RL基础知识 & GRPO公式推导

导读

为彻底理解GRPO，我花费半个月时间研读资料并与AI讨论，最终整理出这份系统的笔记。

市面上的许多教程在可读性上有不足。为此，本文遵循以下原则撰写：

- 聚焦本质：深入GRPO的核心本质（优势A、策略梯度、重要性采样），淡化衍生的trick。
- 紧扣现实：避免空谈概念，确保每个RL概念和公式，都能对应到LLM训练场景与代码框架。
- 直观与严谨：直接给出符合直觉的结论，仅对反直觉的关键结论（如策略梯度定理）进行推导。
- 简洁与易用：对各种公式（尤其期望的公式）既保留简洁版本，也展开为可带入数据计算的版本。
- 精简概念：剔除LLM-RL以及GRPO中不涉及的复杂概念（如非确定环境、贝尔曼方程、时序差分、价值网络、广义优势估计等），保持主线清晰。
- 方法拓展：深入理解GRPO的本质后，自然拓展到PPO、DAPO、GSPO等算法

本文大致结构：

- LLM视角下，重新审视强化学习的各种概念
  - 基础概念：状态s，动作a，奖励r，策略π，环境
  - 进阶概念：轨迹τ，折扣因子γ，收益G，价值V，Q函数，优势A
- 优势A的推导
  - 蒙特卡洛法
  - 蒙特卡洛估计GRPO中的优势A
- 强化学习的目标函数与求导
  - 强化学习的目标函数
  - 对目标函数求导（LLM视角下，策略梯度法的推导）
- 最简单版本的GRPO公式（方法本质）
  - on-policy，无重要性采样，无min-clip，无KL散度
- 完整版GRPO涉及的其他技巧
  - 技巧1：重要性采样（用于off-policy）
  - 技巧2：min-clip操作（更新要谨慎，不要过于乐观/悲观）
  - 技巧3：KL散度（同样为了防止灾难性遗忘，不单独介绍了）
- 从GRPO到其他强化学习算法
  - PPO：用神经网络估计优势A
  - GSPO：将整个序列视为完整动作
  - DAPO：附加大量trick的改良版GRPO

铭记：

- 知其然，亦知其所以然。深刻理解为什么公式长这样，有助于分析LLM训练时的异常现象
- 公式推导，不是为了苛求严谨，而是为了理解算法隐含的假设，进而分析假设是否符合实际
- 万变不离其宗，吃透GRPO的本质，就能自然拓展到其他强化学习算法

RL基础概念：S、A、R、π、环境

状态s、动作a、奖励r、策略π

在用强化学习训练LLM的语境中，有两种建模级别。

- token级建模：每个生成的token视为一个action（对应PPO）
- sequence级建模：每个完整生成序列视为一个action（对应GSPO）

> GRPO是把sequence级建模启发式地分解成token级建模，后面会详细推导

相应的，把强化学习的概念对应到LLM上：
token级建模

- 状态s：
  - 初始状态 s_1 ： prompt x
  - t时刻状态 s_t ： prompt x + response的前t个token $y_{\leq t}$
  - （此处 y_i 指的是序列 y 的第 i 个token）
- 动作a：
  - t时刻的动作 a_t ： response的第t个token y_t
- 奖励r：
  - 最后一个token：完整序列的奖励（例如答案是否正确）
  - 前面的token：奖励为0
- 策略π：
  - LLM生成第t个token的概率 $P(y_t|x, y_{\leq t-1})$

sequence级建模（只有一问一答）

- 状态s：只有初始、结束两个状态
  - 初始状态 s_1 ： prompt x
  - 结束状态 s_2 ： prompt x + response y
- 动作a：只有一步动作
  - LLM生成的response y
- 奖励r：
  - 完整序列的奖励（例如答案是否正确）
- 策略π：
  - LLM生成response y的概率 $P(y|x)$

sequence级建模（多轮问答）：常见于多轮agent的RL训练，但也可以全拆成单轮

- 状态s：
  - 初始状态 s_1 ： 初始prompt x_1
  - t时刻状态 s_t ： 初始prompt x_1 + 第1轮response y_1 + ... + 第t轮prompt x_t + 第t轮response y_t
  - （此处的 x_i, y_i 指代的是第 i 个 x,y 序列，而不是序列中第i个token）
- 动作a：
  - t时刻的动作 a_t ： 第t轮response y_t
- 奖励r：

    - t时刻的奖励 r_t ： 第t轮回复的奖励（有些方法有中间奖励），如果t是最后一轮则对应终局奖励
- 策略π：
    - LLM生成第t个response的概率 $P(y_t|x_1, y_1, ..., x_{t-1}, y_{t-1}, y_t)$

环境：确定性环境 & 概率环境

环境描述了给定状态s，执行动作a时，会转移到什么新状态s'，并且会获得多少奖励r。

- 有时s'和r都是确定的，则环境是确定性环境
- 有时s'和r是有随机性的，则环境是概率环境，此时需要用概率分布 $P(s', r|s, a)$ 来描述

具体而言又可以分为s'有随机性、r有随机性等多种不同情况。以下以“步枪打气球”为例说明。

> 步枪打气球：将枪口移动到某个坐标开枪，打中气球会爆炸
> 理解：
>
> - 可观测的量：枪口的坐标（假设有一个精确的传感器）、气球是否爆炸
> - 不可观测的量：子弹的落点坐标（假设没有相应的传感器）
>
> 形式化：
>
> - 只有一步决策，初始/结束两个状态
> - 初始状态s：枪口当前坐标 (x_1, y_1)
> - 动作a：大脑发出的指令“将枪口移动到新坐标 (x_2, y_2) 并开枪”（把移动和开枪视为一整个动作）
> - 结束状态s'：枪口实际所在的新坐标 (x_2', y_2')
> - 奖励r：气球爆炸则奖励1，否则为0
>   - 我们无法观测子弹的落点坐标，因此无法作为状态的一部分。但只要打中气球就会爆炸，因此可以用气球状态作为奖励。
>
> 随机性的来源：
>
> - s'的随机性：动作执行可能不准确，命令枪口移动到 (x_2, y_2) 和枪实际移动到 (x_2', y_2') 可能并不相同（手稳vs手抖）
> - r的随机性：动作执行效果可能不稳定，就算枪口能准确移动到 (x_2, y_2) ，但如果枪的质量不好，弹道落点是一个很大的范围（瞄准了却没打中、瞄歪却蒙中了）
>
> 具体解释：
>
> - s'随机，r确定：手抖+枪准，手无法准确移到位，但枪本身指哪打哪
> - s'确定，r随机：手稳+枪坏，手能准确移动，但枪的弹道落点范围很大
> - s'随机，r随机：手抖+枪坏
> - s'确定，r确定（确定性环境）：手稳+枪准

强化学习中的随机性：策略随机性 vs 环境随机性

强化学习中的随机性有两个来源：

- 策略随机性 $\pi(a|s)$ ：给定状态s，以根据某种概率分布随机生成动作a。（与之相对的是最优性策略：给定s，生成某个“最优”的a）

- 环境随机性 $P(s', r|s, a)$ ：给状态s和动作a，转移到的新状态s'或获得的奖励r有随机性（与之相对的是确定性环境，转移到的s'和r都确定）

标准的RL各种公式（例如贝尔曼方程）中，是按策略、环境都随机给出的。
但在LLM强化学习的语境下：

- s：prompt+已经生成的token；a：新token或新response序列
- 策略随机性：对于top-p解码，策略是随机的；对于贪婪解码，策略是确定的
- 环境随机性：s无随机性（给定s和a，s'=s拼接a）；r如果由规则判断则是确定的，如果是llm-as-judge可能有随机性（本文暂不考虑这种情况）

对于LLM的强化学习（策略随机、环境确定），比标准的强化学习（策略随机、环境随机）的设定更简单，因此各种公式会有一些简化。

马尔可夫性

很多强化学习的教程会强调马尔可夫性，但对于LLM，我们反而应该**忘记**马尔可夫性。

- 马尔可夫性： $P(a_t|s_1, a_1, ..., s_{t-1}, a_{t-1}, s_t) = P(a_t|s_t)$ ，即下一个动作的概率与历史无关。
- 但在LLM中，预测每个token的概率是 $P(y_t|x, y_{<t})$ ，即每个token是和历史有关的，因此不满足马尔科夫性
- 满足马尔可夫性、历史无关的 $P(y_t|y_{t-1})$ 是传统n-gram模型的做法，不是LLM

当然，也可以“强行”让LLM满足马尔可夫性：

- 把所有历史拼接作为一个状态，它也是满足马尔可夫性的。
- 在LLM中，s_1 是prompt x ，a_t 是response的第t个token y_t ，状态 $s_{t+1} = s_t + a_t = x + y_{\leq t}$
- 这种定义至少在形式上满足“马尔可夫性”：$P(a_t|s_1, a_1, ..., s_{t-1}, a_{t-1}, s_t) = P(a_t|s_t)$
- 虽然看起来有点耍赖

> 核心：
>
> - 在LLM中，我们最好忘记马尔可夫性
> - 非要说马尔可夫性，则LLM的“状态”指的是全量拼接历史，而不是n-gram
> - 后面关于RL的推导，不基于马尔可夫性也一样能推出来

衍生概念：G、V、A、Q

轨迹trace（或称episode）τ

$\tau = (s_1, a_1, r_1, s_2, a_2, r_2, s_3, ..., s_n, a_n, r_n, s_{n+1})$ 从起始到结束称为一个轨迹。

- 第t步的初始状态为 s_t ， 执行动作 a_t ， 此时立刻获得奖励 r_t ，并转移到下个状态 s_{t+1}
- 一共执行t=1...n，共n次行动
- $\tau \sim \pi(a|s)$ 代表从状态s开始，根据策略 π(a|s) 依次采样下一个动作，直到终局，所产生的轨迹，即“从概率π(a|s)采样的轨迹τ”
  - “从状态s开始”不代表s必须是第一个状态 s_1 ，它可以是任何一个中间状态 s_t 。根据马尔科夫性， s_t 之后的演进只和 s_t 有关，与之前如何来到 s_t 无关。

收益（gain）G 与折扣因子 γ

第t步的收益 $G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$

- 折扣因子 0≤γ≤1
- 含义：从第t步开始直到终局，累计reward之和（越未来的步骤通常越“不重要”，因此使用折扣γ衰减。γ=0 则只看当前动作的reward，γ=1 则每一步的reward都同等重要）
- G默认是到终局位置，但也可以计算k步收益： $G_t^{(k)} = r_t + \gamma r_{t+1} + ... + \gamma^{k-1} r_{t+k-1} = \sum_{i=0}^{k-1} \gamma^i r_{t+i}$ ，它在k步时序差分中会用到（本文不介绍）
- 收益（gain）有时也称作回报（return），两者在强化学习中含义相同，只是不同地方的术语有差异。这里称呼为gain，防止return和reward相似混淆

价值（value）函数 $V^{\pi}(s)$：

对于策略π，状态s的价值为：

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{\tau \sim \pi(a|s)}[G_t | S_t = s] \\ &= \mathbb{E}_{\tau \sim \pi(a|s)}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... | S_t = s] \end{aligned}$$

- 其中状态s执行动作a后，获得奖励r，状态变为s'
- **物理意义**：从状态s按策略π(a|s)采样直到终局，**预期**能累积到多少奖励。

几点注意事项：

- 这里大写的 S_t 表示第t个状态的“占位符”，小写的 s 代表状态的实际取值，可以类比“类vs实例”，正好也是类大写、实例小写
- $V^{\pi}(s)$ 和 s 处于序列的第几步无关，只和s与π的取值有关
- 同一个状态s，在不同的策略π下，价值 $V^{\pi}(s)$ 可能不同
- s_t 的价值是从 s_t 的**后续**奖励 r_t 开始算的，而不是从到达s时产生的奖励 r_{t-1} 开始算的

辨析：奖励r、收益G、价值V

- 奖励reward：第t步行动**实际产生**的**即时**“好处”，衡量**状态s+动作a**的好坏
- 收益gain：从第t步到终局**实际产生**的**累计**“好处”，衡量**轨迹τ**（或其中某个片段）的好坏
- 价值V：即从s到终局时，**预期会产生**的**累计**“好处”，衡量**状态s**的好坏

为什么引入价值V（为什么只靠奖励r不够）？

- reward稀疏：有些场景，中间的过程没有奖励，只有终局时才能一次性结算奖励（LLM生成过程就很典型，生成完才能评估是否正确，单个token没有奖励）
- reward好坏 ≠ 动作/状态好坏：即刻的奖励高，可能后续的奖励少，不一定是个好动作/好状态。

因此reward是“局部的、即刻的”，价值V是“全局的、考虑后续的”，它能补齐reward的视角：

- reward稀疏：没有reward时，价值V依然能评估出状态的好坏，并且根据转移到下个状态的V评估动作好坏
- reward好坏 ≠ 动作/状态好坏：考虑对未来的预期，平衡即刻的reward数值

