大模型强化学习

预备知识

NLP 基础：Prompt → Response 形式的任务

核心思想：
给模型一个 **Prompt**（输入/指令），模型输出一个 **Response**（回答）。

例子：
Prompt: “翻译成英文：我爱学习”
Response: “I love studying.”

要点：把一切 NLP 任务（翻译、问答、代码生成、摘要）统一成 **Prompt → Response** 的格式。

Transformer & LLM：了解基础结构

要知道的结构：

- **Self-Attention**：每个 token 能“看见”前面所有 token。
- **Decoder-only LLM**：GPT/Qwen/LLaMA 属于这种结构，训练目标是预测下一个 token。
- **输出概率分布**：LLM 本质就是一个条件概率模型 $P_\theta(y|x)$。

不用深挖公式，但要知道：

- Attention = 信息加权聚合
- LLM = 巨大的自回归语言模型

监督微调 (SFT)：知道怎么用现成数据对模型做有监督训练

目标：让预训练模型更好地“听懂人话”。

方法：用 **(instruction, input, output)** 格式的数据，做交叉熵训练。

本质：就是标准的有监督学习，不涉及奖励，不涉及 RL。

强化学习基础

为“决策”建立数学模型

强化学习的本质是**学习如何做出一系列最优决策**。为了让计算机理解并解决这个问题，我们首先需要一个数学框架来描述它。这个框架就是**马尔可夫决策过程 (MDP)**。

什么是马尔可夫决策过程 (MDP)？

MDP 将决策过程抽象成了五个核心元素 (S,A,P,R,γ)：

- S **(State): 状态空间**。所有可能的状态。在 LLM 中，是用户当前的**输入或整个对话历史**。

- A **(Action): 动作空间**。所有可能的动作。在 LLM 中，是模型可以生成的**所有回答**。
- P **(Transition Probability): 状态转移概率**。P(s′ | s,a)，在状态 s 执行动作 a 后，转移到状态 s′ 的概率。
- R **(Reward): 奖励函数**。R(s,a)，执行动作后获得的即时奖励。在 RLHF 中，是 **Reward Model** 对回答的打分。
- γ **(Discount Factor): 折扣因子**。一个衡量“远见”的参数。

定义“最优目标”

RL 的目标，就是找到一个策略 π，使得期望的**折扣化累积回报 (Return)** Gt 最大化。

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

引入折扣因子 γ 的原因：

1. **数学上保证收敛**：确保总回报是一个有限值。
2. **现实意义上的权衡**：γ 越接近 1，智能体越有“远见”；越接近 0，智能体越“短视”。

价值的递归：贝尔曼方程

为了评估一个状态或动作的“好坏”，我们引入了**价值函数 (Value Function)**。贝尔曼方程是强化学习的基石，它揭示了价值函数之间一个优美的递归关系。

核心思想：今天的价值 = 即时奖励 + 明天的折扣价值

贝尔曼方程的核心思想非常直观：一个状态的价值，可以分解为离开这个状态能立即获得的奖励，加上你将进入的下一个状态的价值（因为是未来的，所以要打个折扣 γ）。

两种价值函数

- **状态价值函数 $V_\pi(s)$**：从状态 s 出发，遵循策略 π 能获得的期望回报。它回答了“处于这个状态有多好？”

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

递归形式体现了**贝尔曼方程**的思想。

- **动作价值函数 $Q_\pi(s,a)$**：在状态 s 执行动作 a 后，再遵循策略 π 能获得的期望回报。它回答了“在这个状态下，执行这个动作有多好？”

$$Q_\pi(s,a) = \mathbb{E}_\pi\Big[R_{t+1} + \gamma Q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\Big]$$

动作价值函数可以用来指导策略更新（例如选择更高价值的动作）。

贝尔曼期望方程 (Bellman Expectation Equation)

这个方程描述了在遵循一个给定的策略 π 时，一个状态的价值与其后续状态价值之间的关系。

1. 价值的定义

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

2. 回报的递归

$$G_t = R_{t+1} + \gamma G_{t+1}$$

（即时奖励 + 未来所有奖励的折扣和）

3. 代入定义

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

4. 展开期望

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

> *注：这里 S_{t+1} 与 s' 是同一概念，即“可能到达的下一状态”。*

5. 完整形式（考虑所有可能性）

$$V_\pi(s) = \sum_{a \in A} \pi(a \mid s) \left[R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a) V_\pi(s') \right]$$

拆解贝尔曼方程的四个关键组件

1. $V_\pi(s')$：下一状态的价值

这是对未来的评估。s' 代表你可能到达的下一个状态。$V_\pi(s')$ 就是该状态的期望累计回报。

2. $P(s' \mid s,a)$：状态转移概率

这是环境的不确定性，表示“如果在状态 s 执行动作 a，有多大概率会到达状态 s'”。
例如，80% 成功，20% 到别的状态。

因此，内层的

$$\sum_{s' \in S} P(s' \mid s,a) V_\pi(s')$$

就是对未来的期望：把所有可能到达的下一状态的价值，乘以到达它们的概率，再加起来。
这就是“选择动作 a 后，对未来的平均期望价值”。

3. $R(s,a) + \gamma \cdot \ldots$：即时奖励与未来价值的结合

- $R(s,a)$ 是你执行动作 a 后能立即获得的奖励。
- γ 是折扣因子，表示对未来的重视程度，$\gamma = 0.9$ 意味着未来的价值折算为今天的 90%。

括号里的

$$R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a) V_\pi(s')$$

表示“在状态 s 选择动作 a”的总价值（即时奖励 + 未来期望奖励），也就是**动作价值** $Q_\pi(s,a)$。

4. $\sum_{a \in A} \pi(a \mid s) \ldots$：策略的加权平均

这是策略的不确定性。$\pi(a \mid s)$ 是策略在状态 s 选择动作 a 的概率。

把所有动作的价值，按概率加权求和，就得到在状态 s 的最终期望价值 $V_\pi(s)$。

贝尔曼最优方程 (Bellman Optimality Equation)

在最优策略 π^* 下，总是选择能带来最大价值的动作，因此期望操作 $\sum \pi(a \mid s)$ 被替换为 $\max_a$：

$$V^*(s) = \max_a \mathbb{E}_{s'}[R(s,a) + \gamma V^*(s')]$$

> *注：这是贝尔曼期望方程在最优策略下的特例，用于求最优值函数和最优策略。*

贝尔曼方程的矩阵形式与求解

贝尔曼方程在理论上可以用矩阵形式表示，但实际求解存在一定困难。下面我们系统梳理。

1. 矩阵形式

对于有限状态空间的马尔可夫决策过程 (MDP)，状态集 $S = \{s_1, s_2, \ldots, s_n\}$，贝尔曼期望方程可以写成矩阵形式：

$$\mathbf{V}_\pi = \mathbf{R}_\pi + \gamma \mathbf{P}_\pi \mathbf{V}_\pi$$

- $\mathbf{V}_\pi$：$n \times 1$ 的状态价值向量
- $\mathbf{R}_\pi$：$n \times 1$ 的状态奖励向量（每个状态下期望即时奖励）
- $\mathbf{P}_\pi$：$n \times n$ 的状态转移概率矩阵（由策略 π 决定）
- γ：折扣因子

