Proximal Policy Optimization

Yiming

February 9, 2025

1 强化学习基础

我们记录强化学习中记号如下，记第 t 步下，状态 (state) 为 s_t，基于状态 s_t 采取的动作为 a_t，策略 (Policy) 为 $\pi(a_t|s_t)$，轨迹 (trajectory) 为 $\tau = \{s_0, a_0, s_1, a_1, \cdots\}$。状态转移我们规定为确定，即 $s_{t+1} = f(s_t, a_t)$。回报 (return) $R(\tau)$，基于路径下总的奖励和，我们总是希望回报最高。

我们训练的模型为策略 π_θ，在此策略模型下，某一路径出现的概率记为，$P_\theta(\tau)$。我们的目标函数为，

$$\mathbb{E}_{\tau\sim P_\theta(\tau)}[R(\tau)] = \sum_\tau R(\tau)P_\theta(\tau) \tag{1}$$

求梯度，

$$\begin{aligned}\nabla_\theta \mathbb{E}_{\tau\sim P_\theta(\tau)}[R(\tau)] &= \nabla_\theta \sum_\tau R(\tau)P_\theta(\tau) = \sum_\tau R(\tau)\nabla_\theta P_\theta(\tau) \\ &= \sum_\tau P_\theta(\tau)R(\tau)\frac{\nabla_\theta P_\theta(\tau)}{P_\theta(\tau)} \approx \frac{1}{N}\sum_{n=1}^{N} R(\tau_n)\frac{\nabla_\theta P_\theta(\tau_n)}{P_\theta(\tau_n)} \\ &= \frac{1}{N}\sum_{n=1}^{N} R(\tau_n)\nabla_\theta \log P_\theta(\tau_n)\end{aligned} \tag{2}$$

这里的 τ_n 意味抽样的含义，计算梯度时实际地抽取 N 条路径。考虑 $P_\theta(\tau_n) = \prod_{t=1}^{T_n} P_\theta(a_n^t|s_n^t)$，带入上式得到，

$$\begin{aligned}\nabla_\theta \mathbb{E}_{\tau\sim P_\theta(\tau)}[R(\tau)] &= \frac{1}{N}\sum_{n=1}^{N} R(\tau_n)\sum_{t=1}^{T_n}\nabla_\theta \log P_\theta(a_n^t|s_n^t) \\ &= \frac{1}{N}\sum_{t=1}^{T_n}\sum_{n=1}^{N} R(\tau_n)\nabla_\theta \log P_\theta(a_n^t|s_n^t)\end{aligned} \tag{3}$$

通过上述梯度对各个神经网络 θ 进行梯度上升，即实现了一次训练，称为 *Policy gradient*。上述的直接含义就是，如果 $R(\tau_n) > 0$，就增加所有 $P(a_n^t|s_n^t), \forall t$ 出现的概

率。一种可能的强化学习模式如同图1，通过尝试每一次模型的尝试，实时地给出对模型自身行为的修正。这种强化学习方式称为 *on policy*，即必须根据模型自身的表现来更新模型行为。不能根据我们对其他结果的批评，离线地进行强化学习。