关于价值V本身的一些澄清：

价值V是客观的 → 客观但不可直接获取 → 不可获取但可以估计 → 估计有误差 → 误差可以收敛

> 逐句解释：
>
> - 价值是客观的：给定策略π和状态s，价值 $V^{\pi}(s)$ 一定是个客观实际的值。（类比：概率分布的期望是个客观实际的值）
> - 客观但不可直接获取：reward和gain是可以直接获取的，但价值V是**期望**，它不能直接获取。（类比：样本的值能直接获取，但如果我们不知道它概率的准确形式，则背后的期望不能直接获取）
> - 不可获取但可以估计：有大量真实数据时，可以构造统计量来近似估计价值 $V^{\pi}(s)$ 。（类比：真实数据的平均数是个人工构造的统计量，可以近似估计概率分布的期望）
>   - 动态规划法中可以为V随机赋值，并且迭代解不动点
>   - PPO中用神经网络计算V
>   - 蒙特卡洛中构造统计量估计V
> - 估计有误差：根据真实数据估计出的价值V，数值上和客观实际的V不一定恰好相等。（类比：数据的平均数≈概率的期望，但不一定恰好相等）
> - 误差可以收敛：数据足够多/迭代步数足够多，估计出的V和客观实际的V误差可忽略。（类比：数据的平均数≈概率的期望，在数据足够多时误差可忽略）

动作价值函数 $Q^{\pi}(s,a)$

对于策略π，在状态s下，动作a的价值为：

$$\begin{aligned} Q^{\pi}(s,a) &= \mathbb{E}_{\tau\sim\pi(a|s)}[G_t|S_t=s,A_t=a] \\ &= \mathbb{E}_{\tau\sim\pi(a|s)}[r_t+\gamma r_{t+1}+\gamma^2 r_{t+2}+...|S_t=s,A_t=a] \end{aligned}$$

- 其中状态s执行动作a后，获得奖励r，状态变为s'
- **物理意义**：从状态s**明确执行动作a**后，未来再按策略π采样直到终局，**预期**能累积到多少奖励。
  - Q也要从s执行a之后产生的奖励开始计算，这点和V的计算一致

Q和V的差异与联系：

- $Q^{\pi}(s,a)$ 衡量**状态s下动作a**的好坏，即状态s**明确选择动作a**后的预期收获（只是说在这一步选择动作a，后续动作还是按策略π采样）
- $V^{\pi}(s)$ 衡量**状态s**的好坏，等价于在状态s按策略π随机动作，并且将这些**动作的效果累加**后的预期收益
  - 也就是**贝尔曼方程**： $V^{\pi}(s)=\sum_a \pi(a|s)Q^{\pi}(s,a)$
  - 贝尔曼方程反映了V和Q的关系
  - 贝尔曼方程还有很多其他形式，这里不展开
  - 这里介绍贝尔曼方程，只是为了解释V和Q的关系，GRPO不涉及贝尔曼方程

优势（advantage）函数 $A^{\pi}(s,a)$

在给定状态s下，动作a的优势为： $A^{\pi}(s,a)=Q^{\pi}(s,a)-V^{\pi}(s)$

- 借助贝尔曼方程辅助理解： $A^{\pi}(s,a)=Q^{\pi}(s,a)-\sum_a \pi(a|s)Q^{\pi}(s,a)$

    - 其中 $V^\pi(s) = \sum_a \pi(a|s) Q^\pi(s,a)$
- 从贝尔曼方程可以看出，优势 $A^\pi(s,a)$ 的**物理意义**：在状态s下，明确选择动作a，比按策略π随机选择一个动作，产生的**额外**的预期收益
    - $A^\pi(s,a) > 0$：a是个好动作
    - $A^\pi(s,a) < 0$：a是个坏动作
    - $A^\pi(s,a) = 0$：a不好不坏
- 数学性质：给定状态s，所有动作a的收益 $A^\pi(s,a)$ 之和为0
    - 因为A衡量的是一个动作相比于所有动作的“相对值”，动作有好有坏，有的相对收益为正，就一定有的相对收益为负（相对收益、额外收益，指的是同一回事）
- 这个优势A和GPRO中的优势A是**同一个东西**，后面会推导。

拓展内容：贝尔曼方程（V←V、Q←Q、V←Q、Q←V 的互相递归计算）

推导蒙特卡洛、GRPO不需要贝尔曼方程，推导时序差分、PPO才需要。故此节可以略过，列出来只是作为参考。

概率环境：

$$V^\pi(s) = \sum_a \pi(a|s) \sum_{s',r} P(s',r|s,a)[r + \gamma V^\pi(s')] \qquad \text{1. V计算V}$$

$$Q^\pi(s,a) = \sum_{s',r} P(s',r|s,a)[r + \gamma \sum_{a'} \pi(a'|s') Q^\pi(s',a')] \qquad \text{2. Q计算Q}$$

$$V^\pi(s) = \sum_a \pi(a|s) Q^\pi(s,a) \qquad \text{3. 用Q计算V}$$

$$Q^\pi(s,a) = \sum_{s',r} P(s',r|s,a)[r + \gamma V^\pi(s')] \qquad \text{4. 用V计算Q}$$

确定性环境：去掉所有 $P(s',r|s,a)$

$$V^\pi(s) = \sum_a \pi(a|s)[r + \gamma V^\pi(s')] \qquad \text{1. V计算V}$$

$$Q^\pi(s,a) = r + \gamma \sum_{a'} \pi(a'|s') Q^\pi(s',a') \qquad \text{2. Q计算Q}$$

$$V^\pi(s) = \sum_a \pi(a|s) Q^\pi(s,a) \qquad \text{3. 用Q计算V}$$

$$Q^\pi(s,a) = r + \gamma V^\pi(s') \qquad \text{4. 用V计算Q}$$

蒙特卡洛：估计V/Q/A

前面介绍价值V时有解释：**价值V是客观的 → 客观但不可直接获取 → 不可获取但可以估计 → 估计有误差 → 误差可以收敛**，这对于Q和优势A也同样成立。

估计V、Q、A的方法很多，其中蒙特卡洛法MC是最直观的。蒙特卡洛的结果带入LLM的场景中做简化，可以直接推导出GRPO中优势A的公式（下一章介绍）

> 除了蒙特卡洛，估计V/Q/A还有其他方法。例如：

> - 动态规划：本质是为V/Q/A随机赋值后，迭代法求不动点。但只适合状态少、动作少的离散toy场景，此处不介绍
> - 时序差分TD：这个分支最终演化为PPO的优势估计（演化过程：时序差分 → 多步时序差分 → 广义优势估计GAE → PPO中的优势A），和本文的主干内容无关。

蒙特卡洛的本质：生成一堆数据，拿到其中的reward，“直接计算” V/Q/A

- 类比：采样一堆数据，用它们的均值“直接计算”分布的期望

回顾V/Q/A的定义：

$$
\begin{aligned}
G_t &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + ... \qquad \text{where } 0 \leq \gamma \leq 1 \\
V^\pi(s) &= \mathbb{E}_{\tau \sim \pi(a|s)}[G_t | S_t = s] \\
Q^\pi(s,a) &= \mathbb{E}_{\tau \sim \pi(a|s)}[G_t | S_t = s, A_t = a] \\
A^\pi(s,a) &= Q^\pi(s,a) - V^\pi(s)
\end{aligned}
$$

蒙特卡洛算法：

- 依据策略π，采样出大量 $\tau = (s_1, a_1, r_1, s_2, a_2, r_2, s_3, ...)$ 轨迹，也就是rollout
- 对于某个状态 s 和 动作 a：
  - 估计 $V^\pi(s)$：
    - 找到所有包含状态 s 的轨迹，轨迹数量记为 $N(s)$
    - 对于每条这样的轨迹，截取从 $S_t = s$ 一直到结尾的轨迹片段，抽取其中的奖励 $r_t, r_{t+1}, ...$，计算该片段的gain：$G_t(S_t = s) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...$
    - 将所有这样轨迹的gain取平均：$\hat{V}^\pi(s) = \frac{\sum_{\text{包含}s\text{的}\tau} G_t(S_t = s)}{N(s)}$
  - 估计 $Q^\pi(s,a)$：
    - 找到所有包含状态-动作对 (s,a) 的轨迹，轨迹数量记为 $N(s,a)$
    - 对于每条这样的轨迹，截取从 $(S_t, A_t) = (s,a)$ 一直到结尾的轨迹片段，抽取其中的奖励 $r_t, r_{t+1}, ...$，计算该片段的gain：$G_t(S_t = s, A_t = a) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...$
    - 将所有这样轨迹的gain取平均：$\hat{Q}^\pi(s,a) = \frac{\sum_{\text{包含}(s,a)\text{的}\tau} G_t(S_t = s, A_t = a)}{N(s,a)}$
  - 估计 $A^\pi(s,a)$：
    - $\hat{A}^\pi(s,a) = \hat{Q}^\pi(s,a) - \hat{A}^\pi(s)$

上述三个统计量 $\hat{V}^\pi, \hat{Q}^\pi, \hat{A}^\pi$，就是对 V^π, Q^π, A^π 的估计。这个估计高方差、无偏差（与此相对，时序差分对 V^π, Q^π, A^π 的估计是低方差、有偏差的。以后PPO中介绍，此处略）

GRPO中的优势A：简化环境下的蒙特卡洛

回到强化学习训练LLM的场景。我们按如下方法定义强化学习问题：

建模：sequence级建模（只有一问一答）

- 只有一个初始状态、一个结束状态、一个动作
- 完整轨迹：只有 $\tau = (s, a, r, s')$

- 初始状态 s： prompt x
- 动作 a： LLM生成的完整response y，（动作空间是无限的）
- 奖励 r： 针对完整response y 的奖励（例如答案是否正确）
- 结束状态 s'： prompt x + response y
- 策略 $\pi(a|s)$： LLM生成response y的概率 $P(y|x)$
- 环境的随机性 $P(s', a'|s, a)$：环境是确定的，没有随机性

蒙特卡洛估计

- rollout：取一个prompt x，生成N个不同的response $\{y_1, ..., y_N\}$，对应N种不同的动作
  - 记 y_i 为第i个rollout， r_i 为对应的奖励
- 各种量的对应：
  - 第i个轨迹 $\tau_i = (s, a, r, s') = (x, y_i, r_i, x + y_i)$
    - 初始状态 s： prompt x
    - 动作 a： 第i个完整response y_i
    - 奖励 r： y_i 的奖励 r_i
    - 结束状态 s'： prompt x + response y_i
  - 包含 $s = x$ 的轨迹数量 $N(s)$：
    - 一共生成N个轨迹，每个轨迹的s都是prompt x
    - $N(s) = N$
  - 包含 $(s, a) = (x, y_i)$ 的轨迹数量 $N(s, a)$：
    - 每个 (prompt, response) 对只出现一次
    - $N(s, a) = 1$
  - 轨迹片段的收益 $G_t(\tau_i)$：
    - 在每一个轨迹 τ_i 上，从第t步直到结束的收益 $G_t(\tau_i) = r_{i,t} + \gamma r_{i,t+1} + \gamma^2 r_{i,t+2} + ...$
    - 因为只有一步动作、一个最终奖励，因此只有 $r_{i,1} = r_i$，不存在 $r_{i,2}, r_{i,3}$， ...
    - 完整轨迹 $G_t(\tau_i) = r_{i,1} = r_i$，不需要考虑折扣因子 γ
    - 轨迹片段 $G_t(s = x) = G_t(s = x, a = y_i) = G_t(\tau_i) = r_i$
- 带入，估计V、Q、A：
  - $\hat{V}^\pi(x) = \hat{V}^\pi(s = x) = \frac{\sum_{包含x的\tau_i} G_t(s=x)}{N(s=x)} = \frac{\sum_{i=1}^N r_i}{N} = \mu$，即奖励 $r_1, ..., r_N$ 的平均值 μ
  - $\hat{Q}^\pi(x, y_i) = \hat{Q}^\pi(s = x, a = y_i) = \frac{\sum_{包含(x,y_i)的\tau_i} G_t(s=x,a=y_i)}{N(s=x,a=y_i)} = \frac{r_i}{1} = r_i$，即奖励 r_i 本身
  - $\hat{A}^\pi(x, y_i) = \hat{Q}^\pi(x, y_i) - \hat{V}^\pi(x) = r_i - \mu$

观察GRPO公式中的优势： $A(x, y_i) = \frac{r_i - \mu}{\sigma}$，可以发现和蒙特卡洛的 $\hat{A}^\pi(x, y_i) = r_i - \mu$ 只差分母σ

- 这里也能看出为什么计算 $A(x, y_i)$ 必须基于相同的prompt：因为 $V^\pi(s)$ 必须基于同一个起始状态s才有意义（这里是prompt x）

为什么引入标准差σ

- 蒙特卡洛法的问题：奖励 r_i 的方差大，导致优势的方差也大
- 如何控制 r_i 的方差：对所有 r_i 的数值进行统一缩放，将标准差控制在1
- 具体操作：所有 r_i 除以标准差 σ
  - 调整后的奖励 $r_i' = \frac{r_i}{\sigma}$
  - 调整后 r_i' 的均值 $\mu' = \frac{\mu}{\sigma}$

