重温 On-Policy Distillation（1）

从 SFT / SeqKD 到 OPD —— GKD-style

Penghui Yang

2026.4.20

引言

本系列为On-Policy Distillation的学习笔记，主要从损失函数、梯度更新、优化方向等角度分析 OPD的内在机理，对“最小化student与teacher之间的概率分布”进行更细致的拆解。

第一节我们从LLM Distillation出发，引出GKD-style OPD。在 LLM 训练中，SFT 是最常见的方法之一。严格来说，SFT 不一定都属于 Distillation：若训练序列来自人工标注，本质为人类监督学习；若序列来自更强的 teacher model 采样，才属于典型蒸馏（student-teacher）语境。

1）SFT：hard label 的 token 学习

给定前缀序列

$$s = (t_1, t_2, \ldots, t_{n-1}),$$

SFT 的目标是在下一位置生成数据中的真实 token t_n。这等价于把目标分布视为 one-hot：t_n 概率为 1，其它 token 概率为 0。对应损失为

$$L_t = -\log \pi_\theta(t_n \mid s).$$

其中 π_θ 是 student 的参数化策略（条件概率分布）。

2）SeqKD：soft label 的分布学习

SeqKD 同样由 teacher 采样完整序列，但 teacher 不只提供“最终采样 token”，还提供该位置完整词表分布。因此 student 的学习目标不再是 one-hot，而是贴近 teacher

的全分布：

$$L_t = -\sum_{i=1}^{V} \pi_{\text{teacher}}(t_i \mid s) \log \pi_\theta(t_i \mid s).$$

该目标本质上与 teacher/student 在该位置分布之间的 KL 相关（称为Forward KL）。

3）SFT 与 SeqKD 的关系

SFT 可以看作 SeqKD 的退化情况：

- 当 $t_i = t_n$ 时，$\pi_{\text{teacher}}(t_i \mid s) = 1$；
- 否则，$\pi_{\text{teacher}}(t_i \mid s) = 0$。

但真实 teacher 分布通常并非严格 one-hot。因此 SeqKD 相比 SFT 提供更丰富的 soft label 信息：不仅告诉 student 哪个 token 对，也告诉它其它候选 token 的相对概率。

4）从 SeqKD 到 OPD 的直觉

在 RL 大规模用于 LLM 之前，SFT 和 SeqKD 可能都是核心范式（今天 SFT 依然很重要）。它们共同点是：都在 token-level 提供监督，并可通过梯度下降直接优化参数。

顺着 SeqKD 的形式看，OPD 与其高度对称：

- 都需要比较 teacher/student 在序列各位置的词表分布；
- 都在 token-level 计算损失；
- 关键差异在于该序列由谁采样（teacher 或 student）。

由此可自然得到 OPD 的一条实现路径：**GKD-style**（分布匹配、可导优化）；另一条路径是 **PG-style**（通过 Policy Gradient 更新 student），这条路径从RL的视角引入或许更加自然，留作下节细讲。

5）补充：SFT和SeqKD的特点和局限性

关于SFT和SeqKD，有不少paper讨论它们在算法层面的效果和局限性（如泛化性、灾难性遗忘等等）。二者都属于off-policy，即采样序列来自teacher，当然更直观的解释（为什么不如on-policy）可以理解为student在学习的过程中，我们不知道它学的怎么样；而on-policy由于每次更新时用到的采样序列都是来自最新（或最近）的student，因此实际上提供了student在学习过程中的逐步变化。

另外，这里再提醒一下，SFT之所以应用更广泛，除了实现上比较简单，另一个特点是SFT可以用在black-box model的蒸馏上。例如一些最先进的闭源模型，我们没法获取token的logit，只能把完整的序列视作one-hot分布让student拟合。此外，也有工作把on-policy用在了black-box的蒸馏上（如general adversarial distillation），感兴趣的朋友可以自行了解。

参考链接： 知乎专栏·两种形式的OPD

重温 On-Policy Distillation（2）

从RL到OPD——PG-style

Penghui Yang

2026.4.21

前言

上一节我们从SFT、SeqKD出发，探讨了几种LLM Distillation之间的联系，并自然引出OPD可以视为 SeqKD的镜像版本，也即得到GKD-style的OPD。关于它的具体定义和损失函数，将留到后文说明。

本节主要探讨一下，同样作为on-policy的学习手段，OPD和RL之间存在什么关联？毕竟RL相对热门的时间更早一些，大家可能对它更熟悉。从RL的角度审视OPD，可以得到OPD的另一种技术路径：PG（Policy Gradient）-style OPD。

Part 1：先从 RL 说起

从去年开始，RLVR在社区大规模兴起，时至今日仍然是LLM post-training中十分重要的一环。它的一个重要特点是鼓励model自己的探索，而不像SFT那样对model每一次生成next token都加以强监督；换句话说，RLVR只关心model生成的完整序列好不好，不关心这个序列的每一处token是否生成的合理。这可以极大地缓解由SFT可能带来的“死记硬背”的泛化性差的问题。

RL作为一种on-policy的手段，每次更新时所用到的序列都是由（最新的，或者最近的）model自己采样得到的。在上一节我们提到，这种on-policy的手段将有助于在LLM的学习过程中提供【实时】的检测效果，即通过每一次实时的采样，我们都能获取到model最新的学习情况，进而提供更精准的指导信号。作为代价，RL在训练中需要增加model实时的采样（而SFT/SeqKD等都是提前把序列准备好了），这显著拖慢了训练速度。

现在让我们再次回顾一下常见RLVR的一个核心特点：只关心【完整序列】的好坏，不关系【每个token】的好坏。上面说到这样做的好处是防止model死记硬背，同时可以激励model自由探索；但是也可以说它所提供的指导信号太稀疏，这样的指导到底

比提供更细粒度的指导是好是坏，我们不去探讨。此外，实际上除了主流的RLVR，在早期传统PPO算法中也可以设计一个专门用来提供逐token监督信号的模型，但是由于训练时model太多、成本太高等因素，现在已经很少见了。

实际上除了训练效果上的讨论，RL与SFT等还有一个重要区别是，如何根据损失函数去优化模型参数？这涉及到loss是否直接针对参数可导的问题，需要仔细说说，方便后面引出PG-style的OPD。

1）RL：通过Policy Gradient更新model参数

在上一节中，我们提到了SFT和SeqKD的损失函数（前者可以视为后者的一个退化），例如对于SeqKD（这里给出完整的KL散度版本，和上一节求梯度是等价的，只是差了个常数项）：

$$L_t = D_{\mathrm{KL}}(\pi_{\text{teacher}}(\cdot \mid s) \,\|\, \pi_\theta(\cdot \mid s)) = \sum_{i=1}^{V} \pi_{\text{teacher}}(t_i \mid s) \log \frac{\pi_{\text{teacher}}(t_i \mid s)}{\pi_\theta(t_i \mid s)}.$$

通过这个Loss，我们要去优化的是student model的权重参数θ，最终达到的目标是在这个token位置处，student生成词表中每一个token的概率$\pi_\theta(t_i \mid s)$，都尽可能与teacher保持一致。

注意，不要忘记我们的优化对象——model的权重参数θ，很明显，上面这个定义在token上的L_t，很明显是直接针对 θ可导的，根本原因是这个token处的监督信号是针对**model概率分布**π_θ的函数，所以可以由链式法则将梯度一路传回去。

那么切换到RL场景，我们前面提到，无论是序列级别的监督信号（以RLVR为典型），还是token级别的监督信号（以PPO为代表），它们的损失（或者称为奖励），都是定义在**【采样出来的token】**上的（前者通过广播或平均机制，从序列级信号得到token级的信号），而不是采样时的概率分布。

需要特别注意的点在于，model通过权重参数从输入一路前向传播到输出最终的词表概率分布，都是可导的（这里只讨论Dense model）。而采样操作：从最终的词表概率分布选出某一个token，这个过程是不可导的（例如argmax的不可导性）。但由于RL中的监督信号都是定义在【采样出来的token（不妨称为y）】上的，因此梯度从监督信号往回传到y之后就被截断掉了，因为无法定义一个采样操作与其概率分布的导数（非光滑映射）。

用符号来表示的话，RL的计算图

$$\theta \leftrightarrow \pi_\theta \rightarrow \text{sample } y \leftrightarrow R(y)$$

在$\pi_\theta \rightarrow$ sample y反向传播时梯度会断掉（即$\frac{\partial y}{\partial \pi_\theta}$不存在）。这里$R(y)$即表示我们的监督信号（或称为奖励）是定义在被采样出来的token上的，而前面的SeqKD的监督信号是直接定义在采样时的概率分布上的。

既然RL的信号梯度无法直接回传到参数 θ上，那么我们又该如何根据这个信号去更新model参数呢？核心思想是**虽然单次 sample 不可导，但期望（多次采样）是可导的**。具体来说，我们不直接去优化$R(y)$，而是去优化$R(y)$的期望：

$$J(\theta) = E_{y\sim\pi_\theta}[R(y)] = \sum_y \pi_\theta(y)R(y)$$

这里$\pi_\theta(y)$就表示在当前输入下，model采样生成y的概率。这样转为求期望，我们就从**【采样出来的token】**引入了**【采样时的概率分布】**，进而实现梯度回传（此时可导）：

$$\nabla_\theta J(\theta) = \sum_y \nabla_\theta\, \pi_\theta(y)\, R(y)$$

为了更清晰的展开梯度表达式，这里需要用到对数导数技巧：

$$\nabla_\theta\, \pi_\theta(y) = \pi_\theta(y)\, \nabla_\theta \log \pi_\theta(y)$$

代入有

$$\nabla_\theta J(\theta) = \sum_y \pi_\theta(y)\, R(y)\, \nabla_\theta \log \pi_\theta(y) = \mathbb{E}_{y\sim\pi_\theta}\left[R(y)\, \nabla_\theta \log \pi_\theta(y)\right]$$

这就是**Policy Gradient**，它避开了不可导的$\frac{\partial y}{\partial \theta}$，转化为 $\nabla_\theta \log \pi_\theta(y)$，就能正常使用梯度下降去更新参数$\theta$了。

2）从RLVR到OPD：稀疏奖励的改进

前面的推导解决了RL如何优化model参数的问题，现在让我们回到RL（后面均指RLVR）的第一个问题：奖励稀疏。也就是前面所说，我们提供的信号是针对完整序列而言，而不关心每一个token的好坏。那我们能不能像SeqKD/SFT那样，也去关注每个token的好坏呢？我们暂且不讨论这样做最终导致的优劣，但从信息量来说明显这样提供的指导信号是更丰富的。这就是OPD的**PG-style**的实现。

具体来说，在上面求RL的梯度时，把$R(y)$直接换成student与teacher生成当前token y的概率比值，即

$$\nabla_\theta J(\theta) = \mathbb{E}_{y\sim\pi_\theta}\left[\log \frac{\pi_\theta(y)}{\pi_{teacher}(y)}\, \nabla_\theta \log \pi_\theta(y)\right]$$

这里需要注意的是，$\log \frac{\pi_\theta(y)}{\pi_{teacher}(y)}$本身是会引入关于$\theta$的梯度项的，但在这里我们相当于施加了**stop-gradient**操作，即梯度完全由最右侧$\nabla_\theta \log \pi_\theta(y)$产生，而前面的$\log \frac{\pi_\theta(y)}{\pi_{teacher}(y)}$只提供一个标量的更新力度。

另外，这里（包括前面RL的实现）的期望在工程上不可直接求解，因此使用Monte

Carlo采样去近似。简单来说就是多次采样取均值，不过需要稍微提醒一下，这里说的多次采样都是针对序列级别，例如给定一个prompt，生成多个响应序列，而上面为了直观对比，推导都是基于token级别的loss，大家也可以换成sequence级别的loss（利用概率乘积即可从token得到完整序列）。

小结：PG-style OPD vs RLVR

最后小结一下，在RLVR中，$R(y)$本质上是基于完整序列的信号，然后通过广播折算到每个token上，也就是说所有token的贡献都是一样的。而OPD则使用$\log \frac{\pi_\theta(y)}{\pi_{teacher}(y)}$作为逐token各自的指导更新力度，因此相比于RLVR可以获得更dense的信号。

