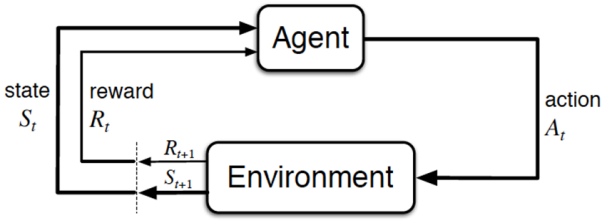
1 RL基础

强化学习(Reinforcement Learning，简称RL)，是指基于智能体在复杂、不确定的环境中最大化它能获得的奖励，从而达到自主决策的目的。

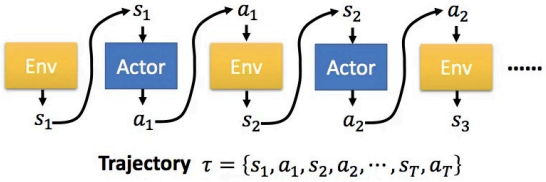


总的而言，Agent依据策略决策从而执行动作action，然后通过感知环境Environment从而获取环境的状态state，进而得到奖励reward(以便下次再到相同状态时能采取更优的动作)，然后再继续按此流程"依据策略执行动作-感知状态--得到奖励"循环进行。RL为得到最优策略从而获取最大化奖励。

- 基于策略的方法：先进行策略评估，即对当前已经搜索到的策略函数进行估值，得到估值后，进行策略改进，不断重复这两步直至策略收敛
- 基于值函数的方法：通过求解一个状态或者状态下某个动作的估值为手段，从而寻找最佳的价值函数，找到价值函数后，再提取最佳策略

1.1 Policy Gradient 算法

参数为 θ 的策略 π_θ 接受状态 s，输出动作概率分布，在动作概率分布中采样动作，执行动作，形成运动轨迹 τ，得到奖励 。



Trajectory $\tau = \{s_1, a_1, s_2, a_2, \cdots, s_T, a_T\}$

给定策略参数θ，可以计算某一条轨迹 τ 发生的概率：

$$\begin{aligned} p_\theta(\tau) &= p(s_1)p_\theta(a_1 \mid s_1)p(s_2 \mid s_1, a_1)p_\theta(a_2 \mid s_2)p(s_3 \mid s_2, a_2)\cdots \\ &= p(s_1)\prod_{t=1}^{T} p_\theta(a_t \mid s_t)p(s_{t+1} \mid s_t, a_t) \end{aligned}$$

评价策略就是看其因此得到的期望奖励：

$$\bar{R}_\theta = \sum_\tau R(\tau)p_\theta(\tau) = \mathbb{E}_{\tau\sim\theta(\tau)}[R(\tau)]$$

想让奖励越大越好，可以使用梯度上升来最大化期望奖励。先要计算期望奖励 $\bar{R}_\theta$ 的梯度：

$$\begin{aligned} \nabla\bar{R}_\theta &= \sum_\tau R(\tau)\nabla p_\theta(\tau) \\ &= \sum_\tau R(\tau)p_\theta(\tau)\frac{\nabla p_\theta(\tau)}{p_\theta(\tau)} \\ &= \sum_\tau R(\tau)p_\theta(\tau)\nabla\log p_\theta(\tau) \\ &= \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[R(\tau)\nabla\log p_\theta(\tau)\right] \\ &\approx \frac{1}{N}\sum_{n=1}^{N} R(\tau^n)\nabla\log p_\theta(\tau^n) \\ &= \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n} R(\tau^n)\nabla\log p_\theta(a_t^n \mid s_t^n) \end{aligned}$$

> $p_\theta(\tau)$ 中含有乘积项不好求，用$\nabla f(x) = f(x)\nabla\log f(x)$将其转换成累加项
>
> $$\begin{aligned} \nabla\log p_\theta(\tau) &= \nabla\left(\log p(s_1) + \sum_{t=1}^{T_n}\log p(s_{t+1} \mid s_t, a_t) + \sum_{t=1}^{T_n}\log p_\theta(a_t \mid s_t)\right) \\ &= \nabla\log p(s_1) + \nabla\sum_{t=1}^{T_n}\log p(s_{t+1} \mid s_t, a_t) + \nabla\sum_{t=1}^{T_n}\log p_\theta(a_t \mid s_t) \\ &= \nabla\sum_{t=1}^{T_n}\log p_\theta(a_t \mid s_t) \\ &= \sum_{t=1}^{T_n}\nabla\log p_\theta(a_t \mid s_t) \end{aligned}$$

