

0.1 RLHF

如何让语言模型的输出符合人类偏好，而不仅仅是流畅呢？DeepMind 将这个定义为“对齐问题”(alignment problem)。大语言模型输出的结果应该满足帮助性 Helpfulness、真实性 Honesty 以及无害性 Harmless 这个 3H 原则。虽然，也有基于其他方法的尝试，但是 OpenAI 和 Anthropic 已经为我们展示了人类反馈的强化学习 (Reinforcement Learning from Human Feedback, RLHF) 是一个有效的途径。**注意，SFT 仅能学习到正反馈，无法让模型学会 reject 有害或者虚假的回答。** [1] 发现，RLHF 微调后的模型，SFT 微调后的模型在回复多样性上好于 RLHF 微调。

0.1.1 强化学习与策略梯度

强化学习中最基本的 MDP(Markov decision process)，如下：

Algorithm 1 MDP 学习过程

```
1:  $t = 1$ ;
2: while True do
3:   agent 观察到系统状态  $s_t$ ;
4:   agent 根据策略  $\pi(s_t)$  执行动作  $a_t$ ;
5:   系统的状态以概率  $p(s_{t+1}|s_t, a_t)$  转移到  $s_{t+1}$ ;
6:   agent 收到回报  $r_t$ ;
7:    $t = t + 1$ ;
8: end while
```

强化学习的目标就是最大化累计回报 R ，策略 π 的累计回报 R^π 有两种定义：

$$\begin{aligned} R^\pi &= \lim_{T \rightarrow \infty} \frac{\mathbb{E}\left(\sum_{t=1}^T r_t | \pi\right)}{T}, && \text{平均回报} \\ R^\pi &= \lim_{T \rightarrow \infty} \mathbb{E}\left(\sum_{t=1}^T \gamma^{t-1} r_t | \pi\right), && \text{折扣回报, } 0 < \gamma < 1 \end{aligned} \tag{1}$$

强化学习有三大组件：

1. policy(策略): 分确定性策略 $\pi(s)$ (状态 s 下应该采取哪个动作) 和随机性策略两种 $\pi(a|s)$ (状态 s 下采取每个动作的概率)。
2. value function(值函数): 表示了 agent 根据策略 π 做动作时，在某个系统状态 s 下对的接下来累计回报的估计：

$$V^\pi(s) = \mathbb{E}(R^\pi | s, \pi) \tag{2}$$

，或者 agent 在某个系统状态 s 下采取某个动作 a 后对接下来累计回报的估计：

$$Q^\pi(s, a) = \mathbb{E}(R^\pi | s, a, \pi) \tag{3}$$

我们还常常使用优势函数:

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s) \quad (4)$$

3. model(模型): 包括两个矩阵, 一个是在状态 s 下采取动作 a 以后, 系统转移到状态 s' 的概率 $P(s'|s, a)$ 。另一个是, 在状态 s 下采取动作 a 所获得的回报的期望值 $R(s, a) = \mathbb{E}(r|s, a)$, 注意即便 s, a 确定后, 其回报也是随机的, 所以这里说的是期望。

强化学习有很多求解算法, 比如传统的查表算法 (采用动态规划列举出全部的 $Q(s, a)$), 然后策略为 $\pi(s) = \arg \max_a Q(s, a)$ 、DQN 类算法 (采用神经网络的方法拟合一个函数表示 $Q(s, a; \mathbf{w}) \approx Q(s, a)$) 等。我们这里关注其中一类叫做策略梯度的算法。策略梯度算法就是用一个带参函数 $\pi = \pi(a|s; \theta)$ 表示策略, 然后迭代地更新参数 θ 使得目标函数 $J(\pi; \theta)$ 最大。REINFORCE 算法 [2] 是一种比较早的策略梯度算法, 计算过程如 **Algorithm2**:

Algorithm 2 REINFORCE 算法

- 1: 初始化参数 θ_1 。
 - 2: **for** $k \in [1, K]$ **do**
 - 3: **for** $i \in [1, N]$ **do**
 - 4: 从系统初始状态开始, 运行策略 $\pi(a|s; \theta_k)$ 直到遇到终结状态, 得到第 i 条轨迹:
 $\tau^i = \{(s_0^i, a_0^i, r_0^i), (s_1^i, a_1^i, r_1^i), \dots, (s_T^i, a_T^i, r_T^i)\}$ 。
 - 5: **end for**
 - 6: 根据 N 个轨迹样本更新 θ 值: $\theta_{k+1} = \theta_k + \alpha \sum_i (\sum_t \nabla_\theta \log \pi(a_t^i | s_t^i; \theta)) (\sum_t r_t^i)$ 。
 - 7: **end for**
-