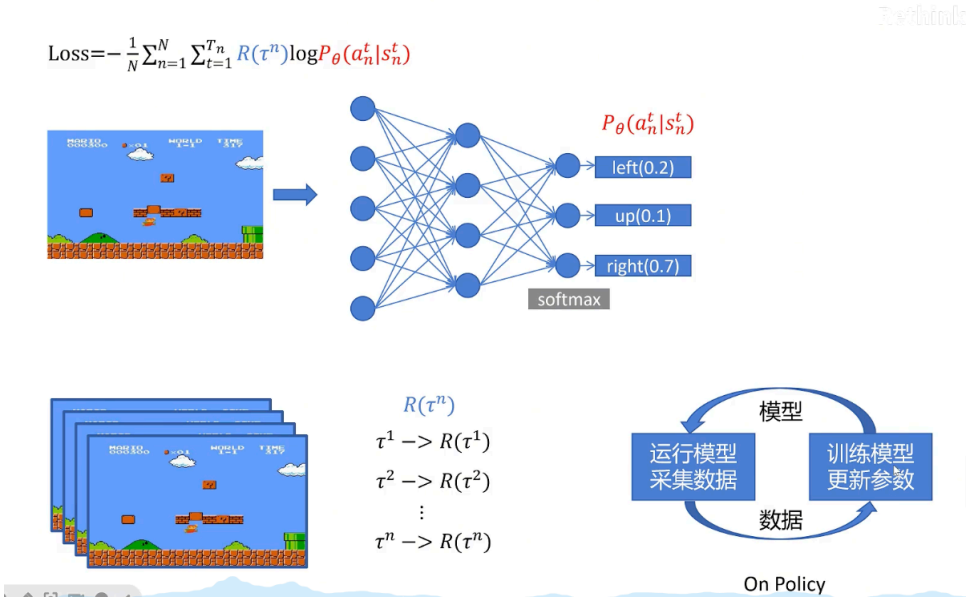

Figure 1: Policy gradient 示例 (on policy)

Improvement on policy gradient

上述目标函数有很多值得改进的空间，以加速训练的收敛效果。目前的目标函数过于笨重，对于所有的 $P(a_n^t|s_n^t), \forall t$ 一荣俱荣，一损俱损。但是我们可以通过更精细的方法去量化每一步 $a_n^t|s_n^t$ 的价值，例如，我们知道，

1. $a_n^t|s_n^t$ 贡献的价值与 $t-1$ 之前的价值无关。

2. $a_n^t|s_n^t$ 贡献的价值不仅仅与自身这一步显性收益相关，还与未来期望收益相关，但是，随着 t 的逐步攀升，这一未来收益考量对于当前步的影响应该降低。

基于此，我们应提出针对每一步行动的价值判断函数 $R_{t|\tau_n}$，经验性地，我们定义为，

$$R_{t|\tau_n} = \sum_{i=t}^{T_n} \gamma^{i-t} r_{i|\tau_n} \quad \gamma < 1 \tag{4}$$

这里，$r(a|s)$ 称为奖励 *(reward)*，来评估单步贡献，奖励模型是手动设置的模型，独立于模型训练过程；奖励模型分为分布奖励 (每一步都给予奖励，例如游戏设计中捡到金币) 和整体奖励 (大部分步骤奖励都是 0，在一个完整的任务结束时进行奖励的结算)。所以，$R_{t|\tau_n}$ 意味着在路径 τ_n 中，第 t 步的回报，它等于在这条路径中之后行动的价值和，通过衰减因子加权。

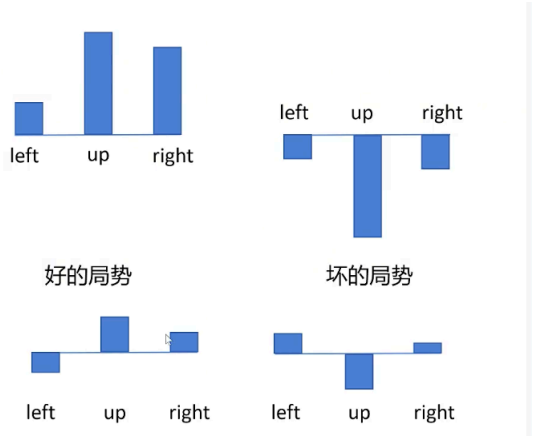

Figure 2: Baseline 示例

另外一个观察是，我们会在某步面临着此时所有行动都有正向回报/负向回报的情形，我们会从中选择最正向的行动，对于每一步设置一个 baseline 是必要的，如同图2所示。

