$\mathcal{L}_{\mathrm{RM}}=-\mathbb{E}_{(x,y_w,y_l)}\left[\log\sigma(r_\phi(x,y_w)-r_\phi(x,y_l))\right]$
RLHF 最优策略	$\pi^*(y\|x)=\frac{1}{Z(x)}\pi_{\mathrm{ref}}(y\|x)\exp\left(\frac{r(x,y)}{\beta}\right)$
DPO Loss	$\mathcal{L}_{\mathrm{DPO}}=-\mathbb{E}\left[\log\sigma\left(\beta\log\frac{\pi_\theta(y_w\|x)}{\pi_{\mathrm{ref}}(y_w\|x)}-\beta\log\frac{\pi_\theta(y_l\|x)}{\pi_{\mathrm{ref}}(y_l\|x)}\right)\right]$
GRPO Advantage	$\hat{A}_i=\frac{R_i-\bar{R}}{\mathrm{Std}(R)}$，其中 $\bar{R}=\frac{1}{G}\sum_{j=1}^{G}R_j$
KL 估计器 (k3)	$\hat{D}_{\mathrm{KL}}^{(k3)}=\frac{\pi_{\mathrm{ref}}(y\|x)}{\pi_\theta(y\|x)}-\log\frac{\pi_{\mathrm{ref}}(y\|x)}{\pi_\theta(y\|x)}-1$

表 6.5: LLM-RL 核心公式

6.3.4 Model-Based 核心公式

公式名称	表达式
World Model	$\hat{P}(s'\|s,a),\hat{r}(s,a)$：学习的转移和奖励函数
UCB 公式	$\mathrm{UCB}(s,a)=Q(s,a)+c\sqrt{\frac{\ln N(s)}{N(s,a)}}$
PUCT (AlphaZero)	$\mathrm{PUCT}(s,a)=Q(s,a)+c\cdot P(s,a)\cdot\frac{\sqrt{N(s)}}{1+N(s,a)}$
Nash 均衡	$\pi_i^*=\mathrm{argmax}_{\pi_i}J_i(\pi_i,\pi_{-i}^*),\ \forall i$

表 6.6: Model-Based 与 MARL 核心公式

6.4 LLM 时代的 RL 演进

6.4.1 从传统 RL 到 LLM-RL

LLM 对齐场景与传统 RL 的关键差异：

维度	传统 RL	LLM-RL
State	游戏画面、机器人传感器	Prompt + 已生成 tokens
Action	离散动作（上下左右）或连续控制	词表中的 token（$\|\mathcal{V}\|\sim 10^5$）
Episode 长度	通常 $< 10^3$ 步	可达 10^4 tokens（Long CoT）
Reward	稠密（每步分数）或稀疏（终局胜负）	极稀疏（仅最终答案）
环境	模拟器（可重置）	无模拟器，生成即交互
Policy 规模	$10^6\sim 10^8$ 参数	$10^9\sim 10^{12}$ 参数

表 6.7: 传统 RL 与 LLM-RL 对比

6.4.2　RLHF → DPO → GRPO 发展脉络

LLM 对齐领域的方法演进体现了“简化 + 高效”的趋势：

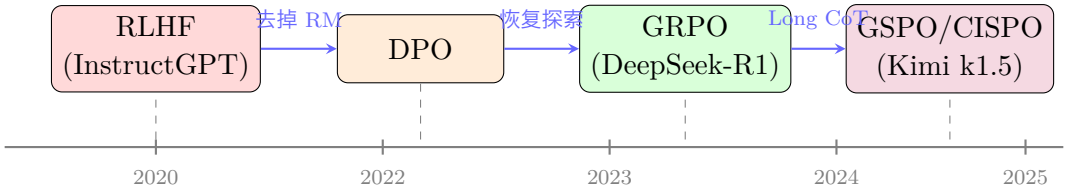


图 6.5: LLM-RL 方法演进时间线

方法	需要 RM	需要 Critic	在线探索	Long CoT
RLHF (PPO)	✓	✓	✓	困难
DPO	×	×	×	不适用
GRPO	×	×	✓	困难
GSPO/CISPO	×	×	✓	✓