重温 On-Policy Distillation（3）

GKD-style的更新模式

Penghui Yang

2026.4.22

前言

第一节我们从SFT和SeqKD出发，得到了GKD-style的OPD，并指出它实际上就是SeqKD的镜像版本；第二节我们从RL出发，得到了PG-style的OPD，并指出它可以视为一种提供密集信号的RLVR。这一节我们先给出GKD-style OPD更正式的数学定义，并通过梯度分析它是如何促进student更新参数的，从动力学优化的角度看看它到底实现了怎样的效果，同时为后面和PG-style的对比做个铺垫。

GKD-style OPD

回顾SeqKD的Loss函数，这里直接给出Sequence版本：

$$L_{SeqKD}(\theta) = \sum_{t} D_{KL}(\pi_{teacher}(\cdot|s_t), \pi_\theta(\cdot|s_t))$$

这里t表示的是当前序列τ的第t个token位置，s_t表示当前位置t时的输入前缀序列（用a_t表示在第t个token位置处实际采样的token），即

$$s_t = (token_{prompt}, a_1, a_2 \dots a_{t-1})$$

同时要注意，上面使用的$\pi(\cdot|s_t)$（表示的是一个概率分布），而不是$\pi(a_t|s_t)$（选择某个token的概率值）。完整的展开即有

$$L_{SeqKD}(\theta) = \sum_{t}\sum_{v} \pi_{teacher}(a_v|s_t) log \frac{\pi_{teacher}(a_v|s_t)}{\pi_\theta(a_v|s_t)}$$

上式的$\sum_t$表示对一个sequence的每一个位置的token的loss求和，$\sum_v$表示在每一个token位置t处，对词表中所有token的概率信号求和（实际上也就是完整KL的展开式，它是定义在每个token上的）。

这里还有一个需要注意的点是，SeqKD中的序列（或者说序列中的每一个token）是由teacher采样而来，这被称为**forward-KL**，即有

$$\tau \sim \pi_{teacher}, a_t \sim \pi_{teacher}(\cdot|s_t)$$

现在我们直接交换KL散度公式中teacher和student（即θ）的位置，是不是就能得到GKD-style OPD呢？

$$L_{OPD}(\theta) = \sum_t D_{KL}(\pi_\theta(\cdot|s_t), \pi_{teacher}(\cdot|s_t)) = \sum_t \sum_v \pi_\theta(a_v|s_t) log \frac{\pi_\theta(a_v|s_t)}{\pi_{teacher}(a_v|s_t)}$$

这样做的话，好像只是在计算KL时换了个算法，但这里忽略了一点：on-policy意味着轨迹是由student自己采样的，因此还有

$$\tau \sim \pi_\theta, a_t \sim \pi_\theta(\cdot|s_t)$$

实际上这就是**reverse-KL**的实现，它不单单只是在计算loss时把SeqKD中的student和teacher换了个位置，更重要的是采样来源也变了。这样做会带来一个与SeqKD不同的后果： SeqKD采用foward KL追求的是student的词表概率分布尽可能和teacher完全对齐（mode-covering），而OPD采用reverse KL追求的是一种mode-seeking，具体见下文。

GKD-style 的梯度

我们推导一下OPD的梯度，将单步 reverse KL 展开：

$$\ell_t(\theta) = \mathrm{KL}(\pi_\theta(\cdot \mid s_t) \,\|\, \pi_{\text{teacher}}(\cdot \mid s_t)) = \sum_v \pi_\theta(a_v \mid s_t) \log \frac{\pi_\theta(a_v \mid s_t)}{\pi_{\text{teacher}}(a_v \mid s_t)}.$$

对参数 θ 求导（此处省略过程，很简单），可得：

$$\boxed{\nabla_\theta \, \ell_t(\theta) = \sum_v \left(\log \frac{\pi_\theta(a_v \mid s_t)}{\pi_{\text{teacher}}(a_v \mid s_t)} + 1 \right) \nabla_\theta \, \pi_\theta(a_v \mid s_t)}$$

因此最终完整的 GKD-style OPD 梯度为：

$$\boxed{\nabla_\theta \, L_{\text{OPD}}(\theta) = \sum_t \sum_v \left(\log \frac{\pi_\theta(a_v \mid s_t)}{\pi_{\text{teacher}}(a_v \mid s_t)} + 1 \right) \nabla_\theta \, \pi_\theta(a_v \mid s_t)}$$

这个式子的关键含义是：**在每个访问到的前缀上，词表中的所有 token 都会同时产生梯度信号**。因此，GKD-style 优化的不是某个 sampled token 的概率，而是 student

在该位置上的**整个 next-token 分布**。这就是所谓的“推动完整词表概率对齐”。

需要特别注意的是，这里不能简单地把

$$\left(\log \frac{\pi_\theta(a_v \mid s_t)}{\pi_{\text{teacher}}(a_v \mid s_t)} + 1\right)$$

理解为“第 v 个 token 单独的更新方向”。因为 $\nabla_\theta \pi_\theta(a_v \mid s_t)$ 是参数空间中的高维向量，不同 token 的梯度彼此耦合；再加上 softmax 的归一化约束，**一个 token 概率的变化会同时影响其他 token 的概率**。因此，如果想分析“某个 token 会被推高还是压低”，更合适的对象不是参数梯度，而是**logit梯度**。

从 logit 角度理解mode-seeking下的概率分布对齐

下面我们先公式推导，然后再列举实际例子，看看reverse-KL会导致student具体怎么更新。

公式分析

设当前位置 student 的 logits 为 z_v（即在最终softmax归一化得到概率之前的值），对应概率为

$$p_v = \pi_\theta(a_v \mid s_t), \qquad q_v = \pi_{\text{teacher}}(a_v \mid s_t).$$

则单步 reverse KL 为

$$\ell_t = \sum_v p_v \log \frac{p_v}{q_v}.$$

对某个 token a_j 的 logit z_j 求导，可以得到（这一步相对比较复杂，后面让GPT写了一个详细过程，此处只给最终结果）：

$$\boxed{\frac{\partial \ell_t}{\partial z_j} = p_j \left(\log \frac{p_j}{q_j} - \sum_v p_v \log \frac{p_v}{q_v}\right)}$$

也就是

$$\boxed{\frac{\partial \ell_t}{\partial z_j} = p_j \left(\log \frac{p_j}{q_j} - \mathrm{KL}(p\|q)\right)}$$

仔细看这个式子，左侧$\frac{\partial \ell_t}{\partial z_j}$表示当前student针对$z_j$的更新方向，即让$z_j$变大或者变小（对应概率$p_j$变大变小）。而右侧括号内，第一项$\log \frac{p_j}{q_j}$可以视为针对词表中的这个token a_j，student和teacher的不匹配程度；第二项 $\mathrm{KL}(p\|q)$可以视为针对词表中的所有token（即完整的词表概率分布），student和teacher的不匹配程度。

也就是说，括号内可以视为：**当前token的不匹配程度与全局token的不匹配程度**

的差异。我们接着看完整的梯度下降更新过程为

$$z_j \leftarrow z_j - \eta \frac{\partial \ell_t}{\partial z_j}.$$

因此：

- 若 $\log \frac{p_j}{q_j} < \mathrm{KL}(p\|q)$，则 $\frac{\partial \ell_t}{\partial z_j} < 0$，梯度下降会**提高**该 token 的 logit；
- 若 $\log \frac{p_j}{q_j} > \mathrm{KL}(p\|q)$，则 $\frac{\partial \ell_t}{\partial z_j} > 0$，梯度下降会**降低**该 token 的 logit。

也就是说：

- 如果某个 token 的不匹配程度高于平均水平，梯度为正，梯度下降会压低它；
- 如果某个 token 的不匹配程度低于平均水平，梯度为负，梯度下降会抬高它。

这说明 reverse KL 的行为并不是“逐 token 地把 student 的概率值硬拉到 teacher 的对应值”，即针对某个token，如果teacher的生成概率大于student，就提升student的概率；如果teacher的生成概率小于student，就降低student的概率。而是更像一种**对整个分布形状的重塑**：它会强烈打压 student 过度押注、而 teacher 不认可的高概率模式，同时把概率重新分配到 teacher 支持的候选上。这也正是 reverse KL 常被称为 *mode-seeking* 的原因。

具体例子

假设当前前缀下 student 和 teacher 的分布分别为：

$$p = (0.2,\ 0.3,\ 0.5), \qquad q = (0.7,\ 0.2,\ 0.1).$$

先计算各 token 的 log-ratio：

$$\begin{aligned} r_1 &= \log \frac{0.2}{0.7} \approx -1.25, \\ r_2 &= \log \frac{0.3}{0.2} \approx 0.41, \\ r_3 &= \log \frac{0.5}{0.1} \approx 1.61. \end{aligned}$$

再计算当前的 reverse KL：

$$\mathrm{KL}(p\|q) = 0.2 \log \frac{0.2}{0.7} + 0.3 \log \frac{0.3}{0.2} + 0.5 \log \frac{0.5}{0.1} \approx 0.68.$$

于是各个 token 的 logit 梯度符号为：

$$\frac{\partial \ell_t}{\partial z_j} = p_j (r_j - 0.68).$$

逐一分析：

- t_1：$r_1 = -1.25 < 0.68$，因此 $\frac{\partial \ell_t}{\partial z_1} < 0$，梯度下降会**提高** t_1 的 logit，从而提高其概率；
- t_2：$r_2 = 0.41 < 0.68$，因此 $\frac{\partial \ell_t}{\partial z_2} < 0$，梯度下降也会**提高** t_2 的 logit；
- t_3：$r_3 = 1.61 > 0.68$，因此 $\frac{\partial \ell_t}{\partial z_3} > 0$，梯度下降会**降低** t_3 的 logit，从而压低其概率。

可见，reverse KL 的行为并不是简单地“student 概率高于 teacher 就压低，低于 teacher 就抬高”。例如在这个例子里，t_2 虽然 student 的概率 0.3 略高于 teacher 的 0.2，但它仍然会被**抬高**，因为当前 student 最大的问题是对 t_3 赋予了过高概率； reverse KL 会优先压低这种 teacher 明显不认可的主峰，并把概率质量重新分配到 teacher 更支持的其它 token 上。

SeqKD采用的forward-KL

同样都是实现“完整词表概率对齐”，相比于 GKD-style OPD 常采用的 reverse-KL， SeqKD 中更自然的目标通常是 forward-KL：

$$\ell_t^{\mathrm{fKL}} = \mathrm{KL}(q \,\|\, p) = \sum_v q_v \log \frac{q_v}{p_v},$$

其中

$$p_v = \pi_\theta(a_v \mid s_t), \qquad q_v = \pi_{\mathrm{teacher}}(a_v \mid s_t).$$

由于 teacher 分布 q 不依赖于 student 参数，因此 forward-KL 也可以写成

$$\ell_t^{\mathrm{fKL}} = \underbrace{\sum_v q_v \log q_v}_{\text{与 student 无关的常数}} - \sum_v q_v \log p_v.$$

因此，最小化 forward-KL 等价于最小化 teacher 软标签下的交叉熵：

$$\ell_t^{\mathrm{fKL}} \equiv -\sum_v q_v \log p_v.$$

对 logit 的梯度

设 student 在该位置的 logits 为 z_j，则 softmax 概率为

$$p_j = \frac{e^{z_j}}{\sum_k e^{z_k}}.$$

forward-KL 对 logit 的梯度有一个非常经典且简洁的结果：

$$\boxed{\frac{\partial \ell_t^{\mathrm{fKL}}}{\partial z_j} = p_j - q_j}$$

因此梯度下降更新为

$$z_j \leftarrow z_j - \eta(p_j - q_j).$$

于是可以直接得到：

- 若 $p_j > q_j$，则 $\frac{\partial \ell_t^{\mathrm{fKL}}}{\partial z_j} > 0$，梯度下降会**降低**该 token 的 logit，从而压低其概率；
- 若 $p_j < q_j$，则 $\frac{\partial \ell_t^{\mathrm{fKL}}}{\partial z_j} < 0$，梯度下降会**提高**该 token 的 logit，从而提升其概率；
- 若 $p_j = q_j$，则该 token 在当前位置上达到局部匹配，不再产生 logit 梯度。

这说明 forward-KL 的行为非常直接：它会**逐 token 地**把 student 的概率往 teacher 的对应概率上拉。也就是说，对每个 token 来说：

