RL 当中的正则化方法

大语言模型时代的强化学习 - 2

前言

本文是大语言模型强化学习系列的第二集，内容承接上集「基础理论」部分而来，主要探讨大语言模型 RL 算法当中的各类正则化方法。正则化方法本身普遍存在于神经网络的各个方面，如模型结构、优化算法等等，其的主要意义在于稳定训练过程以及确保泛化能力。强化学习算法主要围绕着构建训练的目标函数进行，因此此处涉及到的正则化方法都是在目标函数计算过程当中发挥作用的。

当前强化学习训练当中的稳定性也是其关键问题之一[1], [2]，因此它们都属于 RL 算法的必要拼图，并且相关研究也都在快速发展当中。在本文当中我们也仅讨论一些被广泛采纳并验证过的技巧。后续的新技巧可能会更新在新的文章当中。

策略似然比截断 (Ratio Clip)

在前文当中我们有提到过 GRPO 和 GSPO 目标函数的主项本质上是真实目标的一个一阶近似，它在容许一定偏差的情况下换来了估计方差的显著下降，由此取得了明显更好的训练稳定性。如果愿意的话它们实际上也可以被视作一种正则化方法，而且属于几乎是必要的一类 (当前在 RLVR 之前的时代它们本身就是真实目标，偏差则体现在 value 估计上)。

然而仅仅依靠它进行训练的话在很多场景下仍然不足以保持稳定，在 TRPO 和 PPO 等较早期的算法研究当中研究者已经尝试寻找过很多其他的正则化方法。其中 PPO 所提出的核心技巧，似然比截断，在当时就被证明非常简单有效，并且一直流行至今 [3]。回顾上一讲当中我们有提及到 PPO 算法的损失函数主项是：

$$\mathcal{J}_{\text{PPO}}(\theta) = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D},\boldsymbol{y}\sim\mu_{\theta_{\text{old}}}(\cdot\mid\boldsymbol{x})}\left[\frac{1}{|\boldsymbol{y}|}\sum_{t=1}^{|\boldsymbol{y}|} r_t(\boldsymbol{y},\boldsymbol{x})A_t(\boldsymbol{x},\boldsymbol{y})\right]$$

在这个公式当中 $r_t(\boldsymbol{y},\boldsymbol{x})$ 就是逐 token 的重要性权重，也就是新旧策略的似然比。在实际的 RL 训练当中，由于各种原因部分 token 的策略偏差可能非常之大。例如在 [2] 的实验当中，部分 token 在新旧策略下的概率差距可能达到几十倍以上。它们就会造成目标函数估计的显著偏离，并进一步影响训练的稳定性。而 PPO 当中的 clip 方法就可以针对它们进行处理，它所做的就是将优化目标修改为：

$$\mathcal{J}_{\text{PPO clip}}(\theta) = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D},\boldsymbol{y}\sim\mu_{\theta_{\text{old}}}(\cdot\mid\boldsymbol{x})}\left[\frac{1}{|\boldsymbol{y}|}\sum_{t=1}^{|\boldsymbol{y}|} \min(r_tA_t, \text{clip}(r_t, 1-\varepsilon, 1+\varepsilon)A_t)\right]$$

这里 ε 是某一个人为选取的阈值参数，在很多实现当中默认值都会选取为 0.4 左右，误差超过这一范围的似然比就会被截断到这个范围之内。这里在截断之外还有一个取最小值的操作，这其实是一种悲观维稳的方案，也就是要求被训练的模型「不能高估成功，不能低估失败」。

在后来的很多工作当中也都继承了这种截断方法，例如 GRPO 对这一项的处理就和 PPO 完全相同，而 GSPO 则没有采用逐 token 的截断方法，而是先计算出序列层级的近似似然比，而后再进行整体截断。

关于这个方法的另一个重要优化是 DAPO 当中提出的 clip higher [4]，也就是对于似然比的上下限分别采用不同的阈值，并且上界提得更高一些。在最近的一些研究当中也指出，这套方法可以非常有效地实现熵控制，帮助取得稳定的性能提升 [5]。直觉上它相比较原 PPO clip 而言，就会让被训练的模型更加倾向于「适当更加重视成功」，如此就会更有利于鼓励被训练的策略去做更多的探索。在实验上这样一个简单的小技巧也确实能够有效地帮助防范熵坍缩的问题。