表 6.8: LLM-RL 方法特性对比

6.4.3　Long CoT RL 的挑战与解决方案

长链推理（Long Chain-of-Thought）的 RL 训练面临三大核心挑战：

1. **方差爆炸**：Token 级 IS 权重累积，$\prod_{t=1}^{T} \rho_t$ 指数增长
2. **稀疏奖励**：只有最终答案有反馈，中间步骤无信号
3. **信用分配**：长序列中哪一步推理是关键？

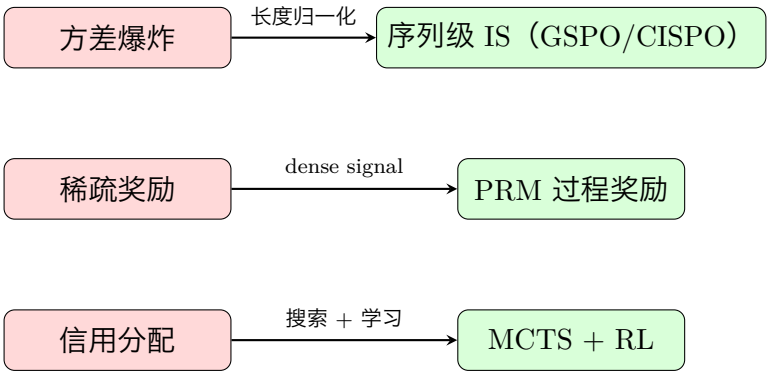


图 6.6: Long CoT RL 的挑战与解决方案

6.5　开放问题与未来方向

6.5.1　理论前沿

尽管 RL 领域已取得巨大进展，仍有许多基础理论问题有待解决：

1. **Token-level vs Sequence-level 目标**

- 两种目标函数的理论关系?
- 何时应该用 token-level，何时用 sequence-level?
- 能否统一两种视角?

2. **KL 正则的最优系数**
   - β 应该如何随训练动态调整?
   - KL 正则与 reward 之间的帕累托前沿?
   - 过大/过小的 β 分别导致什么问题?
3. **长序列 RL 的收敛性**
   - IS 权重的方差如何精确刻画?
   - 序列级 clipping 的理论保证?
   - 最优的轨迹长度与更新频率?
4. **探索与利用的平衡**
   - 在 LLM 场景如何高效探索?
   - 如何避免 mode collapse?

6.5.2　实践挑战

从实践角度，以下问题仍需解决:

挑战	问题描述	可能方向
Reward Hacking	模型学会 "欺骗" 奖励函数, 而非真正提升能力	对抗训练、多 RM 集成、宪法 AI
多目标对齐	同时优化 helpfulness、harmlessness、honesty 可能冲突	帕累托优化、条件生成
计算效率	大规模 RL 训练需要极高计算资源	高效采样、模型蒸馏、LoRA
数据质量	人类偏好标注成本高且有噪声	主动学习、AI 辅助标注
可解释性	RL 训练后模型行为难以解释	可解释 reward、过程监督

表 6.9: LLM-RL 实践挑战与可能方向

6.5.3　新兴方向

以下是当前最活跃的研究前沿:

1. **Agentic RL**
   - LLM 作为 agent 与真实环境（网页、代码执行器、API）交互
   - 挑战：环境反馈稀疏、状态空间巨大、安全约束
   - 代表工作：WebAgent、SWE-agent
2. **多模态 RL**

- 视觉-语言模型（VLM）的对齐
- 机器人控制中的多模态感知-决策
- 代表工作：RT-2、PaLM-E

3. **Reasoning RL**
   - 用 RL 提升模型的推理能力（数学、代码、逻辑）
   - Self-play 生成推理数据
   - 代表工作：DeepSeek-R1、OpenAI o1
4. **World Models for LLM**
   - LLM 是否可以作为 World Model?
   - 基于 LLM 的规划与推理
   - 代表工作：LWM、Genie
5. **Constitutional AI**
   - 用 AI 自我监督替代人类标注
   - 基于原则的 reward 设计
   - 代表工作：Anthropic Constitutional AI