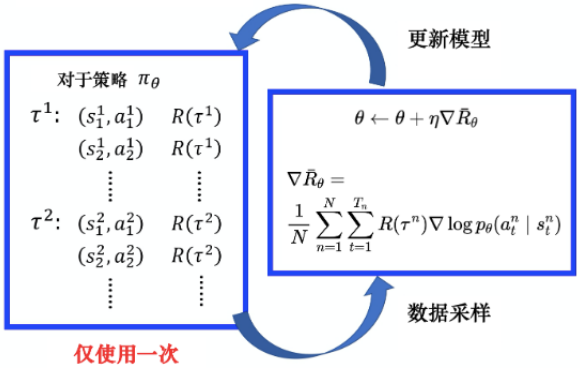
策略梯度 Policy Gradient 算法：

$$\nabla\bar{R}_\theta = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n} R(\tau^n)\nabla\log p_\theta(a_t^n \mid s_t^n)$$

用梯度上升来更新参数：$\theta \leftarrow \theta + \eta\nabla\bar{R}_\theta$。即：

$$J(\theta)=\frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}R\left(\tau^n\right)\log p_\theta\left(a_t^n\mid s_t^n\right)$$

PG方法在更新策略时，基本思想就是增加reward大的动作出现的概率，减小reward小的策略出现的概率。我们首先采集数据，然后基于前面得到的梯度提升的式子更新参数，随后再根据更新后的策略再采集数据，再更新参数，如此循环进行。一个PG方法的完整过程如下：



假设现在有一种情况，我们的reward在无论何时都是正的，对于没有采样到的动作，它的reward是0。因此，如果一个比较好的动作没有被采样到，而采样到的不好的动作得到了一个比较小的正reward，那么没有被采样到的好动作的出现概率会越来越小，这显然是不合适的，于是我们引入一个基线，让奖励有正有负，一般增加基线的方式是所有采样序列的奖励的平均值：

$$\nabla\bar{R}_\theta\approx\frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}\left(R\left(\tau^n\right)-b\right)\nabla\log p_\theta\left(a_t^n\mid s_t^n\right)$$

对于同一个采样序列中的数据点，我们使用相同的奖励 $R(\tau)$，这样的做法实在有点粗糙：

$$R(\tau^n)\Rightarrow\sum_{t'=t}^{T_n}r_{t'}^n\Rightarrow\sum_{t'=t}^{T_n}\gamma^{t'-t}r_{t'}^n$$

- γ 为折扣因子。就像买股票一样，同样一块钱，当前的一块钱比未来期望的一块钱更具有价值。因此在强化学习中，对未来的奖励需要进行一定的折扣

b 的选择有两种：

1. 使用轨迹上的奖励均值 $b=\mathbb{E}[R(\tau)]$
2. 通过critic来计算得到的状态价值函数 $V_\phi(s_t)$，它由一个结构与策略网络相同但参数不同的神经网络构成，主要是来拟合从 s_t 到终局的折扣奖励

优势函数(Advantage Function)：

$$A^\theta\left(s_t,a_t\right)=\sum_{t'>t}\gamma^{t'-t}r_{t'}-V_\phi\left(s_t\right)$$

Actor-Criti算法(演员-评论家算法)，Actor学习参数化的策略即策略函数，Criti学习值函数用来评估状态-动作对，然后根据评估结果改进或更新策略。在学习过程中，Critic试图减小预测的价值和实际经验回报之间的差距，以此来改进我们的策略。A^θ 前半部分是实际的采样折扣奖励，后半部分是拟合的折扣奖励。A^θ 表示了 s_t 下采取动作 a_t，实际得到的折扣奖励相对于模拟的折扣奖励下的优势，因为模拟的折扣奖励是在所有采集过的动作的折扣奖励的拟合（平均），因此这个优势函数也就代表了采用动作 a_t 相对于这些动作的平均优势，这个优势函数由一个critic(评价者)来给出。

既然是监督学习，我们对 $V_\phi(\cdot)$ 的训练就是对每一个数据序列中的每一个动作点的后续折扣奖励作为待学习的特征，来通过最小化预测和特征之间的误差来更新参数。