- 调整后 r_i' 的标准差 $\sigma' = 1$
- 调整后的优势 $\hat{A}^{\pi}(x, y_i) = r_i' - \mu' = \frac{r_i}{\sigma} - \frac{\mu}{\sigma} = \frac{r_i - \mu}{\sigma}$
- 可以发现，**调整后的蒙特卡洛的优势 $\hat{A}^{\pi}(x, y_i)$ 就是GRPO公式中的优势**

token级建模：单个token的优势A

以上介绍的是将完整response视为一个动作，计算出的优势A。但在GRPO中，优势A是针对每个token的，以下将推导单个token的优势。

形式化

- 问题1：如果将n个动作合并为一个“大动作”，则大动作的优势是多少？
  - 结论：记单个动作的优势是 A_i , “大动作”的总优势是 A_{total} , 则 $A_{\text{total}} = A_1 + \gamma A_2 + \gamma^2 A_3 + ...$ 即折扣衰减后的优势之和
- 问题2：反过来，已知 “大动作”的总优势是 A_{total} ，求每个动作的优势 A_i
  - 引入假设：**每一步动作同等重要** （即一个轨迹上所有动作的优势 A_i 相等）
  - 结论：简单的等比数列求和，其中 n 为轨迹τ上的动作数量

$$A_i = \begin{cases} \frac{1}{n} A_{\text{total}} & \text{if} \gamma = 1, \\ \frac{(1-\gamma)}{1-\gamma^n} A_{\text{total}} & \text{if} \gamma \neq 1. \end{cases}$$

> 另一种理解：
>
> - 假设一个序列上，每个动作的优势 $A_1, ..., A_n$ 是n个随机变量，它们的值可以不同
> - 单个动作的优势 $A_1, ..., A_n$ 是隐变量，真实值无法观测
> - 但它们的总和 $A_{\text{total}} = A_1 + A_2 + A_3 + ...$ 可观测（假设折扣因子γ=1）
> - 代替做法：虽然无法观测真实的 A_i ，但可以用 A_i 的期望 $\mathbb{E}[A_i]$ 代替它。
>   - 假设：每个动作优势的期望相同： $\mathbb{E}[A_1] = ... = \mathbb{E}[A_n]$
>   - $\mathbb{E}[A_1 + A_2 + A_3 + ...] = n \cdot \mathbb{E}[A_i] = \mathbb{E}[A_{\text{total}}] \Rightarrow \mathbb{E}[A_i] = \frac{1}{n}\mathbb{E}[A_{\text{total}}]$
>   - 当我们只观测一个序列时，有 $\mathbb{E}[A_{\text{total}}] = A_{\text{total}}$ （把唯一的观测值作为期望）
>   - 于是 $\mathbb{E}[A_i] = \frac{1}{n} A_{\text{total}}$
> - 总结：在这种视角下，我们应该把 $\frac{1}{n} A_{\text{total}}$ 当做单个动作的优势期望 $\mathbb{E}[A_i]$ ，我们用期望 $\mathbb{E}[A_i]$ 来代替无法实际观测的隐变量 A_i

对应到LLM

- 把每个token视为一个动作，则整个response就是一个大动作
- 已知整个response的总优势 $A_{\text{total}} = \frac{r - \mu}{\sigma}$ ，倒推每个token的优势 A_i
- 引入假设：response y 中的每个token同等重要
- 结果：
  - γ=1，即无衰减： $A_i = \frac{1}{|y|} A_{\text{total}} = \frac{1}{|y|} \cdot \frac{r - \mu}{\sigma}$ ，其中 $|y|$ 是y的长度
  - γ=0，即每个token只关心自己： $A_i = A_{\text{total}} = \frac{r - \mu}{\sigma}$
- GRPO通常采用γ=1的设置

总结：GRPO的优势A的本质

sequence级建模：

- 把prompt当成起始状态s
- 把整个response当成一个动作a
- 以同一个prompt的多个rollout作为样本池
- 通过 $\frac{r}{\sigma}$ 将所有的奖励的标准差缩放到1 （实际不用手动调整，公式中已经隐含了这个调整）
- 用蒙特卡洛法估计出的优势A
- 对应GSPO中整个response的优势A

token级建模：

- 把response中的每个token视为一个动作a
- 引入两个假设：
  - 折扣因子γ=1，即无衰减
  - 每个token同等重要，即它们的优势相同
- 则每个token的优势A，就是 $\frac{\text{整个}response\text{的优势}A}{\text{序列长度}}$
- 对应GRPO中每个token的优势A

> 这也是为什么GSPO比GRPO更“自然”：GRPO额外引入了两个假设（折扣因子为1，每个token优势相同）

以上介绍了GRPO公式中优势 A 的来源，接下来将介绍RL的目标函数，如何对目标函数求导，以及将RL目标函数和优势A组合，得到GRPO的公式。

RL的目标函数（原始形式）

RL的目标函数为： $J(\theta) = \mathbb{E}_{s_1 \sim D(s_1), \tau \sim \pi_\theta(\tau|s_1)}[G(\tau)]$

> 该结果对任何强化学习都成立，无论一步决策还是多步决策、把token视为一个动作还是把完整response视为一个动作
> 以下默认将每个token视为一个动作，对应GRPO的结果

物理含义：

- 极大化轨迹的期望gain $G(\tau)$ 。（注：如果是梯度下降，则目标函数取反变成 $-J(\theta)$ 即为损失函数）

公式拆解：

- $s_1 \sim D(s_1)$: 基于初始状态 s_1 的分布 $D(s_1)$ 采样 s_1
  > 对应到LLM：在数据集 D 中采样 prompt x
- $\tau \sim \pi_\theta(s_1)$ ：给定初始状态 s_1 ，通过策略 $\pi_\theta(\tau|s_1)$ 采样出完整轨迹 τ
  > 对应到LLM：给定 prompt x ，用LLM 生成response y
  > 这里将每个token视为一个动作
  > $\pi_\theta(\tau|s_1) = \pi_\theta(y|x) = \pi_\theta(y_1|x)\pi_\theta(y_2|x, y_{<2})\pi_\theta(y_3|x, y_{<3})...$
  > 其中 y_i 是 y 的第 i 个token, π_θ 是LLM，θ是模型权重

- $G(\tau)$：轨迹 τ 的收益（gain），$G(\tau)=r_1+\gamma r_2+\gamma^2 r_3+...$，$r_t$ 为第t步动作的reward
  > 对应到LLM：
  > 令折扣因此 γ=1，即不进行衰减，则 $G(\tau)=r_1+r_2+r_3+...$
  > 每个token没有中间reward，只有最终结果reward，则 $G(\tau)=0+0+...+r_{\text{final}=}r_{\text{final}}$
- $\mathbb{E}_{s_1\sim D(s_1),\tau\sim\pi_\theta(\tau|s_1)}[G(\tau)]$：对采样出的所有 s_1 和 τ 取平均
  > 对应到LLM：
  > $\mathbb{E}_{s_1\sim D(s_1),\tau\sim\pi_\theta(\tau|s_1)}[G(\tau)]=E_{x\sim D(x),y\sim\pi_\theta(y|x)}[r_{\text{final}}]$
  >
  > - 采样出 M 个prompt $x^1,...,x^M$
  > - 每个prompt 采样 N 个 response: $y^{1,1}...,y^{1,N},...,y^{M,N}$
  > - 一共M*N条轨迹τ：$(x^1,y^{1,1}),...,(x^1,y^{1,N}),...,(x^M,y^{M,N})$
  > - 对M*N个样本的 $G(\tau)=r_{\text{final}}$ 取平均

对目标函数求导 & 目标函数的变形

此处先给结论，之后再做推导。该结论实际是**【策略梯度定理】**

变形后“实际使用”的目标函数：

$$
\begin{aligned}
J(\theta)&=\mathbb{E}_{s_1\sim D(s_1),\tau\sim\pi_\theta(\tau|s_1)_{.detach}}\left[G(\tau)\frac{\pi_\theta(\tau|s_1)}{\pi_\theta(\tau|s_1)_{.detach}}\right] && \text{理论结果}\\
&\approx\frac{1}{M}\frac{1}{N}\sum_{\substack{\text{采样}M\text{个}\\ \text{初始状态}s_1}}\sum_{\substack{\text{每个}s_1\text{采样}\\ N\text{个轨迹}\tau}}\left[G(\tau)\frac{\pi_\theta(\tau|s_1)}{\pi_\theta(\tau|s_1)_{.detach}}\right] && \text{实际计算图}\\
&\approx\frac{1}{M}\frac{1}{N}\sum_{\substack{\text{采样}M\text{个}\\ \text{初始状态}s_1}}\sum_{\substack{\text{每个}s_1\text{采样}\\ N\text{个轨迹}\tau}}[G(\tau)] && \text{数值结果}
\end{aligned}
$$

对目标函数求导：

$$
\begin{aligned}
\nabla_\theta J(\theta)&=\mathbb{E}_{s_1\sim D(s_1),\tau\sim\pi_\theta(\tau|s_1)_{.detach}}\left[G(\tau)\frac{\pi_\theta(\tau|s_1)}{\pi_\theta(\tau|s_1)_{.detach}}\nabla_\theta\log\pi_\theta(\tau|s_1)\right] && \text{理论结果}\\
&\approx\frac{1}{M}\frac{1}{N}\sum_{\substack{\text{采样}M\text{个}\\ \text{初始状态}s_1}}\sum_{\substack{\text{每个}s_1\text{采样}\\ N\text{个轨迹}\tau}}\left[G(\tau)\frac{\pi_\theta(\tau|s_1)}{\pi_\theta(\tau|s_1)_{.detach}}\nabla_\theta\log\pi_\theta(\tau|s_1)\right] && \text{实际计算图}\\
&\approx\frac{1}{M}\frac{1}{N}\sum_{\substack{\text{采样}M\text{个}\\ \text{初始状态}s_1}}\sum_{\substack{\text{每个}s_1\text{采样}\\ N\text{个轨迹}\tau}}[G(\tau)\cdot\nabla_\theta\log\pi_\theta(\tau|s_1)] && \text{数值结果}
\end{aligned}
$$

> ### 如何理解 $\pi_\theta(\tau|s_1)_{.detach}$？
>
> - detach指的就是pytorch中的detach操作，即把某个网络分离出来不计算梯度
> - 引入detach，是为了强行让 $\pi_\theta(\tau|s_1)$ 能计算梯度，后面会介绍
> - $\frac{\pi_\theta(\tau|s_1)}{\pi_\theta(\tau|s_1)_{.detach}}$ 类似于gumbel-softmax中 $\text{OneHot}(s)-s_{.detach}+s$，在计算图中不能省略

> - $\frac{\pi_\theta(\tau|s_1)}{\pi_\theta(\tau|s_1)_{.detach}}$ 本质上是一种**重要性采样**，后面会介绍。

在LLM强化学习的语境下， $\pi_\theta(\tau|s_1)$ 和 $\pi_\theta(\tau|s_1)_{.detach}$ 有具体的对应物：

	$\pi_\theta(\tau\|s_1)$	$\pi_\theta(\tau\|s_1)_{.detach}$
计算框架	fsdp 等训练框架部署的 LLM	vllm 等推理框架部署的 LLM（有针对性加速手段）
用处	计算梯度	rollout 和 evaluate
参数更新方式	梯度下降直接更新	将更新后的 $\pi_\theta(\tau\|s_1)$ 参数复制过来（即 detach）

> ## 如何理解两个“约等于≈”？
>
> - 理论结果→实际计算图：把期望 $\mathbb{E}$ 变成采样求平均 $\frac{1}{M}\frac{1}{N}\sum_M\sum_N$ ，本身就有误差
> - 实际计算图→数值结果：因为训练、推理框架的差异，$\frac{\pi_\theta(\tau|s_1)}{\pi_\theta(\tau|s_1)_{.detach}}$ 可能不等于1

> ## 策略梯度定理的两种写法
>
> 很多教程中，策略梯度定理的写法更加简洁：
> $\nabla_\theta J(\theta)=\mathbb{E}_{s_1\sim D(s_1),\tau\sim\pi_\theta(\tau|s_1)}\left[G(\tau)\nabla_\theta\log\pi_\theta(\tau|s_1)\right]$
>
> 但本文的写法更贴近本质，且和现实RL框架的代码吻合：
> $\nabla_\theta J(\theta)=\mathbb{E}_{s_1\sim D(s_1),\tau\sim\pi_\theta(\tau|s_1)_{.detach}}\left[G(\tau)\frac{\pi_\theta(\tau|s_1)}{\pi_\theta(\tau|s_1)_{.detach}}\nabla_\theta\log\pi_\theta(\tau|s_1)\right]$

策略梯度法公式推导

直接求导的困境

观察RL的目标函数 $J(\theta)=\mathbb{E}_{s_1\sim D(s_1),\tau\sim\pi_\theta(\tau|s_1)}[G(\tau)]$

- 该函数实际没办法“直接求导”
- 把期望展开成采样取平均：

$$\begin{aligned}J(\theta)&=\mathbb{E}_{s_1\sim D(s_1),\tau\sim\pi_\theta(\tau|s_1)}[G(\tau)]\\&\approx\frac{1}{M}\frac{1}{N}\sum_{\substack{\text{采样}M\text{个}\\\text{初始状态}s_1}}\sum_{\substack{\text{每个}s_1\text{采样}\\N\text{个轨迹}\tau}}[G(\tau)]\end{aligned}$$