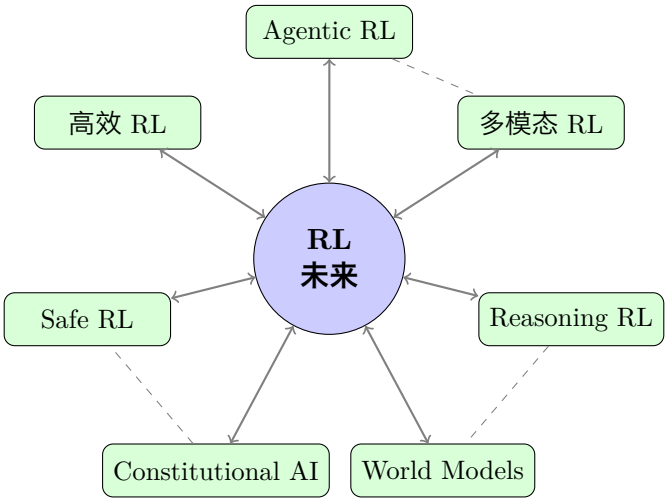


图 6.7: RL 未来研究方向

6.6　学习路径建议

针对不同背景的读者，建议以下学习路径：

6.6.1　入门路径

1. **第 1 章**：理解 MDP、轨迹、价值函数的基本概念
2. **第 2 章**：掌握 Bellman 方程、Q-Learning 的基本思想
3. **第 3 章 (前半)**：理解 Policy Gradient 的直觉
4. 实践：用 DQN 玩 Atari 游戏，用 PPO 解决 CartPole

6.6.2　进阶路径

1. **第 2 章 (深入)**：完整推导 Bellman 方程，理解 DQN 的各种技巧
2. **第 3 章 (完整)**：Policy Gradient 定理完整推导，GAE，PPO
3. **第 4 章**：Model-Based RL，MCTS，AlphaZero
4. 实践：实现 PPO 训练连续控制任务，复现 AlphaZero 简化版

6.6.3　前沿路径

1. **第 5 章**：RLHF、DPO、GRPO 的完整推导
2. **第 6 章**：理解当前开放问题和研究方向
3. 阅读最新论文：DeepSeek-R1、Kimi k1.5、OpenAI o1 技术报告
4. 实践：用 DPO/GRPO 微调开源 LLM

