第 8 章　环境监督与强化学习

智能体适应环境的基本方式

类比生物体在“环境”中的不断学习从而适应环境的过程，智能体也需要在“环境”的监督之下，进行强化学习，以适应环境。

1.1.4 节已经对环境监督和强化学习的一般概念作了一个简要的介绍，其中图 1.4 给出了“模型”和“环境”之间的交互。本章将遵从一般的约定，称“模型”为“智能体”，以更好体现这种学习方式的独特之处。具体来讲，本章将紧接上一章，围绕 ChatGPT 采用的 PPO 模型，探讨环境监督和强化学习。

8.1　ChatGPT 的三阶段训练流程

基于第 7 章介绍的，采用“自监督”方式预训练的大语言模型 GPT-3.5，ChatGPT 首先针对问答系统进行“有监督”的精调，以实现从预训练模型到下游任务的迁移。接下来，采用人类反馈（即“环境监督”）的方式，训练一个奖励模型。最后，基于奖励模型，采用 PPO “强化学习”算法进行训练。如图 8.1 所示。

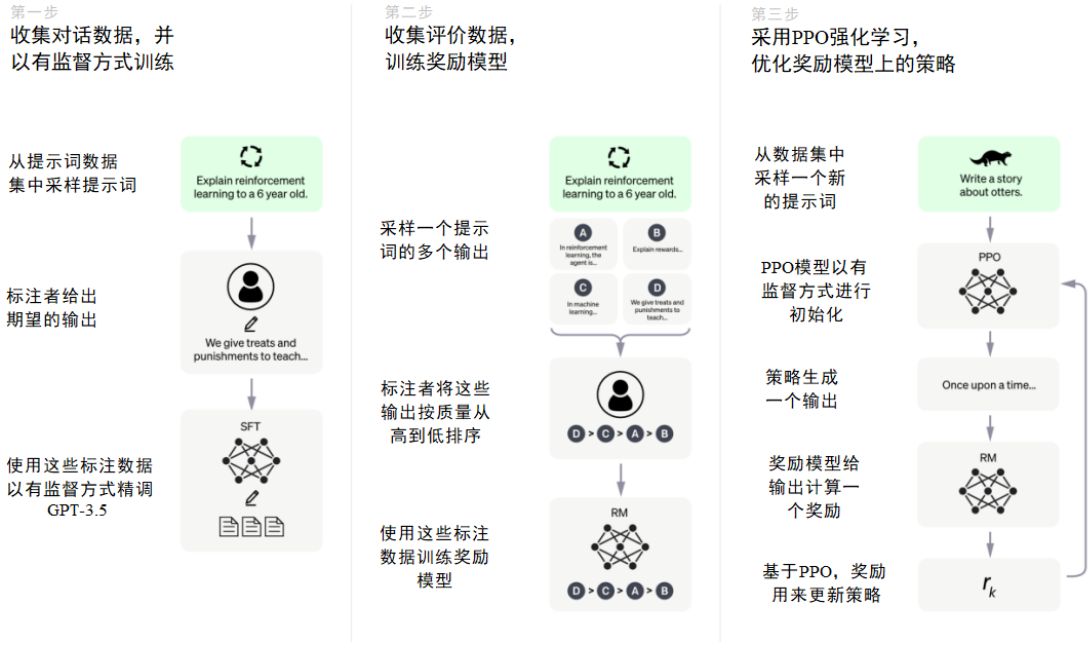


图 8.1 ChatGPT 的三阶段训练流程

有监督前面已经讨论得比较多，本章重点探讨后面两个阶段：环境监督与强化学习。

8.2　强化学习的形式化

由 1.1.4 节约定的符号，环境监督与强化学习的目标就是选择一个策略$\pi_{\theta}(a_t|s_t)$，并依据这个策略在环境中采取行动，从而最大化“总奖励”（称其为“期望回报”）。用公式可以

写为：

$$p(\tau|\pi) = \rho_0(\boldsymbol{s}_0)\prod_{t=0}^{T-1} p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t,\boldsymbol{a}_t)\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t|\boldsymbol{s}_t) \quad (8.1)$$

$$J(\pi) = \int_{\tau} p(\tau|\pi)R(\tau) = \mathop{\mathrm{E}}_{\tau\sim\pi}[R(\tau)] \quad (8.2)$$

$$\pi^* = \arg\max_{\pi} J(\pi) \quad (8.3)$$

下面对式（8.1）~（8.3）进行详细说明。

（1）策略$\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t|\boldsymbol{s}_t)$是关于参数$\boldsymbol{\theta}$的含参分布，典型地可以用一个全连接多层神经网络来表达。