例如 s_t 在 n 个不同采样样本中分别选用了动作 $\alpha_1,\alpha_2,\cdots,\alpha_n$，分别得到折扣奖励（从 s_t 到终局）是 $\gamma_1,\gamma_2,\cdots,\gamma_n$。因为 $V_\phi(s_t)$ 是拟合折扣奖励，所以它表示了在 s_t 下得到的折扣奖励的期望，我们用 $\gamma_i,i=1,2,\cdots,n$，作为特征去拟合，拟合好后，$V_\phi\left(s_t\right)$ 就代表了 s_t的价值（或者说代表了其获得折扣奖励的期望）。那么 $A^\theta\left(s_t,a_t\right)$ 式就表示了 a_t 相对于$\alpha_1,\alpha_2,\cdots,\alpha_n$ 这些动作的平均优势，即 $A^\theta(s_t,a_t)$ 要估测的是在状态 s_t 采取动作 a_t 是好的还是不好的。

- 如果 $A^\theta(s_t,a_t)$ 是正的(即大于0)，意味着在状态 s_t 采取动作 a_t 获得的回报比在状态 s_t 采取任何其他可能的动作获得的回报都要好，就要增加概率
- 如果 $A^\theta(s_t,a_t)$ 是负的(即小于0)，意味着在状态 s_t 采取动作 a_t 得到的回报比其他动作的平均回报要差，要减少概率

Actor-Criti本质上是属于基于策略的算法，毕竟算法的目标是优化一个带参数的策略，只是会额外学习价值函数)，从而帮助策略函数更好的学习，而学习优势函数的Actor-Criti算法被称为Advantage Actor-Criti算法(优势演员-评论家算法，简称A2C)。

最终在更新梯度的时候，如下式所示『我们用演员 θ 去采样出状态跟动作的对(s_t,a_t)，计算这个状态跟动作对的优势 $A^\theta(s_t,a_t)$』

$$\nabla\bar{R}_\theta=\mathbb{E}_{(s_t,a_t)\sim\pi_\theta}\left[A^\theta\left(s_t,a_t\right)\nabla\log p_\theta\left(a_t\mid s_t\right)\right]$$

1.2 PPO算法

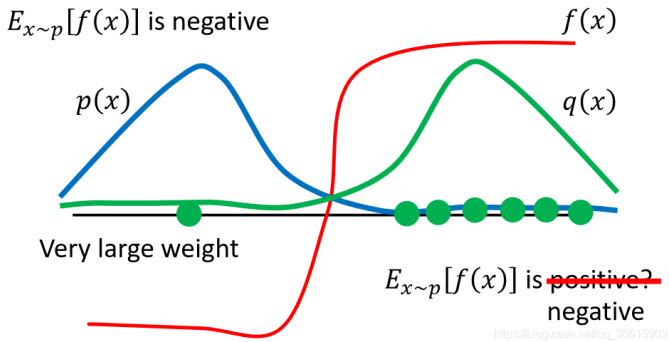
PG 算法有个问题，由于 $A^\theta\left(s_t,a_t\right)$ 是策略参数是 θ 与环境交互的时候计算出来的。一旦更新了参数，从 θ 变成 θ'，在对应状态 s 下采取动作的概率 $p_\theta(\tau)$ 就不对了，之前采样的数据也不能用了。换言之，我们大多数时间都在采样数据。智能体与环境交互以后，接下来就要更新参数，我们只能更新参数一次，然后就要重新采样数据， 才能再次更新参数。

引入重要性采样：对于每一个给定点 x，我们知道其发生的概率，但是我们并不知道 p 的分布,我们可以从一个已知的分布 q 中进行采样。通过对采样点的概率进行比较，确定这个采样点的重要性。

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

> 当不能从 p 里面很好的采样数据，而能从 q 里面很好的采样数据时，虽然我们可以把 p 换成任何的 q，但是在实现上，p 和 q 的差距不能太大，否则会出问题。
> $\mathrm{Var}_{x\sim p}[f(x)] = \mathbb{E}_{x\sim p}\left[f(x)^2\right] - \left(\mathbb{E}_{x\sim p}[f(x)]\right)^2$
> $\mathrm{Var}_{x\sim q}\left[f(x)\frac{p(x)}{q(x)}\right] = \mathbb{E}_{x\sim q}\left[\left(f(x)\frac{p(x)}{q(x)}\right)^2\right] - \left(\mathbb{E}_{x\sim q}\left[f(x)\frac{p(x)}{q(x)}\right]\right)^2$
> $= \int\left(f(x)\frac{p(x)}{q(x)}\right)^2 q(x)dx - \left(\mathbb{E}_{x\sim p}[f(x)]\right)^2$
> $= \int f(x)^2\frac{p(x)}{q(x)}p(x)dx - \left(\mathbb{E}_{x\sim p}[f(x)]\right)^2$
> $= \mathbb{E}_{x\sim p}\left[f(x)^2\frac{p(x)}{q(x)}\right] - \left(\mathbb{E}_{x\sim p}[f(x)]\right)^2$
> 后者的第一项多乘了 $\frac{p(x)}{q(x)}$，如果 $\frac{p(x)}{q(x)}$ 差距很大，$f(x)\frac{p(x)}{q(x)}$ 的方差就会很大。两个随机变量的平均值相同，并不代表它们的方差相同。结论就是，如果我们只要对分布 p 采样足够多次，对分布 q 采样足够多次，得到的期望值会是一样的。但是如果采样的次数不够多，会因为它们的方差差距可能是很大的，所以就可能得到差别非常大的结果。