注意, 每条轨迹就是一个回合 (episode)。REINFORCE 算法优化的目标函数为:

$$J(\pi; \theta) = \mathbb{E}_{\tau \sim p(\tau; \theta)} R(\tau)$$

其中 $p(\tau; \theta)$ 表示在策略 $\pi(a|s; \theta)$ 下轨迹 τ 的概率, $R(\tau)$ 表示轨迹 τ 上的所有回报之和。也就是希望我们的策略在不同回合中平均累计回报最大。我们对该目标函数求导, 则有:

$$\begin{aligned} \nabla_\theta J(\pi; \theta) &= \nabla_\theta E_{\tau \sim p(\tau; \theta)} R(\tau) = \nabla_\theta \frac{1}{N} \sum_\tau p(\tau; \theta) R(\tau) \\ &= \frac{1}{N} \sum_\tau R(\tau) \nabla_\theta p(\tau; \theta) \\ &= \frac{1}{N} \sum_\tau R(\tau) p(\tau; \theta) \nabla_\theta \log p(\tau; \theta) \\ &= E_{\tau \sim p(\tau; \theta)} [R(\tau) \nabla_\theta \log p(\tau; \theta)] \end{aligned}$$

我们有:

$$\begin{aligned} \nabla_\theta \log p(\tau; \theta) &= \nabla_\theta \log (p(s_1) \prod_{t=1} \pi(a_t | s_t; \theta) p(s_{t+1} | a_t, s_t)) \\ &= \nabla_\theta \log p(s_1) + \sum_{t=1} \nabla_\theta \log \pi(a_t | s_t; \theta) + \sum_{t=1} \nabla_\theta \log p(s_{t+1} | a_t, s_t) \\ &= \sum_{t=1} \nabla_\theta \log \pi(a_t | s_t; \theta) \end{aligned}$$

我们这里认为稳态概率 $p(s_1)$ 和状态转移概率 $p(s_{t+1}|a_t, s_t)$ 是环境属性，跟策略无关。所以，最后我们就有：

$$\nabla_\theta J(\pi; \theta) \approx \frac{1}{N} \sum_i^N R(\tau^i) \sum_t \nabla_\theta \log \pi(a_t^i | s_t^i; \theta)$$

可以看到，我们可以用一个神经网络来表示策略 $\pi(a|s; \theta)$ ，并且我们的 loss 定义为：

$$loss = -\frac{1}{N} \sum_i^N R(\tau^i) \sum_t \log \pi(a_t^i | s_t^i; \theta)$$

现在就可以用自动微分求导数了。**小心，有可能出现 REINFORCE 算法中的梯度一直都是负数（策略概率和回报都是正数），所以我们可以对回报减去一个 baseline，然后梯度有正有负从而可以在更复杂的策略函数（比如神经网络）曲面上向最优点移动。**

我们现在介绍策略梯度算法的基础：策略梯度定理 [3]：

定理 0.1.1. (*Policy Gradient Theorem*) 对任意一个可微的带参策略 $\pi(a|s; \theta)$ ，基于平均回报或折扣回报的策略目标函数 $J(\pi; \theta)$ 有

$$\begin{aligned} \nabla_\theta J(\pi; \theta) &= \sum_{s \in S} d^\pi(s) \sum_{a \in A} \nabla_\theta \pi(a|s; \theta) Q^\pi(s, a) \\ &= \sum_{s \in S} \sum_{a \in A} d^\pi(s) \pi(a|s; \theta) \nabla_\theta \log \pi(a|s; \theta) Q^\pi(s, a) \\ &= \mathbb{E}_{(s, a) \sim d^\pi(s) \pi(a|s; \theta)} [\nabla_\theta \log \pi(a|s; \theta) Q^\pi(s, a)] \end{aligned} \quad (5)$$

其中， $d^\pi(s) = \sum_{t=1}^{\infty} \gamma^{t-1} P(s_t = s|\pi)$ ，也就是在策略 π 下状态 s 出现的加权稳态概率。对平均回报也类似，只不过 $d^\pi(s) = \lim_{T \rightarrow \infty} p(s_t = s|\pi)$ ，也即采用策略 π 的情况下状态 s 出现的稳态概率。策略梯度定理是策略梯度算法最核心的理论基础，由策略梯度定理可到，策略目标函数的梯度是一个概率期望，从而我们可以采样估计它，然后利用梯度下降算法更新参数，从而收敛到局部最优解。我们注意到策略梯度定理中需要求解 $Q^\pi(s, a)$ ，实践中，我们常常用一个带参函数进行近似

$$Q(s, a; w) \approx Q^\pi(s, a) \Rightarrow \nabla_\theta J(\pi; \theta) \approx \nabla_\theta \log \pi(a|s; \theta) Q(s, a; w)$$