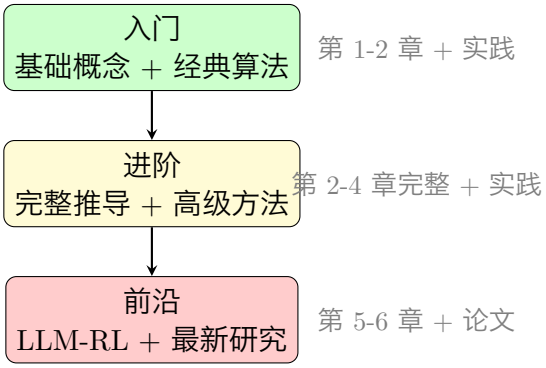


图 6.8: 学习路径建议

6.7　本章小结

关键点

全书总结：

1. **第 1 章**介绍了 RL 的基本概念：MDP 五元组、Markov 性质、轨迹与回报、价值函数 V/Q、优势函数 A、RL 目标
2. **第 2 章**深入 Value-Based 方法：Bellman 方程完整推导（Expectation + Optimality）、动态规划、MC vs TD、Q-Learning、DQN 及其技巧（Experience Replay、Target Network）
3. **第 3 章**系统讲述 Policy-Based 方法：Policy Gradient 定理完整推导、REINFORCE、Baseline 技巧、Actor-Critic、GAE、重要性采样、PPO-Clip
4. **第 4 章**介绍 Model-Based RL 与多智能体学习：World Model、Dyna 架构、MCTS 四步流程、UCB 公式、AlphaGo/Zero、Nash 均衡、Self-Play
5. **第 5 章**聚焦 LLM 对齐：RLHF 三阶段、Bradley-Terry 模型、DPO 完整推导、GRPO、KL 估计器（k1/k2/k3）、PRM、Long CoT RL（GSPO/CISPO）
6. **第 6 章**总结全书：算法分类与选择、核心公式速查、LLM-RL 演进、未来方向

重要

核心思想回顾：

- **RL 的本质**是在不完全信息下通过试错学习最优决策
- **Value-Based** 通过学习价值函数间接得到策略
- **Policy-Based** 直接优化参数化策略
- **Actor-Critic** 融合两者优势
- **Model-Based** 利用环境模型进行规划
- **LLM-RL** 将 RL 应用于语言模型对齐，发展出 RLHF、DPO、GRPO 等方法

强化学习是一个快速发展的领域，新的算法和应用不断涌现。从 AlphaGo 到 ChatGPT，RL 已成为构建智能系统的核心技术。希望本书能为读者提供坚实的理论基础和实践指引，在这一激动人心的领域中继续探索。

附录 A

详细推导与常见问题

A.1 DPO 详细推导

本节给出 DPO Loss 的完整推导过程。

A.1.1 Step 1：RLHF 目标函数

RLHF 的优化目标是：

$$\max_{\pi} \mathbb{E}_{x\sim\mathcal{D},y\sim\pi}\left[r(x,y)\right] - \beta \cdot \mathrm{KL}(\pi(y|x)\|\pi_{\mathrm{ref}}(y|x)) \tag{A.1}$$

A.1.2 Step 2：展开 KL 散度

展开 KL 散度，转为最小化问题：

$$J(\theta) = \max_{\pi} \mathbb{E}_{x,y\sim\pi}\left[r(x,y) - \beta \log \frac{\pi(y|x)}{\pi_{\mathrm{ref}}(y|x)}\right] \tag{A.2}$$

$$= \min_{\pi} \mathbb{E}_{x,y\sim\pi}\left[\beta \log \frac{\pi(y|x)}{\pi_{\mathrm{ref}}(y|x)} - r(x,y)\right] \tag{A.3}$$

$$= \min_{\pi} \mathbb{E}_{x,y\sim\pi}\left[\log \frac{\pi(y|x)}{\pi_{\mathrm{ref}}(y|x)} - \frac{1}{\beta} r(x,y)\right] \tag{A.4}$$

A.1.3 Step 3：引入配分函数 $Z(x)$

定义配分函数：

$$Z(x) = \sum_{y} \pi_{\mathrm{ref}}(y|x) \exp\left(\frac{r(x,y)}{\beta}\right) \tag{A.5}$$

在目标函数中加减 $\log Z(x)$（不影响优化）：

$$J(\theta) = \min_{\pi} \mathbb{E}_{x,y\sim\pi}\left[\log \frac{\pi(y|x)}{\pi_{\mathrm{ref}}(y|x)} - \frac{1}{\beta} r(x,y) + \log Z(x) - \log Z(x)\right] \tag{A.6}$$

$$= \min_{\pi} \mathbb{E}_{x,y\sim\pi}\left[\log \frac{\pi(y|x)}{\frac{1}{Z(x)}\pi_{\mathrm{ref}}(y|x)\exp\left(\frac{r(x,y)}{\beta}\right)} - \log Z(x)\right] \tag{A.7}$$

A.1.4　Step 4：最优策略的闭式解

定义：

$$\pi^*(y|x) = \frac{1}{Z(x)}\pi_{\text{ref}}(y|x)\exp\left(\frac{r(x,y)}{\beta}\right) \tag{A.8}$$