不过其实我们也可以从数学上去考虑这个问题，此前似乎没有什么人去研究。首先我们知道似然比作为两个概率的商，其数值理应分布在 $r_t(\boldsymbol{y}, \boldsymbol{x}) \in [0, +\infty)$，并且每一个比值数值上都接近于 1。但此时它显然并不是一个以 1 为中心的对称分布，而应当呈现出明显的重尾态势。而此时我们知道对它取一个对数的话，就可以得到

$$\log r_t(\boldsymbol{y}, \boldsymbol{x}) = \log \pi_\theta(y_t \mid \boldsymbol{x}, y_{<t}) - \log \mu_{\theta_{\text{old}}}(y_t \mid \boldsymbol{x}, y_{<t}) \in (-\infty, +\infty)$$

它就近似是一个对称分布！并且取对数之后它的数值就分布在 0 附近。对它做左右对称的裁切才是更合理的做法，而后从对数空间放回到原空间之后就自然地得到了 clip higher 这一结果。事实上更进一步地，如果我们去考虑它的期望的话，就会直接发现它和熵之间的关联

$$\mathbb{E}_{\mu_{\theta_{\text{old}}}}[\log r_t] = H\left(\mu_{\theta_{\text{old}}}\right) - H\left(\mu_{\theta_{\text{old}}}, \pi_\theta\right) = -D_{\text{KL}}\left(\mu_{\theta_{\text{old}}}, \pi_\theta\right) \leq 0$$

右侧两项取期望后分别就是两个分布的熵和交叉熵 (使用符号 H 表示)，作差之后刚好就两个策略的 KL 散度的相反数，因此总是负值。这或许意味着随着训练的进行我们还有可能通过动态调整 clip 范围来避免偏差，当然这些都是在对数空间操作更加合适。不过在实践当中暂时还没有见到应用这种细致调控的工作。

KL 散度约束

基础概念

在前文当中我们有提到语言模型的熵、交叉熵以及 KL 散度等概念，这里再给一个更完善的定义。对于两个语言模型的概率分布而言它们的交叉熵就被定义为

$$H\left(\pi_{\theta_{\text{old}}}, \pi_\theta\right) = -\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}, \boldsymbol{y}\sim\pi_{\theta_{\text{old}}}}\left[\log \pi_{\theta_{\text{old}}}(y_t \mid \boldsymbol{x}, y_{<t})\right] \geq H\left(\pi_{\theta_{\text{old}}}, \pi_{\theta_{\text{old}}}\right) \geq 0$$

这些不等关系都可以通过简单的数学分析得到。交叉熵的最小值也就是采样概率分布自身的熵值，也记作 $H\left(\pi_{\theta_{\text{old}}}\right)$。而概率论当中经典的 Kullback-Leibler divergence (KL 散度) 也就根据这一性质进行定义，它就是

$$D_{\text{KL}}\left(\pi_{\theta_{\text{old}}}, \pi_\theta\right) = H\left(\pi_{\theta_{\text{old}}}, \pi_\theta\right) - H\left(\pi_{\theta_{\text{old}}}\right) \geq 0$$

这个指标可以非常好地衡量两个概率分布之间的差异，在大量的工作当中都被用于监控训练稳定性以及直接用于正则化 [2], [3], [6], [7]。此外值得注意的是大语言模型的不同使用方式，预测下一个 token 和自回归生成两种情形下会对应两种不同的熵和 KL 散度计算方式，目前较流行的似乎仍是逐 token 的计算方式。在对数空间当中这两者主要就差在序列长度的影响上。关于这个优化可说的东西也有非常多，其中最直接的两个问题就是：