- 发现：展开后的表达式完全不含参数θ，因此无法对θ求导

> 其中 $G(\tau)$ 用规则计算轨迹的gain，因此不含参数θ

- 问题的本质：参数θ在“分布” $\pi_\theta(\tau|s_1)$ 上，而不在“打分函数” $G(\tau)$ 上

以下先岔开一笔，介绍机器学习中的两种目标函数，从而更好地理解这个目标函数为什么无法直接求导。

两种机器学习目标函数

机器学习中有两种目标函数：

- 参数θ在打分函数上（常见）： $J_\theta = \mathbb{E}_{x\sim p(x)}[f_\theta(x)]$
- 参数θ在分布上（罕见）： $J_\theta = \mathbb{E}_{x\sim p_\theta(x)}[f(x)]$

目标：极大化/极小化采样出的样本的打分期望

- x：采样的样本
- $p(x)$、$p_\theta(x)$：样本的概率分布
- $f(x)$、$f_\theta(x)$：对样本的打分函数

绝大多数机器学习问题，都是参数θ在打分函数 $f(x)$ 上，而分布 $p(x)$ 不含参数

- 这种目标函数可以直接求导：
- $\nabla_\theta J_\theta = \nabla_\theta \mathbb{E}_{x\sim p(x)}[f_\theta(x)] = \mathbb{E}_{x\sim p(x)}[\nabla_\theta f_\theta(x)]$

> 例：LLM的SFT（极大化gold response的概率）
> $J_\theta = \mathbb{E}_{x\sim p(x)}[f_\theta(x)] = \mathbb{E}_{(x,y)\sim D(x,y)}[\pi_\theta(y|x)]$
> $\nabla_\theta J_\theta = \mathbb{E}_{(x,y)\sim D(x,y)}[\nabla_\theta \pi_\theta(y|x)]$
>
> - 样本 $x \Rightarrow (x,y)$： 数据集中提供的prompt x 和response y
> - 分布 $p(x) \Rightarrow D(x,y)$：数据分布，相当于包含配对 prompt-response 的完整数据集
> - 打分函数 $f_\theta(x) \Rightarrow \pi_\theta(y|x)$：给定prompt x，生成指定response y 的概率

然而有少数机器学习问题，参数θ在分布 $p(x)$ 上，打分函数 $f(x)$ 反而不含参数。

> 例：LLM的GRPO（极大化rollout response的收益）
> $J(\theta) = \mathbb{E}_{x\sim p_\theta(x)}[f(x)] = \mathbb{E}_{x\sim D(x),y\sim\pi_\theta(y|x)}[G(x,y)]$
>
> - 样本 $x \Rightarrow (x,y)$： prompt x + response y
> - 分布 $p_\theta(x) \Rightarrow p_\theta(x,y) = D(x)\cdot\pi_\theta(y|x)$：从数据集采样 prompt x ，从LLM采样 response y
> - 打分函数 $f(x) \Rightarrow G(x,y)$：利用规则计算reward（以及相应的gain），不含参数
> - 和SFT的区别：
>   - SFT提供了gold response，且LLM只生成gold response，不生成负例
>   - GRPO不提供gold response（只提供了一个参考标准），LLM随机rollout各种response，其中有正例有负例

另一个典型的对分布 $p_\theta(x)$ 求导的问题：VAE的解码器

> 给定一个噪声 z ，把它随机映射成图像 x' ，极小化它和原始图像 x 的误差
>
> - 带参数的分布 $p_\theta(x) \Rightarrow p_\theta(x'|z)$ ，噪声 z 映射出的图像 x' 不是唯一的，而是一个图像分布
> - 样本 $x \Rightarrow x'$ ，从映射出的分布中随机采样一个图像 x'
> - 不带参数的打分函数 $f(x) \Rightarrow f(x')$： 例如MSE，直接衡量 x 和 x' 的差距
> - 以上只是个不严格的介绍，便于理解。VAE和本文内容无关。

后一类参数在 $p_\theta(x)$ 上的机器学习任务，本质上是要优化一个带参数的采样。

- 传统机器学习问题：给定输入x，输出y是固定的（例如分类器）
- 带参数采样问题：给定输入x，输出y是随机的（例如RL中给定初始状态，随机采样一个动作）

带参数采样的目标函数不能直接求导：

- $\nabla_\theta J_\theta = \nabla_\theta \mathbb{E}_{x\sim p_\theta(x)}[f(x)] \neq \mathbb{E}_{x\sim p_\theta(x)}[\nabla_\theta f(x)]$
- 因为 $f(x)$ 不含 θ，$\nabla_\theta f(x) = 0$

正确的求导结果（策略梯度定理）：

$$\begin{aligned}\nabla_\theta J_\theta &= \nabla_\theta \mathbb{E}_{x\sim p_\theta(x)}[f(x)] \\ &= \mathbb{E}_{x\sim p_\theta(x)}[f(x)\nabla_\theta \log p_\theta(x)] \qquad \text{正确结果（简写）} \\ &= \mathbb{E}_{x\sim p_\theta(x).detach}\left[f(x)\frac{p_\theta(x)}{p_\theta(x).detach}\nabla_\theta \log p_\theta(x)\right] \qquad \text{正确结果（本质）}\end{aligned}$$

下面将推导这个正确结果。

推导：对带参数的采样 $p_\theta(x)$ 求导

- 问题：分布 $p_\theta(x)$ 含参数，打分函数 $f(x)$ 不含参数，直接求导 $\nabla_\theta f(x) = 0$
- 解决思路：把参数θ从分布挪到打分函数上
- 常用做法：重参数化（例：VAE对高斯分布重参数化，gumbel-softmax对离散采样重参数化）
- 此处借助“重参数化+重要性采样+对数导数技巧”，推导 $\nabla_\theta J_\theta$ 的公式

几个数学技巧：

技巧1：“广义的”重参数化方法

> 用途：把参数θ从分布挪到打分函数上

对于参数在分布上的目标函数 $J_\theta = \mathbb{E}_{x\sim p_\theta(x)}[f(x)]$
如果我们能找到另一组无参数分布 $q(x)$ 和含参数的打分函数 $g_\theta(x)$，使得：

- $J_\theta = \mathbb{E}_{x\sim p_\theta(x)}[f(x)] = \mathbb{E}_{x\sim q(x)}[g_\theta(x)]$
- 分布 $p_\theta(x)$ 和 打分函数 $g_\theta(x)$ 使用同一组参数θ

这样就把参数θ从分布转移到打分函数上了，从而能对 J_θ 直接求导：

$$\nabla_\theta J_\theta = \mathbb{E}_{x\sim q(x)}[\nabla_\theta g_\theta(x)]$$

> **“广义的”重参数化**这个名字是我编的，没见过类似写法，但直觉上是对的
> 标准的重参数化方法不长这样，但和本文推导无关，这里就不介绍了

技巧2：重要性采样（下一章会更详细探讨）

> 用途：构造重参数化中的 $q(x)$ 和 $g_\theta(x)$

- 需求：在用采样求期望 $\mathbb{E}_{x\sim p(x)}[f(x)]$ 时，如果从原始分布 $p(x)$ 采样 x 较为困难，但从另一个分布 $q(x)$ 采样 x 较为容易，如何用从 $q(x)$ 采样的 x 估计在 $p(x)$ 上的期望?
- 公式：$\mathbb{E}_{x\sim p(x)}[f(x)] = \mathbb{E}_{x\sim q(x)}\left[\frac{p(x)}{q(x)}f(x)\right]$

> - 证明：$\mathbb{E}_{x\sim p(x)}[f(x)] = \int p(x)f(x)dx = \int q(x)\left[\frac{p(x)}{q(x)}f(x)\right]dx = \mathbb{E}_{x\sim q(x)}\left[\frac{p(x)}{q(x)}f(x)\right]$

> - LLM中的实际场景：
>   - x：采样出的response
>   - $p(x)$：用训练框架fsdp部署的LLM，用于计算梯度，效率低
>   - $q(x)$：用推理框架vllm部署的LLM，用于rollout，效率高
>   - 成本考量：从 $q(x)$ 中采样+计算概率 $q(x)$ 和 $p(x)$，总计成本比直接从 $p(x)$ 中采样还低
>   - 虽然从 $p(x)$ 中采样 x 成本很高，但给定采样好的 x 计算它的概率 $p(x)$ 成本相对不高
>   - 这也是投机解码的原理：用低成本LLM采样response，在高成本LLM上验证response的概率

技巧3：对数导数技巧

> 用途：有利于实际计算时的数值稳定性

- 公式：$\nabla f(x) = f(x) \nabla \log f(x)$

> - 证明：$\nabla \log f(x) = \frac{\nabla f(x)}{f(x)} \Rightarrow f(x) = f(x) \nabla \log f(x)$

整合 & 公式推导

首先依据重参数化，我们要找到另一组无参数分布 $q(x)$ 和含参数的打分函数 $g_\theta(x)$，使得：

- $J_\theta = \mathbb{E}_{x \sim p_\theta(x)}[f(x)] = \mathbb{E}_{x \sim q(x)}[g_\theta(x)]$
- 分布 $p_\theta(x)$ 和 打分函数 $g_\theta(x)$ 使用同一组参数θ

这样的 $q(x)$ 和 $g_\theta(x)$ 可以借助重要性采样来构造：

- $q(x) = p_\theta(x)_{.detach}$，其中detach后就相当于不含参数了
- $g_\theta(x) = \frac{p_\theta(x)}{p_\theta(x)_{.detach}} f(x)$

> 注：为什么不像gumbel-softmax一样用 $g_\theta(x) = (1 - p_\theta(x) + p_\theta(x)_{.detach}) \cdot f(x)$：
>
> - $\frac{p_\theta(x)}{p_\theta(x)_{.detach}}$ 有实际物理意义（重要性采样），而 $1 - p_\theta(x) + p_\theta(x)_{.detach}$ 没有

$q(x)$ 和 $g_\theta(x)$ 显然满足条件：

$$J_\theta = \mathbb{E}_{x \sim p_\theta(x)}[f(x)] = \mathbb{E}_{x \sim p_\theta(x)_{.detach}} \left[\frac{p_\theta(x)}{p_\theta(x)_{.detach}} f(x) \right] = \mathbb{E}_{x \sim q(x)}[g_\theta(x)]$$

这样就成功把参数θ从分布转移到了打分函数上，可以正常求导了。

$$\begin{aligned} \nabla_\theta J_\theta &= \mathbb{E}_{x \sim q(x)}[\nabla_\theta g_\theta(x)] \\ &= \mathbb{E}_{x \sim p_\theta(x)_{.detach}} \left[\frac{\nabla_\theta p_\theta(x)}{p_\theta(x)_{.detach}} f(x) \right] \\ &= \mathbb{E}_{x \sim p_\theta(x)_{.detach}} \left[\frac{p_\theta(x)}{p_\theta(x)_{.detach}} f(x) \nabla_\theta \log p_\theta(x) \right] \qquad \text{对数导数技巧} \end{aligned}$$

对应到RL的场景：

- 样本 x： 初始状态 s_1, 轨迹 τ
- 有参数概率采样 $x \sim p_\theta(x)$：从数据集采样 $s_1 \sim D(s_1)$（这部分没有参数）, 从策略采样 $\tau \sim \pi_\theta(\tau|s_1)$
- 无参数打分函数 $f(x)$： 轨迹收益 $G(\tau)$

带入：

$$
\begin{aligned}
J(\theta) &= \mathbb{E}_{s_1 \sim D(s_1), \tau \sim \pi_\theta(\tau|s_1)_{.detach}} \left[G(\tau) \frac{\pi_\theta(\tau|s_1)}{\pi_\theta(\tau|s_1)_{.detach}} \right] \qquad \text{理论结果} \\
&\approx \frac{1}{M} \frac{1}{N} \sum_{\substack{\text{采样}M\text{个} \\ \text{初始状态}s_1}} \sum_{\substack{\text{每个}s_1\text{采样} \\ N\text{个轨迹}\tau}} \left[G(\tau) \frac{\pi_\theta(\tau|s_1)}{\pi_\theta(\tau|s_1)_{.detach}} \right] \qquad \text{实际计算图} \\
&\approx \frac{1}{M} \frac{1}{N} \sum_{\substack{\text{采样}M\text{个} \\ \text{初始状态}s_1}} \sum_{\substack{\text{每个}s_1\text{采样} \\ N\text{个轨迹}\tau}} [G(\tau)] \qquad \text{数值结果}
\end{aligned}
$$