可以验证 π^* 是合法的概率分布（因为 $Z(x)$ 是归一化常数）。
目标函数变为：

$$J(\theta) = \min_{\pi} \mathbb{E}_x\left[\text{KL}(\pi(\cdot|x)\|\pi^*(\cdot|x)) - \log Z(x)\right] \tag{A.9}$$

由于 KL 散度非负且在 $\pi = \pi^*$ 时为零，最优解为 $\pi = \pi^*$。

关键点

这是 DPO 的核心洞察：KL 正则 RL 问题有闭式解！最优策略是参考策略按 $\exp(r/\beta)$ 重新加权的结果。

A.1.5　Step 5：反解 reward

从 π^* 的定义反解 reward：

$$\pi^*(y|x) = \frac{\pi_{\text{ref}}(y|x)}{Z(x)}\exp\left(\frac{r(x,y)}{\beta}\right) \tag{A.10}$$

$$\log \pi^*(y|x) = \log \pi_{\text{ref}}(y|x) - \log Z(x) + \frac{r(x,y)}{\beta} \tag{A.11}$$

解出 $r(x,y)$：

$$r(x,y) = \beta \log \frac{\pi^*(y|x)}{\pi_{\text{ref}}(y|x)} + \beta \log Z(x) \tag{A.12}$$

A.1.6　Step 6：Bradley-Terry 偏好模型

人类偏好遵循 Bradley-Terry 模型：

$$P(y_w \succ y_l|x) = \sigma(r(x,y_w) - r(x,y_l)) \tag{A.13}$$

A.1.7　Step 7：代入 reward，$Z(x)$ 消除

将 (A.12) 代入：

$$r(x,y_w) - r(x,y_l) = \beta \log \frac{\pi^*(y_w|x)}{\pi_{\text{ref}}(y_w|x)} + \cancel{\beta \log Z(x)} - \beta \log \frac{\pi^*(y_l|x)}{\pi_{\text{ref}}(y_l|x)} - \cancel{\beta \log Z(x)} \tag{A.14}$$

$$= \beta\left[\log \frac{\pi^*(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \log \frac{\pi^*(y_l|x)}{\pi_{\text{ref}}(y_l|x)}\right] \tag{A.15}$$

关键：$\beta \log Z(x)$ 项相消！

A.1.8　Step 8：DPO Loss

最大化 log-likelihood：

$$\boxed{L_{\text{DPO}}(\theta) = -\mathbb{E}_{(x,y_w,y_l)}\left[\log \sigma\left(\beta\left[\log \frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \log \frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)}\right]\right)\right]} \tag{A.16}$$

A.2　KL 估计器推导

本节推导三种 KL 散度估计器：k1、k2、k3。

A.2.1　KL 散度的定义

$$\mathrm{KL}(\pi_\theta \| \pi_{\text{ref}}) = \mathbb{E}_{y\sim\pi_\theta}\left[\log \frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)}\right] = \mathbb{E}_{y\sim\pi_\theta}\left[-\log r\right] \tag{A.17}$$

其中 $r = \frac{\pi_{\text{ref}}(y|x)}{\pi_\theta(y|x)}$。

A.2.2　k1：直接估计

定义 A.1 (k1 估计器).

$$k_1 = -\log r = \log \frac{\pi_\theta}{\pi_{ref}} \tag{A.18}$$

性质:

- $\mathbb{E}_{\pi_\theta}[k_1] = \mathrm{KL}(\pi_\theta \| \pi_{\text{ref}})$（无偏）
- 方差较大，因为 $\log r$ 可能取很大或很小的值

A.2.3　k2：平方形式

定义 A.2 (k2 估计器).

$$k_2 = \frac{1}{2}(\log r)^2 = \frac{1}{2}\left(\log \frac{\pi_{ref}}{\pi_\theta}\right)^2 \tag{A.19}$$

性质:

- $\mathbb{E}_{\pi_\theta}[k_2] \neq \mathrm{KL}$（有偏）
- 但 $\nabla_\theta \mathbb{E}[k_2] = \nabla_\theta \mathrm{KL}$（梯度正确）
- 更平滑，适合做 loss