（2）状态转移分布$p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t,\boldsymbol{a}_t)$表示智能体采取动作$\boldsymbol{a}_t$，从状态$\boldsymbol{s}_t$转移到状态$\boldsymbol{s}_{t+1}$。

（3）从初始状态分布$\rho_0(\boldsymbol{s}_0)$中采样得到初始状态$\boldsymbol{s}_0$。

（4）轨迹$\tau = (\boldsymbol{s}_0,\boldsymbol{a}_0,\boldsymbol{s}_1,\boldsymbol{a}_1,\dots\dots)$，从而$p(\tau|\pi)$表示基于策略$\pi$形成轨迹$\tau$的概率分布。

（5）$R(\tau)$表示轨迹τ的奖励，一般可表达为$R(\tau) = \sum_{t=0}^{\infty}\gamma^t r_t$，其中$r_t = r(\boldsymbol{s}_t,\boldsymbol{a}_t)$是奖励函数，$\gamma \in (0,1)$是折扣因子。引入折扣因子$\gamma$主要出于两方面的考虑：一方面，直觉上我们更关注时间上更近的奖励；另一方面，为了级数（无穷项求和）的收敛性。

（6）$J(\pi)$就是基于策略π的“总奖励”，称其为“期望回报”。

（7）最优策略π^*就是取得最大期望回报$J(\pi)$的策略。

8.3 策略最优化算法

为了找到最优策略π^*，基本的方法就是采用 2.3 节介绍的梯度上升，可用公式写为：

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha\nabla_{\boldsymbol{\theta}}J(\pi_{\boldsymbol{\theta}})|_{\boldsymbol{\theta}_k} \quad (8.4)$$

其中，期望回报$J(\pi_{\boldsymbol{\theta}})$的梯度$\nabla_{\boldsymbol{\theta}}J(\pi_{\boldsymbol{\theta}})$称为策略梯度，因此这种优化策略的方式被称为“策略梯度算法”。

下面对策略梯度的表达式进行推导：

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}}J(\pi_{\boldsymbol{\theta}}) &= \nabla_{\boldsymbol{\theta}}\mathop{\mathrm{E}}_{\tau\sim\pi_{\boldsymbol{\theta}}}[R(\tau)] \\
&= \nabla_{\boldsymbol{\theta}}\int_{\tau} p(\tau|\pi_{\boldsymbol{\theta}})R(\tau) \\
&= \int_{\tau}\nabla_{\boldsymbol{\theta}}p(\tau|\pi_{\boldsymbol{\theta}})R(\tau) \\
&= \int_{\tau} p(\tau|\pi_{\boldsymbol{\theta}})\nabla_{\boldsymbol{\theta}}logp(\tau|\pi_{\boldsymbol{\theta}})R(\tau) \\
&= \mathop{\mathrm{E}}_{\tau\sim\pi_{\boldsymbol{\theta}}}[\nabla_{\boldsymbol{\theta}}logp(\tau|\pi_{\boldsymbol{\theta}})R(\tau)] \\
&= \mathop{\mathrm{E}}_{\tau\sim\pi_{\boldsymbol{\theta}}}[\textstyle\sum_{t=0}^{T}\nabla_{\boldsymbol{\theta}}log\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t|\boldsymbol{s}_t)R(\tau)]
\end{aligned} \quad (8.5)$$

推导过程用到了式（8.1）~（8.3）。第三行到第四行用了一个恒等变换：$\nabla_{\boldsymbol{\theta}}p(\tau|\pi_{\boldsymbol{\theta}}) = p(\tau|\pi_{\boldsymbol{\theta}})\nabla_{\boldsymbol{\theta}}logp(\tau|\pi_{\boldsymbol{\theta}})$。第五行到第六行是把式（8.1）代入，并且注意到$\rho_0(\boldsymbol{s}_0)$、$p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t,\boldsymbol{a}_t)$都与参数$\boldsymbol{\theta}$无关，从而求梯度只剩下$\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t|\boldsymbol{s}_t)$这一项。