理论上可以直接求解：

$$\mathbf{V}_\pi = (\mathbf{I} - \gamma \mathbf{P}_\pi)^{-1} \mathbf{R}_\pi$$

---

2. 为什么难求解

1. 状态空间过大
   实际问题中状态空间可能非常大，甚至连续，构造 $\mathbf{P}_\pi$ 或求逆矩阵几乎不可能。
2. 计算复杂度高
   矩阵求逆复杂度 $O(n^3)$，状态数 n 很大时不可行。
3. 稀疏性与内存问题
   尽管矩阵可能稀疏，但直接求逆仍然内存消耗大。
4. 环境未知
   在强化学习中通常不知道完整的 $\mathbf{P}_\pi$ 和 $\mathbf{R}_\pi$，矩阵方法无法使用。

---

3. 常见的计算方式

（**1**）迭代方法（动态规划类）

- 价值迭代（**Value Iteration**）

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} P(s' \mid s, a) \big[R(s, a) + \gamma V_k(s') \big]$$

迭代直到收敛，逼近最优价值 $V^*(s)$。

- 策略迭代（**Policy Iteration**）

  交替进行：

  a. 策略评估：用当前策略计算 V_π（可用迭代逼近）

  b. 策略改进：每个状态选择能使价值最大的动作，更新策略

  重复迭代直到策略收敛。

> *这类方法不需要矩阵求逆，可处理较大状态空间。*

（**2**）采样方法（强化学习类）

- 蒙特卡洛方法（**Monte Carlo**）
  根据实际采样轨迹计算累计回报，逼近 $V_\pi(s)$，不依赖模型。
- 时序差分学习（**TD Learning**）

$$V(s_t) \leftarrow V(s_t) + \alpha\big(R_{t+1} + \gamma V(s_{t+1}) - V(s_t)\big)$$

利用每一步奖励和下一状态估计值进行在线更新，适合大规模或未知环境。

- **Q-learning / SARSA**
  直接估计动作价值 $Q(s,a)$，无需知道环境的转移概率。

---

4. 总结对比

方法类别	特点	是否需要模型
矩阵求解	精确解，可理论使用	需要完整 P_π, R_π
迭代方法	数值逼近，可处理较大空间	需要模型
采样方法	在线学习，无需完整模型	不需要模型

On-Policy 与 Off-Policy：两种不同的学习范式

在采样方法（如蒙特卡洛、TD学习）中，智能体通过与环境交互产生的数据（轨迹）来学习。根据产生这些数据的策略与我们想要评估和改进的策略是否为同一个，可以将学习方法分为两大类。

- **On-Policy (同策略)：“边玩边学”**。
  - **核心思想**：用来学习的策略（我们想要优化的目标策略）与用来生成数据的策略（行为策略）是**同一个**。
  - **比喻**：你想学习如何成为一名顶级的 F1 赛车手。On-Policy 的方法就是你亲自上场比赛，根据自己每次比赛的经验来调整和提高驾驶技术。你犯的错误、取得的成功都直接来自于你自己的行为。
  - **典型算法：SARSA**
    - 更新规则为：

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[R_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

    - 注意这里的 a_{t+1}，它是在状态 s_{t+1} 时，智能体**实际执行**的下一个动作。整个更新过程依赖于 (s, a, r, s', a') 这条完整的路径。
- **Off-Policy (异策略)：“看别人玩，自己学”**。
  - **核心思想**：用来学习的策略（目标策略）与用来生成数据的策略（行为策略）**可以不同**。
  - **比喻**：你仍然想成为 F1 赛车手。Off-Policy 的方法是你可以观看过去所有比赛的录像，无论是舒马赫的冠军录像，还是新手的第一场比赛录像。你从这些多样化的数据中学习，总结出最优的驾驶策略，而不需要自己亲自去开每一场。
  - **典型算法：Q-Learning**
    - 更新规则为：

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[R_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right]$$

    - 关键区别在于 $\max_{a'} Q(s_{t+1}, a')$。它在更新时，总是假设在下一个状态 s_{t+1} 会选择**当前最优**的动作，而不管行为策略实际选择了哪个动作。

特性	ON-POLICY (如 SARSA)	OFF-POLICY (如 Q-LEARNING)
学习数据来源	必须是当前策略产生的数据	可以是任何策略产生的数据（经验回放）
策略关系	行为策略 = 目标策略	行为策略 ≠ 目标策略
优点	通常更稳定，收敛性较好	数据利用率高，可以从历史数据中学习

特性	ON-POLICY (如 SARSA)	OFF-POLICY (如 Q-LEARNING)
缺点	数据利用率低，探索性策略可能导致学习缓慢	方差较大，可能不稳定，收敛更困难
典型应用	需要稳定控制的场景，如机器人行走	需要高效利用数据的场景，如 AlphaGo

探索与利用的权衡 (Exploration vs. Exploitation)

- **利用 (Exploitation)**：根据现有知识，选择当前看来能获得最大奖励的动作。
- **探索 (Exploration)**：尝试新的、未知的动作，期望能发现更好的选择，从而获得长期的更大利益。

只利用可能会陷入局部最优，只探索则无法获得稳定的高回报。关键在于平衡。

如何实现策略中的随机探索？

最常用的方法是 **ϵ-greedy (epsilon-greedy) 策略**：

1. 设定一个小的概率值 ϵ（例如 0.1）。
2. 在每个状态需要做决策时：
    - 以 $1-\epsilon$ 的概率，选择当前 Q 值最高的动作（**利用**）。
    - 以 ϵ 的概率，从所有可能的动作中随机选择一个（**探索**）。

这种方法保证了智能体既能大部分时间执行最优策略，又有一定机会去探索未知的可能性。通常，ϵ 的值会随着训练的进行逐渐减小，让智能体从“好奇的探索者”慢慢转变为“自信的执行者”。

当数据流不具备马尔可夫性时

马尔可夫性质（当前状态包含了决定未来的所有必要信息）是一个强假设，在现实世界中常常不成立。

当环境是**部分可观测 (Partially Observable)**，即观测 o_t 不等于真实状态 s_t 时，我们进入**部分可观测马尔可夫决策过程 (POMDP)**。

解决方法

1. **状态增强 (State Augmentation)**
    - 将历史信息加入当前状态。例如，将过去 k 步的观测 $(o_t, o_{t-1}, \dots, o_{t-k+1})$ 拼接起来，形成新的状态表示。
    - **优点**：简单直观。
    - **缺点**：需要手动选择历史长度 k，且状态维度会急剧增加。
2. **使用循环神经网络 (Recurrent Neural Networks, RNN)**
    - RNN（特别是 LSTM 和 GRU）内部有一个“记忆单元”或“隐藏状态” h_t。
    - 每一步，RNN 接收当前观测 o_t 和上一时刻的隐藏状态 h_{t-1}，输出新的隐藏状态 h_t：

$$h_t = f(h_{t-1}, o_t)$$

    - h_t 可以看作到目前为止所有历史信息的浓缩摘要，用 h_t 替代 s_t 进行决策。
    - 即使原始环境不是马尔可夫的，由 RNN 构建的隐藏状态空间也近似满足马尔可夫性质。