梯度等价性证明.

$$\nabla_\theta k_2 = \nabla_\theta \frac{1}{2}(\log r)^2 = \log r \cdot \nabla_\theta \log r \tag{A.20}$$

$$= -\log r \cdot \nabla_\theta \log \pi_\theta = k_1 \cdot \nabla_\theta \log \pi_\theta \tag{A.21}$$

由于 $\mathbb{E}[\nabla_\theta \log \pi_\theta] = 0$（score function 的期望为零），可以证明 $\mathbb{E}[\nabla_\theta k_2] = \mathbb{E}[\nabla_\theta k_1]$。 □

A.2.4　k3：Control Variate

定义 A.3 (k3 估计器).

$$k_3 = (r-1) - \log r = \frac{\pi_{ref}}{\pi_\theta} - 1 - \log \frac{\pi_{ref}}{\pi_\theta} \tag{A.22}$$

性质:

- $\mathbb{E}_{\pi_\theta}[r-1] = 0$（因为 $\mathbb{E}_{\pi_\theta}[r] = \sum_y \pi_\theta \cdot \frac{\pi_{\text{ref}}}{\pi_\theta} = 1$）

- $(r-1)$ 作为 control variate，降低方差
- $\mathbb{E}_{\pi_\theta}[k_3] = \mathrm{KL}$（无偏）
- 方差比 k1 小

关键点

k3 的巧妙之处：利用 $\mathbb{E}[r-1]=0$ 这个性质，加入 $(r-1)$ 项不改变期望，但可以降低方差。这是 control variate 技术的典型应用。

A.2.5　三种估计器的对比

估计器	公式	偏差	方差	推荐用法
k1	$-\log r$	无偏	高	in reward
k2	$\frac{1}{2}(\log r)^2$	有偏	低	as loss
k3	$(r-1)-\log r$	无偏	低	as loss

表 A.1: KL 估计器对比

A.3　Off-Policy 状态分布修正的理论基础

本节给出“严格的 off-policy policy gradient 需要状态分布修正”的完整数学推导。

A.3.1　性能差引理（Performance Difference Lemma）

定理 A.1 (Kakade & Langford, 2002)**.** 对于任意两个策略 π_θ 和 π_{old}:

$$J(\pi_\theta) - J(\pi_{old}) = \frac{1}{1-\gamma}\mathbb{E}_{s\sim d_{\pi_\theta}, a\sim\pi_\theta(\cdot|s)}\left[A^{\pi_{old}}(s,a)\right] \tag{A.23}$$

其中 $d_\pi(s) = (1-\gamma)\sum_{t=0}^{\infty}\gamma^t P(s_t = s \mid \pi)$ 是 *discounted state visitation distribution*。

证明. 定义状态价值函数的差:

$$J(\pi_\theta) - J(\pi_{\mathrm{old}}) = \mathbb{E}_{s_0}\left[V^{\pi_\theta}(s_0) - V^{\pi_{\mathrm{old}}}(s_0)\right] \tag{A.24}$$

利用 advantage 的定义 $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$，通过 telescope sum 展开:

$$V^{\pi_\theta}(s) - V^{\pi_{\mathrm{old}}}(s) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty}\gamma^t\left(r_t + \gamma V^{\pi_{\mathrm{old}}}(s_{t+1}) - V^{\pi_{\mathrm{old}}}(s_t)\right)\,\middle|\, s_0 = s\right] \tag{A.25}$$

$$= \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty}\gamma^t A^{\pi_{\mathrm{old}}}(s_t,a_t)\,\middle|\, s_0 = s\right] \tag{A.26}$$

对初始状态求期望并整理:

$$J(\pi_\theta) - J(\pi_{\mathrm{old}}) = \sum_{t=0}^{\infty}\gamma^t\mathbb{E}_{s_t\sim P^t_{\pi_\theta}, a_t\sim\pi_\theta}\left[A^{\pi_{\mathrm{old}}}(s_t,a_t)\right] \tag{A.27}$$