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{s_1 \sim D(s_1), \tau \sim \pi_\theta(\tau|s_1)_{.detach}} \left[G(\tau) \frac{\pi_\theta(\tau|s_1)}{\pi_\theta(\tau|s_1)_{.detach}} \nabla_\theta \log \pi_\theta(\tau|s_1) \right] \qquad \text{理论结果} \\
&\approx \frac{1}{M} \frac{1}{N} \sum_{\substack{\text{采样}M\text{个} \\ \text{初始状态}s_1}} \sum_{\substack{\text{每个}s_1\text{采样} \\ N\text{个轨迹}\tau}} \left[G(\tau) \frac{\pi_\theta(\tau|s_1)}{\pi_\theta(\tau|s_1)_{.detach}} \nabla_\theta \log \pi_\theta(\tau|s_1) \right] \qquad \text{实际计算图} \\
&\approx \frac{1}{M} \frac{1}{N} \sum_{\substack{\text{采样}M\text{个} \\ \text{初始状态}s_1}} \sum_{\substack{\text{每个}s_1\text{采样} \\ N\text{个轨迹}\tau}} [G(\tau) \cdot \nabla_\theta \log \pi_\theta(\tau|s_1)] \qquad \text{数值结果}
\end{aligned}
$$

这就是**策略梯度定理**的详细推导版本

实际常用形式

把轨迹展开为动作

前面的策略梯度定理中使用的是轨迹τ的概率 $\pi_\theta(\tau|s_1)$ ，实际应用中通常会进一步展开为每个动作的概率 $\pi_\theta(a_t|s_t)$ 。

在确定性环境下：

$$
\begin{aligned}
\nabla_\theta \log \pi_\theta(\tau|s_1) &= \nabla_\theta \log \prod_{t=1}^{|\tau|} \pi_\theta(a_t|s_t) \qquad |\tau|\text{代表轨迹的步骤数} \\
&= \sum_{i=t}^{|\tau|} \nabla_\theta \log \pi_\theta(a_t|s_t)
\end{aligned}
$$

> 在非确定环境下也成立，但和LLM无关，这里不介绍了

带入：

$$
\begin{aligned}
\nabla_\theta J(\theta) &\approx \frac{1}{M}\frac{1}{N}\sum_{\substack{\text{采样}M\text{个}\\ \text{初始状态}s_1}}\sum_{\substack{\text{每个}s_1\text{采样}\\ N\text{个轨迹}\tau}}\left[G(\tau)\cdot\nabla_\theta\log\pi_\theta(\tau|s_1)\right] \\
&= \frac{1}{M}\frac{1}{N}\sum_{\substack{\text{采样}M\text{个}\\ \text{初始状态}s_1}}\sum_{\substack{\text{每个}s_1\text{采样}\\ N\text{个轨迹}\tau}}\left[\sum_{t=1}^{|\tau|}G(\tau)\cdot\nabla_\theta\log\pi_\theta(a_t|s_t)\right] \\
&\approx \mathbb{E}_{s_0\sim D(s_0),\tau\sim\pi_\theta(\tau|s_0)}\left[\sum_{t=1}^{|\tau|}G(\tau)\cdot\nabla_\theta\log\pi_\theta(a_t|s_t)\right]
\end{aligned}
$$

> 这里只在求导版本 ∇J 下展开，目标函数 J 没有log不能这么展开
> GRPO的目标函数实际是用 ∇J 倒推回去的一个“代理目标函数” $\hat{J}$
> $\hat{J}$ 和 J 本质上不是同一个函数，但恰好 $\nabla J = \nabla\hat{J}$ ，且 J 和 $\hat{J}$ 的极值点相同
> 最后一步的约等于：如果忽略 $\pi_\theta(\tau|s_0)$ 和 $\pi_\theta(\tau|s_0)_{.detach}$ 的数值差异，可以做这种“形式简化”

导数的变体：替换 $G(\tau)$

实际应用中，一般会把轨迹收益 $G(\tau)$ 替换成别的东西，公式还是成立的。

> 这部分的推导很复杂，就不管了。但公式看起来很符合直觉

$$
\begin{aligned}
\nabla_\theta J(\theta) &\approx \frac{1}{M}\frac{1}{N}\sum_{\substack{\text{采样}M\text{个}\\ \text{初始状态}s_1}}\sum_{\substack{\text{每个}s_1\text{采样}\\ N\text{个轨迹}\tau}}\left[\sum_{t=1}^{|\tau|}G(\tau)\cdot\nabla_\theta\log\pi_\theta(a_t|s_t)\right] \\
&= \frac{1}{M}\frac{1}{N}\sum_{\substack{\text{采样}M\text{个}\\ \text{初始状态}s_1}}\sum_{\substack{\text{每个}s_1\text{采样}\\ N\text{个轨迹}\tau}}\left[\sum_{t=1}^{|\tau|}G_t(\tau)\cdot\nabla_\theta\log\pi_\theta(a_t|s_t)\right] \\
&= \frac{1}{M}\frac{1}{N}\sum_{\substack{\text{采样}M\text{个}\\ \text{初始状态}s_1}}\sum_{\substack{\text{每个}s_1\text{采样}\\ N\text{个轨迹}\tau}}\left[\sum_{t=1}^{|\tau|}Q(s_t,a_t)\cdot\nabla_\theta\log\pi_\theta(a_t|s_t)\right] \\
&= \frac{1}{M}\frac{1}{N}\sum_{\substack{\text{采样}M\text{个}\\ \text{初始状态}s_1}}\sum_{\substack{\text{每个}s_1\text{采样}\\ N\text{个轨迹}\tau}}\left[\sum_{t=1}^{|\tau|}A(s_t,a_t)\cdot\nabla_\theta\log\pi_\theta(a_t|s_t)\right]
\end{aligned}
$$

其中：

- 替换成从第t步开始的轨迹片段收益 $G_t(\tau)$： REINFORCE算法，$G_t(\tau)$ 从实际数据中收集
- 替换成Q函数 $Q(s_t,a_t)$：Actor-critic算法，Q函数用一个critic网络来计算
- 替换成优势 $A(s_t,a_t)$：PPO/GRPO等现代方法。PPO用一个critic网络来计算，GRPO直接对数据归一化计算

其中替换成优势A训练最稳定，因此现代方法一般使用A。
公式理解：用优势A控制梯度的方向和大小

- A>0：好动作，沿梯度方向走
- A<0：坏动作，逆着梯度方向走
- A>0且数值很大：非常好的动作，沿梯度方向走一大步

> 这里也能看出为什么引入 $\nabla_\theta \log \pi_\theta(\tau|s_1)$ 来代替 $\nabla_\theta \pi_\theta(\tau|s_1)$：
>
> - 轨迹拆分成动作：通过log把动作概率相乘变成相加
> - logits：$\log \pi_\theta(a_t|s_t)$ 实际是每个动作的logits，而神经网络给每个token的打分在softmax归一化之前，天然就是logits，省略一步softmax有利于数值稳定
> - 约分：$\frac{\nabla_\theta \pi_\theta(\tau|s_1)}{\pi_\theta(\tau|s_1)_{.detach}} = \frac{\pi_\theta(\tau|s_1)}{\pi_\theta(\tau|s_1)_{.detach}} \nabla_\theta \log \pi_\theta(\tau|s_1) \approx \nabla_\theta \log \pi_\theta(\tau|s_1)$，在不考虑训练-推理的数值差异时，可以约分

目标函数的变体：动作优势之和

如果把RL的目标函数从轨迹τ的收益 $G(\tau)$ 改成动作优势之和:

$$J(\theta) = \mathbb{E}_{s_1 \sim D(s_1), \tau \sim \pi_\theta(\tau|s_1)} \left[\sum_{t=1}^{|\tau|} \gamma^{t-1} A(s_t, a_t) \right]$$

当折扣因子γ=1，即无衰减时：

$$J(\theta) = \mathbb{E}_{s_1 \sim D(s_1), \tau \sim \pi_\theta(\tau|s_1)} \left[\sum_{t=1}^{|\tau|} A(s_t, a_t) \right]$$

其导数恰好和对 $\mathbb{E}_\tau[G(\tau)]$ 求导的结果相同：

$$\nabla_\theta J(\theta) = \mathbb{E}_{s_1 \sim D(s_1), \tau \sim \pi_\theta(\tau|s_1)} \left[\sum_{t=1}^{|\tau|} A(s_t, a_t) \cdot \nabla_\theta \log \pi_\theta(a_t|s_t) \right]$$

> 以上是简化形式，完整形式是：
>
> - $J(\theta) = \mathbb{E}_{s_1 \sim D(s_1), \tau \sim \pi_\theta(\tau|s_1)_{.detach}} \left[\sum_{t=1}^{|\tau|} \gamma^{t-1} A(s_t, a_t) \frac{\pi_\theta(a_t|s_t)}{\pi_\theta(a_t|s_t)_{.detach}} \right]$
> - $\nabla_\theta J(\theta) = \mathbb{E}_{s_1 \sim D(s_1), \tau \sim \pi_\theta(\tau|s_1)_{.detach}} \left[\sum_{t=1}^{|\tau|} A(s_t, a_t) \frac{\pi_\theta(a_t|s_t)}{\pi_\theta(a_t|s_t)_{.detach}} \nabla_\theta \log \pi_\theta(a_t|s_t) \right]$
>
> 如果忽略训练、推理的差异，即认为在数值上 $\pi_\theta(a_t|s_t) = \pi_\theta(a_t|s_t)_{.detach}$，得到的就是简化形式

这实际就是最简单版本的GRPO公式。

最简单版本的GRPO公式

前提设置

- token级建模，每个token是一个动作
- 衰减因子γ=1
- 一个response中每个token同等重要（即优势相同）
- 目标函数：极大化response中token优势之和的期望（此设置下，等价于极大化轨迹收益）
- 优势估计方式：用同一个prompt rollout出一批数据后，做蒙特卡洛估计
- 不考虑KL散度

- on-policy，即rollout出一批数据只用来更新一次梯度，用后即丢弃（因此没有旧策略 π_{old} ）
- 训练、推断的LLM用两个不同框架部署

符号约定：

- 从数据集 $D(x)$ 采样 N 个prompt x （为了简化公式，不引入下标 x_i ）
- 为每个 x rollout G 个response $\{y_i\}_{i=1}^{G}$
- $\{y_i\}_{i=1}^{G}$ 对应的reward为 $\{r_i\}_{i=1}^{G}$
- y_i 长度为 $|y_i|$
- $y_{i,t}$ 为 y_i 的第t个token，$y_{i,<t}$ 为 y_i 的第1,2,...,t-1个token
- $A_{y_{i,t}}$ 为token $y_{i,t}$ 的优势
- π_θ 是训练框架（例如fsdp）部署的LLM，$\pi_{\theta.detach}$ 是推理框架（例如vllm）部署的LLM
- 所有response y_i 是从 $\pi_{\theta.detach}$ rollout出来的

目标函数：

$$
\begin{aligned}
J(\theta) &= \mathbb{E}_{x\sim D(x),y_i\sim\pi_\theta(y_i|x)_{.detach}}\left[\sum_{t=1}^{|y_i|} A_{y_{i,t}}\cdot\frac{\pi_\theta(y_{i,t}|x,y_{i,<t})}{\pi_\theta(y_{i,t}|x,y_{i,<t})_{.detach}}\right] \\
&\approx \frac{1}{N}\sum_{\substack{\text{采样}\\ N\text{个}x}}\frac{1}{G}\sum_{\substack{\text{每个}x\\ \text{生成}G\text{个}y_i}}\left[\sum_{t=1}^{|y_i|} A_{y_{i,t}}\cdot\frac{\pi_\theta(y_{i,t}|x,y_{i,<t})}{\pi_\theta(y_{i,t}|x,y_{i,<t})_{.detach}}\right]
\end{aligned}
$$

其中单个token $y_{i,t}$ 的优势为 $A_{y_{i,t}}$ ，完整response的优势为 A_{y_i} ，则：

$$
A_{y_{i,t}} = \frac{1}{|y_i|}\cdot A_{y_i},\qquad A_{y_i} = \frac{r_i - \text{mean}\left(\{r_i\}_{i=i}^{G}\right)}{\text{std}\left(\{r_i\}_{i=i}^{G}\right)},\qquad \text{都与}t\text{无关}
$$

求导

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{x\sim D(x),y_i\sim\pi_\theta(y_i|x)_{.detach}}\left[\sum_{t=1}^{|y_i|} A_{y_{i,t}}\cdot\frac{\pi_\theta(y_{i,t}|x,y_{i,<t})}{\pi_\theta(y_{i,t}|x,y_{i,<t})_{.detach}}\nabla_\theta\log\pi_\theta(y_{i,t}|x,y_{i,<t})\right] \\
&\approx \frac{1}{N}\sum_{\substack{\text{采样}\\ N\text{个}x}}\frac{1}{G}\sum_{\substack{\text{每个}x\\ \text{生成}G\text{个}y_i}}\left[\sum_{t=1}^{|y_i|} A_{y_{i,t}}\cdot\frac{\pi_\theta(y_{i,t}|x,y_{i,<t})}{\pi_\theta(y_{i,t}|x,y_{i,<t})_{.detach}}\nabla_\theta\log\pi_\theta(y_{i,t}|x,y_{i,<t})\right]
\end{aligned}
$$