这样我们就有 θ 和 w 两个参数需要估计。这也就是所谓的 actor-critic 算法，critic 负责更新 w ，actor 负责更新 θ 。

根据策略梯度定理， $\nabla_\theta J(\pi; \theta)$ 的一个无偏估计为： $\nabla_\theta \log \pi(a|s; \theta) Q^\pi(s, a)$ ，而：

$$\nabla_\theta \log \pi(a|s; \theta) (Q^\pi(s, a) - V^\pi(s)) = \nabla_\theta \log \pi(a|s; \theta) A^\pi(s, a)$$

也是其无偏估计且方差更小。在自动微分框架下，我们定义 loss 如下：

$$loss = \mathbb{E}_{(s, a) \sim d^\pi(s) \pi(a|s; \theta)} [\log \pi(a|s; \theta) A^\pi(s, a)]$$

也就是执行策略 π 得到一批样本 $(s_i, a_i), i = 1, 2, \dots, N$ ，然后计算：

$$loss = \frac{1}{N} \sum_i \log \pi(a_i | s_i; \theta) A^\pi(s_i, a_i)$$

接着 `loss.backward()` 即可求得到 θ 参数集合中各个参数的梯度。进一步： $\nabla_\theta \log \pi(a_t | s_t; \theta) (r_t - V(s_t))$ 也是 $\nabla_\theta J(\pi; \theta)$ 的一个无偏估计。

0.1.2 PPO 算法

先看下重要性采样的背景知识：假设我们要计算 $E_{x \sim p}[f(x)]$ ，于是我们从分布 $p(x)$ 中采样一批 x_i ，然后计算 $f(x_i)$ ，那么

$$\frac{1}{N} \sum_i^N f(x_i)$$

就是 $E_{x \sim p}[f(x)]$ 的近似值了。但是如果我们不能从分布 $p(x)$ 采样了，只能从分布 $q(x)$ 采样，该怎么办呢？看下面的公式：

$$E_{x \sim p}[f(x)] = \int f(x)p(x)dx = \int f(x)\frac{p(x)}{q(x)}q(x)dx = E_{x \sim q}[f(x)\frac{p(x)}{q(x)}]$$

所以，我们可以从 $q(x)$ 采样一批 x_i ，那么

$$\frac{1}{N} \sum_i^N f(x_i)\frac{p(x_i)}{q(x_i)}$$

也是 $E_{x \sim p}[f(x)]$ 的近似值。注意，二者的期望虽然相等，但是方差并不相等，如下：

$$\begin{aligned} Var_{x \sim p}[f(x)] &= E_{x \sim p}[f^2(x)] - (E_{x \sim p}[f(x)])^2 \\ Var_{x \sim q}[f(x)\frac{p(x)}{q(x)}] &= E_{x \sim q}[f(x)\frac{p(x)}{q(x)}f(x)\frac{p(x)}{q(x)}] - (E_{x \sim q}[f(x)\frac{p(x)}{q(x)}])^2 \\ &= E_{x \sim p}[f^2(x)\frac{p(x)}{q(x)}] - (E_{x \sim p}[f(x)])^2 \end{aligned}$$

可以看出，如果 $\frac{p(x)}{q(x)}$ 很大，那么二者的方差也会相差很大。

我们现在来介绍 PPO 算法 (proximal policy optimization) [4]。REINFORCE 算法²每次花很多时间做采样，然后只做一次参数更新，是低效的。所以，我们想有两个 actor: $\pi(a|s; \theta)$ 和 $\pi(a|s; \theta')$ ，用 $\pi(a|s; \theta')$ 去与环境互动做采样，然后用样本更新 $\pi(a|s; \theta)$ 的参数。根据式策略梯度定理和重要性采样，我们有：