代入 discounted state visitation distribution 的定义得：

$$\boxed{J(\pi_\theta) - J(\pi_{\text{old}}) = \frac{1}{1-\gamma}\mathbb{E}_{s\sim d_{\pi_\theta}, a\sim\pi_\theta}\left[A^{\pi_{\text{old}}}(s,a)\right]} \tag{A.28}$$

□

重要

注意右边的期望是关于 **新策略的状态分布** d_{π_θ}，而非旧策略的 $d_{\pi_{\text{old}}}$。这是状态分布修正的理论来源。

A.3.2　严格的 Off-Policy 估计

我们的数据来自旧策略 π_{old}：

- 状态：$s \sim d_{\pi_{\text{old}}}$
- 动作：$a \sim \pi_{\text{old}}(\cdot|s)$

要用这些数据无偏地估计性能差引理中的期望，需要两个重要性权重：

$$\mathbb{E}_{s\sim d_{\pi_\theta}, a\sim\pi_\theta}[f(s,a)] = \mathbb{E}_{s\sim d_{\pi_{\text{old}}}, a\sim\pi_{\text{old}}}\left[\underbrace{\frac{d_{\pi_\theta}(s)}{d_{\pi_{\text{old}}}(s)}}_{\text{状态分布修正}} \cdot \underbrace{\frac{\pi_\theta(a|s)}{\pi_{\text{old}}(a|s)}}_{\text{动作概率修正}} \cdot f(s,a)\right] \tag{A.29}$$

因此，严格的 off-policy policy gradient 为：

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim d_{\pi_{\text{old}}}}\left[\frac{d_{\pi_\theta}(s)}{d_{\pi_{\text{old}}}(s)} \cdot \mathbb{E}_{a\sim\pi_{\text{old}}}\left[\frac{\pi_\theta(a|s)}{\pi_{\text{old}}(a|s)}\nabla_\theta \log \pi_\theta(a|s)\hat{A}(s,a)\right]\right] \tag{A.30}$$

A.3.3　PPO/TRPO Surrogate Objective 的近似

PPO/TRPO 使用的 surrogate objective 为：

$$L(\theta) = \mathbb{E}_{s\sim d_{\pi_{\text{old}}}, a\sim\pi_{\text{old}}}\left[\frac{\pi_\theta(a|s)}{\pi_{\text{old}}(a|s)}\hat{A}(s,a)\right] \tag{A.31}$$

对比严格的 off-policy 梯度，**缺少了状态分布比** $\frac{d_{\pi_\theta}(s)}{d_{\pi_{\text{old}}}(s)}$，即隐式假设：

$$\frac{d_{\pi_\theta}(s)}{d_{\pi_{\text{old}}}(s)} \approx 1 \tag{A.32}$$

A.3.4　近似误差界

定理 A.2 (TRPO 误差界，Schulman et al., 2015)**.** *Surrogate objective* 与真实性能差的偏差有界：

$$\left|J(\pi_\theta) - J(\pi_{old}) - \frac{1}{1-\gamma}L(\theta)\right| \le \frac{2\gamma\epsilon}{(1-\gamma)^2} \cdot \max_s D_{KL}(\pi_{old}(\cdot|s)\|\pi_\theta(\cdot|s)) \tag{A.33}$$

其中 $\epsilon = \max_{s,a}|A^{\pi_{old}}(s,a)|$。

关键点

这个误差界说明：

1. 当 π_θ 与 π_{old} 的 KL 散度足够小时，surrogate objective 是真实性能差的良好近似
2. TRPO 的 KL 约束和 PPO 的 clip 机制正是为了控制这个误差界
3. 这解释了为什么 PPO/TRPO 能直接使用 token-level 的 $\rho_t = \frac{\pi_\theta(a|s)}{\pi_{\text{old}}(a|s)}$ 而不需要显式计算状态分布修正——**前提是 trust region 约束成立**