- 如何计算 KL 散度？它本身就是一个期望形式的数值，我们无法计算精确的 KL 散度数值，因此必然需要使用到某些估计方法，而它们的性质则可能各有不同。
- 如何使用 KL 散度？已有方法包括直接使用数值用于监测训练，加入 reward 计算或者目标函数计算当中直接起到约束与正则化等等功效。KL 散度的不同使用场景也会直接影响我们在计算上的决策。

首先对于第二个问题，当前使用 KL 散度进行正则化的方法主要有两种。一者是类 PPO 的做法，在每一个数据点的优势函数当中额外减去一个正则化系数乘以 KL 散度值；二者是类 GRPO 的路线，它将 KL 散度乘以一个正则化系数后直接加在目标函数之后。这两种做法都能起到约束策略更新幅度的作用，但是在实际计算上侧重点却不同。前者要求更精确地计算出 KL 散度的具体数值，以更好地完成优势估计。由于优势函数本身不参与梯度计算，因此它对于 KL 散度的梯度并没有要求。后者则实际上并不需要关心 KL 散度的具体数值，但作为目标函数的一部分，我们必然需要更好地估计 KL 散度的正确梯度，否则正则化项也可能成为额外的不稳定因素。但是值得注意的是 AReaL 当中的 GRPO 算法实现实际上使用的是类 PPO 的 KL 正则化，并且使用方式上也因为异步训练的特性而有所不同。

KL 散度约束的目的都是限制训练后模型相比较原模型的更新幅度。由于原模型一般都被认为是表现相对稳定的版本，因此加上这一项优化后就可以更好地保持训练稳定。另一方面如果训练目标本身和真实需求之间存在偏差，可能造成 reward hack，那么使用 KL 散度约束也可以取得更好的稳定性以及泛化能力。当然在某些训练目标非常明确且有效的场景，例如针对数学习题的 RLVR 当中，就可以完全不使用 KL 散度约束，且这样的训练效果还会更好 [8]。

除去使用 KL 散度进行正则化外，它本身也是一个很好的监控指标。如在 [2] 当中使用它来检测训练稳健性时，我们所需的也是更好的数值估计而非梯度估计。

数值估计

此处内容部分参考自王锡淮大佬的博客，但细节设定上有所不同。在开始这两个问题之前，其实我们还有另一个基础性的问题。两个概率分布之间的 KL 散度计算实际

上存在两种不同的变体，分别是反向 KL ($D_{\text{KL rev}}$) 和正向 KL ($D_{\text{KL fwd}}$)。KL 散度本身并不是一个度量，其两个分布的顺序也会对结果产生实质性的影响，进而也会影响到训练的效果。具体而言两种 KL 散度的定义方式是

$$D_{\text{KL fwd}} = D_{\text{KL}}\left(\mu_{\theta_{\text{old}}}, \pi_\theta\right) = \mathbb{E}_{\boldsymbol{y}\sim\mu_{\theta_{\text{old}}}}\left[\log \mu_{\theta_{\text{old}}}(\boldsymbol{y}) - \log \pi_\theta(\boldsymbol{y})\right]$$

$$D_{\text{KL rev}} = D_{\text{KL}}\left(\pi_\theta, \mu_{\theta_{\text{old}}}\right) = \mathbb{E}_{\boldsymbol{y}\sim\pi_\theta}\left[\log \pi_\theta(\boldsymbol{y}) - \log \mu_{\theta_{\text{old}}}(\boldsymbol{y})\right]$$

我们不难看出正向 KL 的含意是直接使用参考分布的样本估计散度，而反向 KL 则是使用当前策略样本估算散度，在实践上前者其实也可以通过重要性采样的方法获得。参考 DeepSeek-V3.2 的技术报告 [8] 以及知名训练框架 AReaL 的源代码，可见 DeepSeek-V3.2 的 GRPO 算法当中使用的是反向 KL，而 AReaL 当中实现的则是正向 KL。后面我们主要以 AReaL 的版本为例。