$$\begin{aligned} \nabla_\theta J(\pi; \theta) &= \mathbb{E}_{(s,a) \sim \pi_\theta} [\nabla_\theta \log \pi(a|s; \theta) A^\pi(s, a)] && \text{策略梯度定理} \\ &= \mathbb{E}_{(s,a) \sim \pi'_\theta} \left[\frac{p(s,a;\theta)}{q(s,a;\theta')} \nabla_\theta \log \pi(a|s; \theta) A^\pi(s, a) \right] && \text{重要性采样} \\ &= \mathbb{E}_{(s,a) \sim \pi'_\theta} \left[\frac{\pi(a|s;\theta)p(s;\theta)}{\pi(a|s;\theta')p(s;\theta')} \nabla_\theta \log \pi(a|s; \theta) A^\pi(s, a) \right] \\ &= \mathbb{E}_{(s,a) \sim \pi'_\theta} \left[\frac{\pi(a|s;\theta)}{\pi(a|s;\theta')} \nabla_\theta \log \pi(a|s; \theta) A^\pi(s, a) \right] && \text{假设 } p(s; \theta') \text{ 和 } p(s; \theta) \text{ 相等} \\ &= \mathbb{E}_{(s,a) \sim \pi'_\theta} \left[\frac{\pi(a|s;\theta)}{\pi(a|s;\theta')} \nabla_\theta \log \pi(a|s; \theta) A^{\pi'}(s, a) \right] && \text{假设 } A^\pi(s, a) \text{ 和 } A^{\pi'}(s, a) \text{ 相等} \end{aligned}$$

其中， $(s, a) \sim \pi_\theta$ 表示执行策略 $\pi(a|s; \theta)$ 采样得到的样本， $(s, a) \sim \pi'_\theta$ 表示执行策略 $\pi(a|s; \theta')$ 采样得到的样本。 $p(s, a; \theta)$ 是执行策略 $\pi(a|s; \theta)$ 下 (s, a) 的出现概率， $q(s, a; \theta')$ 是执行策略 $\pi(a|s; \theta')$ 下 (s, a) 的出现概率。

接下来我们从梯度反推目标函数。在神经网络或者自动微分框架下，当我们的 objective 定义为：

$$L = \mathbb{E}_{(s,a) \sim \pi'_\theta} \left[\frac{\pi(a|s;\theta)}{\pi(a|s;\theta')} A^{\pi'}(s, a) \right]$$

那么我们可以得到：

$$\nabla_{\theta} L = \mathbb{E}_{(s,a) \sim \pi'_{\theta}} \left[\frac{\pi(a|s; \theta)}{\pi(a|s; \theta')} A^{\pi'}(s, a) \nabla_{\theta} \log \pi(a|s; \theta) \right]$$

因为 $\nabla_{\theta} \pi(a|s; \theta) = \pi(a|s; \theta) \nabla_{\theta} \log \pi(a|s; \theta)$ 。所以我们的 loss 定义为 $-L$ ，然后让自动微分框架按正常的逻辑求导数即可。到此，我们可以总结为：用策略 $\pi(a|s; \theta')$ 与环境互动过，从而得到大量样本，然后根据这些样本计算 loss 为 $-L$ ，接着用 pytorch 算梯度并更新策略 $\pi(a|s; \theta)$ 。

为了防止两个策略差异过大导致方差过大，可以在 *objective* 定义为加入了惩罚项：

$$L = \mathbb{E}_{(s,a) \sim \pi'_{\theta}} \left[\frac{\pi(a|s; \theta)}{\pi(a|s; \theta')} A^{\pi'}(s, a) \right] - \beta KL(\pi(a|s; \theta), \pi(a|s; \theta'))$$

而 PPO 提出了如下的目标函数 (这个算法也被称为 PPO2 或者 PPO-Clip，实验发现这比加入 KL 惩罚项 (PPO1，也称 PPO-Penalty) 效果更好，)：

$$L^{CLIP} = \mathbb{E}_{(s,a) \sim \pi'_{\theta}} \left[\min \left\{ \frac{\pi(a|s; \theta)}{\pi(a|s; \theta')}, \text{clip}\left(\frac{\pi(a|s; \theta)}{\pi(a|s; \theta')}, 1 - \epsilon, 1 + \epsilon\right) \right\} A^{\pi'}(s, a) \right]$$

其中 clip 是限制 $\pi(a|s; \theta) / \pi(a|s; \theta')$ 相差不要过大，太大的话就裁剪。在 PPO 论文中发现 $\epsilon = 0.2$ 取得了最好的效果。PPO 的算法过程如 Algorithm 3：