- 在 LLM 中，Transformer 的注意力机制扮演类似角色，通过回顾整个对话历史理解当前输入的上下文。

强化学习的“安全绳”：KL 散度约束

在之前的讨论中，我们了解了强化学习的基本框架和不同的学习范式。现在，我们将深入探讨一个在现代深度强化学习（特别是 **RLHF** 中使用的 **PPO** 算法）中至关重要的概念：**KL 散度约束**。

你可以将它理解为在模型训练过程中，给智能体系上的一根“**安全绳**”，防止它因为一步迈得太大而“坠崖”（即策略崩溃）。

1. 问题根源：朴素策略梯度 (Vanilla Policy Gradient) 的不稳定性

策略梯度方法的核心思想很简单：直接用一个神经网络（策略网络）来输出在某个状态下应该执行各个动作的概率。然后，通过梯度上升来更新网络参数 θ，使得能够获得更高总回报的“好”动作的概率变大。

目标函数梯度的大致形式为：

$$\nabla_\theta J(\theta) \approx \mathbb{E}\Big[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \cdot A(s_t, a_t)\Big]$$

- $\pi_\theta(a_t \mid s_t)$ 是我们的策略网络。
- $A(s_t, a_t)$ 是**优势函数 (Advantage Function)**，简单理解就是动作 a_t 比平均水平好多少。

策略梯度公式推导：

首先，我们希望最大化期望总回报：

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]$$

其中 $\tau = (s_0, a_0, s_1, a_1, \ldots)$ 是轨迹，$R(\tau) = \sum_{t=0}^{T} r(s_t, a_t)$ 是总回报。对 θ 求梯度：

$$\nabla_\theta J(\theta) = \sum_{\tau} \nabla_\theta P(\tau;\theta) R(\tau)$$

使用 **log trick**：

$$\nabla_\theta P(\tau;\theta) = P(\tau;\theta) \nabla_\theta \log P(\tau;\theta)$$

于是：

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\Big[\nabla_\theta \log P(\tau;\theta) R(\tau)\Big]$$

轨迹概率可以分解为：

$$P(\tau;\theta) = \rho_0(s_0) \prod_{t=0}^{T} \pi_\theta(a_t|s_t) P(s_{t+1}|s_t, a_t)$$

因为环境转移概率不依赖 θ，梯度只作用在策略上：

$$\nabla_\theta \log P(\tau;\theta) = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

代入期望公式：

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

为了降低方差，引入**优势函数** $A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$，最终得到策略梯度公式：

$$\boxed{\nabla_\theta J(\theta) \approx \mathbb{E}\Big[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot A(s_t, a_t)\Big]}$$

- 其中 $d^\pi(s)$ 是策略 π 下状态的分布，$Q^\pi(s,a)$ 是动作价值函数，$A(s,a)$ 是优势函数。

这个简单的更新方式存在一个致命缺陷：**步长（学习率）非常敏感**。

- 如果学习率太小，训练会非常慢。
- 如果学习率太大，一次更新就可能让策略网络发生剧变。新的策略可能会表现得非常糟糕，进入一个“万劫不复”的状态，之前学到的所有知识都可能被瞬间遗忘。这就是所谓的“**策略崩溃**”或“**模型跑飞**”。

2. 解决方案：从参数空间到策略空间的约束

上面的梯度更新公式告诉了我们优化参数 θ 的**方向**，但更新的**幅度**（步长）仅由一个固定的学习率控制。这正是问题的核心：参数空间中的一小步，可能导致策略空间（即模型实际输出的动作概率分布）发生一场“地震”。

想象一下，我们只稍微调整了一下神经网络的权重，结果模型对于某个关键状态的决策从“90%概率向左”突变为“90%概率向右”。这种剧变是毁灭性的。

因此，一个更稳健的想法是：我们不应该只限制参数 θ 的更新步长，而应该**直接限制新旧两个策略分布之间的变化幅度**。我们需要一个“尺子”来衡量两个策略（概率分布）之间的差异。

这个“尺子”就是 **KL 散度 (Kullback-Leibler Divergence)**。

什么是 KL 散度？（比喻与公式）

再次请出我们的**老师傅**（旧策略 P 或 π_{old}）和**徒弟**（新策略 Q 或 π_{new}）。KL 散度衡量的是，用徒弟的视角来看老师傅的行为，会有多么“**出格**”或“**令人意外**”。

它的数学公式是：

$$D_{\text{KL}}(P \parallel Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)}$$

这个公式计算的是，在老师傅（P）看来，徒弟（Q）的做法有多么“出格”。核心是 $\log \frac{P(x)}{Q(x)}$，它衡量了在某个具体情况 x 下，师徒俩做法概率的差异有多大。最后，用老师傅的经验 $P(x)$ 作为权重再求和，意味着我们更关心**老师傅常遇到的那些情况**下的差异。

3. KL 散度的实际应用：如何“管住”徒弟

现在我们有了一把叫做“KL散度”的尺子，可以衡量徒弟有没有跑偏。TRPO 和 PPO 就是两种主流的解决方案。

(1) TRPO：设定一个“规矩”

TRPO (Trust Region Policy Optimization) 的方法是**直接立规矩**。它对徒弟说：“你可以自由学习和改进，但我给你定一个**绝对不能超过的纪律红线**。”

数学公式：

$$\text{maximize}_\theta \ \mathbb{E}\Big[\frac{\pi_\theta(a \mid s)}{\pi_{\theta_{\text{old}}}(a \mid s)} A_{\theta_{\text{old}}}(s,a)\Big], \quad \text{subject to } \mathbb{E}[D_{\text{KL}}(\pi_{\theta_{\text{old}}}(\cdot \mid s) \parallel \pi_\theta(\cdot \mid s))] \le \delta$$

- **目标 (maximize)**：让徒弟学习能带来“惊喜”（好的优势 A）的操作。
- **约束 (subject to)**：徒弟的新方法 (π_θ) 和老师傅的方法 ($\pi_{\theta_{\text{old}}}$) 之间的平均差异度 ($D_{\text{KL}}$) 必须小于一个很小的数 δ。

(2) PPO：用一个“橡皮筋”来拉住

PPO (Proximal Policy Optimization) 给徒弟系上了一根“橡皮筋”，通过截断目标函数实现：

$$L_{\text{CLIP}}(\theta) = \mathbb{E}\Big[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)\Big]$$

- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$：徒弟和师傅做法的比率。
- A_t：动作带来的“惊喜”好坏。
- $\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)$：限制比率在小区间 [1-ϵ, 1+ϵ] 内，即“橡皮筋”。

(3) 在 RLHF 中的直接应用

在训练大语言模型时，KL 散度的作用更加直接。模型是一个**想拿高分的学生**（π_{RL}），它最初有一个**基础老师**版本（π_{SFT}），还有一个**评分系统**（奖励模型 R_{RM}）。

最终评分公式：

$$R_{\text{final}} = R_{\text{RM}} - \beta \cdot D_{\text{KL}}(\pi_{\text{RL}} \parallel \pi_{\text{SFT}})$$