在 John Schulman 这位大佬的一篇经典博客当中他就给出了三种关于 KL 散度的估计方法，在不同的应用场景下各有优劣。在很多地方它们分别被称作 k_1, k_2, k_3 估计量，对于我们的正向 KL 而言它们分别是：

$$k_1 = \log \mu_{\theta_{\text{old}}}(\boldsymbol{y}) - \log \pi_\theta(\boldsymbol{y}) = -\log r$$

$$k_2 = \frac{1}{2}\left(\log \mu_{\theta_{\text{old}}}(\boldsymbol{y}) - \log \pi_\theta(\boldsymbol{y})\right)^2 = \frac{1}{2}(\log r)^2$$

$$k_3 = \frac{\pi_\theta(\boldsymbol{y})}{\mu_{\theta_{\text{old}}}(\boldsymbol{y})} - \log \frac{\pi_\theta(\boldsymbol{y})}{\mu_{\theta_{\text{old}}}(\boldsymbol{y})} - 1 = r - \log r - 1$$

此处

$$r := \frac{\pi_\theta(\boldsymbol{y})}{\mu_{\theta_{\text{old}}}(\boldsymbol{y})} = 1 + \delta$$

我们知道参与 KL 散度计算的两个分布之间差距并不大，因此可以假设 δ 是一个绝对值较小的随机变量。接着我们可以逐一去分析它们的性质。

在三者之中，最简单的就是 k_1 估计量，它本身就是对正向 KL 散度原定义的直接的 Monte-Carlo 估计，显然是无偏的 (也就是期望等于真实值)。但是在实践当中它的主要问题在于估计方差较大，因为我们知道真实的 KL 散度本身是恒正值，但 k_1 却几乎是正负对半的均匀分布。

k_2 估计量就很好地解决了这一问题，因为它本身作为平方值总是恒正的，所以估计方差会小很多。与此同时它的估计偏差实际上也不大，对于这一性质的证明稍微有一些小 tricky。

在此前的假设当中，除去已知 δ 数值较小之外，更妙的是我们知道它的期望就是精确的 0。这是因为：

$$\begin{aligned}\mathbb{E}_{\mu_{\theta_{\text{old}}}}[\delta] &= \mathbb{E}_{\mu_{\theta_{\text{old}}}}[1+\delta] - 1 = \mathbb{E}_{\mu_{\theta_{\text{old}}}}[r] - 1 \\ &= \int \frac{\pi_\theta(\boldsymbol{y})}{\mu_{\theta_{\text{old}}}(\boldsymbol{y})} \mu_{\theta_{\text{old}}}(\boldsymbol{y})\,\mathrm{d}\boldsymbol{y} - 1 = \int \pi_\theta(\boldsymbol{y})\,\mathrm{d}\boldsymbol{y} - 1 = 0\end{aligned}$$

由此如果我们对 $\log r = \log(1+\delta)$ 做 Taylor 展开而后近似就会得到：

$$\mathbb{E}_{\mu_{\theta_{\text{old}}}}[k_1] = \mathbb{E}_{\mu_{\theta_{\text{old}}}}[-\log(1+\delta)] \approx \mathbb{E}_{\mu_{\theta_{\text{old}}}}\left[-\delta + \frac{1}{2}\delta^2\right] = \mathbb{E}_{\mu_{\theta_{\text{old}}}}\left[\frac{1}{2}\delta^2\right]$$

$$\mathbb{E}_{\mu_{\theta_{\text{old}}}}[k_2] = \mathbb{E}_{\mu_{\theta_{\text{old}}}}\left[\frac{1}{2}(\log(1+\delta))^2\right] \approx \mathbb{E}_{\mu_{\theta_{\text{old}}}}\left[\frac{1}{2}\left(\delta - \frac{\delta^2}{2}\right)^2\right] \approx \mathbb{E}_{\mu_{\theta_{\text{old}}}}\left[\frac{1}{2}\delta^2\right]$$