Algorithm 3 PPO 算法

- 1: 初始化策略网络的参数 θ_0 ，初始化值网络的参数 ϕ_0 ，
- 2: **for** $k \in [0, 1, 2, \dots]$ **do**
- 3: 从系统初始状态开始，运行策略 $\pi(a|s; \theta_k)$ 直到遇到终结状态，得到很多条轨迹

$$\{\tau^i = \{(s_t, a_t, r_t)\}\}$$

- 4: 计算每条轨迹中每个动作-状态对的优势函数值：

$$A_t = A(s_t, a_t; \phi_k) = \sum_{t' > t} \gamma^{t'-t} r_t - V(s_t; \phi_k)$$

- 5: 更新策略网络的参数。loss 为：

$$L^{CLIP} = \sum_{\tau^i} \sum_t \left[\min \left\{ \frac{\pi(a_t|s_t; \theta_k)}{\pi(a_t|s_t; \theta_{old})}, \text{clip}\left(\frac{\pi(a_t|s_t; \theta_k)}{\pi(a_t|s_t; \theta_{old})}, 1 - \epsilon, 1 + \epsilon\right) \right\} A(s_t, a_t; \phi) \right]$$

采用梯度更新： $\theta_{k+1} = \arg \max_{\theta} L^{CLIP}$ 。

- 6: 基于 MSE loss 更新值网络的参数，更新完成后得到 ϕ_{k+1} 。

$$\phi_{k+1} = \arg \min_{\phi} \sum_{\tau^i} \left(\sum_{t' > t} \gamma^{t'-t} r_t - V(s_t; \phi_k) \right)^2$$

- 7: 更新 θ_{old} 。

- 8: **end for**
-

以前，人们认为用强化学习训练 LM 是不可能的。而 PPO 被发现可行。PPO 目前是 OpenAI

默认的强化学习算法。GAE(Generalized Advantage Estimation) [5] 对优势函数采取如下的估计方法：

$$\hat{A}_t^{GAE} = \hat{A}^{GAE}(s_t, a_t) = \sum_{l=0}^{\infty} (\gamma\lambda)^l (r_{t+l} + \gamma V(s_{t+l+1}) - V(s_{t+l}))$$

$\lambda = 0$ 对应高偏差， $\lambda = 1$ 对应高方差。

0.1.3 RLHF 微调

强化学习中，agent 只会学习如何最大(折扣)累积回报 R^π ，如果想让它完成某些指定任务，就必须保证我们设计的 reward 可以使得智能体最大化 R^π 的同时也能实现我们的目标。RLHF [6] 根据人类反馈的偏好来学习一个好的 reward 函数，而非由环境给到 reward 或者由人工设计精巧(well-specified)的 reward。只需要人类评判一个行为的好坏，而无需专家演示大量优秀示例(这是 imitation learning 和 Inverse Reinforcement Learning 的思想)，因为很多时候人类也不是轻易知道该怎么做才是优秀的(比如开飞机)，但很多时候人类很容易知道一系列行为导致的最终结果到底好不好。此外，让人类比较两套动作最后导致结果哪个好，远比给每套动作打个绝对值的分更容易。我们需要学习一个 reward 函数： $r(s, a; w)$ 。策略函数 $a = \pi(s; \theta)$ 和 reward 函数 $r(s, a; w)$ 都用深度神经网络表示。整个训练过程如下：

- Optimizing the Policy. $\pi(s; \theta)$ 跟环境互动，产生多条轨迹 τ^i 。然后网络 $\pi(s; \theta)$ 的参数通过强化学习更新以最大化下面的值：

$$\sum_i \sum_{(s_j^i, a_j^i) \in \tau^i} r(s_j^i, a_j^i)$$

我们可以看到，此时的问题跟 REINFORCE 要求解的问题几乎一模一，唯一的问题在于 $r(s, a; w)$ 此时还在训练中，是不稳定的，所以可以采用 PPO [4] 算法来求解。

- Preference Elicitation. 从多条轨迹 τ^i 中选择多个 pair，让人类比较哪个更好，人类比较的结果为 (τ^1, τ^2, μ) ， μ 是一个大小为 2 的向量，取值有三种 $(1, 0), (0, 1), (0.5, 0.5)$ ，前两种取值表示 τ^1 或者 τ^2 更好，第三种表示两条轨迹差不多。
- Fitting the Reward Function. 我们先定义 τ^1 比 τ^2 好的概率为：

$$p(\tau^1 > \tau^2) = \frac{\exp \sum_{(s_j^1, a_j^1) \in \tau^1} r(s_j^1, a_j^1)}{\exp \sum_{(s_j^1, a_j^1) \in \tau^1} r(s_j^1, a_j^1) + \exp \sum_{(s_j^2, a_j^2) \in \tau^2} r(s_j^2, a_j^2)}$$

网络 $r(s, a)$ 的参数被更新去拟合人类的偏好，loss 如下：

$$-\sum_{(\tau^1, \tau^2, \mu) \in D} \mu_1 \log p(\tau^1 > \tau^2) + \mu_2 \log p(\tau^2 > \tau^1)$$