- R_{RM}：评分系统的分数。
- $D_{\text{KL}}(\pi_{\text{RL}} \parallel \pi_{\text{SFT}})$：学生答案与老师范文的差异。
- β：控制“老师严格程度”的参数。

总结

算法	核心思想	优点	缺点
Vanilla PG	直接沿梯度方向更新策略	简单，理论基础	极其不稳定，对学习率敏感
TRPO	在 KL 散度定义的信任区域内更新	非常稳定，理论保证	计算复杂，难以实现
PPO	用截断的目标函数间接限制更新幅度	稳定，效果好，实现简单	理论保证不如 TRPO 强

KL 散度及其在 PPO 中的变体，是解决策略梯度方法不稳定性的关键技术。它通过限制策略更新幅度，确保学习过程平稳高效，是从理论走向实践的重要一步。

强化学习进阶

模块一：基础准备 - PPO 与 RLHF 的数学语言

引言：在我们深入 DPO 的巧妙之处前，必须先用统一的数学语言来精确描述 RLHF 的目标。DPO 并非凭空创造，而是站在 PPO 的肩膀上，通过精妙的代数变换解决了 PPO 流程繁琐的核心痛点。因此，理解 PPO 在做什么、优化什么，是理解 DPO 为何有效的关键。

1.1 RLHF 的最终目标

在 RLHF 的第三阶段，我们的核心任务是利用 PPO 算法，根据奖励模型 (RM) 的反馈来微调 SFT 模型。这个过程的最终优化目标可以表示为以下函数：

$$\text{Objective}(\pi_\theta) = \mathbb{E}_{(x,y)\sim D_{\pi_{\text{SFT}}}}\Big[R(x,y) - \beta \log \frac{\pi_{\text{SFT}}(y|x)}{\pi_\theta(y|x)}\Big]$$

让我们来手撕这个公式的每一个部分：

- $\pi_\theta(y|x)$：这是我们**正在训练的策略模型**。它的参数是 θ。给定一个 prompt x，它会生成一个回答 y。我们的目标就是找到最优的参数 θ。

- $\pi_{\text{SFT}}(y|x)$：这是**参考模型**，通常就是训练好但未进行强化学习的 SFT 模型。它在这里的作用是作为“锚点”或“基准”。
- $R(x, y)$：这是**奖励函数**。它由一个独立的奖励模型 (RM) 提供，用于评估在 prompt x 下，生成的回答 y 有多好。$R(x, y)$ 的值越高，说明回答质量越好，越符合人类偏好。
- $\mathbb{E}_{(x,y)\sim D_{\pi_{\text{SFT}}}}$：这个期望符号表示，我们从 SFT 模型生成的 (x, y) 数据分布中进行采样，来计算括号内表达式的平均值。
- 核心部分 1: $R(x, y)$ (奖励最大化)

  这部分非常直观：我们希望调整策略 π_θ，使得它生成的回答 y 能够从奖励模型那里获得尽可能高的分数。这是驱动模型向“好”的方向优化的主要动力。
- **核心部分 2: $-\beta \log \frac{\pi_{\text{SFT}}(y|x)}{\pi_\theta(y|x)}$ (KL 散度惩罚)**
  - $\log \frac{\pi_{\text{SFT}}(y|x)}{\pi_\theta(y|x)}$ 正是 **KL 散度** $D_{\text{KL}}(\pi_\theta || \pi_{\text{SFT}})$ 的一种形式。它衡量了我们正在训练的模型 π_θ 和原始 SFT 模型 π_{SFT} 之间的“距离”。
  - **为什么要有这一项？** 这是为了防止“模型跑飞”。如果我们只看重奖励 $R(x, y)$，模型可能会为了迎合奖励模型而生成一些奇怪、不自然的文本（即所谓的“奖励过拟合”）。这一项就像一根**安全绳**，它会惩罚那些与原始 SFT 模型风格差异过大的生成结果。
  - β 是一个超参数，控制着这根“安全绳”的松紧程度。β 越大，惩罚越强，模型就越倾向于保守，不敢偏离 SFT 模型太远。

关键问题：

这个目标函数看起来很完美，但有一个致命的问题：奖励 $R(x, y)$ 是一个黑盒。它是由另一个神经网络（奖励模型）给出的，我们并不知道它的确切函数形式。PPO 的流程就是，先花大力气训练一个奖励模型，然后把它当作一个固定的“裁判”来用。

1.2 奖励模型 (RM) 的学习目标

现在，我们来看看这个“裁判”——奖励模型 (RM) 是如何训练出来的。

- 偏好数据

  RM 的训练数据不是简单的 (输入, 输出) 对，而是一种偏好对 (Preference Pair)，其形式为 (x, y_w, y_l)。
  - x：输入的 prompt。
  - y_w (winner)：人类标注员认为**更好**的回答。
  - y_l (loser)：人类标注员认为**更差**的回答。
- Bradley-Terry 模型

  为了从偏好数据中学习，我们需要一个数学模型来描述“偏好”这件事。Bradley-Terry 模型是一个经典的选择，它假设：任何两个选项被偏好的概率，可以由它们各自的潜在“分数”之差的 Sigmoid 函数来表示。

  应用到 RLHF 中，这个模型可以写成：

  $$P(y_w \succ y_l|x) = \sigma(r(x, y_w) - r(x, y_l))$$
  - $P(y_w \succ y_l|x)$：给定 prompt x，人类更偏好 y_w 而不是 y_l 的概率。
  - $r(x, y_w)$ 和 $r(x, y_l)$：分别是奖励模型为 y_w 和 y_l 打出的潜在分数。
  - $\sigma(z) = \frac{1}{1+e^{-z}}$：是我们熟悉的 Sigmoid 函数，它能将任意实数映射到 $(0, 1)$ 区间，正好可以表示概率。

  这个公式非常直观：两个回答的分数差距越大，其中一个比另一个更好的概率就越接近 1。
- RM 的损失函数

  我们的目标是训练奖励模型 r，使其预测的偏好概率 $P(y_w \succ y_l|x)$ 与人类标注的数据尽可能一致。在机器学习中，这通常通过最大化对数似然来实现。

对于一个数据点 (x, y_w, y_l)，其对数似然为 $\log P(y_w \succ y_l|x)$。为了将其转化为一个最小化的损失函数，我们取其负数。因此，RM 的最终损失函数就是：

$$L_{\text{RM}} = -\mathbb{E}_{(x,y_w,y_l)}\big[\log P(y_w \succ y_l|x)\big] = -\mathbb{E}_{(x,y_w,y_l)}\big[\log \sigma(r(x,y_w) - r(x,y_l))\big]$$

我们通过梯度下降最小化这个 L_{RM}，就能训练出奖励模型 r。

承上启下：DPO 的突破点

至此，我们理清了 PPO 的两步走流程：

1. **第一步**：用 L_{RM} 训练一个奖励模型 $r(x,y)$。
2. **第二步**：将训练好的 $r(x,y)$ 视为固定奖励，代入 1.1 的目标函数，用 PPO 算法进行强化学习。