由此可知两者的均值近似相等，k_2 估计确实存在偏差，但偏差也仅是关于 delta 的三阶项以下。

作为绝对数值估计，k_3 估计则采用了另一种方式实现恒正的估计量，它也是目前 DeepSeek-V3.2 的 GRPO [8]，经典的 PPO 等训练算法当中首选的估计方法。实际上它就是在 k_1 的基础上减去了其在 1 处的切线值，加上函数的凸性就可以得到一个恒正的二阶误差值。它有着显著小的方差，同时甚至还是一个无偏的估计量，这是因为：

$$\mathbb{E}_{\mu_{\theta_{\text{old}}}}[k_3] = \mathbb{E}_{\mu_{\theta_{\text{old}}}}[r - \log r - 1] = \mathbb{E}_{\mu_{\theta_{\text{old}}}}[r] + \mathbb{E}_{\mu_{\theta_{\text{old}}}}[-\log r] - 1 = \mathbb{E}_{\mu_{\theta_{\text{old}}}}[-\log r]$$

这一期望就是我们所需的真实值。由此我们也不难理解为什么它成为了很多类算法的首选。

梯度估计

接下来我们考虑三个估计量对于真实 KL 散度梯度的估计情况。首先我们来看一下真实的正向 KL 散度的梯度数值，注意在这里我们始终假设积分和求导可以换序，并且事实上正向 KL 和反向 KL 对应的梯度方向也是不同的。直接推导可以得到：

$$\begin{aligned}\nabla_\theta D_{\text{KL fwd}} &= \nabla_\theta \int \Big(\log \mu_{\theta_{\text{old}}}(\boldsymbol{y}) - \log \pi_\theta(\boldsymbol{y})\Big)\mu_{\theta_{\text{old}}}(\boldsymbol{y})\,\mathrm{d}\boldsymbol{y} \\ &= -\int \nabla_\theta \log \pi_\theta(\boldsymbol{y}) \mu_{\theta_{\text{old}}}(\boldsymbol{y})\,\mathrm{d}\boldsymbol{y} \\ &= -\mathbb{E}_{\mu_{\theta_{\text{old}}}}[\nabla_\theta \log \pi_\theta(\boldsymbol{y})] = -\mathbb{E}_{\mu_{\theta_{\text{old}}}}[s_\theta] \\ \nabla_\theta D_{\text{KL rev}} &= \nabla_\theta \int \Big(\log \pi_\theta(\boldsymbol{y}) - \log \mu_{\theta_{\text{old}}}(\boldsymbol{y})\Big)\pi_\theta(\boldsymbol{y})\,\mathrm{d}\boldsymbol{y} \\ &= \int \nabla_\theta(\pi_\theta(\boldsymbol{y}) \log \pi_\theta(\boldsymbol{y}))\,\mathrm{d}\boldsymbol{y} = \mathbb{E}_{\pi_\theta}[s_\theta \log r]\end{aligned}$$

此处 θ_{old} 本身是一个常数，因此相关梯度均为 0。另一方面我们知道 $s_\theta := \nabla_\theta \log \pi_\theta(\boldsymbol{y})$ 就是概率论当中的分数函数 (score function)。关于它还有一个经典的性质是

$$\mathbb{E}_{\pi_\theta}[s_\theta] = \int \pi_\theta(\boldsymbol{y}) \nabla_\theta \log \pi_\theta(\boldsymbol{y}) \, \mathrm{d}\boldsymbol{y} = \nabla_\theta \int \pi_\theta(\boldsymbol{y}) \, \mathrm{d}\boldsymbol{y} = \nabla_\theta 1 = 0$$

上述推导当中就使用到了这一技巧。接下来我们就看一下这几个估计量对应的梯度如何。注意由于我们的样本都是直接采样自 $\mu_{\theta_{\text{old}}}$，因此后续计算当中期望都理应是针对这一个策略进行计算的。在近期 DeepSeek-V3.2 的技术报告当中他们额外对 KL 散度估计应用了重要性采样，由此才可以得到对 KL 散度正确无偏的估计 [8]。我们同样计算这些无偏版本的 KL 估计值的梯度，注意由于 DeepSeek-V3.2 使用反向 KL 估计，因此推导结果也会有所不同。