讨论

- 以上是GRPO公式**最本质**的部分。完整的GRPO只是加上了一些trick：
  - π_{old} ：为了把on-policy变成off-policy以节约rollout成本，需要使用重要性采样
  - min-clip操作：off-policy情况下为了防止 π_{old} 和 π_θ 差距过大，导致训练不稳定
  - KL散度：为了防止RL训练后的 π_θ 和其最初始版本 π_{ref} 偏离太远，导致灾难性遗忘
- $\frac{1}{|y_i|}$ 在公式中的位置？

    - GRPO公式的常见写法中，$\frac{1}{|y_i|}$ 不写在优势 A 里，而写在 $\frac{1}{|y_i|}\sum_{i=1}^{|y_i|}$ 位置
    - 两种写法是等价的，我的写法更本质： $\frac{1}{|y_i|}$ 来自于严格推导出的单个token的优势
- 公式中 $\frac{\pi_\theta}{\pi_{\theta.detach}}$ 为什么不移除?
    - 其反映了RL实质的训练过程：从 $\pi_{\theta.detach}$ rollout，从 π_θ 更新参数，两者采用不同的部署框架（而且数值经常有差异，训练-推理不一致是RL中一个头疼的问题）
    - 即使二者数值相等也不能移除，它们在计算图上有实际对应，移除后无法对 π_θ 求导
- on-policy版本的GRPO，目标函数是0吗?
    - $J(\theta)$ 确实等于0，但梯度 $\nabla J(\theta) \neq 0$
    - 因为 A 是通过reward归一化得到的，如果不考虑 $\pi_{\theta.detach}$ 和 π_θ 的数值差异，$J(\theta) = 0$
    - 后面章节会详细介绍为什么梯度 $\nabla J(\theta) \neq 0$

重要性采样

前面推导策略梯度定理时，已经介绍了重要性采样，而本章将进行更深入的探讨。

基于重要性采样的公式： $\mathbb{E}_{x\sim p(x)}[f(x)] = \mathbb{E}_{x\sim q(x)}\left[\frac{p(x)}{q(x)}f(x)\right]$ ，我们可以从任意一个分布中rollout数据，来训练GRPO：

$$\mathbb{E}_{x\sim D(x),y\sim\pi_\theta(y|x)}[G(y)] = \mathbb{E}_{x\sim D(x),y\sim q(y|x)}\left[\frac{\pi_\theta(y|x)}{q(y|x)}G(y)\right]$$

- $\pi_\theta(y|x)$ 是待训练的LLM
- $q(y|x)$ 是用来rollout的数据分布（通常也是个LLM）

基于重要性采样，我们可以统一理解 on-policy GRPO、off-policy GRPO、LLM知识蒸馏，甚至SFT：它们都是重要性采样的特例，只是从不同的分布 $q(y|x)$ 中来rollout。

on-policy：用 $\pi_{\theta.detach}$ rollout

前面推导GRPO的公式，用的就是on-policy。其中用于rollout的 $\pi_{\theta.detach}$ ，对应了用vllm等推理专用框架部署的LLM，其参数值和 π_θ 相同。这些内容已经详细探讨过了。

这里重点解释一个问题：on-policy GRPO中，为什么**目标函数（优势之和）是0，但梯度不是0**。

- 计算图的角度：虽然目标函数值是0，但计算图中有 $\pi_\theta(y|x)$ 节点，保证它会有梯度
- 动作的角度：把优势视为一种“效应”，虽然所有动作的“效应”之和是0，但单个动作的“效应”都不是0。而且这些效应更新的方向不是均衡的：好的动作会被加强，坏的动作会被削弱
    - 类比：左右两边都有力拉着小车，力平衡，小车静止。但更新时会加强向左的力，削弱向右的力，下个时刻小车往左走。
- 数值的角度（随着训练动态更新）：当前优势的期望之和恰好是0，但策略更新后，如果恰好还是rollout出这些数据，在旧的策略视角下，期望之和就不是0了（虽然在新策略的视角下还是0）

response空间无限的例子不容易理解，这里举一个有限的例子

- prompt：选择题，ABCD四选一，其中正确选项是C
- response：ABCD四个token其中之一

- 采样：采样20个response $\{y_1, ..., y_{20}\}$，每个选项一定会重复多次。选项的分布就是策略 π_θ
- 对20个response归一化，得到 μ 和 σ，计算A，它们的和确实是0
- 梯度的方向：加强选项C，削弱ABD
- 参数更新后，会得到一个新策略 π_θ^*，用新策略采样20个新response $\{y'_1, ..., y'_{20}\}$
- 结果：$\{y'_1, ..., y'_{20}\}$ 中的正确选项C比 $\{y_1, ..., y_{20}\}$ 多
- 对20个新response归一化，得到 μ' 和 σ'，计算A'，它们的和确实还是0
  - 但A'之和为0是在 新策略 π_θ^* 视角下的
  - 在旧策略 π_θ 视角下如果计算 $\{y'_1, ..., y'_{20}\}$ 的优势，应当使用旧的 μ 和 σ，而不是新的 μ' 和 σ'，这样算出的优势就不是0了
- 也就是说，优势和为0只是“临时”的，如果我们永远从固定策略中rollout，下个时刻优势和就不是0了
- 之所以每次目标函数的优势和为0，是因为on-policy每次都重新换策略
- 因此on-policy GRPO的目标函数一直是0，但梯度不是0

off-policy: 用模型的历史版本 π_{old} rollout

GRPO公式中，更常见的是off-policy形式：

$$J(\theta) = \mathbb{E}_{x\sim D(x), y_i\sim\pi_{old}(y_i|x)} \left[\sum_{t=1}^{|y_i|} A_{y_{i,t}} \cdot \frac{\pi_\theta(y_{i,t}|x, y_{i,<t})}{\pi_{old}(y_{i,t}|x, y_{i,<t})} \right]$$
$$\approx \frac{1}{N} \sum_{\substack{\text{采样}\\ N\text{个}x}} \frac{1}{G} \sum_{\substack{\text{每个}x\\ \text{生成}G\text{个}y_i}} \left[\sum_{t=1}^{|y_i|} A_{y_{i,t}} \cdot \frac{\pi_\theta(y_{i,t}|x, y_{i,<t})}{\pi_{old}(y_{i,t}|x, y_{i,<t})} \right]$$

它实际对应了这样的训练过程：

- 在某一时刻，用于训练的 π_θ 和用于推断的 π_{old} 参数相同（但部署框架不同）
  - 用推理模型 π_{old} rollout出 M*N 条数据
  - 对于训练模型 π_θ，每次取 M 条数据用于更新梯度，一共更新 N 轮
  - 最初始 $\pi_\theta = \pi_{old}$，但之后参数 π_θ 更新了N轮，而 π_{old} 一直保持不变，且两者相差越来越大
  - 第N轮更新完后，使用 π_θ 参数重新部署用于rollout的 π_{old}
- 此时两者参数又相等了，开始下个 N 轮的训练

训练时为什么不把 M*N 条数据一次全用掉?

- 直观理解：推理框架（如vllm）比训练框架（如fsdp）显存占用少，如果rollout M*N条数据能把显存跑满，则训练用M*N条数据一定会爆显存

为什么不通过梯度累加，用M*N条数据更新一次参数?

- 没有必要，更新N次 vs 更新1次，前者更新频率更高，训练更“快”(但方差也更大)
- 使用M条数据更新时，本身往往也用了梯度累加（即真实的batch size比M更小，现在的框架如verl可以自动分配实际batch size并自动累加达到M）

off-policy相比于on-policy是一种trade-off，能提升训练效率，代价是方差变大，训练更不稳定。

> 某些教程中写off-policy是为了rollout一次训练多次，这个说法有些误导：并不是rollout出M条数据，每轮更新都用这M条，一共更新N次；而是一次rollout出M*N条数据，每次用M条，一共分N次用完

verl中对应的训练config（暂时不管evaluate）：

> verl config：https://verl.readthedocs.io/en/latest/examples/config.html

- `data.train_batch_size`： 一次选择多少个prompt来rollout
- `actor_rollout_ref.rollout.n`：一个prompt rollout出多少个response
- `actor_rollout_ref.actor.ppo_mini_batch_size`： π_θ 更新一次使用多少个prompt（以及相应的全部response）
- `actor_rollout_ref.actor.ppo_micro_batch_size_per_gpu`： 在gpu上计算时实际的batch size（会通过梯度累加达到ppo_mini_batch_size）
- `actor_rollout_ref.actor.use_dynamic_bsz`：是否自动、动态设置ppo_micro_batch_size_per_gpu

例：

- vllm一次选择 `train_batch_size` 个prompt，每个rollout出 `n` 个response，一共 `train_batch_size * n` 个response
- fsdp一次选择 `ppo_mini_batch_size` 个prompt（相应 `ppo_mini_batch_size * n` 个response）用于更新模型，一共更新 `train_batch_size / ppo_mini_batch_size` 轮
  - 如果未设置 `use_dynamic_bsz` ，则gpu实际使用 `ppo_micro_batch_size_per_gpu` 个prompt，并且累加到 `ppo_mini_batch_size` 完成一次梯度更新，一共累加 `ppo_mini_batch_size / ppo_micro_batch_size_per_gpu` 轮
  - 如果设置 `use_dynamic_bsz` ，则框架自动设置最合适的 `ppo_micro_batch_size_per_gpu` ，且会随着prompt长度动态调整

LLM知识蒸馏：用另一个更强大的LLM rollout

如果我们的 $\pi_\theta(y|x)$ 是个弱LLM，而采样分布 $q(y|x)$ 是个强LLM，且我们始终从强LLM rollout并实时计算 $q(y|x)$ ，则对应了LLM知识蒸馏：用强模型生成的数据训练弱模型。

这也是种极端版本的off-policy：用于采样的模型永远不变。

> 这只是种理论做法。实际的知识蒸馏，通常是用强LLM rollout出大量数据，给弱模型SFT。

SFT：从数据集的（prompt, response）分布rollout

SFT其实也可以套到策略梯度+重要性采样的框架下。只不过场景特殊：

- rollout采样过程：
  - 采样数据 $x \sim D(x)$ ： 从数据集采样 prompt x
  - rollout $y \sim q(y|x)$ ： 这样定义 $q(y,x)$ ，则采样出的 y 有且仅有数据集中的gold response。
  - 此结果等价于采样 $(x,y) \sim D(x,y)$

$$q(y|x) = \begin{cases} 1 & \text{如果y是数据集中的gold response} \\ 0 & \text{otherwise.} \end{cases}$$

- RL建模：
  - token级建模
  - reward粒度：token级，生成gold token则为1，否则为0
  - 折扣因子γ=0，即只看当前token reward
  - 轨迹片段收益 $G_t = \sum_{i=0}^{|y|} \gamma^i r_{t+i} = r_t$
    - 但因为只会rollout出gold response，所以实际上只有 $G_t = r_t = 1$

- 带入：
  - G_t 版本的导数（因此实际是REINFORCE而不是GRPO）：

$$\begin{aligned}\nabla_\theta J_\theta &= \nabla_\theta \mathbb{E}_{x\sim D(x),y\sim q(y|x,y)}\left[\sum_{t=1}^{|y|}\frac{\pi_\theta(y_t|x,\ y_{<t})}{q(y_t|x,y_{<t})}\cdot G_t\right] \\ &= \mathbb{E}_{(x,y)\sim D(x,y)}\left[\sum_{t=1}^{|y|}\frac{\nabla_\theta\pi_\theta(y_t|x,\ y_{<t})}{1}\cdot 1\right] \qquad \text{因为只会采样出gold } y\end{aligned}$$

可以发现该结果就是SFT监督学习的公式。

min-clip操作：参数更新要谨慎

完整的GRPO公式通常还有min-clip操作，这是一种让训练更稳定的技巧。

目标函数

$$\begin{aligned}J(\theta) &= \mathbb{E}_{x\sim D(x),y_i\sim\pi_{old}(y_i|x)}\left[\sum_{t=1}^{|y_i|}\text{min-clip}\left(A_{y_{i,t}}\cdot\frac{\pi_\theta(y_{i,t}|x,y_{i,<t})}{\pi_{old}(y_{i,t}|x,y_{i,<t})}\right)\right] \\ &\approx \frac{1}{N}\sum_{\substack{\text{采样}\\ N\text{个}x}}\frac{1}{G}\sum_{\substack{\text{每个}x\\ \text{生成}G\text{个}y_i}}\left[\sum_{t=1}^{|y_i|}\text{min-clip}\left(A_{y_{i,t}}\cdot\frac{\pi_\theta(y_{i,t}|x,y_{i,<t})}{\pi_{old}(y_{i,t}|x,y_{i,<t})}\right)\right]\end{aligned}$$

其中单个token $y_{i,t}$ 的优势：

$$A_{y_{i,t}} = \frac{1}{|y_i|}\cdot A_{y_i} = \frac{1}{|y_i|}\cdot\frac{r_i - \text{mean}\left(\{r_i\}_{i=i}^{G}\right)}{\text{std}\left(\{r_i\}_{i=i}^{G}\right)} \qquad \text{与}t\text{无关}$$

其中的min-clip算子：

$$\text{min-clip}\left(A\cdot\frac{\pi_\theta}{\pi_{old}}\right) = \min\left(A\cdot\frac{\pi_\theta}{\pi_{old}}, A\cdot\text{clip}\left(\frac{\pi_\theta}{\pi_{old}}, 1-\varepsilon, 1+\varepsilon\right)\right)$$