- teacher 给得比 student 高，就把 student 往上抬；
- teacher 给得比 student 低，就把 student 往下压。

因此，forward-KL 更像是一种**逐坐标的分布校正**。

具体例子

仍然使用前面的同一个例子：

$$p = (0.2,\ 0.3,\ 0.5), \qquad q = (0.7,\ 0.2,\ 0.1).$$

则各 token 的 logit 梯度为：

$$\frac{\partial \ell_t^{\mathrm{fKL}}}{\partial z_j} = p_j - q_j.$$

逐一计算：

$$\begin{aligned}
\frac{\partial \ell_t^{\mathrm{fKL}}}{\partial z_1} &= 0.2 - 0.7 = -0.5, \\
\frac{\partial \ell_t^{\mathrm{fKL}}}{\partial z_2} &= 0.3 - 0.2 = +0.1, \\
\frac{\partial \ell_t^{\mathrm{fKL}}}{\partial z_3} &= 0.5 - 0.1 = +0.4.
\end{aligned}$$

因此梯度下降会：

- **提高** t_1 的 logit（因为 student 明显低估了 teacher 最支持的 token）；
- **轻微压低** t_2 的 logit（因为 student 略微高估了它）；

- **明显压低** t_3 的 logit（因为 student 严重高估了它）。

把它整理成表格，更直观一些：

Token	p_j	q_j	rKL 方向	rKL 力度	fKL 方向	fKL 力度
t_1	0.2	0.7	↑	强	↑	强
t_2	0.3	0.2	↑ **(注意!)**	弱	↓	弱
t_3	0.5	0.1	↓	强	↓	强

可以看到：t_1 和 t_3 在两种 KL 下更新方向一致，但 t_2 的更新方向**完全相反**——这正是 reverse-KL（mode-seeking）与 forward-KL（mode-covering）在优化行为上的核心差异。forward-KL 的行为与直觉高度一致：**哪个 token 概率偏高就压低，哪个 token 概率偏低就抬高。** 与前面的 reverse-KL 相比，它不会先看“这个 token 的失配程度是否高于全局平均”，而是直接做**逐 token 的点对点校正**。

forward-KL 与 reverse-KL 的差异

虽然两者都属于“完整词表概率对齐”，但它们的优化偏好并不相同：

- **forward-KL** 更关注 teacher 所有的区域是否被 student 覆盖到。因此它倾向于让 student 去**覆盖 teacher 的所有主要模式**，因此被称为 *mode-covering*。
- **reverse-KL** 更关注 student 当前押注的区域是否得到了 teacher 的认可。因此它倾向于**打压 student 错押的主峰，并集中到 teacher 更认可的少数模式上**，因此被称为 *mode-seeking*。

小结

所以，SeqKD 与 GKD-style OPD 的共同点在于：它们都属于**可导的完整词表分布对齐**；而它们的关键区别在于：

- SeqKD 是 **teacher rollout / off-policy**，并常采用 **forward-KL**；
- GKD-style OPD 是 **student rollout / on-policy**，并常采用 **reverse-KL**。

下一篇预告

PG-style OPD在梯度更新时会产生和GKD-style不一样的走向吗？

补充：针对 logit 的梯度推导

(以下内容由GPT5.4生成)

前文在讨论 GKD-style OPD 的 reverse-KL 与 SeqKD 的 forward-KL 时，直接给出了它们对 student logits 的梯度结论。为了让推导更完整，这里单独证明这两个结果。

设在某个固定前缀 s_t 下，student 的词表大小为 $|\mathcal{V}|$，其 logits 为

$$z = (z_1, z_2, \dots, z_{|\mathcal{V}|}),$$

softmax 概率为

$$p_j = \pi_\theta(a_j \mid s_t) = \frac{e^{z_j}}{\sum_k e^{z_k}},$$

teacher 分布记为

$$q_j = \pi_{\text{teacher}}(a_j \mid s_t).$$

其中：

- p_j 依赖于 student 的 logits z；
- q_j 是 teacher 给定的常数，与 student 参数无关。

softmax 的导数

后续推导都要用到 softmax 的经典导数公式：

$$\boxed{\frac{\partial p_v}{\partial z_j} = p_v(\delta_{vj} - p_j)}$$

其中 δ_{vj} 是 Kronecker delta：

$$\delta_{vj} = \begin{cases} 1, & v = j, \\ 0, & v \neq j. \end{cases}$$

这个结果可以由 softmax 直接求导得到：

$$p_v = \frac{e^{z_v}}{\sum_k e^{z_k}}.$$

对 z_j 求导，

$$\frac{\partial p_v}{\partial z_j} = \frac{\delta_{vj} e^{z_v} \sum_k e^{z_k} - e^{z_v} e^{z_j}}{\left(\sum_k e^{z_k}\right)^2} = \frac{e^{z_v}}{\sum_k e^{z_k}} \left(\delta_{vj} - \frac{e^{z_j}}{\sum_k e^{z_k}}\right),$$

即

$$\frac{\partial p_v}{\partial z_j} = p_v(\delta_{vj} - p_j).$$

这个式子揭示了 softmax 的耦合性：提高某个 token 的 logit z_j，不仅会提高它自己的概率 p_j，还会压低其他所有 token 的概率。

1. reverse-KL 对 logit 的梯度推导

单步 reverse-KL 定义为

$$\ell_t^{\mathrm{rKL}} = \mathrm{KL}(p\|q) = \sum_v p_v \log \frac{p_v}{q_v}.$$

我们的目标是计算

$$\frac{\partial \ell_t^{\mathrm{rKL}}}{\partial z_j}.$$

第一步：对 p_v 求偏导

将 loss 看成 p 的函数：

$$\ell_t^{\mathrm{rKL}} = \sum_v p_v \log \frac{p_v}{q_v}.$$

对某个 p_v 求偏导：

$$\frac{\partial \ell_t^{\mathrm{rKL}}}{\partial p_v} = \frac{\partial}{\partial p_v}\left(p_v \log \frac{p_v}{q_v}\right) = \log \frac{p_v}{q_v} + 1.$$

第二步：链式法则

利用链式法则，

$$\frac{\partial \ell_t^{\mathrm{rKL}}}{\partial z_j} = \sum_v \frac{\partial \ell_t^{\mathrm{rKL}}}{\partial p_v}\frac{\partial p_v}{\partial z_j}.$$

代入前面的两个结果：

$$\frac{\partial \ell_t^{\mathrm{rKL}}}{\partial z_j} = \sum_v \left(\log \frac{p_v}{q_v} + 1\right) p_v(\delta_{vj} - p_j).$$

第三步：展开并化简

将上式拆成两项：

$$\frac{\partial \ell_t^{\mathrm{rKL}}}{\partial z_j} = \sum_v \left(\log \frac{p_v}{q_v} + 1\right) p_v\delta_{vj} - \sum_v \left(\log \frac{p_v}{q_v} + 1\right) p_v p_j.$$

第一项中，δ_{vj} 只在 $v = j$ 时为 1，因此

$$\sum_v \left(\log \frac{p_v}{q_v} + 1\right) p_v\delta_{vj} = p_j \left(\log \frac{p_j}{q_j} + 1\right).$$

第二项中，p_j 与求和变量 v 无关，可以提出来：

$$\sum_v \left(\log \frac{p_v}{q_v} + 1\right) p_v p_j = p_j \sum_v p_v \left(\log \frac{p_v}{q_v} + 1\right).$$

继续展开：

$$= p_j \left(\sum_v p_v \log \frac{p_v}{q_v} + \sum_v p_v\right).$$

由于 $\sum_v p_v = 1$，因此

$$= p_j \left(\sum_v p_v \log \frac{p_v}{q_v} + 1\right).$$

于是整体变为

$$\frac{\partial \ell_t^{\mathrm{rKL}}}{\partial z_j} = p_j \left(\log \frac{p_j}{q_j} + 1\right) - p_j \left(\sum_v p_v \log \frac{p_v}{q_v} + 1\right).$$

两边的常数项 $+1$ 相消，得到

$$\boxed{\frac{\partial \ell_t^{\mathrm{rKL}}}{\partial z_j} = p_j \left(\log \frac{p_j}{q_j} - \sum_v p_v \log \frac{p_v}{q_v}\right)}$$

而

$$\sum_v p_v \log \frac{p_v}{q_v} = \mathrm{KL}(p\|q),$$

因此也可写成更紧凑的形式：

$$\boxed{\frac{\partial \ell_t^{\mathrm{rKL}}}{\partial z_j} = p_j \left(\log \frac{p_j}{q_j} - \mathrm{KL}(p\|q)\right)}$$

结果解释

这个式子表明，reverse-KL 对某个 logit 的更新，并不只由该 token 的局部差异 $\log \frac{p_j}{q_j}$ 决定，还会与整个分布的平均失配程度 $\mathrm{KL}(p\|q)$ 比较。因此它体现的是一种**全局重分配**行为，而不是简单的逐 token 点对点拉齐。

2. forward-KL 对 logit 的梯度推导

单步 forward-KL 定义为

$$\ell_t^{\mathrm{fKL}} = \mathrm{KL}(q\|p) = \sum_v q_v \log \frac{q_v}{p_v}.$$

由于 q_v 与 student 参数无关，这个式子可以写成

$$\ell_t^{\mathrm{fKL}} = \underbrace{\sum_v q_v \log q_v}_{\text{常数}} - \sum_v q_v \log p_v.$$

因此只需要对

$$-\sum_v q_v \log p_v$$

求导。

第一种推法：直接使用 $\log p_v$ 的导数

先写出 softmax 的一个常用恒等式：

$$\log p_v = z_v - \log \sum_k e^{z_k}.$$

因此对 z_j 求导：

$$\frac{\partial \log p_v}{\partial z_j} = \delta_{vj} - p_j.$$

于是

$$\frac{\partial \ell_t^{\mathrm{fKL}}}{\partial z_j} = -\sum_v q_v \frac{\partial \log p_v}{\partial z_j} = -\sum_v q_v(\delta_{vj} - p_j).$$

展开得

$$= -\sum_v q_v \delta_{vj} + \sum_v q_v p_j.$$

第一项为

$$-\sum_v q_v \delta_{vj} = -q_j,$$

第二项中 p_j 可提出求和号外，并利用 $\sum_v q_v = 1$，得到

$$\sum_v q_v p_j = p_j.$$

因此

$$\boxed{\frac{\partial \ell_t^{\mathrm{fKL}}}{\partial z_j} = p_j - q_j}$$

第二种推法：通过链式法则

为了与 reverse-KL 的推法形式保持一致，也可以从链式法则出发：

$$\ell_t^{\mathrm{fKL}} = \sum_v q_v \log \frac{q_v}{p_v}.$$

对 p_v 求偏导：

$$\frac{\partial \ell_t^{\mathrm{fKL}}}{\partial p_v} = -\frac{q_v}{p_v}.$$

再利用

$$\frac{\partial p_v}{\partial z_j} = p_v(\delta_{vj} - p_j),$$

得到

$$\frac{\partial \ell_t^{\mathrm{fKL}}}{\partial z_j} = \sum_v \left(-\frac{q_v}{p_v}\right) p_v(\delta_{vj} - p_j) = -\sum_v q_v(\delta_{vj} - p_j),$$

后续化简与上面完全相同，最终得到

$$\boxed{\frac{\partial \ell_t^{\mathrm{fKL}}}{\partial z_j} = p_j - q_j}$$

结果解释

forward-KL 的梯度形式非常直接：

$$\frac{\partial \ell_t^{\mathrm{fKL}}}{\partial z_j} = p_j - q_j.$$

因此它是一个典型的**逐 token 点对点校正**：

- 若 $p_j > q_j$，则梯度为正，梯度下降会压低该 token；
- 若 $p_j < q_j$，则梯度为负，梯度下降会抬高该 token。

这与 reverse-KL 的全局重分配行为形成了鲜明对比：

- **forward-KL** 更像“把 student 的每个概率值逐项拉向 teacher”；
- **reverse-KL** 更像“优先打压 student 错押的主峰，并重塑整个分布形状”。

重温 On-Policy Distillation（4）

PG-style的更新模式

Penghui Yang

2026.4.23

前言

前两节我们分别从经典的Knowledge Distillation和RL的视角介绍了GKD-style和PG-style 的OPD，第三节我们对其中的GKD-style的优化动力学做了更细致的分析，特别是与SeqKD相比其mode-seeking的优化差异。这一节我们介绍PG-style OPD是如何对model参数进行更新的，并在最后与GKD-style做一个正面对比。