这个过程不仅复杂、计算昂贵，而且两个阶段是割裂的。

DPO 的核心思想（我们将在下一章推导）：我们能否找到一种方法，将 1.1 的 RL 目标和 1.2 的 RM 目标巧妙地结合起来，从而**绕过显式的奖励模型训练**，直接用偏好数据 (x, y_w, y_l) 来优化策略模型 π_θ 呢？

答案是可以的，而连接这两个世界的桥梁，正是我们下一章要推导的核心关系式。

模块二：核心突破 - DPO 的公式推导

引言：
在第一章中，我们明确了 PPO 的优化目标和奖励模型（RM）的学习目标是两个独立的阶段。DPO 的天才之处在于，它证明了这两个阶段可以被一个单一的、等效的损失函数所取代。本章的使命就是完整地推导出这个损失函数，揭示其背后的数学之美。

2.1 关键洞察：从 PPO 目标反推最优策略

我们的起点是第一章中 RLHF 的最终目标函数。第一个问题是：

> *给定一个确切的奖励函数 $R(x,y)$，满足这个目标的最优策略 $\pi^*(y \mid x)$ 应该是什么样的？*

回顾目标函数：

$$\text{Objective}(\pi_\theta) = \mathbb{E}_{(x,y)\sim D^{\pi_{\text{SFT}}}}\left[R(x,y) - \beta \log \frac{\pi_{\text{SFT}}(y \mid x)}{\pi_\theta(y \mid x)}\right]$$

通过变分法求解，其最优策略为（DPO 论文关键结论）：

$$\pi^*(y \mid x) = \frac{1}{Z(x)} \pi_{\text{SFT}}(y \mid x) \exp(\beta R(x,y))$$

解释：

- $\pi^*(y \mid x)$：最优策略
- $\pi_{\text{SFT}}(y \mid x)$：参考策略
- $\exp(\beta R(x,y))$：奖励加权项，奖励越高概率越大
- $Z(x) = \sum_y \pi_{\text{SFT}}(y \mid x) \exp(\beta R(x,y))$：归一化因子

> *直观意义：最优策略在 SFT 的基础上，对高奖励回答提高概率，对低奖励回答降低概率。β 控制奖励对策略的影响程度。*

2.2 关键转换：用策略表示奖励

从“奖励”推导出“最优策略”后，DPO 逆向操作：**用策略表示奖励**。

两边取对数：

$$\log \pi^*(y \mid x) = -\log Z(x) + \log \pi_{\text{SFT}}(y \mid x) + \beta R(x, y)$$

移项解出 $R(x, y)$：

$$\beta R(x, y) = \log \pi^*(y \mid x) - \log \pi_{\text{SFT}}(y \mid x) + \log Z(x)$$

简化写成：

$$R(x, y) = \beta \log \frac{\pi^*(y \mid x)}{\pi_{\text{SFT}}(y \mid x)} + \text{常数项}$$

> *这个核心关系式告诉我们：奖励可以由最优策略和参考策略的对数概率比值线性表示，为消除独立奖励模型铺平了道路。*

2.3 收官之战：代入 RM 损失函数

回顾 RM 的损失函数：

$$L_{\text{RM}} = -\mathbb{E}_{(x, y_w, y_l)} \big[\log \sigma(R(x, y_w) - R(x, y_l)) \big]$$

将上一步的 $R(x, y)$ 代入：

$$\begin{aligned} R(x, y_w) - R(x, y_l) &= (\beta \log \pi_{\text{SFT}}(y_w \mid x) \pi^*(y_w \mid x) + \beta \log Z(x)) \\ &\quad -(\beta \log \pi_{\text{SFT}}(y_l \mid x) \pi^*(y_l \mid x) + \beta \log Z(x)) \\ &= \beta \log \pi_{\text{SFT}}(y_w \mid x) \pi^*(y_w \mid x) - \beta \log \pi_{\text{SFT}}(y_l \mid x) \pi^*(y_l \mid x) \end{aligned}$$

> *归一化项 $\beta \log Z(x)$ 消掉了！*

将 π^* 替换为当前策略 π_θ：

$$L_{\text{DPO}}(\pi_\theta; \pi_{\text{SFT}}) = -\mathbb{E}_{(x, y_w, y_l) \sim D} \Big[\log \sigma \big(\beta \log \pi_{\text{SFT}}(y_w \mid x) \pi_\theta(y_w \mid x) - \beta \log \pi_{\text{SFT}}(y_l \mid x) \pi_\theta(y_l \mid x) \big) \Big]$$

结论与解读

1. **没有奖励模型**：公式中已经完全没有 $R(x, y)$
2. **只依赖策略**：计算仅需当前策略 π_θ 和参考策略 π_{SFT}
3. **单一优化目标**：合并了 RLHF 的两个阶段（训练 RM + PPO）
4. **训练方式简单**：直接用偏好数据 (x, y_w, y_l) 通过梯度下降最小化损失，类似 SFT 微调

> *通过三步推导，我们从 PPO 复杂目标出发，得到 DPO 简洁且强大的损失函数，实现 RLHF 算法的一次重大飞跃。*

模块三：思想进阶 - GRPO 的公式推导

引言：DPO 通过其简洁的损失函数，极大地简化了 RLHF 的流程，成为 PPO 的有力替代方案。然而，任何算法都有其适用范围和局限性。GRPO (Group Reward Policy Optimization) 的提出，正是为了解决 DPO 在处理偏好数据时的一个核心问题。本章将带你理解 DPO 的局限性，并完整推导出 GRPO 的损失函数。

3.1 DPO 的局限性

DPO 的整个理论基础都建立在**成对比较 (Pairwise Comparison)** 之上，即数据点总是 (x, y_w, y_l) 的形式。这种形式虽然简单，但在现实世界的标注中存在一个问题：

信号可能很弱或有噪声。

想象以下几种情况：

1. **“差” vs “更差”**：标注员得到的两个回答 y_w 和 y_l 可能质量都很差。虽然 y_w 相对好一点，但模型从这种“矮子里面拔将军”的比较中学到的信号非常微弱，甚至可能学到一些不好的模式。
2. **“好” vs “也挺好”**：两个回答质量都很高，只是风格略有不同。这种情况下，y_w 和 y_l 的奖励差异非常小，梯度信号也很微弱。
3. **标注噪声**：人类标注员的判断是主观的，有时可能会出现错误或不一致的标注。DPO 对这种单个数据点的噪声比较敏感。

这些情况的共同点是，仅仅依赖单个的 `chosen` vs `rejected` 对，可能会让模型在学习过程中受到噪声的干扰，或者因为信号太弱而学习效率不高。

3.2 GRPO 的核心思想：从“单挑”到“群战”

GRPO 的解决方案非常直观：既然单个比较不可靠，那我们就**比较两组**！

- **新的数据形式**：GRPO 的训练数据不再是成对的，而是成组的 (x, Y_w, Y_l)。
  - x：输入的 prompt。
  - $Y_w = \{y_{w1}, y_{w2}, \ldots, y_{wm}\}$：这是一组被认为是**好的**回答（Group of Winners）。
  - $Y_l = \{y_{l1}, y_{l2}, \ldots, y_{ln}\}$：这是一组被认为是**坏的**回答（Group of Losers）。
- **分组奖励假设**：