我们可以看到，每次要输入两条轨迹，然后计算每条轨迹中每个点的 reward $r(s_j, a_j)$ ，然后才可以更新 $r(s, a; w)$ 的参数。

以上三个步骤异步执行。

[6] 考虑的是 Atari games 和 simulated robotics tasks 两个场景，输入的是视频，输出的是动作(上下左右等)。而 [7] 利用 RLHF 对语言模型上进行微调并取得不错的效果。给定输入的 prompt x 和 4 个输出 y_0, y_1, y_2, y_3 (由已经预训练好的语言模型输出)，人类选择哪个输出最好，用下标 $b \in \{0, 1, 2, 3\}$ 表示。训练 reward 模型时的 loss 定义为：

$$\sum_x \log \frac{\exp r(x, y_b)}{\sum_j \exp r(x, y_j)}$$

reward 模型与 GPT-2 的网络结构一样，只不过在最后加了一个线性层，也就是：多层 GPT-2 的 decoder+FC，且 reward 模型的参数用 GPT-2 初始化(线性层采用随机初始化)。然后用 PPO 这种策略梯度算法更新 $\pi(x|y)$ 中的参数，但是此时 reward 定义如下：

$$R(x, y) = r(x, y) - \beta KL(\pi, p) = r(x, y) - \beta \log \frac{\pi(x|y)}{p(x|y)} \quad (6)$$

KL 是为了防止微调后的模型与原始模型差别太大。其中， $p(x|y)$ 是 GPT-2， $\pi(x|y)$ 是微调中的模型。[8] 以英语摘要作为评测任务，利用 RLHF 来微调语言模型。训练过程总共涉及到 4 个模型：

- 预训练好的模型，具体是 GPT-3 6B。
- 对 GPT-3 6B 做 SFT 后得到的模型 $\pi^{SFT}(y|x)$ 。这个模型后续被称为 Reference 模型。
- 用 SFT 后的模型加一个 FC 层来训练一个 reward 模型 $r_\theta(x, y)$ 。此时 loss 为：

$$-E_{(x, y_w, y_l) \in D} [\log \sigma(r_\theta(x, y_w) - r_\theta(x, y_l))]$$

其中，对于 x ， y_w 比 y_l 更好。

- RLHF 微调模型 $\pi^{RL}(y|x)$ (初始化为 $\pi^{SFT}(y|x)$)。训练这个模型时， $R(x, y)$ 的定义跟 [7] 一样，且也是用 PPO 算法训练。需要注意的是，PPO 中需要一个 Critic 网络，这个 Critic 初始化为 reward 模型，然后 PPO 算法中也会更新其参数。

整个过程见图1，具体说来如下(注意，这个过程可以反复进行)：

- step1. 从日志中采集 prompt。
- step2. 用预训练好的多个模型(包括上一轮 RLHF 微调后的模型 $\pi_k^{RL}(y|x)$)为每个 prompt x 的输出多个回答 y ，并选择多个回答的 pair 对儿。
- step3. 人类对回答的 pair 对儿标注哪个好，哪个差，得到很多样本 (x, y_w, y_l)
- step4. 用全部历史样本训练 reward 模型。
- step5. 用 RLHF 技术微调 $\pi_k^{RL}(y|x)$ 得到 $\pi_{k+1}^{RL}(y|x)$ ：收集一批新的 prompt x ， $\pi_k^{RL}(y|x)$ 输出回答，reward 模型计算 $r(x, y)$ ，然后通过 PPO 更新 $\pi_k^{RL}(y|x)$ 的参数。