1）Policy Gradient 的更新

Loss 函数

第二节我们简单提到了PG-style OPD可简单视为把RLVR的奖励信号密集化到每个token上，这里我们直接给出PG-style OPD 的sequence级损失函数（注意这里没有采用RL的advantage写法；在RL场景下，我们希望reward/ advantage越大越好）为：

$$L_{\text{PG-OPD}}(\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\cdot|x)}\left[\sum_{t=1}^{T} \text{sg}(C_t)\cdot \log \pi_\theta(a_t \mid s_t)\right]$$

其中 a_t 是 student 在位置 t 实际采样出的 token， $s_t = (x, a_1, \ldots, a_{t-1})$ 是对应的前缀，而 C_t 是 reverse-KL 的单样本近似（Schulman K1 近似）：

$$C_t = \log \pi_\theta(a_t \mid s_t) - \log \pi_{teacher}(a_t \mid s_t).$$

注意这里**只计算了实际采样出来的 token** a_t 的 log-ratio，而不是对完整词表分布求和的精确 reverse-KL。这里的 C_t 更适合理解为一个单样本的 cost / penalty 信号

（不是 reward）。它在单样本上可正可负，但其在 student 分布下的期望为 reverse-KL，因此优化目标并不是让每一步的 C_t 都尽可能小，而是让其期望尽可能趋近于0（即student和teacher分布完全一致）。

sg(·) 表示 stop-gradient： C_t 在反向传播时被当作常数，不对 θ 产生梯度。

另外一个需要注意的是，PG-style由于涉及到求期望（解决不可导的问题），而在实际工程中，我们一般通过Monte Carlo 采样去近似期望。具体来说，需要通过采样 N 条轨迹做蒙特卡洛近似（这和所有基于policy gradient的RL方法是一致的）：

$$L_{\text{PG-OPD}}(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T^{(i)}} \text{sg}(C_t^{(i)}) \cdot \log \pi_\theta(a_t^{(i)} \mid s_t^{(i)}), \qquad \tau^{(i)} \sim \pi_\theta(\cdot \mid x)$$

梯度

下面我们依然从梯度去分析model的更新方式。为了方便书写，下面都只考虑采样一次的情形，即$N = 1$ （严格来说，即使在单次采样情形下，采样轨迹 $\tau \sim \pi_\theta$ 本身也依赖于参数 θ，所以完整求导会包含对轨迹的求导，这里我们暂时不考虑轨迹的变化，这也是RL领域常见做法，感兴趣的朋友可以了解REINFORCE风格的surrogate）。

对Loss求导（C_t 被 stop-gradient，视为常数），得到 PG-style OPD 的梯度：

$$\boxed{\nabla_\theta L_{\text{PG-OPD}}(\theta) = \sum_t \text{sg}(C_t) \cdot \nabla_\theta \log \pi_\theta(a_t \mid s_t)}$$

注意和上一节 GKD-style 梯度的关键区别：

- GKD-style 的梯度是 $\sum_t \sum_v (\cdots) \nabla_\theta \pi_\theta(a_v \mid s_t)$ ——对**词表中所有 token** v 求和；
- PG-style 的梯度是 $\sum_t (\cdots) \nabla_\theta \log \pi_\theta(a_t \mid s_t)$ ——只涉及**实际采样出的那一个 token** a_t。

这意味着：PG-style 的梯度信息来源只在被采样到的token上（当然也会间接影响其他token的概率），而 GKD-style 的梯度针对所有token产生，后者的信息量天然更大。

和上一节一样，为了直观分析"某个 token 的 logit 会被推高还是压低"，我们仍然把梯度推到 logit 层面。

2）从 logit 角度理解 PG-style 的更新

设 student 在位置 t 的 logits 为 $z = (z_1, \ldots, z_{|\mathcal{V}|})$， softmax 概率为 $p_v = \pi_\theta(a_v \mid s_t)$，teacher 概率简记为 $q_v = \pi_{teacher}(a_v \mid s_t)$。

PG-style 的 loss 在单步上为

$$\ell_t^{\mathrm{PG}} = \mathrm{sg}(C_t) \cdot \log p_k,$$

其中 k 是实际被采样出的那个 token 的索引（即 $a_t = a_k$），$C_t = \log p_k - \log q_k$。

利用上一节推导过的 softmax 求导恒等式

$$\frac{\partial \log p_k}{\partial z_j} = \delta_{kj} - p_j,$$

可以得到：

$$\frac{\partial \ell_t^{\mathrm{PG}}}{\partial z_j} = \mathrm{sg}(C_t) \cdot (\delta_{kj} - p_j).$$

这个式子展开来看就很直观了：

- 对当前**被采样的 token** a_k（即 $j = k$）：

$$\frac{\partial \ell_t^{\mathrm{PG}}}{\partial z_k} = \mathrm{sg}(C_t) \cdot (1 - p_k);$$

- 对当前**未被采样的 token** a_j（即 $j \neq k$）：

$$\frac{\partial \ell_t^{\mathrm{PG}}}{\partial z_j} = \mathrm{sg}(C_t) \cdot (0 - p_j) = -\,\mathrm{sg}(C_t) \cdot p_j.$$

注意p_k和p_j都是大于0小于1的，那么可以看出，$\frac{\partial \ell_t^{\mathrm{PG}}}{\partial z_k}$和 $\frac{\partial \ell_t^{\mathrm{PG}}}{\partial z_j}$的正负始终是相反的！且梯度下降更新为 $z_j \leftarrow z_j - \eta \frac{\partial \ell_t^{\mathrm{PG}}}{\partial z_j}$，因此：

- 若 $C_t > 0$（即 student 概率高于 teacher，student 过度押注了这个 token）：
  - 被采样 token a_k：$\frac{\partial \ell_t^{\mathrm{PG}}}{\partial z_k} > 0$，梯度下降会**降低** $z_k \Rightarrow$ 减小该 token 的概率；
  - 词表中其他所有 token a_j：$\frac{\partial \ell_t^{\mathrm{PG}}}{\partial z_j} < 0$，梯度下降会**提高** $z_j \Rightarrow$ 无差别地抬高其他所有 token。
- 若 $C_t < 0$（即 student 概率低于 teacher，teacher 更认可这个 token）：
  - 被采样 token a_k：$\frac{\partial \ell_t^{\mathrm{PG}}}{\partial z_k} < 0$，梯度下降会**提高** $z_k \Rightarrow$ 增大该 token 的概率；
  - 词表中其他所有 token a_j：$\frac{\partial \ell_t^{\mathrm{PG}}}{\partial z_j} > 0$，梯度下降会**降低** $z_j \Rightarrow$ 无差别地压低其他所有 token。

这揭示了 PG-style 更新的一个本质特征：**它对未被采样的 token 是完全"盲目"的**。当 teacher 不认可 student 采样的 token 时，PG-style 会降低该 token 的概率，但它**无差别地把概率质量分散给词表中所有其他 token**（当然，这里说无差别可

能不太严谨，或者说不区分 teacher 偏好）——不管这些 token 是 teacher 高度认可的，还是同样不认可的。它没有能力区分"应该把概率转移给谁"。

3）具体例子

仍然使用上一节的三 token 例子：

$$p = (0.2,\ 0.3,\ 0.5), \qquad q = (0.7,\ 0.2,\ 0.1).$$

假设 student 在这个位置采样到了 t_3（概率最高的那个 token），则

$$C_t = \log p_3 - \log q_3 = \log 0.5 - \log 0.1 \approx +1.61.$$

$C_t > 0$，说明 student 在 t_3 上过度押注了（teacher 只给 t_3 分配了 0.1 的概率，student 却给了 0.5）。

各个 token 的 logit 梯度为：

- **被采样的 t_3：**
$$\frac{\partial \ell_t^{\mathrm{PG}}}{\partial z_3} = (+1.61)(1 - 0.5) = +0.81.$$
梯度下降会**降低** $z_3 \Rightarrow$ 压低 t_3 的概率。**方向正确。**

- **未被采样的 t_1：**
$$\frac{\partial \ell_t^{\mathrm{PG}}}{\partial z_1} = -(+1.61) \cdot 0.2 = -0.32.$$
梯度下降会**提高** $z_1 \Rightarrow$ 抬高 t_1 的概率。 **方向正确**（teacher 确实最支持 t_1），但这**只是巧合**—— PG 并不知道 teacher 对 t_1 的态度，它只是在无差别地抬高所有非采样 token。

- **未被采样的 t_2：**
$$\frac{\partial \ell_t^{\mathrm{PG}}}{\partial z_2} = -(+1.61) \cdot 0.3 = -0.48.$$
梯度下降会**提高** $z_2 \Rightarrow$ 抬高 t_2 的概率。这里可以发现t_2**的更新方向就错了**，因为PG在提高除了被采样token以外的所有token的概率。

4）与 GKD-style 的正面对比

现在我们把同一个例子下，GKD-style（reverse-KL）和 PG-style 的 logit 梯度放在一起比较。

回顾上一节的 GKD-style 结果：

$$\frac{\partial \ell_t^{\mathrm{rKL}}}{\partial z_j} = p_j \left(\log \frac{p_j}{q_j} - \mathrm{KL}(p\|q) \right), \qquad \mathrm{KL}(p\|q) \approx 0.68.$$

整理为表格（红色表示与teacher对齐方向相反）：

Token	p_j student概率	q_j teacher概率	GKD logit 梯度	GKD 方向	PG logit 梯度	PG 方向
t_1	0.2	0.7	$0.2 \times (-1.93) = -0.39$	↑ 提升	-0.32	↑ 提升
t_2	0.3	0.2	$0.3 \times (-0.27) = -0.08$	↑ 微弱提升	-0.48	↑ 较强提升
t_3（采样）	0.5	0.1	$0.5 \times (0.93) = +0.47$	↓ 压低	$+0.81$	↓ 压低

从这张表可以读出两种方法的一些差异：

1. t_1（**teacher 最支持的 token**）：两者都抬高。
2. t_2（**teacher 不太在意的 token**）： GKD-style 几乎不动它（梯度仅 -0.08），因为它知道 teacher 对 t_2 的评价只是中性的； PG-style 却给了较大的抬升力度（-0.48），因为它不知道 teacher 对 t_2 的态度，只是在盲目地把从 t_3 拿走的概率然后分配给所有非采样 token。当然，如果是SeqKD式的forward-KL，那么t_2概率应该下降以对齐teacher。
3. t_3（**student 错押的主峰**）：两者都压低。

总结来看：

- **GKD-style** 像一位知道答案的老师：它清楚词表中每个 token 应该得到多少概率，因此能**精准地**告诉 student “把概率从哪里拿走、分配给谁”。
- **PG-style** 像一位只能打分的考官：它只能告诉 student “你刚才采样的这个 token 好不好”，至于概率应该转移给谁，student 只能**盲目地**重新分配。

这也解释了为什么在实践中，PG-style 通常方差更高、收敛更慢（也包括RL）：每一步更新只利用了 teacher 在**一个 token** 上的信息，而 GKD-style 同时利用了 teacher 在**整个词表** 上的信息。

5）PG-style 的采样依赖性

上面的分析还揭示了 PG-style 的另一个重要特性： **它的更新结果高度依赖于实际采样到了哪个 token**。

仍然用同一个例子，如果 student 这次碰巧采样（假设非贪婪采样）到的不是 t_3，而是 t_1，则

$$C_t = \log p_1 - \log q_1 = \log 0.2 - \log 0.7 \approx -1.25.$$

$C_t < 0$，teacher 非常认可这个选择。此时的 logit 梯度为：

Token	p_j	q_j	logit 梯度	更新方向	力度
t_1 (采样)	0.2	0.7	$(-1.25)(1-0.2) = -1.00$	↑ 提升	强
t_2	0.3	0.2	$-(-1.25)(0.3) = +0.38$	↓ 压低	中
t_3	0.5	0.1	$-(-1.25)(0.5) = +0.63$	↓ 压低	强

有趣的是，这次 t_2 会被**压低**（而采样 t_3 时它会被抬高）。至于 t_2 到底应该被抬高还是压低？PG-style **完全无法给出一致的答案**—— 它的判断完全取决于"这次碰巧采样到了谁"。