由式（8.5）可知，策略梯度$\nabla_{\boldsymbol{\theta}}J(\pi_{\boldsymbol{\theta}})$是关于轨迹$\tau$的期望，因此可以通过样本均值来进行估计。如果让智能体采用策略$\pi_{\boldsymbol{\theta}}$在环境中行动，就可以得到一系列轨迹，记为$D = \{\tau_i\}_{i=1\dots N}$，由此可以对策略梯度估计如下：

$$\hat{g} = \frac{1}{|D|}\sum_{\tau \in D}\sum_{t=0}^{T}\nabla_{\boldsymbol{\theta}} log\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t|\boldsymbol{s}_t)R(\tau) \tag{8.6}$$

其中，$|D|$为D中轨迹数量（此处为N）。

只要知道策略$\pi_{\boldsymbol{\theta}}$的表达式，并且可以计算$\nabla_{\boldsymbol{\theta}} log\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t|\boldsymbol{s}_t)$，式（8.6）就可以基于收集到的轨迹集合$D$计算出来，然后根据式（8.4）对参数$\boldsymbol{\theta}$进行更新。

8.3.1 事后奖励

式（8.5）存在一个问题：不管当前t取何值，轨迹τ的奖励$R(\tau) = \sum_{t=0}^{T}\gamma^t r_t$都会把$t$从 0 到$T$所有动作$\boldsymbol{a}_t$的奖励$r_t$累加起来。显然，这是不合理的，因为动作是“因”，奖励是“果”，先有“因”后有“果”，过去的奖励不能算在当前动作的头上。由此，式（8.5）应该修改为：

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \mathop{\mathrm{E}}_{\tau \sim \pi_{\boldsymbol{\theta}}}\left[\sum_{t=0}^{T}\nabla_{\boldsymbol{\theta}} log\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t|\boldsymbol{s}_t)\sum_{t'=t}^{T}R(\boldsymbol{s}_{t'}, \boldsymbol{a}_{t'}, \boldsymbol{s}_{t'+1})\right] \tag{8.7}$$

式（8.7）被称为“事后奖励策略梯度”。为了方便，将事后奖励$\sum_{t'=t}^{T}R(\boldsymbol{s}_{t'}, \boldsymbol{a}_{t'}, \boldsymbol{s}_{t'+1})$记为$\widehat{R_t}$。

8.3.2 基于优势函数的策略梯度

首先引入一个结论：

$$\mathop{\mathrm{E}}_{x \sim p_{\boldsymbol{\theta}}}[\nabla_{\boldsymbol{\theta}} log p_{\boldsymbol{\theta}}(x)] = 0 \tag{8.8}$$

这个结论的证明比较简单，用到概率的归一化和前面证明式（8.5）时使用的恒等变换，具体证明过程留给读者自己来完成。

由式（8.8）可知：

$$\mathop{\mathrm{E}}_{a_t \sim \pi_{\boldsymbol{\theta}}}[\nabla_{\boldsymbol{\theta}} log\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t|\boldsymbol{s}_t)b(\boldsymbol{s}_t)] = 0 \tag{8.9}$$

因为$\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t|\boldsymbol{s}_t)$是一个概率分布，而函数$b(\boldsymbol{s}_t)$仅依赖于状态。

由此，式（8.7）可以等价地写为：

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \mathop{\mathrm{E}}_{\tau \sim \pi_{\boldsymbol{\theta}}}\left[\sum_{t=0}^{T}\nabla_{\boldsymbol{\theta}} log\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t|\boldsymbol{s}_t)(\sum_{t'=t}^{T}R(\boldsymbol{s}_{t'}, \boldsymbol{a}_{t'}, \boldsymbol{s}_{t'+1}) - b(\boldsymbol{s}_t))\right] \tag{8.10}$$

因为加上或减去一项只依赖于状态的项并不影响最终的期望值。类似这样使用的任意函数$b(\boldsymbol{s}_t)$被称为“基线”（表示“起点”的意思）。

最常用的基线是如下函数：

$$V^{\pi}(\boldsymbol{s}_t) = \mathop{\mathrm{E}}_{\tau \sim \pi}[R(\tau)|\boldsymbol{s}_0 = \boldsymbol{s}_t] \tag{8.11}$$