> 注意：
>
> - $\text{min-clip}\left(A\cdot\frac{\pi_\theta}{\pi_{old}}\right)$ 中要把 A 和 $\frac{\pi_\theta}{\pi_{old}}$ 看成两个自变量，而不是把 $A\cdot\frac{\pi_\theta}{\pi_{old}}$ 整体看成一个自变量，即把其中的乘号 $\cdot$ 也视为 min-clip 的一部分。严格来说写成 $\text{min-clip}\left(A, \frac{\pi_\theta}{\pi_{old}}\right)$ 更合适，但不直观
> - 相比于最简版本GRPO，这里用 $\text{min-clip}\left(A\cdot\frac{\pi_\theta}{\pi_{old}}\right)$ 代替了 $A\cdot\frac{\pi_\theta}{\pi_{old}}$

求导

min-clip的意义直接从目标函数上比较难懂，从导数的角度更容易理解：

$$\nabla_\theta J(\theta) = \mathbb{E}_{x\sim D(x), y_i\sim\pi_{old}(y_i|x)}\left[\sum_{t=1}^{|y_i|}\nabla_\theta \text{min-clip}\left(A_{y_{i,t}}\cdot\frac{\pi_\theta(y_{i,t}|x,y_{i,<t})}{\pi_{old}(y_{i,t}|x,y_{i,<t})}\right)\right]$$
$$\approx \frac{1}{N}\sum_{\substack{\text{采样}\\ N\text{个}x}}\frac{1}{G}\sum_{\substack{\text{每个}x\\ \text{生成}G\text{个}y_i}}\left[\sum_{t=1}^{|y_i|}\nabla_\theta \text{min-clip}\left(A_{y_{i,t}}\cdot\frac{\pi_\theta(y_{i,t}|x,y_{i,<t})}{\pi_{old}(y_{i,t}|x,y_{i,<t})}\right)\right]$$

其中

$$\nabla_\theta \text{min-clip}\left(A\cdot\frac{\pi_\theta}{\pi_{old}}\right) = \begin{cases} 0 & \text{触发clip：}\ A>0, \frac{\pi_\theta}{\pi_{old}}>1+\varepsilon \text{ 或者 } A<0, \frac{\pi_\theta}{\pi_{old}}<1-\varepsilon \\ A\cdot\frac{\pi_\theta}{\pi_{old}}\nabla_\theta\log\pi_\theta & \text{不触发clip：}\ otherwise \end{cases}$$

min-clip做了什么：

- 从求导公式本身来理解：
  - 当触发clip时，梯度会被**置为0**，而不是设置一个梯度的上/下限
  - 当不触发clip时，和标准的 $A\cdot\frac{\pi_\theta}{\pi_{old}}$ 求导结果一致
- 实际含义：
  - **对于概率过大的好动作，或者概率过小的坏动作，不更新模型**

注意事项：

- min-clip是“不对称”的clip：触发clip的条件是 $A>0, \frac{\pi_\theta}{\pi_{old}}>1+\varepsilon$ 或者 $A<0, \frac{\pi_\theta}{\pi_{old}}<1-\varepsilon$，即对于好动作只触发上clip，对坏动作只触发下clip。而不是对于任何动作都clip到 $1-\varepsilon<\frac{\pi_\theta}{\pi_{old}}<1+\varepsilon$ 的范围。
- 概率大/小指的不是 π_θ 的绝对大小，而是 $\frac{\pi_\theta}{\pi_{old}}$ 的相对大小
- 对于on-policy， $\frac{\pi_\theta}{\pi_{old}}=1$,等价于没有min-clip。

从表格会更好理解这种不对称性：

	A>0 好动作	**A<0 坏动作**
$\frac{\pi_\theta}{\pi_{old}}>1+\varepsilon$ 概率过大	不继续增大（**丢弃**）	应当减小（更新）
$\frac{\pi_\theta}{\pi_{old}}>1$ 概率增大	应当增大（更新）	应当减小（更新）
$\frac{\pi_\theta}{\pi_{old}}<1$ 概率减小	应当增大（更新）	应当减小（更新）
$\frac{\pi_\theta}{\pi_{old}}<1-\varepsilon$ 概率过小	应当增大（更新）	不继续减小（**丢弃**）

实际意义

min-clip本质是一种**谨慎**的更新策略：

- 模型本身希望加大好动作的概率、减小坏动作概率
- 但更新尽量谨慎，不要乐观地大步前进（高概率好动作），也不要悲观地大步后退（低概率坏动作），因为模型估计可能不准
  - 乐观/悲观指的是策略 π 的大小（主观的），而不是优势A的正负（客观的）
  - 允许因为客观优势A而大步前进、后退（因此不clip A），但限制因为主观策略π而大步前进、后退（因此clip π）

如果反过来变成max-clip：

- 就会变成一种激进更新策略：
  - 乐观地大步前进（高概率好动作，进一步加大概率）
  - 悲观地大步后退（低概率坏动作，进一步减小概率）
  - 否则就原地踏步（触发clip，不更新）
- 这种策略更新会很“神经质”
  - 通常一动不动（clip了）
  - 偶尔一下更新特别猛（模型恰好蒙对了：给好动作大概率，给坏动作小概率）
- 从动力学角度：
  - max-clip是一种正反馈，鼓励激进更新，越偏越远，无法回头
  - min-clip是一种**负反馈**，限制极端情况提供了一种“阻尼”，能在一个稳定区间内往复调整

拓展：PPO、GSPO、DAPO

有了以上GRPO的公式，可以很容易拓展出PPO、GSPO、DAPO等各种变体公式

PPO：用网络估计优势A

PPO中，第t个token的优势 A_t 不是通过数据归一化得到的，而是用神经网络计算的：

$$A_t = Q_t - V_t = \gamma^{|y|-t} \cdot r - V_\theta(s_t = x + y_{<t})$$

- r 是最终response
- $V_\theta(s_t = x + y_{<t})$ 是个神经网络，用于估计状态 $s_t = x + y_{<t}$ 的价值。

价值网络 $V_\theta(x + y_{<t})$ 的形式，与LLM计算句子概率的形式类似： $\pi_\theta(x + y_{<t})$ ，因此在实现上，可以和 $\pi_\theta(x + y_{<t})$ 共用相同的LLM，只是在输出时换用不同的head。

> 以上只是PPO最核心的思想。更多细节就不展开了（例如价值网络 V_θ 如何训练、用广义优势估计GAE计算A）

PPO与GRPO比较：

- GRPO的A直接统计得到，本质是蒙特卡洛；PPO的A用网络估计，本质是时序差分（时序差分是RL中另一大类方法，这里不做探讨）
- PPO和GRPO的训练效果：PPO低方差、高偏差；GRPO高方差、低偏差
  - 如何理解?
  - PPO低方差：用神经网络估计A，本质是用大量数据的滑动平均，波动比GRPO每次采样小
  - PPO高偏差：（1）网络不一定有能力拟合真实的A；（2）训练过程中，网络可能还未收敛
- 从历史的角度，是先有PPO再简化出的GRPO。但从理解的角度，不妨先理解GRPO再看PPO。

GSPO：把整个response作为一个大动作

如果把整个response y_i 视为一个完整动作，得到的就是GSPO。

目标函数：

$$J_{GSPO}(\theta) = \mathbb{E}_{x\sim D(x), y_i\sim\pi_{old}(y_i|x)}\left[\text{min-clip}\left(A_{y_i}\cdot\frac{\pi_\theta(y_i|x)}{\pi_{old}(y_i|x)}\right)\right]$$
$$\approx \frac{1}{N}\sum_{\substack{\text{采样}\\ N\text{个}x}}\frac{1}{G}\sum_{\substack{\text{每个}x\\ \text{生成}G\text{个}y_i}}\left[\text{min-clip}\left(A_{y_i}\cdot\frac{\pi_\theta(y_i|x)}{\pi_{old}(y_i|x)}\right)\right]$$

其中完整response y_i 的优势：

$$A_{y_i} = \frac{r_i - \text{mean}\left(\{r_i\}_{i=i}^{G}\right)}{\text{std}\left(\{r_i\}_{i=i}^{G}\right)}$$

它去掉了GRPO中每个token优势相等的假设。

> 注：
>
> - on-policy下GSPO和GRPO是完全等价的，因为 $\frac{\pi_\theta}{\pi_{\theta.detach}}$ 都是1.
> - off-policy下把 $\pi_{\theta.detach}$ 换成 π_{old} ，两者就不相等了。
> - off-policy下，GSPO训练比GRPO更稳定，尤其在使用大batch以及训练MoE的时候
> - GSPO中如果clip，会把整个response都clip，而不是像GRPO中按token clip

DAPO：附加大量trick的GRPO

> verl中的DAPO：https://verl.readthedocs.io/en/latest/perf/best_practices.html

DAPO是GRPO的魔改版，相当于添加了很多trick，来修正训练时的各种问题：

- token优势不受输出长度影响
- clip-higher
- 长度软惩罚
- 动态采样

修改1：token优势不受输出长度影响

Motivation： GRPO对长输出不友好

- 原始GRPO中，如果输出 y_i 过长，会稀释单个token的优势 $A_{y_{i,t}} = \frac{1}{|y_i|}A_{y_i}$ ，导致正确的长序列（尤其带CoT的情况）梯度更新幅度不够大
- 解决方案：把 $\frac{1}{|y_i|}$ 换成 $\frac{1}{\text{mean}\{|y_i|\}}$ ，单个token的优势就不受序列长度影响了。

理解：对GRPO中优势的“修正”

DAPO将token优势中的 $\frac{1}{|y_i|}$ 换成 $\frac{1}{\text{mean}\{|y_i|\}}$ ，可以认为是对原始GRPO中单个token优势、完整序列优势的修正：

- 初始化：给定GRPO中完整response y_i 优势 $A_{y_i} = \frac{r_i - \text{mean}\left(\{r_i\}_{i=i}^{G}\right)}{\text{std}\left(\{r_i\}_{i=i}^{G}\right)}$
- 重新定义单个token的优势： $A'_{y_{i,t}} = \frac{1}{\text{mean}\{|y_i|\}}A_{y_i}$ （而在原始GRPO中是 $A_{y_{i,t}} = \frac{1}{|y_i|}A_{y_i}$ ）
- 修正后完整response y_i 优势： $A'_{y_i} = \frac{|y_i|}{\text{mean}\{|y_i|\}}A_{y_i}$
- 依然假设同一个回复中每个token优势相同，且折扣因子为1

修正后的性质

- 不改变优势的正负：好/坏动作依然是好/坏动作
- 对长度超过平均值的序列，优势幅度被放大；长度低于平均值的序列，优势幅度被缩小
- 修正后，同一个prompt的所有response优势之和（即所有token的优势之和）**期望**依然是0（但数值不一定不是0）

证明（不重要）：修正后token优势之和的期望依然是0

- 在GRPO公式中：

记 $B_i = \text{min-clip}\left(\sum_{t=1}^{|y_i|} A_{y_{i,t}} \cdot \frac{\pi_\theta(y_{i,t}|x,y_{i,<t})}{\pi_{old}(y_{i,t}|x,y_{i,<t})}\right) = \text{min-clip}\left(\sum_{t=1}^{|y_i|} \frac{1}{|y_i|} A_{y_i} \cdot \frac{\pi_\theta(y_{i,t}|x,y_{i,<t})}{\pi_{old}(y_{i,t}|x,y_{i,<t})}\right)$

随机变量 B_i 的值依赖于response y_i ，于是GRPO的目标函数可改写成：

$$
\begin{aligned}
J_{GRPO}(\theta) &= \mathbb{E}_{x\sim D(x), y_i\sim\pi_{old}(y_i|x)}\left[B_i\right] \\
&= \mathbb{E}_{x\sim D(x)}\left[\mathbb{E}_{y_i\sim\pi_{old}(y_i|x)}\left[B_i\right]\right] \\
&\approx \mathbb{E}_{x\sim D(x)}\left[\frac{1}{G}\sum_{\substack{\text{每个}x\\ \text{生成}G\text{个}y_i}} B_i\right] \\
&= \mathbb{E}_{x\sim D(x)}\left[\sum_{\substack{\text{每个}x\\ \text{生成}G\text{个}y_i}} \frac{1}{G}\cdot B_i\right]
\end{aligned}
$$

- 在DAPO公式中：

仿照GRPO，记 $B'_i = \text{min-clip}\left(\sum_{t=1}^{|y_i|} A'_{y_{i,t}} \cdot \frac{\pi_\theta(y_{i,t}|x,y_{i,<t})}{\pi_{old}(y_{i,t}|x,y_{i,<t})}\right)$