而 GKD-style 则完全不受采样影响：不管 student 采样到了哪个 token， GKD-style 在这个前缀上的 logit 梯度永远是确定的（因为它依赖的是完整的 p 和 q 分布，而不是某个 sampled token）。

这就是 PG-style 方差高的**根本来源**：同一个前缀、同一对分布，仅仅因为采样到不同的 token，梯度更新的方向就可以截然不同。而 GKD-style 在相同条件下的梯度是**确定性的**。

当然，这不是说PG-style优化方向是错的。虽然单样本梯度高度随机，但对采样分布取期望后，它所优化的目标仍然是有确定方向的，只是问题在于单样本估计噪声大。

6）小结

	GKD-style	**PG-style**
梯度涉及的 token	词表中所有 token	仅采样到的 1 个 token
teacher 信息利用	完整词表分布	仅 sampled token 的 logprob
对未采样 token 的处理	精准分配（知道每个 token 该得多少）	盲目均分
梯度确定性	确定的（给定 p, q）	随采样结果波动
方差	低	高
更像什么	监督学习（SFT/KD）	强化学习（REINFORCE）

从这个对比可以看出，**若只看蒸馏信号本身，GKD-style 提供了比 PG-style 更细粒度、更低方差的监督：** PG-style 能做到的（压低错误 token、抬高正确 token），GKD-style 都能做到，而且 GKD-style 还能精确控制概率应该转移给谁——这是 PG-style 做不到的。

那 PG-style 还有存在的价值吗？有。它的独特优势不在"信息利用效率"，而在于：

- **与 RLVR 的无缝融合**：PG-style 的 C_t 接口可以直接叠加传统 RL 的标量 reward（如 outcome correctness），这是 GKD-style 难以自然做到的；
- **对采样策略本身的优化**：PG-style相对来说对teacher的依赖较弱，一方面的缺点当然是学习起来可能比较慢，毕竟指导不足；但这也可以说是它的独特之处：充分发挥student的主观能动性，让他自己充分探索，我们只关注最终的结果，不关注过程中可能出现的错误或不合理的地方。从这一点来说，PG-style天然更有利于 student突破teacher的能力——而GKD-style说到底还是让student尽可能和teacher保持一致，从而上限也被teacher锁死了。（当然个人认为与纯RLVR相比，这两种模式下student的更新都是受限的，这也是蒸馏的一个长久命题吧——如何让student达到甚至超越teacher，也已经有不少工作了。）

重温 On-Policy Distillation（5）

标准OPD的KL散度

Penghui Yang

2026.4.24

前言

前几节我们对常见OPD的两种实现：GKD-style和PG-style，从梯度的视角分析了模型参数的更新过程。当然二者并不是相互割裂的，也有一些工作是在考虑二者的融合。后面打算分享一下社区对标准OPD的一些改进工作，例如与RL的结合、KL的实现选择、算法原理上的推陈出新等等。

这一节我们再一次系统看看OPD中的reverse-KL，前几节把GKD-style和PG-style分开来看了梯度更新过程，这里再以统一的视角看看二者的联系，顺便对之前在xhs上留的一个关于full-vocab KL问题做讨论。

0）预备知识：概率分布差异与KL散度

在进入OPD里的KL之前，我们先简单回顾一个更基础的问题：如果有两个概率分布，我们该如何衡量它们到底“差多少”？

最朴素的方式当然是直接比较每个位置上的概率差，例如L_1距离、L_2距离等。但在很多机器学习问题中，我们更关心的是两个分布在**相对概率结构**上的差异，而不只是逐点相减的绝对误差。因此，大家更常使用一些专门用于比较分布的量，比如KL散度、Jensen-Shannon散度、Wasserstein距离等。

其中最常见的一个就是**KL散度**（Kullback-Leibler divergence）。对于两个离散分布p和q，其定义为：

$$\mathrm{KL}(p\|q)=\sum_v p_v\log\frac{p_v}{q_v}=\mathbb{E}_{a\sim p}\left[\log p(a)-\log q(a)\right].$$

直观上，它衡量的是：如果真实分布是p，但我们用q来描述它，那么会产生多大的“分布失配”。

KL散度有两个最重要的性质：

- **非负性：**

$$\mathrm{KL}(p\|q) \geq 0,$$

且当且仅当$p = q$时取0；

- **不对称性：**

$$\mathrm{KL}(p\|q) \neq \mathrm{KL}(q\|p).$$

第二点尤其重要，因为这意味着：虽然都叫KL，但两个方向对应的优化偏好通常是不一样的。

在OPD中，我们最常见的是

$$\mathrm{KL}(\pi_\theta\|\pi_T),$$

也就是让student在**自己的采样分布**下尽量和teacher保持一致。这也是为什么reverse-KL会天然适合on-policy distillation。

有了这个基础，下面我们再来看OPD里reverse-KL到底是如何被实现和优化的。

1）OPD中的KL

现在大家可能有一个common-sense——标准OPD的目标函数就是去最小化 student与teacher之间的reverse-KL：

$$\mathrm{KL}(\pi_\theta\|\pi_T) = \mathbb{E}_{a\sim\pi_\theta(\cdot|s)}\left[\log \pi_\theta(a|s) - \log \pi_T(a|s)\right].$$

之所以它天然适合on-policy distillation，是因为这个期望本身就在student自己的分布下取。也就是说，student不是去模仿teacher真实采样出的轨迹，而是在**自己会采到的轨迹上**，要求自己的行为尽量和teacher一致。这和on-policy RL的精神是非常一致的。

不过，再让我们审视“OPD优化的是reverse-KL”，真正落到算法实现时，这里面其实至少包含两个不同层面：

1. **目标层面**：理论上想最小化的量是不是reverse-KL？
2. **梯度层面**：我们到底是怎样对这个目标求梯度并实现训练的？

这两件事并不等价。很多论文在叙述时会把它们混在一起说，容易造成一种错觉：似乎“只要目标是reverse-KL，那么不同算法只是它的不同工程近似”。但前几节我们在讨论GKD-style和PG-style时其实已经看到，事情并没有这么简单。

设当前位置t的student分布与teacher分布分别为

$$p_v = \pi_\theta(a_v|s_t), \qquad q_v = \pi_T(a_v|s_t),$$

则单步reverse-KL为：

$$\ell_t^{\mathrm{rKL}}(p,q) = \sum_v p_v \log \frac{p_v}{q_v}.$$

这个量至少可以有两种完全不同的实现方式：

- **GKD-style**：直接保留完整词表分布，对上式精确求导；
- **PG-style**：利用

  $$\mathbb{E}_{a\sim p}\left[\log p(a) - \log q(a)\right] = \mathrm{KL}(p\|q)$$

  的单样本形式，在sampled token上构造REINFORCE风格的梯度估计器（我们在后文给出梯度证明）。

因此，**“优化reverse-KL”并不唯一对应某一种训练算法**；更准确地说，reverse-KL更像是一个**理论目标**，而GKD-style和PG-style是两种不同的**梯度实现方式**。

这一点非常重要，因为后面很多所谓“改进OPD”的工作，表面上是在“改KL”，但本质上可能是在做以下改动：

- 改**目标函数（不同的散度）**：比如不用reverse-KL，而换成forward-KL、JSD、α-divergence等；
- 改**梯度估计器**：比如仍然优化reverse-KL，但从sample-level estimator换成full-vocab exact gradient，或者加入baseline降方差；
- 改**训练框架**：比如把OPD和RLVR的reward信号统一起来。

关于reverse-KL和forward-KL的比较，我们已经在第3节GKD-style的更新模式中，通过logit视角做过说明了。核心区别就是：reverse-KL重点关注当前student认为概率最大的方向是否符合teacher， forward-KL则强制让student在所有方向都和teacher对齐。当然，在理想的分布空间中，二者的全局最优解都是student=teacher（即对应KL为0）；真正的差异主要体现在优化路径、局部更新偏好以及有限样本训练行为上。

2）PG-style与GKD-style：同一个rKL，不同的梯度实现

虽然我们说两种OPD都在“优化reverse-KL”，但前面已经讨论过： **它们的根本区别不在目标名字，而在梯度来源。**

2.1 GKD-style：full-vocab exact gradient

GKD-style直接对

$$\ell_t^{\mathrm{rKL}} = \sum_v p_v \log \frac{p_v}{q_v}$$

求导，得到logit梯度

$$\frac{\partial \ell_t^{\mathrm{rKL}}}{\partial z_j} = p_j \left(\log \frac{p_j}{q_j} - \mathrm{KL}(p\|q) \right).$$

这个梯度的特点是：

- 对词表中**所有token**都有显式更新；
- 每个token的更新都知道teacher对它的态度（通过q_j体现）；
- 给定同一对分布(p, q)，也即在某一个序列前缀下，梯度是**确定性的**。

所以GKD-style本质上是一种精确的**distribution-level distillation**。

这里有一个非常值得注意的结构：

$$\log \frac{p_j}{q_j} - \mathrm{KL}(p\|q).$$

由于

$$\mathrm{KL}(p\|q) = \mathbb{E}_{v\sim p}\left[\log \frac{p_v}{q_v}\right],$$

因此这个量可以理解为：

当前token的局部log-ratio，相对于student分布下全局平均log-ratio的偏离程度。

也就是说，GKD-style的梯度天然带有一个**center / baseline**结构（从“减去全局平均量”的结构上看，它和RL里常见的baseline / center思路有些相似，例如GRPO中的group-mean reward；当然二者在具体对象和统计含义上并不相同）。

2.2 PG-style：sample-level stochastic gradient estimator

PG-style则利用

$$\mathrm{KL}(p\|q) = \mathbb{E}_{a\sim p}\left[\log \frac{p(a)}{q(a)}\right]$$

这一恒等式，把sampled token上的log-ratio当作随机梯度估计器的一部分。那么这样的做法和上面使用完整词表的做法，在梯度上有何联系呢？

实际上可以不复杂地得到：**PG-style这种做法的梯度本身是完整reverse-KL梯度的无偏估计**。证明：由

$$\mathrm{KL}(p\|q) = \sum_k p_k \log \frac{p_k}{q_k}$$

出发，对θ求导：

$$\nabla_\theta \mathrm{KL}(p\|q) = \sum_k \nabla_\theta p_k \log \frac{p_k}{q_k} + \sum_k p_k \nabla_\theta \log p_k.$$

由于

$$\sum_k p_k \nabla_\theta \log p_k = \sum_k \nabla_\theta p_k = \nabla_\theta \sum_k p_k = 0,$$

再利用

$$\nabla_\theta p_k = p_k \nabla_\theta \log p_k,$$

便得到

$$\nabla_\theta \mathrm{KL}(p\|q) = \sum_k p_k \log \frac{p_k}{q_k} \nabla_\theta \log p_k = \mathbb{E}_{k\sim p}\left[\log \frac{p_k}{q_k} \nabla_\theta \log p_k\right].$$

注意看，左边是GKD-style下使用完整rKL的梯度，右边期望的括号内是PG-style下的单样本梯度估计。

这就是说，**最原始的sampled-token log-ratio estimator本身就是reverse-KL梯度的无偏估计器**。这点非常重要，因为它说明尽管PG-style没有显式利用teacher在未采样token上的完整分布信息，但并不意味着它只是“和reverse-KL沾边”，而是在严格意义上可以对应到reverse-KL的policy gradient形式。

不过，这里也恰恰暴露了PG-style和GKD-style最本质的区别：

- **GKD-style**：直接利用**全词表**信息形成精确梯度；
- **PG-style**：只利用**单次采样token**上的log-ratio，形成随机梯度估计器。

前者是**full-vocab exact gradient**，后者是**single-sample stochastic estimator**。因此，PG-style的问题不在于它“目标不对”，而在于它每一步只使用了teacher在**一个sampled token**上的信息，从而天然带来更高方差与更强的采样依赖性。这也再次说明，两者的关键差异并不是“一个优化KL，一个不优化KL”，而是：

同一个理论目标，对应了两种完全不同的梯度实现方式。

2.3 full-vocab信息在PG-style里能起什么作用？

（这里是对xhs上留的问题的回答）上面看起来似乎会让人有一个很自然的想法：既然GKD-style利用了完整词表KL的信息，那能不能也把这个**full-vocab KL**塞进PG-style里，让PG-style既保留RL框架兼容性，又利用更多teacher信息？