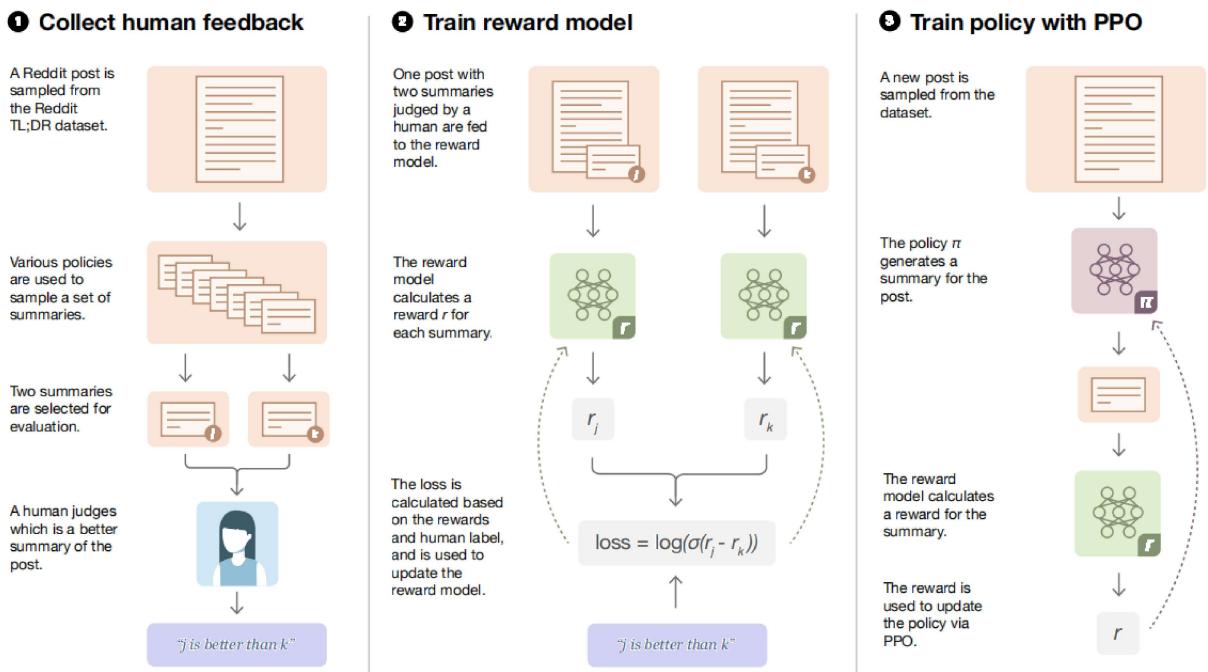


图 1: RLHF 微调过程示意图

[9] 使用如下 loss 函数来训练 reward model:

$$-\gamma E_{(x,y_w,y_l) \in D} [\log \sigma(r_\theta(x, y_w) - r_\theta(x, y_l))] + \beta E_{(x,y_w) \in D} [\log r'_\theta(x, y_w)]$$

这里 r' 除了最后一层，其他跟 r 一模一样且共享参数， r' 的最后一层输出的不是一个标量分，而是一个词表中每个 token 的概率， $r'_\theta(x, y_w)$ 便是输入 x 后输出 y_w 的概率，其实是一个语言模型。这样设计损失函数目的是要尽量生成 y_w ，因为 y_w 是个好的回答。

在使用 RLHF 微调时，每条训练数据都需要包含 prompt(提问)，chosen(正向回答) 和 rejected(负向回答)。

[10] 提出 RL4LMs 和 NLPO。Constitutional AI [11] 也是一项很重要的工作。

到目前有个问题：如果我们把输入的 prompt x 当做强化学习的 state，输出 y 当做 action，reward model 的输出当做 reward，一切看起来都很通畅，因为我们有 (s_i, a_i, r_i) 。但 y 真的是 action 吗？语言模型一般都是自回归的， y 不是一次性输出的，而是一个一个 token 地输出的，似乎每次输出 token 对应 action 更合理，但是这样一来，我们不知道每个 action 的 reward，因为我们目前要输出全部 token 才能算 reward。而且我们的轨迹是什么呢？难道就是由一个 point (s_i, a_i, r_i) 组成的吗？还有一个问题，是不是我们有了 reward model 了，PPO 中的 $V(s_t)$ 就可以不要了呢？毕竟我们都有最终的回报了，何必还要估计一个呢？而且如果输入的 prompt 是 s 的话， $V(s)$ 的意义是什么呢？因为输入根本不受 action 的影响，action 无法使得环境的状态发生迁移。这些点之前的论文中没有明确这几个问题。

按照 [12]，我们首先将用 RLHF 微调语言模型问题表述为传统强化学习范式：

- policy 就是我们的语言模型 $\pi(\mathbf{o}|\mathbf{c})$, 给到一个 prompt/context \mathbf{c} , 他输出一个由多个 token 组成的 answer \mathbf{o} 。注意, 输出是一个 token 一个 token 地输出, $o_t = \arg \max_o \pi(o|[\mathbf{c}, \mathbf{c}_{0:t-1}])$ 。
- state 是将输入的 prompt 和已经预测得到的 token concatenate 起来的 token 序列 $[\mathbf{c}, \mathbf{c}_{0:t-1}]$ 。初始状态是输入的 prompt \mathbf{o} 。
- action 是输出下一个 token。 t 时刻对应 answer 里的第 t 个 token o_t , s_t 表示 prompt+answer 中已经预测得到的 $t-1$ 个 token $[\mathbf{c}, \mathbf{c}_{0:t-1}]$, a_t 表示输出 o_t 这个 action。action space 对应词典大小, 大约 50K 左右。注意, 状态转移概率为:

$$p(s_{t+1}|s_t, a_t) = \begin{cases} 1 & \text{if } s_{t+1} = [\mathbf{c}, \mathbf{c}_{0:t}] \\ 0 & \text{else} \end{cases}$$