启发性地，我们可以写出如下形式目标函数梯度，

$$\frac{1}{N}\sum_{t=1}^{T_n}\sum_{n=1}^{N}(R_{t|\tau_n} - B(s_{t|\tau_n}))\nabla_\theta \log P_\theta(a_n^t|s_n^t) \tag{5}$$

使得 $B(s_{t|\tau_n})$ 作为基线。这已经非常好了，然而，现在唯一的问题是，$R_{t|\tau_n}$ 是基于对路径 τ 进行 N 次采样的估计，具有相当大的不确定性。

Canonical form

基于上述启发示探索，我们引入完整的强化学习概念，我们通过引入价值函数 (Value)，

1. 状态价值函数 (State-value function) $V_\theta(s)$，在当前状态 s 下，回报 R 的期望值。

2. 行动价值函数 (Action-value function) $Q_\theta(a|s)$，在状态 s 下，作出动作 a 的回报期望。

3. 优势函数 (Advantage function)，$A_\theta(s,a) = Q_\theta(a|s) - V_\theta(s)$，在状态 s 下，作出动作 a 能带来多少提升。

在此定义下，自然的目标函数梯度写为，

$$\frac{1}{N}\sum_{t=1}^{T_n}\sum_{n=1}^{N} A_\theta(s_t,a_t|\tau_n)\nabla_\theta \log P_\theta(a_n^t|s_n^t) \tag{6}$$

Generalized Advantage Estimation

在 PPO 中，优势函数的构建往往采用 *Generalized Advantage Estimation* (GAE)，方式如下：首先，我们选择构建经验关系来关联起状态价值函数和行动价值函数，防止多一个模型的训练，定义

$$Q_\theta(s_t, a) \equiv r_t + \gamma V_\theta(s_{t+1}) \tag{7}$$

从而有，

$$A^1_\theta(s_t, a) = r_t + \gamma V_\theta(s_{t+1}) - V_\theta(s_t) \tag{8}$$

根据上式可知，

$$V_\theta(s_t) \equiv A_\theta(s_t, a) - Q_\theta(s_t, a) = r_t + \gamma V_\theta(s_{t+1}) \tag{9}$$

自然有诸如此类的递推关系，

$$V_\theta(s_{t+1}) = r_{t+1} + \gamma V_\theta(s_{t+2}) \tag{10}$$

这是基于 1 步运算 $V_\theta(s_{t+1}), V_\theta(s_t)$ 进行的优势函数计算，同样的，代入上式子的下子式，我们可以通过 2 步运算来计算优势函数，

$$\begin{aligned} A^2_\theta(s_t, a) &\sim r_t + \gamma V_\theta(s_{t+1}) - V_\theta(s_t) \\ &\equiv r_t + \gamma(r_{t+1} + \gamma V_\theta(s_{t+2})) - V_\theta(s_t) \\ &= r_t + \gamma r_{t+1} + \gamma^2 V_\theta(s_{t+2}) - V_\theta(s_t) \end{aligned} \tag{11}$$

自然也有，

$$A^3_\theta(s_t, a) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 V_\theta(s_{t+3}) - V_\theta(s_t) \tag{12}$$

从自洽性来说，如果我们有完全自洽的状态函数 $V_\theta(s_t)$，那么应该有 $A^1_\theta(s_t, a) = A^2_\theta(s_t, a) = A^3_\theta(s_t, a)$，然而实际情况不然。更小的采样步数采样方差更小，但是系统偏差更大。

我们定义时序差分误差，

$$\delta_t = r_t + \gamma V_\theta(s_{t+1}) - V_\theta(s_t) \tag{13}$$

从而有，

$$\begin{aligned} A^1_\theta(s_t, a) &= \delta_t \\ A^2_\theta(s_t, a) &= \delta_t + \gamma\delta_{t+1} \\ A^3_\theta(s_t, a) &= \delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2} \end{aligned} \tag{14}$$