一个最直接但其实**错误**的想法是：既然

$$\mathrm{KL}(p\|q)$$

是prefix级别的full-vocab量，那能不能直接把它拿来代替sampled token上的C_k，也就是令

$$A = \mathrm{KL}(p\|q),$$

然后做PG更新

$$g = \mathrm{sg}(A)\nabla_\theta \log p_k, \qquad k \sim p?$$

乍看之下，这似乎结合了两边的优点：

- advantage里用了全词表信息；
- 更新仍然保持PG-style，可与RL框架兼容。

但如果仔细算一下，会发现这样做其实**没有有效学习信号！**

原因在于，policy gradient本质上使用的是一个随机梯度估计器，我们真正关心的是它在采样分布下的**期望更新方向**。如果某个估计器的期望为0，就意味着它平均上不提供任何系统性的学习信号，只是在做零均值噪声更新。

对采样$k \sim p$取期望，有

$$\boxed{\mathbb{E}_{k\sim p}\left[\nabla_\theta \log p_k\right] = \sum_k p_k \nabla_\theta \log p_k = \sum_k \nabla_\theta p_k = \nabla_\theta \sum_k p_k = \nabla_\theta 1 = 0.}$$

而$\mathrm{KL}(p\|q)$在给定前缀时又是**不依赖sampled token的常数**，因此

$$\boxed{\mathbb{E}_{k\sim p}\left[\mathrm{KL}(p\|q)\nabla_\theta \log p_k\right] = \mathrm{KL}(p\|q)\,\mathbb{E}_{k\sim p}\left[\nabla_\theta \log p_k\right] = 0.}$$

也就是说，如果advantage与sampled action无关，它在policy gradient里就退化成了一个常数baseline，平均起来什么也学不到。

这其实和RL中的经典结论完全一致：

- policy gradient前面的权重必须和action相关，才能产生有效更新；
- 如果只是一个与action无关的常数，它的期望梯度就是0。

总而言之，在PG-style中强行把单采样估计器更换为全词表求和计算advantage，吃力不讨好（感兴趣的朋友可以实验看看效果）。从这也可以看出，社区中所说的"full-vocab KL"，一般都是针对GKD-style而言的。

这里留一个思考方向：如果不考虑计算成本，就是想把full-vocab KL以特定方式（显然上面的做法是不行的）引入PG-style，能否得到什么好处？如果有好处，在引入额外计算开销的前提下是否值得？以及同样都是full-vocab rKL，它和GKD-style是等价的吗？

3）OPD的改进方向

上面这些讨论可以帮助我们更清楚地看待社区里所谓“改KL”的工作。很多时候，“改KL”并不是单纯换一个散度名字那么简单，而可能是在以下几个层面之一上做文章。

A. 改目标函数

最直接的一类，就是把reverse-KL本身换掉，比如：

- forward-KL；
- Jensen-Shannon divergence；
- α-divergence；
- 某些对称化的KL或混合散度。

这类工作的核心在于改变优化偏好：

- reverse-KL偏mode-seeking；
- forward-KL偏mass-covering；
- 介于二者之间的散度，则希望在稳定性、覆盖性、探索性之间做折中。

B. 改梯度估计器

另一类工作表面上仍然在优化reverse-KL，但并不直接使用最原始的sampled estimator，而是去改它的梯度估计方式，例如：

- 用full-vocab exact gradient替代sample-level estimator；
- 对sample-level estimator加入baseline / normalization / importance weighting；
- 使用importance sampling修正采样偏差（training-inference差异或者重复利用rollout更新的策略不一致）；
- 对大偏移token进行截断或过滤，减少极端比值带来的不稳定性。

C. 改训练框架

还有一类工作并不执着于“纯蒸馏teacher”，而是把OPD作为RL训练中的一个额外优势信号，例如：

$$\hat{A}_t = \hat{A}_t^{\mathrm{OPD}} + \alpha \hat{A}_t^{\mathrm{ORM}}.$$

此时teacher提供的log-ratio不再是唯一信号，而是与outcome reward、process reward、rule reward等一起共同决定更新方向。这类方法的优势是：

- 能自然融合teacher知识和外部反馈；
- student不一定只会“贴着teacher学”，而可能在RL奖励驱动下超越teacher；

但它的代价也很明显：训练目标会变得更混合、更复杂，而“最终学到的是teacher知识，还是reward偏好”也会变得不那么容易分辨。

4）这一节的小结

这一节其实想澄清一个经常被一笔带过的问题：

OPD里说“优化KL”，到底是在说目标，还是在说梯度？

简单总结一下：

- **reverse-KL之所以流行**，是因为它天然和on-policy学习对齐，更关注student自己会采到的行为；
- **同样都是reverse-KL**，GKD-style和PG-style的根本区别不在名字，而在梯度来源：前者是full-distribution exact gradient，后者是sample-level stochastic gradient estimator；
- **sampled-token log-ratio本身就是reverse-KL梯度的无偏估计器**，因此PG-style在理论上并不是“另一个目标”，而是同一目标的随机梯度实现；
- **不能把full-vocab KL直接当成PG advantage**，否则由于它与sampled action无关，期望梯度为0，只会退化成baseline；
- 从数学结构上看，如果直接需要利用完整分布信息，那就更自然地会走向GKD-style的distribution-level distillation；
- 很多所谓“改KL”的工作，真正改的未必是散度本身，也可能是在改梯度估计器或训练框架。

从这个角度看，后续很多工作其实都可以放进一个统一视角里理解：

我们到底是在改“学什么”，还是在改“怎么学”？

前者对应目标函数，后者对应梯度估计与训练框架。

重温 On-Policy Distillation（6）

Policy Gradient的优化目标: 当前位置需要对未来负责吗？

Penghui Yang

2026.4.29

前言

前面我们在分析GKD-style和PG-style两种OPD时，其实隐含了一个思想：我们想要在每个给定前缀下，使得student在next-token-prediction时的行为尽可能与teacher对齐。具体体现在，我们考虑每个前缀下student和teacher生成下一token的概率差异（GKD关注全词表，PG关注sampled-token），然后我们把这些不同前缀的差异求和（即$\sum_{t=1}^{T}$），用于表征在整个序列下student和teacher的差异。

这看起来十分自然，但如果直接从【对齐**完整序列的分布**】出发，实际上会得到一个略有差异的求解形式，核心其实就是 sequence差异是否直接等价于token差异求和？这里主要针对PG-style而言，因为这个问题其实在经典RL中已经有所体现；至于GKD-style则是另一种思路，避开了这个问题，我们也会在后文说明。

参考资料

- 知乎专栏· 重探 On-Policy Distillation（OPD）：三类典型失败以及修复路径
- Revisiting On-Policy Distillation: Empirical Failure Modes and Simple Fixes
- FIPO: Eliciting Deep Reasoning with Future-KL Influenced Policy Optimization

0）预备知识：“完整序列”的概率分布

我们平时最熟悉的是 LLM 在每一步输出一个 next-token 概率分布（这里仅考虑next-token-prediction，不考虑multi-token-preditcion）：

$$\pi_\theta(a_t|x, a_{<t}).$$

但这并不意味着模型只定义了局部条件分布，而没有定义整条序列的概率。事实上，自回归模型天然就在整个序列空间上定义了一个联合分布：

$$\pi_\theta(\tau|x) = \prod_{t=1}^{T} \pi_\theta(a_t|x, a_{<t}), \qquad \tau = (a_1, \ldots, a_T).$$

也正因为如此，我们完全可以在**序列空间**上定义 reverse-KL：

$$\mathrm{KL}(\pi_\theta(\cdot|x)\|\pi_T(\cdot|x)) = \mathbb{E}_{\tau\sim\pi_\theta(\cdot|x)}\left[\log \pi_\theta(\tau|x) - \log \pi_T(\tau|x)\right].$$

这里τ即为在给定prompt下LLM生成的完整轨迹序列。这里从直观意义上来讲，我们想要让student学习的其实是在给定一个prompt时， teacher完整的回复应该长什么样，而不是仅仅关注某一个token下student应该去如何学习teacher。

而常见的 token-level 写法，其实都只是这个 sequence-level 目标在自回归结构下的近似。

1）sequence-level reverse-KL

对一个给定 prompt x，student 和 teacher 在完整序列空间上的 reverse-KL 为：

$$\mathrm{KL}(\pi_\theta(\cdot|x)\|\pi_T(\cdot|x)) = \mathbb{E}_{\tau\sim\pi_\theta(\cdot|x)}\left[\log \pi_\theta(\tau|x) - \log \pi_T(\tau|x)\right].$$

将序列概率做自回归分解：

$$\log \pi_\theta(\tau|x) = \sum_{t=1}^{T} \log \pi_\theta(a_t|s_t), \qquad \log \pi_T(\tau|x) = \sum_{t=1}^{T} \log \pi_T(a_t|s_t),$$

其中

$$s_t = (x, a_{<t}).$$

于是可得：

$$\log \pi_\theta(\tau|x) - \log \pi_T(\tau|x) = \sum_{t=1}^{T} \left(\log \pi_\theta(a_t|s_t) - \log \pi_T(a_t|s_t)\right).$$

现在我们把右边括号里定义为每一步的即时 KL 信号（注意这里都是只针对sampled-token，不是full-vocabulary）：

$$r_t^{\mathrm{KL}} = \log \pi_\theta(a_t|s_t) - \log \pi_T(a_t|s_t).$$

那么 sequence-level reverse-KL 就可以写成：

$$\mathrm{KL}(\pi_\theta(\cdot|x)\|\pi_T(\cdot|x)) = \mathbb{E}_{\tau\sim\pi_\theta(\cdot|x)}\left[\sum_{t=1}^{T} r_t^{\mathrm{KL}}\right].$$

从这个角度看，（PG-style）OPD 可以被理解为一个特殊的 policy optimization 问题：每一步的“reward”不是环境直接给出的分数（与RLVR形成对比），而是 student 与 teacher 在该位置上的 log-ratio。也正是从这里开始，reward-to-go （考虑未来完整收益）会自然出现。

2）sequence-level 下的 reward-to-go

现在从上面的 sequence-level 目标出发，用最标准的 REINFORCE 形式去求梯度（也就是PG-style OPD的来源）。设

$$J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\cdot|x)}\left[\sum_{t=1}^{T} r_t^{\mathrm{KL}}\right].$$

其中

$$r_t^{\mathrm{KL}} = \log \pi_\theta(a_t|s_t) - \log \pi_T(a_t|s_t), \qquad s_t = (x, a_{<t}).$$

为了简化书写，记整条序列上的总回报为

$$R(\tau) = \sum_{t=1}^{T} r_t^{\mathrm{KL}}.$$

则

$$J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\cdot|x)}[R(\tau)] = \sum_{\tau} \pi_\theta(\tau|x) R(\tau).$$

对 θ 求导，注意同时考虑两部分：

$$\nabla_\theta J(\theta) = \sum_{\tau} \nabla_\theta \pi_\theta(\tau|x)\, R(\tau) + \sum_{\tau} \pi_\theta(\tau|x)\, \nabla_\theta R(\tau).$$

第一项是由采样分布 $\pi_\theta(\tau|x)$ 带来的 score-function 项；第二项则来自 $R(\tau)$ 对 θ 的显式依赖。

对于第一项，利用 log-derivative trick

$$\nabla_\theta \pi_\theta(\tau|x) = \pi_\theta(\tau|x) \nabla_\theta \log \pi_\theta(\tau|x),$$

可得

$$\sum_{\tau} \nabla_\theta \pi_\theta(\tau|x)\, R(\tau) = \sum_{\tau} \pi_\theta(\tau|x) R(\tau) \nabla_\theta \log \pi_\theta(\tau|x) = \mathbb{E}_{\tau\sim\pi_\theta(\cdot|x)} \left[R(\tau) \nabla_\theta \log \pi_\theta(\tau|x) \right].$$

再看第二项。由于

$$R(\tau) = \sum_{t=1}^{T} \left(\log \pi_\theta(a_t|s_t) - \log \pi_T(a_t|s_t) \right),$$

而 teacher 分布 π_T 不依赖于 θ，因此

$$\nabla_\theta R(\tau) = \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) = \nabla_\theta \log \pi_\theta(\tau|x).$$