即从状态$\boldsymbol{s}_t$开始基于策略π行动而得到的期望回报。将式（8.11）作为事后奖励$\widehat{R_t} = \sum_{t'=t}^{T}R(\boldsymbol{s}_{t'}, \boldsymbol{a}_{t'}, \boldsymbol{s}_{t'+1})$的起点是符合直觉的。

进一步，可以将式（8.10）中的事后奖励$\widehat{R_t}$这一项替换为：

$$Q^{\pi}(\boldsymbol{s}_t, \boldsymbol{a}_t) = \mathop{\mathrm{E}}_{\tau \sim \pi}[R(\tau)|\boldsymbol{s}_0 = \boldsymbol{s}_t, \boldsymbol{a}_0 = \boldsymbol{a}_t] \tag{8.12}$$

注意关键的区别：式（8.12）中起始动作$\boldsymbol{a}_0 = \boldsymbol{a}_t$是任意选取的，之后才是根据策略$\pi$来行动。

定义$A^{\pi}(\boldsymbol{s}_t, \boldsymbol{a}_t) = Q^{\pi}(\boldsymbol{s}_t, \boldsymbol{a}_t) - V^{\pi}(\boldsymbol{s}_t)$为“优势函数”，即在状态$\boldsymbol{s}_t$任意选取一个起始动作$\boldsymbol{a}_t$，得到的相对于基线的期望回报“优势”。

由此，最终得到基于优势函数的策略梯度：

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \mathop{\mathrm{E}}_{\tau \sim \pi_{\boldsymbol{\theta}}}[\sum_{t=0}^{T}\nabla_{\boldsymbol{\theta}} log\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t|\boldsymbol{s}_t)A^{\pi}(\boldsymbol{s}_t, \boldsymbol{a}_t)] \tag{8.13}$$

补充一点。类似用一个全连接多层神经网络来表达策略函数$\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t|\boldsymbol{s}_t)$，同样可以用一个全连接多层神经网络$V_{\boldsymbol{\varphi}}(\boldsymbol{s}_t)$来逼近基线$V^{\pi}(\boldsymbol{s}_t)$，然后类似 2.5 节采用均方差损失函数和随机梯度下降来训练$V_{\boldsymbol{\varphi}}(\boldsymbol{s}_t)$，即：

$$\boldsymbol{\varphi}_k = \underset{\boldsymbol{\varphi}}{\operatorname{argmin}} \underset{\boldsymbol{s}_t,\widehat{R_t}\sim\pi_k}{\mathrm{E}} [(V_{\boldsymbol{\varphi}}(\boldsymbol{s}_t) - \widehat{R_t})^2] \tag{8.14}$$

其中，π_k表示第 k 轮训练的策略。

算法 1 给出了完整的基于优势函数的策略梯度算法。

算法 1：基于优势函数的策略梯度算法

1：输入：初始策略函数参数$\boldsymbol{\theta}_0$，初始基线函数参数$\boldsymbol{\varphi}_0$

2：for k=0,1,2,… do

3：　在环境中运行策略$\pi_k = \pi(\boldsymbol{\theta}_k)$，从而收集轨迹集合$D_k = \{\tau_i\}_{i=1\ldots N}$

4：　计算事后奖励$\widehat{R_t}$

5：　基于基线$V_{\boldsymbol{\varphi}_k}$计算优势函数的估计值$\widehat{A_t}$

6：　估计策略梯度：

$$\hat{g}_k = \frac{1}{|D_k|}\sum_{\tau\in D_k}\sum_{t=0}^{T}\nabla_{\boldsymbol{\theta}} log\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t|\boldsymbol{s}_t)|_{\boldsymbol{\theta}_k}\widehat{A_t}$$

7：　更新策略参数：

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha_k\hat{g}_k$$

8：　基于均方差损失函数回归基线函数参数$\boldsymbol{\varphi}$：

$$\boldsymbol{\varphi}_{k+1} = \underset{\boldsymbol{\varphi}}{\operatorname{argmin}}\frac{1}{|D_k|T}\sum_{\tau\in D_k}\sum_{t=0}^{T}(V_{\boldsymbol{\varphi}}(\boldsymbol{s}_t) - \widehat{R_t})^2$$

9：end for

8.3.3 近端策略最优化

近端策略最优化（Proximal Policy Optimization）是一个一阶方法，实现较为简单。该方法在每一步更新中，一方面最大限度改进策略；另一方面确保新策略和老策略比较接近，以避免策略发散而导致的性能坍塌。