GAE 优势函数计算通过对不同阶优势函数通过附加不同的权重来计算最终的结果，

$$A^{\mathrm{GAE}}_\theta(s_t, a) \equiv (1-\lambda)(A^1_\theta + \lambda A^2_\theta + \lambda^2 A^3_\theta + \cdots) \tag{15}$$

注意到 $\sum_{i=0}^{\infty}\lambda^i = 1/(1-\lambda),\ (\lambda<0)$。上述式子可进一步写为，

$$\begin{aligned} A_\theta^{\mathrm{GAE}}(s_t,a) &\equiv (1-\lambda)(A_\theta^1 + \lambda A_\theta^2 + \lambda^2 A_\theta^3 + \cdots) \\ &= (1-\lambda)(\delta_t + \lambda(\delta_t + \gamma\delta_{t+1}) + \lambda^2(\delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2}) + \cdots) \\ &= (1-\lambda)(\delta_t(1+\lambda+\lambda^2+\cdots) + \gamma\delta_{t+1}(\lambda+\lambda^2+\cdots) + \cdots) \\ &= \sum_{b=0}^{\infty}(\gamma\lambda)^b\delta_{t+b} \end{aligned} \tag{16}$$

状态价值函数 V_θ 也是通过神经网络进行训练的，是有监督的训练，label 为，

$$V_\theta(s_t) = \sum_{i=t}^{T_n}\gamma^{i-t}r_{i|\tau_n} \tag{17}$$

通过对多路径 τ_n 进行训练。这也符合我们假设的状态价值函数和行动价值函数的关系。

2 Proximal Policy Optimization

PPO 首先实现的是将 on policy 训练改变为 *off policy*，以便充分利用采集到的数据。即并不是直接对模型的决策进行评估，而是让模型通过给出的示例及评价，来进行强化学习，这需要引入重要性采样的概念。

我们首先写出 on policy 的目标函数梯度，

$$\frac{1}{N}\sum_{t=1}^{T_n}\sum_{n=1}^{N} A_\phi(s_t,a_t|\tau_n)\nabla_\theta \log P_\theta(a_n^t|s_n^t) \tag{18}$$

注意，这里的

$$\frac{1}{N}\sum_{t=1}^{T_n}\sum_{n=1}^{N}|_{\tau\sim P_\theta} \sim \int P_\theta(\tau)\mathrm{d}\tau = \int \frac{P_\theta(\tau)}{q(\tau)}q(\tau)\mathrm{d}\tau \sim \frac{1}{N}\sum_{t=1}^{T_n}\sum_{n=1}^{N}|_{\tau\sim q_\theta}\cdot\frac{P_\theta(\tau)}{q(\tau)} \tag{19}$$

从而有，off policy 的目标函数为

$$\begin{aligned} &\frac{1}{N}\sum_{t=1}^{T_n}\sum_{n=1}^{N}|_{\tau\sim q_\theta} A_\phi(s_t,a_t|\tau_n)\frac{P_\theta(s_t,a_t|\tau_n)}{q(s_t,a_t|\tau_n)}\nabla_\theta \log P_\theta(a_n^t|s_n^t) \\ &= \frac{1}{N}\sum_{t=1}^{T_n}\sum_{n=1}^{N}|_{\tau\sim q_\theta} A_\phi(s_t,a_t|\tau_n)\frac{\nabla_\theta P_\theta(s_t,a_t|\tau_n)}{q(s_t,a_t|\tau_n)} \end{aligned} \tag{20}$$

其中 q 是离线基准模型。从而，我们可以定义损失函数为，

$$\mathrm{Loss}(\theta) = -\frac{1}{N}\sum_{t=1}^{T_n}\sum_{n=1}^{N}|_{\tau\sim q_\theta} A_\phi(s_t,a_t|\tau_n)\frac{P_\theta(s_t,a_t|\tau_n)}{q(s_t,a_t|\tau_n)} \tag{21}$$