于是

$$\sum_{\tau} \pi_\theta(\tau|x)\, \nabla_\theta R(\tau) = \sum_{\tau} \pi_\theta(\tau|x)\, \nabla_\theta \log \pi_\theta(\tau|x) = \sum_{\tau} \nabla_\theta \pi_\theta(\tau|x) = \nabla_\theta \sum_{\tau} \pi_\theta(\tau|x) = \nabla_\theta 1 = 0.$$

也就是说，虽然 $R(\tau)$ 本身显含 θ，但对这个特殊的 sequence-level reverse-KL 目标而言，它带来的显式导数项在 student 分布下的**理论期望**恰好为 0。需要注意的是，这里为 0 的是对完整轨迹分布取期望后的精确恒等式；在实际训练中，若只用有限条采样轨迹做 Monte Carlo 近似，对应的样本均值一般不会恰好为 0，但其期望仍然为 0。

而我们上一节在分析单步rKL时也有一个梯度为0的项（见上一节2.2），那个地方是由于全词表概率求和为1，任意条件下都成立。

因此最终仍有

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\cdot|x)} \left[R(\tau) \nabla_\theta \log \pi_\theta(\tau|x) \right].$$

代回

$$R(\tau) = \sum_{u=1}^{T} r_u^{\mathrm{KL}},$$

得到

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\cdot|x)} \left[\left(\sum_{u=1}^{T} r_u^{\mathrm{KL}} \right) \nabla_\theta \log \pi_\theta(\tau|x) \right].$$

接下来利用自回归分解

$$\log \pi_\theta(\tau|x) = \sum_{t=1}^{T} \log \pi_\theta(a_t|s_t),$$

于是

$$\nabla_\theta \log \pi_\theta(\tau|x) = \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t).$$

代入上式可得

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\cdot|x)} \left[\left(\sum_{u=1}^{T} r_u^{\mathrm{KL}} \right) \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \right) \right].$$

交换求和顺序：

$$= \mathbb{E}_{\tau\sim\pi_\theta(\cdot|x)} \left[\sum_{t=1}^{T} \left(\sum_{u=1}^{T} r_u^{\mathrm{KL}} \right) \nabla_\theta \log \pi_\theta(a_t|s_t) \right].$$

到这里，每个位置 t 前面的权重（优势）还是整条序列（包括这个位置之前）的总回报

$$\sum_{u=1}^{T} r_u^{\mathrm{KL}}.$$

但实际上，对于任意固定的 t，过去的回报项

$$\sum_{u=1}^{t-1} r_u^{\mathrm{KL}}$$

只依赖于当前动作 a_t 之前的前缀，因此在给定 s_t 后，它与当前 action a_t 无关，于是有

$$\mathbb{E}_{a_t\sim\pi_\theta(\cdot|s_t)} \left[\left(\sum_{u=1}^{t-1} r_u^{\mathrm{KL}} \right) \nabla_\theta \log \pi_\theta(a_t|s_t) \right] = \left(\sum_{u=1}^{t-1} r_u^{\mathrm{KL}} \right) \mathbb{E}_{a_t\sim\pi_\theta(\cdot|s_t)} \left[\nabla_\theta \log \pi_\theta(a_t|s_t) \right] = 0.$$

这里最后一步用到了一个基本恒等式

$$\mathbb{E}_{a_t\sim\pi_\theta(\cdot|s_t)} \left[\nabla_\theta \log \pi_\theta(a_t|s_t) \right] = \sum_{a_t} \pi_\theta(a_t|s_t) \nabla_\theta \log \pi_\theta(a_t|s_t) = \sum_{a_t} \nabla_\theta \pi_\theta(a_t|s_t) = \nabla_\theta 1 = 0.$$

因此，所有与当前位置动作无关的过去回报项，在期望梯度中都为0，可以略去，只剩下从当前位置开始的未来项，也就是 reward-to-go：

$$G_t = \sum_{u=t}^{T} r_u^{\mathrm{KL}}.$$

也就是说梯度可以改写为

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

这就是这一节最关键的式子。它告诉我们：如果严格从 sequence-level reverse-KL 的 PG 形式出发，那么第 t 步动作的更新权重应该是

$$G_t = \sum_{u=t}^{T} r_u^{\mathrm{KL}},$$

也就是从当前位置开始，未来所有 KL 项的和。

直观上其实很好理解：

- 第 t 步的动作不仅决定当前 token 本身，它还会影响后续前缀；
- 后续前缀又会影响未来 token 的 teacher-student 对齐。

因此，在严格的 sequence-level policy gradient 视角下，当前位置动作理应为未来的 KL 后果负责。

3）PG-style的token-level近似

然而在实践中，大家最常见的 PG-style OPD 写法并不是使用

$$G_t = \sum_{u=t}^{T} r_u^{\mathrm{KL}},$$

而是直接把当前位置的即时项

$$r_t^{\mathrm{KL}} = \log \pi_\theta(a_t|s_t) - \log \pi_T(a_t|s_t)$$

当作该位置的 advantage / cost 来做更新：

$$g_t \approx r_t^{\mathrm{KL}} \nabla_\theta \log \pi_\theta(a_t|s_t).$$

也就是说，常见的 token-level PG-style 实际做的是下面这个近似：

$$\boxed{G_t = \sum_{u=t}^{T} r_u^{\mathrm{KL}} \quad \Longrightarrow \quad G_t \approx r_t^{\mathrm{KL}}.}$$

换句话说，它把**未来 KL 的耦合项全部丢掉了**，只保留当前位置的即时 teacher-student 偏差。这也是为什么前面几节分析 PG-style 时，我们默认每个位置的训练信号就是 sampled-token 的 log-ratio：那对应的是一种 **token-level / immediate-reward** 的 PG 近似，而不是严格的 sequence-level reward-to-go 形式。

注意区分：rKL的两种近似

严格的 sequence-level PG 形式使用的是 reward-to-go：

$$G_t = \sum_{u=t}^{T} r_u^{\mathrm{KL}},$$

也就是说，第 t 步动作的更新不仅取决于当前位置的 teacher-student 差异，还取决于它对未来所有 KL 项的影响。而常见的 token-level PG-style 则进一步把它近似为只使用当前项

$$G_t \approx r_t^{\mathrm{KL}}.$$

因此，它并不是完整 sequence-level reverse-KL policy gradient 的无偏估计，而是通过丢弃未来 KL 项对当前动作的 credit assignment，得到的一个更局部、更低方差的近似更新。

这里需要特别区分两种“无偏/有偏”：

- 前面我们说 PG-style 是 GKD-style 梯度的无偏估计，指的是在**固定前缀** s_t 下，对局部 reverse-KL

  $$\mathrm{KL}(\pi_\theta(\cdot|s_t)\|\pi_T(\cdot|s_t))$$

  而言，sampled-token estimator

  $$\log \frac{p_k}{q_k} \nabla_\theta \log p_k, \qquad k \sim p$$

  对该**局部 token-level KL 梯度**是无偏的；

- 而这里说 token-level PG-style 相对 sequence-level 目标是有偏的，指的是它并不是完整 sequence-level reverse-KL policy gradient 的无偏估计，因为它把严格应当使用的 reward-to-go

  $$G_t = \sum_{u=t}^{T} r_u^{\mathrm{KL}}$$

  近似成了当前即时项 r_t^{KL}，从而忽略了未来项。

换句话说，前一种“无偏”说的是：

在一个固定前缀上，如何估计当前这一步局部 KL 的梯度；

而后一种“有偏”说的是：

如果把整个序列级目标放到 PG 视角下，当前位置是否还承担了未来 KL 项的责任。

因此，这两者讨论的并不是同一个层面的“偏差”。

为什么使用immediate reward？

既然这个近似是有偏的，那我们为什么还要广泛使用它？这里涉及到方差的问题。如果使用 reward-to-go：

$$G_t = \sum_{u=t}^{T} r_u^{\mathrm{KL}},$$

那么：

- 每个位置的梯度都要背负未来很多随机项，序列越长，future coupling 越强；
- 在长推理、多步 agent interaction 等场景下，梯度波动往往会非常大。

而如果只使用即时项 r_t^{KL}：

- 每一步只看当前位置的局部 teacher-student 差异；
- 不把未来随机性显式回传到前面；
- 梯度通常更短视，但也更稳定。

所以它本质上是在做这样一个 trade-off：

token-level PG-style 用偏差换低方差。

4）RL中的 reward-to-go

我们在第二讲介绍 PG-style OPD 时，本身就是通过 RLVR 的视角，直接把 student-teacher 的概率分布差异作为 reward 去展开并引入 Policy Gradient 的。前面说 PG-OPD 更严格的方式是使用 reward-to-go，那对标准 RL 来说其实也是一样：只要采用的是 REINFORCE 形式的 policy gradient，理论上都会遇到“当前动作到底应该承担多少未来责任”的问题。区别只是：在 OPD 中，这个 reward 来自 teacher-student 的分布差异；而在 RL / RLVR 中，这个 reward 则来自环境、规则系统或外部打分模型。

但这里要再次提醒：存在这个问题，并不意味着实践中一定要严格使用最完整的 reward-to-go。正如前面所说，更完整的 credit assignment 往往意味着更高方差；而更局部、更即时的近似虽然有偏，却可能更稳定。因此，在不同场景下，到底采用哪一种形式，往往是一个 bias-variance trade-off，而不是简单的“理论上更严格就一定更好”。

4.1 经典 RL 中的 reward-to-go

这里简单介绍下经典的 RL 设定。其实如果对之前介绍的PG-style OPD体会比较深的话，理解RL是顺理成章的，毕竟我们一开始就是从RL的视角引出的PG-OPD。

设一条轨迹为（这里s表示状态，a表示采取的动作）

$$\tau = (s_1, a_1, s_2, a_2, \ldots, s_T, a_T),$$

每一步环境给出即时 reward r_t，则整条轨迹的总回报写作

$$R(\tau) = \sum_{t=1}^{T} r_t.$$

对应的 policy gradient 目标为

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

使用 REINFORCE 求导可得

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[R(\tau)\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)\right].$$

进一步同样利用因果性（当前step之前的reward梯度为0），可以把总回报 $R(\tau)$ 改写成当前位置开始的 reward-to-go：

$$G_t = \sum_{u=t}^{T} r_u.$$

于是有

$$\boxed{\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=1}^{T} G_t \nabla_\theta \log \pi_\theta(a_t|s_t)\right].}$$

这个式子就是经典 RL 里 reward-to-go 的标准形式。它表达的含义很直接：

- 第 t 步动作的更新，不只取决于当前 reward；
- 它还要为该动作之后带来的所有未来回报负责。

所以，reward-to-go 本质上是一种**credit assignment**机制：它回答的是“当前动作到底应该分到多少未来结果”。

4.2 常见变体

虽然 reward-to-go 是最标准、最严格的 PG 形式，但实践中很少直接使用最裸的

$$G_t \nabla_\theta \log \pi_\theta(a_t|s_t)$$

做训练，原因也很熟悉：它的方差往往太大。

因此，RL 社区通常会进一步做很多处理，例如：

- 减 baseline，得到 advantage：

$$A_t = G_t - V(s_t);$$

- 使用 GAE 等更平滑的 return 估计；
- 使用 bootstrapping，把未来回报折叠进 value function；
- 在 actor-critic 框架下，把“未来责任”通过 critic 来近似。

这些方法本质上都没有否定 reward-to-go 的理论地位，而是在回答另一个问题：

“如何在保留合理 *credit assignment* 的同时，把方差压低到可训练？”

4.3 RLVR 场景下的“即时 reward”

到了 LLM 后训练，尤其是 RLVR / outcome-based RL 场景， reward 往往和经典 RL 里的 dense environment reward 不一样。很多时候，模型只在整条回复完成后，才拿到一个最终分数，例如：

- 答案是否正确；
- 单元测试是否通过；
- 代码执行结果是否成功；
- ORM / RM 给出的整体质量分数。

也就是说，原始环境信号更像是一个**sequence-level terminal reward**：

$$R(\tau) \in \mathbb{R}.$$

但由于 $R(\tau)$ 对所有位置都相同，它可以视为一种最极端的 reward-to-go：整条轨迹上的每个动作都背同一个最终结果。具体可以看5.2中的讨论。

当然，RLVR 也有一些“密集化”处理，例如：

- 使用 process reward model，对中间步骤单独打分；
- 利用规则或 verifier，把最终奖励拆成更细粒度的 token-level / step-level 信号；