A.4　常见问题解答

A.4.1　Q1：为什么 RLHF 要加 KL 正则？

答：KL 正则有多重作用：

1. **防止 Reward Hacking**：Reward Model 不是完美的，模型可能学到“欺骗” RM 的方式（如生成特定模式获得高分但质量差）。KL 项限制策略偏离 SFT 太远。
2. **保持生成质量**：SFT 模型已经学会了基本的语言能力，KL 项确保这些能力不丢失。
3. **稳定训练**：约束优化空间，避免策略崩溃或发散。
4. **理论保证**：KL 正则 RL 问题有闭式解（DPO 的理论基础）。

A.4.2　Q2：长序列 token-level PPO/GRPO 为什么容易崩？

答：主要原因是 importance sampling 权重的累积：

1. **权重累积**：序列级 IS 权重是 token 级权重的乘积：

$$\rho_{\text{seq}} = \prod_{t=1}^{T} \rho_t \tag{A.34}$$

   即使每个 ρ_t 接近 1，累积后 ρ_{seq} 可能非常大或非常小。
2. **方差爆炸**：大的 IS 权重导致梯度估计方差急剧增大。
3. **Policy Staleness**：长序列生成耗时长，策略 π_θ 在生成过程中已经更新多次，导致 π_θ 和 π_{old} 偏离更大。

解决方案：

- GSPO：序列级 IS + 长度归一化
- CISPO：clip IS 权重而非 loss
- 减少更新步数，保持 policy 接近

A.4.3　Q3：如何同时优化 Helpfulness / Harmlessness / Honesty？

答：多目标对齐的常见方法：

1. **多 Reward Model**：为每个目标训练独立的 RM，加权求和：

$$r_{\text{total}} = w_1 r_{\text{helpful}} + w_2 r_{\text{harmless}} + w_3 r_{\text{honest}} \tag{A.35}$$

2. **Constitutional AI**：用规则和原则指导生成，自我批评和修正。
3. **多阶段训练**：先优化 harmlessness，再优化 helpfulness。
4. **Pareto 优化**：寻找多目标的 Pareto 前沿。

挑战：不同目标可能冲突（如 helpful 但 harmful 的回答）。

A.4.4　Q4：如何避免 Reward Hacking?

答：Reward Hacking 是指模型学到获得高奖励但实际质量差的行为。防范方法：

1. **KL 正则**：限制策略偏离参考模型。
2. **多样化 RM 训练数据**：覆盖更多场景，减少 RM 的漏洞。
3. **RM Ensemble**：使用多个 RM，取平均或最小值。
4. **人工评估**：定期人工检查模型输出。
5. **对抗训练**：生成可能 hack RM 的样本，加入训练。
6. **过程监督**：使用 PRM 提供更细粒度的反馈。

关键点

Reward Hacking 的根本原因是 Reward Model 不完美，无法完全捕捉人类偏好。长期解决方案是提升 RM 质量和多维度评估。

参考文献

[1] Rishabh Agarwal et al. “On-Policy Distillation of Language Models: Learning from Self-Generated Mistakes”. In: *International Conference on Learning Representations (ICLR)*. GKD: Generalized Knowledge Distillation. 2024. URL: https://arxiv.org/abs/2306.13649.

[2] Yuxian Gu et al. “MiniLLM: Knowledge Distillation of Large Language Models”. In: *International Conference on Learning Representations (ICLR)*. Reverse KL for LLM distillation. 2024. URL: https://arxiv.org/abs/2306.08543.

[3] An Yang et al. “Qwen3 Technical Report”. In: *arXiv preprint arXiv:2505.09388* (2025). URL: https://arxiv.org/abs/2505.09388.

[4] Haoran Xu et al. “KDRL: Post-Training Reasoning LLMs via Unified Knowledge Distillation and Reinforcement Learning”. In: *arXiv preprint arXiv:2506.02208* (2025). URL: https://arxiv.org/abs/2506.02208.