有两种主要的 PPO 算法：惩罚式与裁剪式。所谓惩罚式 PPO 是指，在目标函数里增加相应的惩罚项（也就是正则项），并且在训练过程中自动调整惩罚系数，从而避免策略发散。裁剪式 PPO 不采用惩罚项，而是采用带有特别裁剪方式的目标函数，从而确保新策略和老策略比较接近。ChatGPT 采用的是后者，因此下面详细介绍它。

裁剪式 PPO 的损失函数为：

$$L(\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta}_k,\boldsymbol{\theta}) = \min\left(\frac{\pi_{\theta}(\boldsymbol{a}|\boldsymbol{s})}{\pi_{\theta_k}(\boldsymbol{a}|\boldsymbol{s})}A^{\pi_{\theta_k}}(\boldsymbol{s},\boldsymbol{a}), g(\epsilon, A^{\pi_{\theta_k}}(\boldsymbol{s},\boldsymbol{a}))\right) \tag{8.15}$$

其中：

$$g(\epsilon, A) = \begin{cases}(1+\epsilon)A, A \geq 0\\(1-\epsilon)A, A < 0\end{cases} \tag{8.16}$$

式（8.16）中的超参数ϵ是一个小的正数，用来控制新策略和老策略的接近程度。如果优势函数值$A^{\pi_{\theta_k}}(\boldsymbol{s},\boldsymbol{a})$为正，则式（8.15）成为：

$$L(\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta}_k,\boldsymbol{\theta}) = \min\left(\frac{\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})}{\pi_{\boldsymbol{\theta}_k}(\boldsymbol{a}|\boldsymbol{s})}, (1+\epsilon)\right) A^{\pi_{\boldsymbol{\theta}_k}}(\boldsymbol{s},\boldsymbol{a}) \tag{8.17}$$

由于$A^{\pi_{\boldsymbol{\theta}_k}}(\boldsymbol{s},\boldsymbol{a})$为正，那么如果策略$\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})$的概率值升高（意味着状态$\boldsymbol{s}$下采取行为$\boldsymbol{a}$的概率升高），则损失函数的值$L(\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta}_k,\boldsymbol{\theta})$也将增加。但是取最小值的操作min将确保，一旦$\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s}) > (1+\epsilon)\pi_{\boldsymbol{\theta}_k}(\boldsymbol{a}|\boldsymbol{s})$，则最大能取到的值就是$(1+\epsilon)A^{\pi_{\boldsymbol{\theta}_k}}(\boldsymbol{s},\boldsymbol{a})$，这就是所谓的“裁剪”。由此，新策略不会离老策略太远。类似地，可以分析$A^{\pi_{\boldsymbol{\theta}_k}}(\boldsymbol{s},\boldsymbol{a})$为负的情况，同样可以确保新策略不会离老策略太远。后面这种情况的分析留给读者自己来完成。

算法 2 给出了完整的裁剪式 PPO 算法。

算法 2：裁剪式 PPO 算法

1：输入：初始策略函数参数$\boldsymbol{\theta}_0$，初始基线函数参数$\boldsymbol{\varphi}_0$

2：for k=0,1,2,… do

3：　在环境中运行策略$\pi_k = \pi(\boldsymbol{\theta}_k)$，从而收集轨迹集合$D_k = \{\tau_i\}_{i=1\ldots N}$

4：　计算事后奖励$\widehat{R_t}$

5：　基于基线$V_{\boldsymbol{\varphi}_k}$计算优势函数的估计值$\widehat{A_t}$

6：　更新策略参数：

$$\boldsymbol{\theta}_{k+1} = arg\max_{\boldsymbol{\theta}} \frac{1}{|D_k|T} \sum_{\tau\in D_k}\sum_{t=0}^{T} \min\left(\frac{\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t|\boldsymbol{s}_t)}{\pi_{\boldsymbol{\theta}_k}(\boldsymbol{a}_t|\boldsymbol{s}_t)} A^{\pi_{\boldsymbol{\theta}_k}}(\boldsymbol{s}_t,\boldsymbol{a}_t), g(\epsilon, A^{\pi_{\boldsymbol{\theta}_k}}(\boldsymbol{s}_t,\boldsymbol{a}_t))\right)$$