重要性采样会自然存在权重分歧 (weight divergence) 问题，只有 proposal 和 target 分布足够相似，重要性采样不会崩坏，同样的，我们也不希望最终模型偏离基准模型 q 太远，从而，我们希望加一项 KL 散度项，

$$\text{Loss}(\theta) = -\frac{1}{N}\sum_{t=1}^{T_n}\sum_{n=1}^{N}|_{\tau\sim q_\theta} A_\phi(s_t, a_t|\tau_n)\frac{P_\theta(s_t, a_t|\tau_n)}{q(s_t, a_t|\tau_n)} + \beta\text{KL}(P_\theta||q) \tag{22}$$

另外，PPO 规定了重要性权重 $P_\theta(s_t, a_t|\tau_n)/q(s_t, a_t|\tau_n)$ 的改变“增益”的上限，防止模型剧烈的变化，从而 PPO 最终的 Loss 增加了截取函数，

$$-\frac{1}{N}\sum_{t=1}^{T_n}\sum_{n=1}^{N}|_{\tau\sim q_\theta} \min\left(A_\phi\frac{P_\theta}{q}, \text{clip}\left(\frac{P_\theta}{q}, 1-\epsilon, 1+\epsilon\right)A_\phi\right) \tag{23}$$

上述公式通过 min 和截取函数 clip 实现的功能是，增加高增益点的分布/减少低收益点的分布得到的增益是有限的，被压缩在 $1\pm\epsilon$ 之内，意味不要通过此来盲目追求某个收益点的极端高概率；但对于惩罚是不设置下界的，从而保证了性能不退化。

3 PPO for LLM

在通过 PPO 进行 LLM 强化学习训练时，需要构建离线数据库，通过需要提供一个问题 (question)，一个正例 (chosen)，一个反例 (rejected)。在进行强化学习模型之前，我们需要训练一个 Reward Model，一般会选择和当前大模型差不多的模型作为 Reward Model 的基座模型。

4 Else

一些有用的链接，

1. PPO 原理，讲得很好。https://www.bilibili.com/video/BV1iz421h7gb

Github (PPO 原理): https://github.com/RethinkFun/trian_ppo/tree/main/train_ppo

2. GRPO 漫谈，思路很清晰，但是有些地方感觉略有错误。https://www.youtube.com/watch?v=Yi1UCrAsf4o

Github(GRPO): https://github.com/krillinai/KrillinAI

3. GRPO，总体还行。https://zhuanlan.zhihu.com/p/19840603214

在写 note 的时候，我产生了如下问题：
1、是否需要构建 token level 的 value function。

众所周知，无论是 PPO 还是 GRPO，reward function 都是基于序列的，无论是在最后序列给一个分数还是分步骤给，但总归是基于 sequence 的。但是可以看到，无论是 PPO 还是 GRPO，定义的 Loss 都需要 t 步 action 的“优势函数”，这是基于 token 的。PPO 的选择是通过构建一个 value function，基于 reward function 有监督地学习基于 token 的 value function；GRPO 是直接把基于序列的 reward 一致地分配给这个回答下的每个 token，即公式 $A_{i,t} = \cdots$，注意到右侧式子与 t 独立。从而也实现了 Loss 的构建。

我一直在想为什么 GRPO 能绕开 value function 的构建，为什么不能把 PPO 视为只有正/反例的一种 GRPO 的退化，取缔掉 PPO 的优势函数，类似 GRPO 一样一致地为每个 token 赋 reward 值?

目前我的理解可能是，GRPO 量变带来了质变，探索了更多的 token 可能。如果像 PPO 采用 reward 一致赋给 token 的话，那么其实 PPO 只探索了两种可能的 token 类型，对于未知的 token 情况是不知道的，这非常坏，可能会影响泛化，所以我们构建 value function，理论上获知了一切的 token 情形下的结果；而 GRPO 则通过更多的，但是有限的路径覆盖了 token 参数空间，作为对 value function 无限可知的一种近似。