  GRPO 扩展了 Bradley-Terry 模型，它不再假设单个回答的奖励有差异，而是假设“好回答组”的平均奖励高于“坏回答组”的平均奖励。

  这个假设用数学公式表达为：

  $$P(Y_w \succ Y_l \mid x) = \sigma(\bar{r}(x, Y_w) - \bar{r}(x, Y_l))$$

  其中，$\bar{r}(x, Y)$ 代表一组回答的平均奖励：

  $$\bar{r}(x, Y) = \frac{1}{|Y|} \sum_{y \in Y} r(x, y)$$

> *这个核心思想通过聚合一组样本的奖励，有效地平滑了单个样本可能带来的噪声，从而获得更稳定、更鲁棒的学习信号。*

3.3 GRPO 的损失函数推导

GRPO 的推导过程与 DPO 惊人地相似，我们只需将“单挑”的逻辑替换为“群战”即可。

1. **起点**：GRPO 的损失函数与 RM 损失函数形式相同，只是奖励变成了组的平均奖励：

   $$L_{\text{GRPO}} = -\mathbb{E}_{(x, Y_w, Y_l)} \left[\log \sigma(\bar{r}(x, Y_w) - \bar{r}(x, Y_l)) \right]$$

2. **代入奖励表达式**：使用模块二中推导出的核心关系式：

   $$r(x, y) = \beta \log(\pi_{\text{SFT}}(y \mid x) \pi^*(y \mid x)) + \text{常数项}$$

   将其代入平均奖励差 $\bar{r}(x, Y_w) - \bar{r}(x, Y_l)$：

$$\bar{r}(x, Y_w) - \bar{r}(x, Y_l) = \frac{1}{|Y_w|} \sum_{y_w \in Y_w} r(x, y_w) - \frac{1}{|Y_l|} \sum_{y_l \in Y_l} r(x, y_l)$$

代入 $r(x, y)$ 的表达式，并注意到常数项在相减后消掉：

$$= \frac{1}{|Y_w|} \sum_{y_w \in Y_w} \beta \log(\pi_{\text{SFT}}(y_w \mid x) \pi^*(y_w \mid x)) - \frac{1}{|Y_l|} \sum_{y_l \in Y_l} \beta \log(\pi_{\text{SFT}}(y_l \mid x) \pi^*(y_l \mid x))$$

3. **最终损失函数**：将 π^* 替换为当前策略 π_θ：

$$L_{\text{GRPO}}(\pi_\theta; \pi_{\text{SFT}}) = -\mathbb{E}_{(x, Y_w, Y_l) \sim D} \left[\log \sigma \Big(\frac{\beta}{|Y_w|} \sum_{y_w \in Y_w} \log(\pi_{\text{SFT}}(y_w \mid x) \pi_\theta(y_w \mid x)) - \frac{\beta}{|Y_l|} \sum_{y_l \in Y_l} \log(\pi_{\text{SFT}}(\right.$$

结论与解读

1. **DPO 的自然扩展**：GRPO 只是将 DPO 中的单个对数概率比，换成了组内所有样本的**平均对数概率比**。
2. **鲁棒性增强**：求平均操作使得模型不再聚焦于单个 (y_w, y_l) 对，而是优化整体组间差异，对标注噪声不敏感。
3. **实现简单**：在实现上，GRPO 只是比 DPO 多了一步对组内样本循环求平均，改动成本很小。

大模型强化学习QA

引言：本手册旨在成为一份全面、深入的大模型强化学习参考指南。我们将通过超过40个精心设计的问题，系统性地梳理从经典RLHF到前沿GRPO算法的核心概念、关键差异、实践挑战与未来方向，助你构建一个完整、扎实的知识体系。

第一部分：基础原理与框架 (Foundations & Framework)

Q1: 什么是强化学习 (RL)？它和普通监督学习有何不同？

A1:

- **强化学习 (RL)**：一种通过“试错”进行学习的范式。智能体 (Agent) 在与环境 (Environment) 的交互中，根据获得的奖励 (Reward) 或惩罚来调整其行为策略 (Policy)，目标是最大化长期累积奖励。
- **核心区别**：监督学习依赖带有明确“正确答案”的标签数据进行训练；而强化学习没有固定答案，它通过探索来发现能带来最高奖励的行为序列，更侧重于序贯决策。

Q2: 什么是 RLHF？它为什么要引入如此复杂的 RL 框架？

A2:

- **RLHF (Reinforcement Learning from Human Feedback)**：是一个将强化学习应用于 LLM 对齐的框架。核心思想是：先用人类偏好数据训练一个**奖励模型 (RM)** 来模仿人类的判断标准，然后用这个 RM 作为“环境”，通过 RL 算法（如 PPO）来优化 LLM 的生成策略。
- **引入原因**：因为“好的回答”是一个模糊、主观且复杂的概念，无法用简单的规则或损失函数来定义。RLHF 通过学习一个 RM，将这个模糊的人类偏好转化为了可优化的数学奖励信号，从而解决了对齐任务中“目标函数难以定义”的核心问题。

Q3: 在 LLM 的 RLHF 中，MDP（马尔可夫决策过程）的各个元素分别是什么？

A3:

- **状态 (State)**：到目前为止的**全部对话历史** (Prompt + 已生成的 Tokens)。
- **动作 (Action)**：模型在当前状态下，从词汇表中选择并生成的**下一个 Token**。

- **奖励 (Reward)**：通常是在整个回答**生成完毕后**，由 RM 对**完整回答**进行一次性打分。这是一个稀疏奖励。
- **策略 (Policy)**：**LLM 本身**，即 $\pi\theta(a \mid s)$，给定状态 s 生成下一个 Token a 的概率分布。

Q4: 为什么不能直接用 SFT (监督微调) 代替 RLHF？

A4: SFT 使用高质量的 (prompt, response) 数据对，教模型“如何回答”。但它有两大局限：

1. **数据成本极高**：编写大量高质量范例回答的成本，远高于仅仅判断两个回答哪个更好的成本。
2. **缺乏探索性**：SFT 只能让模型模仿数据集中已有的模式，无法探索和发现数据集中不存在的、但可能更好的回答方式。RLHF 则可以通过探索机制找到更优的解。

Q5: SFT 和 RLHF 的关系是什么？是替代还是互补？

A5: 绝对是互补关系，且 SFT 是 RLHF 的必要前置步骤。

- **SFT 是基础**：SFT 负责教模型“听懂人话”，即理解指令并生成符合格式、有基本逻辑的回答。它为模型注入了基础的指令遵循能力。没有 SFT，一个预训练模型直接进行 RLHF 会非常困难，因为它可能根本无法生成有意义的回答来获得奖励。
- **RLHF 是进阶**：在 SFT 的基础上，RLHF 负责教模型“说好听的话”，即根据人类的复杂偏好（如无害性、帮助性、风格）来优化回答的质量。它是一种更精细的“品味”调优。

Q6: 什么是“策略” (Policy) 和“价值函数” (Value Function)？

A6:

- **策略 (Policy)**：即 LLM 本身，它是一个从状态到动作的映射。具体来说，它定义了在给定对话历史（状态）下，模型生成每个可能 token（动作）的概率。
- **价值函数 (Value Function)**：用于评估一个状态或“状态-动作”对的长期价值。
    - **V(s)**：状态价值函数，表示从状态 s 出发，遵循当前策略，未来能获得的期望总奖励。
    - Q(s, a)：动作价值函数，表示在状态 s 执行动作 a 后，再遵循当前策略，未来能获得的期望总奖励。

      在 PPO 中，价值函数由 Critic 网络学习；而在 DPO/GRPO 中，它们被隐式地绕过了。

第二部分：经典 RLHF 流程 (PPO-based)

Q7: 经典 RLHF (基于 PPO) 的完整流程是怎样的？

A7: 分为三个核心阶段：

1. **监督微调 (SFT)**：使用高质量的指令和回答数据，对预训练 LLM 进行微调，使其初步具备理解和执行指令的能力。产出 SFT 模型。
2. **奖励模型训练 (RM Training)**：用 SFT 模型对同一个 prompt 生成多个回答，由人类标注员对这些回答进行排序或选出最优/最差。利用这些偏好数据 `(prompt, chosen, rejected)` 训练一个奖励模型，使其能模仿人类的判断标准。
3. **PPO 强化学习**：将 SFT 模型作为初始策略，用 RM 作为奖励函数。通过 PPO 算法，让策略模型不断生成回答、获得 RM 的奖励、然后更新自身参数，最终得到对齐后的模型。

Q8: PPO 训练阶段为什么需要 4 个模型？它们分别是什么？

A8: 这是 PPO 资源消耗大的根源：

1. **Actor (策略模型)**：正在被训练和更新的模型，负责生成回答。
2. **Reference Model (参考模型)**：通常是初始的 SFT 模型，保持参数固定。用作计算 KL 散度的“锚点”，防止 Actor 跑飞。
3. **Reward Model (奖励模型)**：参数固定，负责给 Actor 生成的回答打分。
4. **Critic (价值模型)**：负责评估当前状态的“价值”，用于计算优势函数 (Advantage)，以稳定 PPO 的训练过程。

Q9: 什么是优势函数 (Advantage Function)？Critic 模型的作用是什么？

A9:

- **优势函数 A(s,a)**：衡量在状态 s 下，执行动作 a 相对于当前策略的平均水平“好多少”。即 $A(s,a)=Q(s,a)-V(s)$。一个正的优势意味着这个动作比平均表现要好。
- **Critic 模型的作用**：就是为了学习和预测价值函数 V(s)。在 PPO 中，通过计算优势函数，可以大大降低策略梯度的方差，使得训练过程更稳定、收敛更快。可以说，Critic 是 PPO 训练的“稳定器”。

Q10: 什么是 GAE (广义优势估计)？

A10: GAE (Generalized Advantage Estimation) 是 PPO 中计算优势函数的一种先进技术。它通过一个参数 λ 来平衡偏差和方差，巧妙地结合了不同时间步的奖励信息，从而得到比简单计算更准确和稳定的优势估计，是提升 PPO 性能的关键组件之一。

Q11: PPO 是 on-policy 还是 off-policy 算法？这在 LLM 训练中意味着什么？

A11:

- **PPO 本质上是 on-policy 算法**。这意味着它理论上需要用当前正在优化的策略（Actor）所产生的数据来进行训练。
- **在 LLM 训练中的妥协**：然而，每次更新策略后都重新生成所有数据，成本太高。因此，PPO 引入了**重要性采样 (Importance Sampling)** 技术，允许它在一定程度上重用旧策略产生的数据，但会对新旧策略的概率比率进行限制（通过 Clip），从而在 on-policy 和 off-policy 之间做了一个巧妙的平衡，使其既高效又稳定。

Q12: 经典 RLHF 流程有哪些核心痛点？

A12:

1. **流程复杂**：SFT→RM→PPO 三个独立阶段，流程长，调试难，迭代慢。
2. **成本高昂**：PPO 阶段需要同时维护 4 个模型，对显存要求极高。
3. **训练不稳定**：PPO 对超参数敏感，训练过程容易崩溃。
4. **数据成本高**：人工标注偏好数据成本高，难以规模化。

第三部分：算法演进 (DPO & GRPO)

Q13: DPO 是如何解决 PPO 上述痛点的？

A13: DPO 通过两大核心创新：

1. **简化流程**：在数学上将“训练 RM”和“PPO 优化”合并为**单一损失函数**，将三阶段流程缩短为两阶段 (SFT → DPO)。
2. **降低资源**：将在线 RL 转为离线优化，训练时**不再需要独立的奖励和价值模型**，显存占用减少约一半。

Q14: DPO 的损失函数和对比学习 (Contrastive Learning) 有关系吗？

A14: 有很强的联系。DPO 的损失函数可以被看作是对比学习思想的一种体现。它通过最大化 chosen 回答的概率（正样本）和最小化 rejected 回答的概率（负样本），来拉开两者在模型概率空间中的距离。这种“拉近正样本，推远负样本”的思想，正是对比学习的核心。

Q15: GRPO 的核心思想是什么？它相比 DPO 改进了什么？

A15:

- **核心思想**：将 DPO 的**成对比较 (Pairwise)** 升级为**分组比较 (Groupwise)**。它不再比较哪个回答更好，而是比较哪**一组**回答更好。
- **改进之处**：DPO 对单个标注噪声敏感（例如，“差”vs“更差”的比较信号很弱）。GRPO 通过比较两组回答的“平均水平”，聚合了更强的学习信号，使得训练更稳定、对噪声数据更鲁棒。

Q16: GRPO 与 PPO 的核心差异是什么？

A16:

- **优势估计方式**：
    - **PPO**：依赖一个独立的 **Critic 网络**来学习价值函数 V(s)，并计算优势 A(s,a)。
    - **GRPO**：**无需 Critic 网络**。它开创性地使用**组内奖励的标准化**来直接计算相对优势，即一个回答的奖励高出同组平均水平多少。
- **资源效率**：GRPO 因为移除了 Critic 网络，显著降低了计算和显存开销。

Q17: GRPO 相比 PPO 有哪些具体的优劣势？

A17:

- **优势**：
    a. **资源节约 (核心)**：移除 Critic，显著降低显存占用和计算量。
    b. **过程奖励**：组内对比的范式，非常适合对“过程正确性”（如代码步骤）建模。
- **劣势**：
    a. **依赖样本质量**：组内样本多样性差或质量低时，优势估计会失真。
    b. **探索能力**：理论上，探索新状态的能力可能不如带 Critic 的 PPO。

Q18: 为什么 DeepSeek 在训练代码模型时会选择 GRPO？

A18: 因为任务特点与算法优势高度契合：

1. **奖励规则化**：代码任务的奖励信号明确（如单元测试通过与否），无需复杂的 Critic 评估。
2. **资源优化**：训练千亿级代码模型，GRPO 带来的显存节省至关重要。
3. **过程监督**：代码的结构、风格等过程信号，很适合用 GRPO 的分组比较来学习。