7：　基于均方差损失函数回归基线函数参数$\boldsymbol{\varphi}$：

$$\boldsymbol{\varphi}_{k+1} = \operatorname*{argmin}_{\boldsymbol{\varphi}} \frac{1}{|D_k|T} \sum_{\tau\in D_k}\sum_{t=0}^{T} (V_{\boldsymbol{\varphi}}(\boldsymbol{s}_t) - \widehat{R_t})^2$$

9：end for

8.4 环境构建与训练奖励模型

8.2 和 8.3 节探讨了强化学习，本节探讨环境监督部分，对应图 8.1 中的第二阶段。

由 8.2 和 8.3 节的探讨可知，“环境”提供的奖励$r_t = r(\boldsymbol{s}_t,\boldsymbol{a}_t)$是强化学习的最基本要素，但是对于 ChatGPT 这一类以聊天方式呈现的智能体，还需要人为构建对话“环境”，训练一个符合人类价值观的奖励模型$r_t = r(\boldsymbol{s}_t,\boldsymbol{a}_t)$。有了这个奖励模型，环境监督与强化学习就可以最终得以实现，这就是所谓的“对齐”过程，即基于人类反馈的强化学习（RLHF）。

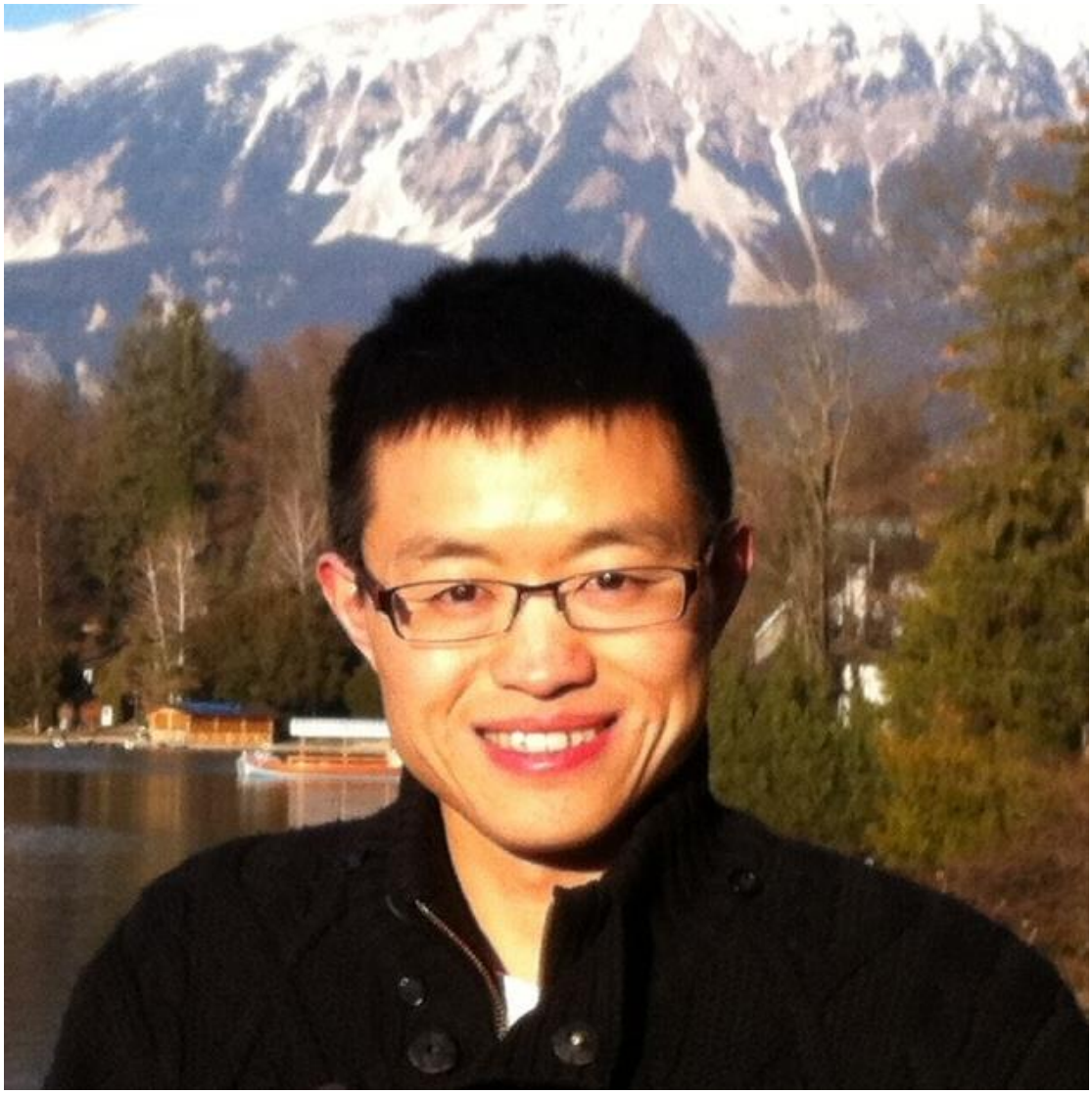
具体来讲，首先是构建对话“环境”，即建立有监督训练数据集D；然后，以梯度下降方式训练一个基于神经网络的奖励模型$r_{\boldsymbol{\theta}}$。训练集D中每个样本，由聊天模型（图 8.1 中的第一阶段有监督训练得到的模型）对同一个输入$\boldsymbol{x}$的两个回答$\boldsymbol{y} = \{\boldsymbol{y}_0,\boldsymbol{y}_1\}$构成，人工会标定出这两个回答哪一个更好。接下来，样本输入到奖励模型中，得到同一个输入$\boldsymbol{x}$的两个回答的奖励$r_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{y}_0)$和$r_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{y}_1)$。设回答$\boldsymbol{y}_i$更好，则如下计算损失：