$$\begin{aligned}
\nabla_\theta \mathbb{E}_{\mu_{\theta_{\text{old}}}}[k_1] &= -\mathbb{E}_{\mu_{\theta_{\text{old}}}}[s_\theta] = \nabla_\theta D_{\text{KL fwd}} \\
\nabla_\theta \mathbb{E}_{\mu_{\theta_{\text{old}}}}[\text{sg}[r] k_2] &= \mathbb{E}_{\pi_\theta}[\log r \nabla_\theta \log r] = \nabla_\theta D_{\text{KL rev}} \\
\nabla_\theta \mathbb{E}_{\mu_{\theta_{\text{old}}}}[k_3] &= \mathbb{E}_{\mu_{\theta_{\text{old}}}}[\nabla_\theta r - \nabla_\theta \log r] = \mathbb{E}_{\mu_{\theta_{\text{old}}}}[(r-1) s_\theta] \\
&= \mathbb{E}_{\pi_\theta}[s_\theta] - \mathbb{E}_{\mu_{\theta_{\text{old}}}}[s_\theta] = -\mathbb{E}_{\mu_{\theta_{\text{old}}}}[s_\theta] = \nabla_\theta D_{\text{KL fwd}}
\end{aligned}$$

此处 $\text{sg}[r]$ 表示对其中表达式停止梯度计算，这在 PyTorch 当中是很容易实现的功能。关于这些估计量梯度方差的情况实际上与原数值类似，k_2, k_3 都具有较小的估计方差，而 k_1 的方差则相对较高。具体的推导可以给读者留作习题，在本文当中就不细致推导了。

策略似然比截断和 KL 散度约束就是当前 RL 算法当中应用最广泛的正则化方法，在此前提到的 RL 目标函数的基础上应用它们之后就可以得到各论文当中完整的 RL 训练目标了。除去这两者之外关于稳定 RL 的工作还非常多，并且很多新的 trick 都已经被广泛应用在了工程实践当中，后面我们也举几个例子来。

几何序列遮罩

几何序列遮罩 (Geometric Sequence Masking) 由字节跳动的研究团队在一篇 blog 当中首次提出 [2]，理论和工程实验都相当完备，并且已经在 DeepSeek-V3.2 等前沿的大模型训练当中得到了应用 [8]，因此也非常值得一说。

这一技巧最早来自于字节对于大模型 RL 训练当中闪崩现象的研究。这就是说，在当前很多大语言模型的 RL 训练进行当中会时常出现训练突然崩溃的情况，reward 值突然大幅下降并且几乎无法恢复。经研究之后这个问题的成因主要是大模型的训推不一致问题，尤其是部分低概率样本的训练与推理似然可能相差甚大，由此在某些样本当中可能对目标函数估计产生巨大的偏差，进而造成训练崩溃。

当前主流的类 GRPO 的一阶近似方法就可能在概率偏离过多时失效，因此一个直接的解决方案就可以是，主动监控并过滤掉偏离过多的样本。而几何序列遮罩就是其中一个稳定且有效的方案，在 DeepSeek-V3.2 的实现当中它的操作方式是在每一个序列的 loss 上额外乘以这一项：

$$M_i = \begin{cases} 0 & A_i < 0 \wedge \frac{1}{|\boldsymbol{y}_i|} \log\left(\frac{\mu_{\theta_{\text{old}}}(\boldsymbol{y}_i, \boldsymbol{x})}{\pi_\theta(\boldsymbol{y}_i, \boldsymbol{x})}\right) > \delta \\ 1 & \text{otherwise} \end{cases}$$

也就是当优势函数值为负，并且序列对数似然的均值超过某一个阈值时，就会触发遮罩，停止针对这一条数据进行训练。由于对数似然的均值本身可以视作对似然比的几何平均，这也是其名称的由来。由于序列的似然比会随着长度指数增长，数值范围不一，因此为了统一信任区间的范围就需要做这样的长度平均。这一方法本身是和策略似然截断配合使用的，并且着重针对重要性权重过高的负样本进行屏蔽，因为它们被认为对训练造成的危害最多。关于这一技术的更多理论细节可参考这篇博客。