变形：

$$
\begin{aligned}
B'_i &= \text{min-clip}\left(\sum_{t=1}^{|y_i|} A'_{y_{i,t}} \cdot \frac{\pi_\theta(y_{i,t}|x,y_{i,<t})}{\pi_{old}(y_{i,t}|x,y_{i,<t})}\right) \\
&= \text{min-clip}\left(\sum_{t=1}^{|y_i|} \frac{1}{\text{mean}\{|y_i|\}} A_{y_i} \cdot \frac{\pi_\theta(y_{i,t}|x,y_{i,<t})}{\pi_{old}(y_{i,t}|x,y_{i,<t})}\right) \\
&= \text{min-clip}\left(\sum_{t=1}^{|y_i|} \frac{|y_i|}{\text{mean}\{|y_i|\}}\cdot\frac{1}{|y_i|} A_{y_i} \cdot \frac{\pi_\theta(y_{i,t}|x,y_{i,<t})}{\pi_{old}(y_{i,t}|x,y_{i,<t})}\right) \\
&= \text{min-clip}\left(\sum_{t=1}^{|y_i|} \frac{|y_i|}{\text{mean}\{|y_i|\}}\cdot A_{y_{i,t}} \cdot \frac{\pi_\theta(y_{i,t}|x,y_{i,<t})}{\pi_{old}(y_{i,t}|x,y_{i,<t})}\right) \\
&= \frac{|y_i|}{\text{mean}\{|y_i|\}}\cdot\text{min-clip}\left(\sum_{t=1}^{|y_i|} A_{y_{i,t}} \cdot \frac{\pi_\theta(y_{i,t}|x,y_{i,<t})}{\pi_{old}(y_{i,t}|x,y_{i,<t})}\right) \\
&= \frac{|y_i|}{\text{mean}\{|y_i|\}}\cdot B_i
\end{aligned}
$$

带入DAPO的目标函数：

$$
\begin{aligned}
J_{DAPO}(\theta) &= \mathbb{E}_{x\sim D(x)}\left[\frac{1}{G}\sum_{\substack{\text{每个}x\\ \text{生成}G\text{个}y_i}} B_i'\right] \qquad \text{与}GRPO\text{形式相同} \\
&= \mathbb{E}_{x\sim D(x)}\left[\sum_{\substack{\text{每个}x\\ \text{生成}G\text{个}y_i}} \frac{1}{G}\cdot B_i'\right] \\
&= \mathbb{E}_{x\sim D(x)}\left[\sum_{\substack{\text{每个}x\\ \text{生成}G\text{个}y_i}} \frac{1}{G}\cdot\left(\frac{|y_i|}{\text{mean}\{|y_i|\}}\cdot B_i\right)\right] \\
&= \mathbb{E}_{x\sim D(x)}\left[\sum_{\substack{\text{每个}x\\ \text{生成}G\text{个}y_i}} \frac{|y_i|}{\sum_{1=i}^{G}|y_i|}\cdot B_i\right] \qquad \text{其中} \quad token\text{总数} = G\cdot\text{mean}\{|y_i|\} = \sum_{1=i}^{G}|y_i|
\end{aligned}
$$

对比 $J_{GRPO}(\theta) = \mathbb{E}_{x\sim D(x)}\left[\sum_{\substack{\text{每个}x\\ \text{生成}G\text{个}y_i}} \frac{1}{G}\cdot B_i\right]$，可以发现二者形式是相似的：

- $J_{GRPO}(\theta)$ 是对 B_i 的**算数平均**
- $J_{DAPO}(\theta)$ 是对 B_i 的**加权平均**

在**期望**意义上，二者相等（但在一组具体的采样上，**数值**可能不相等）。因为GRPO中优势之和的期望为0，故DAPO中优势之和的期望也为0（期望为0，数值不一定为0）

> 原理：算数平均与加权平均，在什么条件下期望相等？
> 对于n个随机变量 $\{X_i\}_{i=1}^n$ 和相应的权重 $\sum_{i=1}^n w_i = 1$
> 如果想让算数平均和加权平均的期望相等，即 $\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^n X_i\right] = \mathbb{E}\left[\sum_{i=1}^n w_i X_i\right]$，则需要满足的条件（二者满足其一即可）：
>
> - 平凡情况：所有 X_i 相等（更一般的情况：所有 X_i 的期望相等）
> - 非平凡情况：w_i 和 X_i 互相独立
>
> 此处考虑非平凡情况，即**假设：权重 $\frac{|y_i|}{\sum_{1=i}^{G}|y_i|}$ 和随机变量 B_i 无关，即序列长度和序列的概率、序列的正确性无关**

DAPO公式的另一种形式

DAPO公式更常见的形式是这样的：

$$
J_{DAPO}(\theta) = \mathbb{E}_{x\sim D(x),\{y_i\}_{i=1}^{G}\sim\pi_{old}(y_i|x)}\left[\frac{1}{\sum_{i=1}^{G}|y_i|}\sum_{i=1}^{G}\sum_{t=1}^{|y_i|}\text{min-clip}\left(A_{y_i}\cdot\frac{\pi_\theta(y_{i,t}|x,y_{i,<t})}{\pi_{old}(y_{i,t}|x,y_{i,<t})}\right)\right]
$$

其中 $A_{y_i} = \frac{r_i - \text{mean}\left(\{r_i\}_{i=i}^{G}\right)}{\text{std}\left(\{r_i\}_{i=i}^{G}\right)}$

这种形式和我定义的，修正token优势的形式是等价的：

$$
J_{DAPO}(\theta) = \mathbb{E}_{x\sim D(x),\{y_i\}_{i=1}^{G}\sim\pi_{old}(y_i|x)}\left[\frac{1}{G}\sum_{i=1}^{G}\sum_{t=1}^{|y_i|}\text{min-clip}\left(A'_{y_{i,t}}\cdot\frac{\pi_\theta(y_{i,t}|x,y_{i,<t})}{\pi_{old}(y_{i,t}|x,y_{i,<t})}\right)\right]
$$

其中 $A'_{y_{i,t}} = \frac{1}{\text{mean}\{|y_i|\}} A_{y_i}$

它们都等于

$$J_{DAPO}(\theta) = \mathbb{E}_{x\sim D(x),\{y_i\}_{i=1}^{G}\sim\pi_{old}(y_i|x)} \left[\sum_{i=1}^{G} \frac{|y_i|}{\sum_{i=1}^{G}|y_i|} \cdot \text{min-clip}\left(A_{y_i} \cdot \frac{\pi_\theta(y_{i,t}|x, y_{i,<t})}{\pi_{old}(y_{i,t}|x, y_{i,<t})} \right) \right]$$

比较：GRPO、GSPO、DAPO

优势的计算：

- GRPO：蒙特卡洛计算序列级优势 → 启发式token级优势（算数平均）
- DAPO（另一种token级）：GRPO的序列级优势 → 另一种启发式的token级优势（排除序列长度影响） →累加得到校正后的序列级优势
- GSPO（序列级）：GRPO的序列级优势（不拆解为token级）

背后隐含的假设：

- GRPO：序列中每个token优势相等，token优势受序列长度影响
- DAPO：序列中每个token优势相等，但每个token优势不受序列长度影响
- GSPO：不要启发式计算单个token优势

另一种理解：统计学意义

形式化：

- 从同一个prompt采样G个不同序列 $\{y_i\}_{i=1}^{G}$ （称为一组）
- 每个序列token数为 $\{L_i\}_{i=1}^{G}$
- 序列平均长度 $\bar{L} = \text{mean}(L_i)$
- 第i个序列、第t个token的优势 A_{it} 是个随机变量，且**不可观测**（隐变量）
- 第i个序列，所有token的优势之和 A_i 也是个随机变量，可以观测（蒙特卡洛估计）

现构造几个统计量（可基于观测的 A_i 计算）：

- $Y_1 = \frac{1}{L_i}\sum_{t=1}^{L_i} A_{it} = \frac{1}{L_i} A_i$ （GRPO单个token优势）
- $Y_2 = \frac{1}{G}\sum_{i=1}^{G} Y_1 = \frac{1}{G}\sum_{i=1}^{G} \frac{1}{L_i} A_i$ （GRPO目标函数）
- $Y_3 = \frac{1}{L}\sum_{t=1}^{L_i} A_{it} = \frac{1}{L} A_i$ （DAPO单个token优势）
- $Y_4 = \frac{1}{G}\sum_{i=1}^{G} Y_3 = \frac{1}{G}\sum_{i=1}^{G} \frac{1}{L} A_i = \frac{1}{token总数}\sum_{i=1}^{G} A_i$ （DAPO目标函数）

以及估计隐变量 A_{it} 期望的几种不同口径（期望不可观测）：

- 单个序列内： $\mu_{第i桶内} = \mathbb{E}_{序列内}[A_{it}]$，$i$ 固定，$t = 1...L_i$
- 全量统计： $\mu_{全量} = \mathbb{E}_{全部token}[A_{it}]$，$i = 1...G$ ，$t = 1...L_i$
- 分桶统计（每个序列为一桶）：$\mu_{分桶} = \mathbb{E}_{全部序列}[\mathbb{E}_{序列内}[A_{it}]]$，$i = 1...G$ ，$t = 1...L_i$

则它们在统计上的对应关系：

统计量\期望	$\mu_{\text{第}i\text{桶内}}$	$\mu_{\text{全量}}$	$\mu_{\text{分桶}}$
Y_1	**无偏**	有偏 （无偏条件：所有 $\mu_{\text{第}i\text{桶内}}$ 相等）	有偏 （无偏条件：所有 $\mu_{\text{第}i\text{桶内}}$ 相等）
Y_2	有偏 （无偏条件：$G=1$ 或所有 $\mu_{\text{第}i\text{桶内}}$ 相等）	有偏 （无偏条件：所有 L_i 相等且所有 $\mu_{\text{第}i\text{桶内}}$ 相等）	**无偏**
Y_3	有偏 （无偏条件：所有 $L_i=\bar{L}$）	有偏 （无偏条件：所有 L_i 相等且所有 $\mu_{\text{第}i\text{桶内}}$ 相等）	有偏 （无偏条件：所有 $L_i=\bar{L}$ 且所有 $\mu_{\text{第}i\text{桶内}}$ 相等）
Y_4	有偏 （无实际无偏条件）	**无偏**	有偏 （无偏条件：所有 $\mu_{\text{第}i\text{桶内}}$ 相等）

由此理解GRPO,DAPO,GSPO：

- GRPO：用统计量 Y_2 估计期望 $\mu_{\text{分桶}}$ （假设：每个序列贡献相等）
- DAPO：用统计量 Y_4 估计期望 $\mu_{\text{全量}}$ （假设：每个token贡献相等，等价于用token数量对每个序列的贡献加权）
- GSPO：不考虑隐变量 A_{ij} ，直接使用可观测的 A_i

> 注：
>
> - 虽然DAPO用 Y_3 估计 $\mu_{\text{第}i\text{桶内}}$ 有偏，但各个序列的偏差能互相抵消，于是最后用 Y_4 估计 $\mu_{\text{全量}}$ 是无偏的。
> - 有偏估计并不总是坏的：有偏估计的方差通常比无偏估计小（有偏→欠拟合→模型简单→偏差大方差小）
> - 单个估计有偏，但偏差互相抵消，总体无偏的例子：boosting分类器
>   - 单个分类器只使用部分特征，是有偏的
>   - 所有分类器的平均，偏差互相抵消，是无偏的
>   - boosting既实现了有偏估计的低方差特性，又互相抵消了偏差，从而达到低方差+低偏差的效果

修改2：clip-higher

把min-clip算子中，clip的上限调高：

$$\text{min-clip}\left(A\cdot\frac{\pi_\theta}{\pi_{old}}\right)=\min\left(A\cdot\frac{\pi_\theta}{\pi_{old}},A\cdot\text{clip}\left(\frac{\pi_\theta}{\pi_{old}},1-\varepsilon_{low},1+\varepsilon_{high}\right)\right)$$

其中通常 $\varepsilon_{low}=\varepsilon<\varepsilon_{high}$

目的：

- 原本GRPO的min-clip：更新时丢弃概率过大的好token、概率过小的坏token，以防止和旧策略相差过大
- 实际训练中，“丢弃概率过大的好token”（即clip上限）更容易触发，但人们发现这种上限clip很多是“aha-moment”，（生成aha对于旧策略是低概率的，但在新策略的CoT中是高概率的，而且有利于引导出正确答案）。因此上限clip会抑制CoT训练。
- 因此把clip的上限调高（即对概率大的好token更加宽容）

> GSPO虽然没有clip-higher，但因为是计算整个序列的概率，即使个别token概率高，但整个序列的概率可能被其他token“压下来”，因此这样的高概率token也不会被clip，起到和clip-higher类似的效果

修改3：长度软惩罚

DAPO对回复长度的阈值惩罚比GRPO更缓和：

$$reward_{len}(y_i) = \begin{cases} 0 & |y_i| \le l_{low} \\ -\frac{|y_i|-l_{low}}{l_{high}-l_{low}} & l_{low} < |y_i| < l_{high} \\ -1 & |y_i| \ge l_{high} \end{cases}$$

其中长度在 $l_{low} < |y_i| < l_{high}$ 设置一个线性的软惩罚。

修改4：动态采样

GRPO中，如果生成的一组回复全错（例如训练早期）或全对（训练晚期），这种情况下所有token优势都是0，没办法训练。因此DAPO会丢弃这种数据，只留下回复有对也有错的数据。