一个回合就是输出当前 \mathbf{c} 的全部 token。

- reward model 的输入是 prompt \mathbf{c} 和 answer \mathbf{o} , 输出一个向量, 其中第 t 个元素 $r(s_t, a_t)$ 表示到 answer 的第 t 个 token 这个时间点预测的未来收益。最后一个 token 对应的元素作为整个 answer 的 reward。

– 强化学习的目标便是最大化累积回报的期望值:

$$J = E_{\tau \sim \pi} \left[\sum_{t=0}^T r(s_t, a_t) \right]$$

, 也就是最大化回复句子的平均回报, 而这个回报是可以通过累加句子中每个 token 的即时回报得到。

- reward model 输出的 $r(s_t, a_t)$ 是一个即时回报, 也可以输出整个 answer 的回报, 但是不是所有回复句子的回报的均值。注意, 所有回复句子指的是训练和推理时可能输出所有回复。
- critic model 则是试图估算 $Q(s_t, a_t)$, 这表示当 $s_t = [\mathbf{c}, \mathbf{c}_{0:t-1}]$ 时输出下一个 token 后 J 的值。

给到 PPO 的 reward 依然按照式6来计算。reward model 和 critic model 共享全部参数 (其实也可以独立参数, 但是网络结构都几乎一样, 也就最后一层有差异)。二者的参数量通常比语言模型小一些 (比如 6B VS 175B)。Reward Model 在模型架构上一般是预训练的语言模型的最后一个非 embedding 层 (就是非 transformer decoder block) 换成一个全新的 FC 层。

参考文献

- [1] Robert Kirk, Ishita Mediratta, Christoforos Nalmpantis, Jelena Luketina, Eric Hambro, Edward Grefenstette, and Roberta Raileanu. Understanding the effects of rlhf on llm generalisation and diversity, 2024.
- [2] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, May 1992.
- [3] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS’99, pages 1057–1063, Cambridge, MA, USA, 1999. MIT Press.
- [4] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- [5] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation, 2018.
- [6] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.
- [7] Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.
- [8] Nisan Stiennon, Long Ouyang, Jeff Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul Christiano. Learning to summarize from human feedback, 2022.
- [9] Amanda Askell, Yuntao Bai, Anna Chen, Dawn Drain, Deep Ganguli, Tom Henighan, Andy Jones, Nicholas Joseph, Ben Mann, Nova DasSarma, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Jackson Kernion, Kamal Ndousse, Catherine Olsson, Dario

Amodei, Tom Brown, Jack Clark, Sam McCandlish, Chris Olah, and Jared Kaplan. A general language assistant as a laboratory for alignment, 2021.

- [10] Rajkumar Ramamurthy, Prithviraj Ammanabrolu, Kianté Brantley, Jack Hessel, Rafet Sifa, Christian Bauckhage, Hannaneh Hajishirzi, and Yejin Choi. Is reinforcement learning (not) for natural language processing: Benchmarks, baselines, and building blocks for natural language policy optimization, 2023.
- [11] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson, Ethan Perez, Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua Landau, Kamal Ndousse, Kamile Lukosuite, Liane Lovitt, Michael Sellitto, Nelson Elhage, Nicholas Schiefer, Noemi Mercado, Nova DasSarma, Robert Lasenby, Robin Larson, Sam Ringer, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Tamera Lanham, Timothy Telleen-Lawton, Tom Conerly, Tom Henighan, Tristan Hume, Samuel R. Bowman, Zac Hatfield-Dodds, Ben Mann, Dario Amodei, Nicholas Joseph, Sam McCandlish, Tom Brown, and Jared Kaplan. Constitutional ai: Harmlessness from ai feedback, 2022.
- [12] Shreyas Chaudhari, Pranjal Aggarwal, Vishvak Murahari, Tanmay Rajpurohit, Ashwin Kalyan, Karthik Narasimhan, Ameet Deshpande, and Bruno Castro da Silva. Rlhf deciphered: A critical analysis of reinforcement learning from human feedback for llms, 2024.