其他的小技巧

在前文之外的其他强化学习当中的正则化方法还有很多，并且在工程应用当中也非常普遍，在这里仅对其中的部分稍作提及，不再细讲了。这些方法起作用的主要原理都是在于维持训练与推理的一致性，以及稳定估计方差等等，在统计上能产生的效果不难想象。

首先 DeepSeek-V3.2 [8] 当中就提及了其中在算法上应用的其他技术：

- 维持采样遮罩 (Keep Sampling Mask)。当前的大语言模型在采样时往往并不是直接根据概率采样的，而是会有 top-p，top-k 等等控制参数，仅在一个子集当中采样得到结果。因此在训练时也应当维持同样的遮罩，如此才能正确保证训推一致。此前也存在很多 temperature 参数不同的情况，不过现在业界逐渐产生的共识是无论训练、推理以至最终部署，都始终保持 temperature 为 1 才能取得最好的性能。
- 维持路由 (Keep Routing)。这主要是针对 MoE 架构的语言模型设计的方法，由于 MoE 模型在推理时会由 Router 选择激活某一部分的神经网络，因此其输出概率事实上并不连续依赖于模型参数，而是可能存在很多跳跃点。这会对 RL 训练的稳定性造成很大的影响，而一个简单的解决方案就是人为限定推理和训练时路由到的神经网络保持一致。在实践上针对训推不一致的两个成因它还可以进一步被区分为 R2 和 R3 两套方案，可以不同程度地维持训练稳定性，但仍会引入一定的偏差影响性能表现 [1]。

在此之外社区内常常提及的技巧还有如 tokenizer 上的处理等等，由于当前的 tokenzier 实际上存在多义性，即一段文本可能对应多个 token 序列，这也会产生一定的训推不一致现象，需要在工程实现上加以处理。

后记

在本文当中我们讲解了当前强化学习算法当中主要的一些正则化方法，其中策略似然比截断和 KL 散度约束是其中最常用的两种方案。而随着当前大模型 RL 技术的发展，也涌现出了很多新的维稳方法，其中很多都已经被广泛地应用在了工程实践之上。从个人的技术偏好上来说我并不喜欢在算法上增加太多的复杂性，即使它可能在工程上无法避免。复杂系统各个组件之间的相互影响会使得各种实验结果的归因变得格外困难，并明显地限制了方法的泛化能力和后续改进的可能。因此相比较添加新的技术而言、删减与重构带来的优化可能更加令人兴奋。此前的 JustRL 就是一次拨乱反正式的尝试 [5]，而我们在 RL 算法上还有非常多研究的空间在。

Bibliography

[1] C. Zheng *et al.*, "Stabilizing Reinforcement Learning with LLMs: Formulation and Practices." Accessed: Dec. 06, 2025. [Online]. Available: http://arxiv.org/abs/2512.01374

[2] J. Liu, Y. Li, Y. Fu, J. Wang, Q. Liu, and Y. Shen, "When Speed Kills Stability: Demystifying RL Collapse from the Training-Inference Mismatch." [Online]. Available: https://richardli.xyz/rl-collapse

[3] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms." Accessed: Jan. 22, 2025. [Online]. Available: http://arxiv.org/abs/1707.06347

[4] Q. Yu *et al.*, "DAPO: An Open-Source LLM Reinforcement Learning System at Scale." Accessed: Mar. 20, 2025. [Online]. Available: http://arxiv.org/abs/2503.14476

[5] B. He *et al.*, "JustRL: Scaling a 1.5B LLM with a Simple RL Recipe." Accessed: Dec. 23, 2025. [Online]. Available: http://arxiv.org/abs/2512.16649

[6] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust Region Policy Optimization." Accessed: Jan. 02, 2026. [Online]. Available: http://arxiv.org/abs/1502.05477

[7] Z. Shao *et al.*, "DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models." Accessed: Jan. 22, 2025. [Online]. Available: http://arxiv.org/abs/2402.03300

[8] DeepSeek-AI *et al.*, "DeepSeek-V3.2: Pushing the Frontier of Open Large Language Models." Accessed: Dec. 05, 2025. [Online]. Available: http://arxiv.org/abs/2512.02556