第四部分：关键技术细节 (In-depth Mechanisms)

Q19: KL 散度约束 (KL Penalty) 的作用是什么？如果 β=0 会怎样？

A19:

- **作用**：KL 散度是 RLHF 的**“安全绳”**。它通过惩罚当前策略与初始 SFT 策略的差异，起到三大作用：1) **防止模型遗忘** SFT 阶段学到的语言能力；2) **保证生成多样性**；3) **稳定训练过程**。
- **如果 β=0**：相当于解开安全绳，模型会不惜一切代价迎合奖励目标，导致**奖励过拟合 (Reward Overfitting)**，生成内容可能在人类看来质量严重下降。

Q20: PPO/GRPO 中的 Clip 方法是什么？和 KL Penalty 有何区别？

A20:

- **Clip 方法**：PPO/GRPO 中另一种约束策略更新的机制。它通过**硬截断**的方式，直接限制新旧策略的概率比率 $r_t(\theta)$ 不能超过 $[1-\epsilon, 1+\epsilon]$ 这个区间。
- 区别：
    - **Clip**：**硬约束**，直接设定更新边界，实现简单。
    - **KL Penalty**：**软约束**，通过惩罚项引导更新方向，更灵活。

Q21: GRPO 为何要同时使用 Clip 和 KL Penalty？

A21: 对于超大规模模型训练，单一约束可能不够。同时使用 Clip 和 KL Penalty 可以形成双重保险，极大地增强了训练的稳定性，是一种追求极致稳健的工程选择。

Q22: GRPO 的组大小 (N) 如何影响训练效果？

A22:

- **小 N (如 16)**：优势估计方差大，不稳定，但资源友好。适合简单任务。
- **大 N (如 128)**：优势估计更稳定，但资源消耗大。适合复杂任务。

Q23: 如何缓解 GRPO 对样本质量的依赖问题？

A23:

1. **数据预处理**：在构建分组时，通过聚类去重等方法，确保组内样本的多样性。
2. **混合训练**：大部分时间用 GRPO，但周期性地引入 Critic 校准优势估计。

Q24: 在 GRPO 中，当 reward 突然下降时，可能的原因是什么？

A24: 这通常与优势估计失真有关：

1. **批次数据质量差**：当前训练批次中的分组恰好质量很差或多样性不足，导致计算出的平均奖励基准有偏，使得优势估计出现大的负值，策略被错误地惩罚。
2. **学习率过大**：即使有双重约束，过大的学习率仍然可能导致策略更新步子太大，跳出了最优区域。
3. **KL 惩罚系数不当**：β 值设置不合理，可能导致对策略的约束过强或过弱。

Q25: 在 DPO/GRPO 中，参考模型 π_{SFT} 可以被更新吗？比如用 EMA (Exponential Moving Average)？

A25: 可以，这是一种常见的进阶技巧。

- **固定参考模型 (标准做法)**：优点是稳定，KL 散度的计算基准始终不变。缺点是如果训练时间很长，策略模型 π_θ 可能会离固定的 π_{SFT} 越来越远，导致 KL 惩罚过大，限制了模型的进一步优化。
- **EMA 更新参考模型**：优点是可以让“锚点”随着训练的进行而缓慢移动，使得 KL 约束总是施加在策略模型的“附近”，让训练更平滑。缺点是可能会引入一些不稳定性。这是一种在稳定性和持续优化之间的权衡。

第五部分：实践与展望

Q26: 除了 PPO/DPO/GRPO，还有哪些重要的对齐算法？

A26:

- **KTO (Kahneman-Tversky Optimization)**：进一步简化数据要求，不再需要成对的偏好数据，只需要知道哪些是“好”的回答，哪些是“坏”的回答。
- **Constitutional AI (CAI)**：让 AI 自己帮助自己对齐。通过定义一套“宪法”（原则），让模型自己判断和修改回答，并生成偏好数据来微调自己。
- **RAG (Retrieval-Augmented Generation)**：通过将模型的回答“锚定”在可信的外部知识源上，在“真实性”维度上实现对齐。

Q27: 什么是“奖励过拟合” (Reward Overfitting / Hacking)？如何缓解？

A27:

- **定义**：指模型学会了利用奖励模型的漏洞来获得高分，但其生成的回答在人类看来质量很差。
- **缓解方法**：
    a. **加强 KL 约束**：提高 β 值，让模型不敢偏离 SFT 模型太远。
    b. **高质量 RM**：训练一个更强大、更不容易被欺骗的奖励模型。
    c. **多样的 RM 集成**：使用多个不同的奖励模型进行集成打分。
    d. **人类评估**：在训练过程中定期进行人工评测，及时发现并纠正问题。

Q28: 什么是“对齐税” (Alignment Tax)？

A28: 指的是在进行对齐（如 RLHF）之后，模型在某些基础能力（如常识推理、知识问答、代码生成）上性能出现下降的现象。这可以理解为，模型为了学习人类偏好，牺牲了一部分在预训练和 SFT 阶段学到的通用能力。如何在对齐的同时最小化对齐税，是当前研究的一个热点。

Q29: RLHF/DPO 会损害模型的知识或推理能力吗？

A29: 有可能，这就是“对齐税”的具体体现。 如果奖励模型或偏好数据过分强调“无害性”或某种特定风格，而忽略了事实性或逻辑性，那么模型在优化过程中就可能学会生成更“安全”但更“愚蠢”的回答，从而损害其知识和推理能力。平衡不同对齐目标是关键。

Q30: 如何客观地评估一个对齐后的模型？

A30: 这是一个核心难题，通常采用组合评估：

1. **自动化评估**：使用奖励模型打分，或在标准 benchmark（如 MT-Bench, AlpacaEval）上进行评估。
2. **人工盲测**：这是黄金标准。让人类裁判在不知道模型来源的情况下，对不同模型的回答进行成对比较或打分。
3. **红队测试 (Red Teaming)**：专门寻找模型的弱点和漏洞，测试其安全性、鲁棒性和有害性。

Q31: 如何处理多轮对话中的对齐问题？

A31: 多轮对话的对齐更具挑战性，因为奖励不仅取决于当前回答，还取决于整个对话的连贯性、记忆性和上下文理解能力。

- **奖励设计**：RM 需要能评估整个对话历史，而不仅仅是最后一轮。
- **数据收集**：偏好数据需要以整个对话为单位进行标注。
- **模型能力**：模型本身需要有更强的长上下文处理能力。

Q32: 未来的对齐技术可能会朝哪个方向发展？

A32:

1. **更高效的数据**：从需要昂贵人类偏好的 RLHF，到只需要“好/坏”标签的 KTO，未来可能会出现对数据要求更低的算法。

2. **AI 辅助对齐**：类似 Constitutional AI，更多地利用 AI 自身来监督和对齐 AI，降低对人类的依赖。
3. **可解释性与可控性**：不仅仅是对齐结果，更要理解模型为何这样回答，并能对模型的行为进行更精细的控制。
4. **多模态对齐**：将对齐技术从文本扩展到图像、语音、视频等多模态领域。