$$L(r_{\boldsymbol{\theta}}) = -\mathop{\mathrm{E}}_{(\boldsymbol{x},\boldsymbol{y}_0,\boldsymbol{y}_1,i)\sim D}[\log(\sigma(r_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{y}_i) - r_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{y}_{1-i})))] \tag{8.18}$$

其中，$\sigma(x)$为对数几率函数（见式(2.3.4)）。如 2.3.2 节所述，对数几率函数把实数域的输入“挤压”到(0,1)的输出范围内。这样，如果$r_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{y}_i) - r_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{y}_{1-i})$为正，则$\sigma(x)$的值在$(0.5,1)$之间，取对数再取反后就在$(0,-log0.5)$之间，$r_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{y}_i) - r_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{y}_{1-i})$越大（意味着奖励越符合预期）则损失越小；如果$r_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{y}_i) - r_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{y}_{1-i})$为负，则可知损失函数值在$(-log0.5,+\infty)$

之间，$r_{\theta}(x, y_i) - r_{\theta}(x, y_{1-i})$越小（意味着奖励越符不合预期）则损失越大。

小故事：国产 AI 基础模型 GLM



Jie Tang (Tang, Jie) 唐 杰

（清华大学计算机科学与技术系长聘教授，智源研究院副院长）

超大规模预训练模型（也称基础模型、大模型，英文为 Foundation Model、Big Model、Large Model 等）快速发展，成为国际人工智能领域研究和应用的前沿焦点。OpenAI ChatGPT 和 Sora 的推出引发了社会和公众的广泛关注，并引起了大模型是否会引发新一轮行业变革甚至新一次工业革命的讨论。

大模型作为 ChatGPT 和 Sora 等生成式人工智能技术产品的核心技术基座，正在快速改变产业格局，孕育出全新的用户交互模式，形成舆论引导、社会治理、信息服务等方面的不对称优势。大模型也被认为是通向通用人工智能(Artificial General Intelligence，AGI)的重要途径之一，成为各国人工智能发展的新方向，正在成为新一代人工智能的基础设施。人工智能大模型已成为国际科技“必争之地”，实现国产全自研、自主可控的人工智能基础模型迫在眉睫。

当前，我国人工智能基础模型研究、应用与产业化发展正处于“从模仿追赶迈向创新引

领”的关键时期。大模型的快速发展给全球科技创新带来全新挑战：超大算力需求、超大规模数据需求、全新模型训练算法与框架、安全可信的软硬件系统。同时大模型的应用需求也更加动态多样，要求对大模型的不同层次进行深入研究。这是个全新的人工智能科学难题，也是我们赶超国际的机会。发展可媲美人类智能的人工智能系统已经成为人工智能领域研究的国际共识。

启迪：

（1）从模仿追赶迈向创新引领。

（2）实现国产全自研、自主可控的根本重要性。

（3）繁荣 AI 科技生态，推动应用落地和产业发展。