但是相对来说，直接把outcome作为reward还是比较主流的，毕竟实现比较简单，以GRPO为代表的算法也验证了在LLM领域中的有效性。

4.4 PG-style下的统一视角

从更统一的角度看，无论是 RL、RLVR，还是 PG-style OPD，一旦采用的是

$$(\text{某个权重}) \times \nabla_\theta \log \pi_\theta(a_t|s_t)$$

这样的 score-function / REINFORCE 更新形式，核心问题其实都是一样的：

这个权重到底应该只看当前，还是应该包含未来？

- 如果只看当前，通常更稳定，但可能有偏；
- 如果包含未来，通常更完整，但方差更高。

所以 reward-to-go 并不是 OPD 特有的问题，而是所有 policy-gradient 风格方法都会面对的一个基本选择题。

5）reward-to-go 的相关工作

前面提到，在不同场景下到底采用哪一种形式，往往是一个 bias-variance trade-off，而不是简单的“理论上更严格就一定更好”。近期刚好有两篇工作分别探讨了OPD和RL中的相关问题，这里顺便分享一下。当然，这两篇工作内容都很丰富，这里简单讲讲其中关于reward-to-go的insight，不做过多展开。

5.1 PG-style OPD中的reward-to-go

前面在RL中提到，可以使用 GAE 等更平滑的 return 估计，这里不展开它的完整形式，只简单说明其中最核心的思想：**通过衰减因子控制未来 reward 的权重。**

最原始的 reward-to-go 写作

$$G_t = \sum_{u=t}^{T} r_u,$$

也就是当前位置之后的所有 reward 都等权累加。这种形式最完整，但也最容易带来高方差：越远处的 reward 波动，也会被同样地传回当前位置。

一个很自然的想法就是：让越远的未来，权重越小。例如引入折扣因子 $\gamma \in [0, 1]$，定义折扣回报：

$$G_t^{(\gamma)} = \sum_{u=t}^{T} \gamma^{u-t} r_u.$$

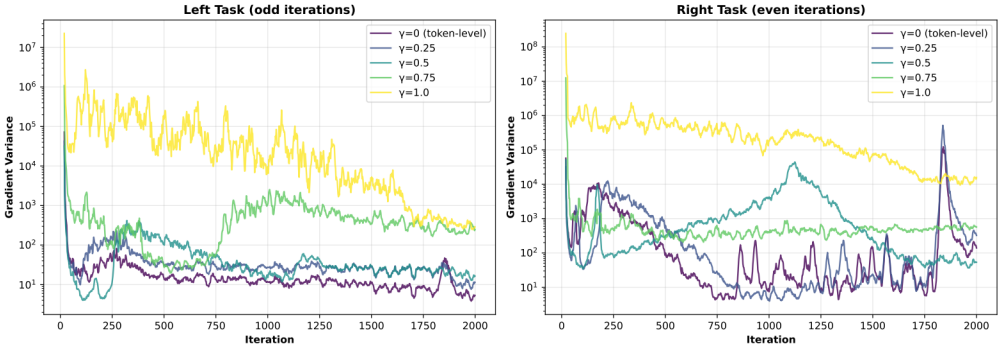
这样一来：

- 当 $\gamma = 1$ 时，就退化回原始的 reward-to-go；

- 当 $\gamma < 1$ 时，越远的 future reward 权重越小；
- 当 $\gamma = 0$ 时，就退化回原始的 immediate reward。

因此，引入 γ 的本质就是：**不是简单地在“只看当前”与“看全部未来”之间二选一，而是允许对未来信息进行连续加权。**这个思想当然也可以直接放到PG-style OPD中来。

Revisiting On-Policy Distillation: Empirical Failure Modes and Simple Fixes 这篇文章以一个很简单的MLP模型为例，通过一个toy experiment（控制物体移动）展示了在OPD中γ 和方差的关系。这个实验并不是直接在LLM上做的（当然OPD也不仅限于language model，这种基于Neural Network的模型都可以去蒸），不过用于直观展示“引入未来奖励会导致梯度方差显著上升”足够了。



具体来说，作者先训练了2个teacher model，分别用于完成两类task(上图中的左右子图)，然后分别用这2个teacher交替去蒸馏同一个student，使其具备完成这两个task的能力。

上图的横坐标可以理解为训练的step，纵坐标表示的是梯度的方差。可以明显发现，随着γ增大，也就是对未来reward给的权重越大，梯度的方差就越高。当然这确实只是个toy exp，主要是想给大家直观展示下加上未来reward之后，梯度方差确实会变大。 paper同时还证明了，若只考虑token-level（即immediate reward），则梯度方差的上界是 $O(T^2)$ （这里T是序列长度）；若考虑完整的sequence-level（即reward-to-go），则梯度方差的上界是 $O(T^4)$。

这篇paper后面重点给出了常见的在$\gamma = 0$，即只考虑当前token的reward时，通过将sampled-token扩展到teacher top-K 去计算梯度的方式，以稳定只使用immediate-reward的效果。不过个人感觉后文其实和前面探讨的由reward-to-go引起的方差问题没什么关联，按我们前几节的分析来看其实更适合作为GKD-styel与PG-style的比较，即都站在token-level的角度，比较使用何种方式的rKL梯度（full-vocab / sampled-token / top-k 等等），所以这里就不继续展开了，感兴趣的朋友可以自行阅读。

5.2 RLVR中的reward-to-go

主流 outcome-based RLVR 通常是在整条 sequence 完成后，先对这条完整回复打一个最终 reward

$$R(\tau),$$

然后将这个 sequence-level reward 广播给整条序列中的所有 token。也就是说，更常见的做法是令

$$r_t = R(\tau), \qquad \forall t = 1, \ldots, T.$$

从这个角度看，RLVR 的 token-level reward本质上仍然是由同一个 sequence-level outcome 派生出来的，因此是高度稀疏、缺乏局部区分度的——每个token的reward都一样。这也是为什么常说 PG-style OPD 可以被看作一种带有 dense-reward 的 RL：相较于 RLVR 把同一个最终 reward 广播给所有 token， OPD 在每个位置上天然提供了一个 teacher-derived 的局部信号。

在这种设定下，如果进一步考虑 reward-to-go，是否可以带来有用的信号以区别不同token的收益？由于

$$r_t = R(\tau),$$

于是

$$G_t = \sum_{u=t}^{T} r_u = \sum_{u=t}^{T} R(\tau) = (T - t + 1)\, R(\tau).$$

也就是说，这时 reward-to-go 并不会引入真正新的“局部内容信息”，它只会让越靠前的位置拿到越大的同号权重：

- 前面的 token 权重大一些；
- 后面的 token 权重小一些；
- 但它们共享的本质上仍然是同一个 sequence-level outcome reward。

因此，在 outcome-based RLVR 中， reward-to-go 的作用往往更像是一种**位置重加权**，而不是像一般 dense-reward RL 那样，真正把未来回报细粒度地分配回不同动作。那么在 RLVR 中，有什么办法可以比较好地考虑未来收益以区别开不同 token 的作用吗？ FIPO: Eliciting Deep Reasoning with Future-KL Influenced Policy Optimization 这篇 paper 就研究了这个问题。

FIPO 的核心出发点是利用了一个token级别的指标 log-prob shift：

$$\Delta \log p_t = \log \pi_\theta(o_t \mid q, o_{<t}) - \log \pi_{\theta_{\text{old}}}(o_t \mid q, o_{<t}),$$

至于这个指标的出处，是作者团队的另一篇工作On the Direction of RLVR Updates for LLM Reasoning: Identification and Exploitation。直观上讲，这个值越大表明当前

位置的采样token越好（即优化方向是增大它的概率）。

FIPO在此基础上定义了一个 Future-KL / Future-Shift 风格的量，把当前位置之后未来 token 的概率变化累积起来，作为该 token 的“未来影响”估计。也即：

$$\text{FutureKL}_t = \sum_{k=t}^{T} \Delta \log p_k.$$

直观上，这个量想回答的是：

“当前这个 *token* 所开启的后续轨迹，在整体上是被新策略强化还是抑制了？”

如果 $\text{FutureKL}_t > 0$，说明从当前位置开始的未来轨迹整体上更被当前策略支持；反之若 $\text{FutureKL}_t < 0$，则说明这条后续轨迹整体上在被压制。

不过，直接累加未来项很容易把方差重新拉高，尤其是在长序列和训练/推理分布不一致时，远处 token 的概率漂移会带来很大的噪声。因此，FIPO 又进一步做了两件事：

- **mask / clipping**：对那些 importance ratio 过大、明显不稳定的 token，不让它们继续参与未来累积。具体实现上来讲就是通过定义一个指标进行过滤：

$$M_k = \mathbf{1}\left(\frac{\pi_\theta(o_k \mid o_{<t})}{\pi_{\text{old}}(o_k \mid o_{<t})} \le c\right)$$

- **soft decay**：对更远处的 future signal 施加衰减因子，使得越远的未来影响越小，也就是我们前面说的GAE：

$$\text{FutureKL}_t = \sum_{k=t}^{T} M_k \, \gamma^{k-t} \, \Delta \log p_k, \qquad \gamma \in (0, 1),$$

当然，作者并不是直接使用这个量作为reward，而是把它和 outcome-reward / advantage 结合起来，具体做法是把这个 Future-KL 映射成一个额外的权重因子，去重加权当前位置的 advantage：

$$A_t = \hat{A}_t \cdot f_t$$

这里$\hat{A}_t$就是原始的基于outcome的reward计算的advantage，f_t就是上面所说 FutureKL_t。具体实验效果还是很不错的，可以参考原文，这里就不赘述了。

当然引入未来的credit assignment也可能不止文中这一种基于log-prob shift的做法。如果去考虑其他的信号，可能也需要权衡一下计算成本，毕竟传统RLVR实现上还是最精简的。

6）GKD-style：前缀分布下的未来耦合

这里还需要特别澄清一点：这一节讲的 reward-to-go，本质上是 **sequence-level reverse-KL 在 policy gradient / REINFORCE 视角下的展开**，因此主要针对的是 **PG-style**。

对于 GKD-style，训练不是通过

$$G_t \nabla_\theta \log \pi_\theta(a_t|s_t)$$

这种 sampled-action score-function credit assignment 来实现的，而是在每个已访问 prefix 上，直接对 full-vocab distillation loss 求导：

$$\ell_t^{\mathrm{rKL}} = \sum_v p_v^{(t)} \log \frac{p_v^{(t)}}{q_v^{(t)}}.$$

因此，GKD-style 中并不存在与 PG-style 完全对应的 reward-to-go 对象。它的“未来耦合”更多体现为：

- 前面的采样决定了未来会访问哪些 prefix；
- 在这些 prefix 上再计算局部 full-vocab KL。

也就是说，GKD-style 的 future effect 更像一种**隐式的状态分布偏移**，而不是 PG-style 里那种显式的 return-to-go 权重。

所以可以简单概括为：

reward-to-go 是 PG-style 的显式 credit assignment 问题，而不是 GKD-style 的关注对象。

7）小结

这一节其实主要是在补一个前面默认略去、但理论上非常重要的问题：

PG-style OPD / RL 里，我们是否只需要考虑“即时reward”？

简单总结如下：

- LLM 不仅定义了每一步的 next-token 条件分布，也通过链式法则在整个序列空间上定义了联合分布；
- 因而 sequence-level reverse-KL 是一个完全合法的分布差异目标；

- 若从 sequence-level reverse-KL 的 policy gradient 严格出发，那么第 t 步动作对应的权重应为

$$G_t = \sum_{u=t}^{T} r_u^{\mathrm{KL}},$$

即 reward-to-go；

- 常见的 token-level PG-style 则进一步近似为只使用当前位置的即时项

$$r_t^{\mathrm{KL}},$$

因而它相对完整 sequence-level PG 是有偏的；

- 但这种近似通常也能显著降低方差，因此在长时程场景中更 practical；
- 在 OPD 里，即时 KL 本身已经是一个局部 teacher alignment 信号，因此 token-level 近似虽然有偏，但仍然往往可以 work；
- 但它的脆弱性也恰恰来自忽略 future coupling、teacher 在 student prefix 上可能失真，以及 sampled-token 信号过于局部；
- reward-to-go 主要是 PG-style 的显式 credit assignment 问题，而不是 GKD-style 的核心对象。