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



回到PPO中，假设我们收集数据时使用的策略参数是 θ'，此时收集到的数据 τ 保存到记忆库中，但收集到足够的数据后，我们对参数按照前述的PG方式进行更新，更新后，策略的参数从 $\theta' \rightarrow \theta$，此时如果采用PG的方式，我们就应该用参数 θ 的策略重新收集数据，但是我们打算重新利用旧有的数据再更新更新 θ。注意到我我们本来应该是基于 θ 的策略来收集数据，但实际上我们的数据是由 θ' 收集的，所以就需要引入重要性采样来修正这二者之间的偏差，这也就是前面要引入重要性采样的原因。

$$\begin{aligned}\nabla\bar{R}_\theta &= \mathbb{E}_{(s_t,a_t)\sim\pi_{\theta'}}\left[\frac{p_\theta(s_t,a_t)}{p_{\theta'}(s_t,a_t)}A^{\theta'}(s_t,a_t)\nabla\log p_\theta(a_t^n \mid s_t^n)\right]\\ &= \mathbb{E}_{(s_t,a_t)\sim\pi_{\theta'}}\left[\frac{p_\theta(a_t \mid s_t)}{p_{\theta'}(a_t \mid s_t)}\frac{p_\theta(s_t)}{p_{\theta'}(s_t)}A^{\theta'}(s_t,a_t)\nabla\log p_\theta(a_t^n \mid s_t^n)\right]\\ &= \mathbb{E}_{(s_t,a_t)\sim\pi_{\theta'}}\left[\frac{p_\theta(a_t \mid s_t)}{p_{\theta'}(a_t \mid s_t)}A^{\theta'}(s_t,a_t)\nabla\log p_\theta(a_t \mid s_t)\right]\end{aligned}$$

- 假设 $p_\theta(s_t) = p_{\theta'}(s_t)$。另外 $p_\theta(s_t)$ 很难算，需要拿 θ 去跟环境做互动算 s_t 出现的概率，有可能 s_t 根本就不会出现第二次。我们根本没有办法估这一项，所以就直接无视这个问题。

从而最终可以从 $\nabla f(x) = f(x)\nabla\log f(x)$ 来反推目标函数，当使用重要性采样的时候，要去优化的目标函数如下式：

$$J^{\theta'}(\theta) = \mathbb{E}_{(s_t,a_t)\sim\pi_{\theta'}}\left[\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)}A^{\theta'}(s_t,a_t)\right]$$

如果 $p_\theta(a_t|s_t)$ 与 $p'_\theta(a_t|s_t)$ 相差太多，即这两个分布相差太多，重要性采样的结果就会不好。怎么避免它们相差太多呢？增加了一个KL散度约束，这就是信任区域策略优化(Trust Region Policy Opimization，简称TRPO)算法。

$$J^{\theta'}_{\mathrm{TRPO}}(\theta) = \mathbb{E}_{(s_t,a_t)\sim\pi_{\theta'}}\left[\frac{p_\theta(a_t \mid s_t)}{p_{\theta'}(a_t \mid s_t)}A^{\theta'}(s_t,a_t)\right]$$
$$s.t.\ \mathrm{KL}(\theta,\theta') < \delta$$

TRPO 就是考虑到连续动作空间无法每一个动作都搜索一遍，因此大部分情况下只能靠猜。如果要猜，就最好在信任域内部去猜。而TRPO将每一次对策略的更新都限制了信任域内，从而极大地增强了训练的稳定性。TRPO的问题在于把 KL 散度约束当作一个额外的约束，没有放在目标里面。因为信任域的计算量太大了，导致TRPO很难计算。

近端策略优化PPO算法是针对TRPO计算量的大的问题提出来的。PPO算法有两个主要的变种：近端策略优化惩罚（PPO-penalty）和近端策略优化裁剪（PPO-clip）。

PPO-penalty把KL散度约束作为惩罚项放在了目标函数中(可用梯度上升的方法去最大化它):

$$J_{\mathrm{PPO}}^{\theta'}(\theta)=\mathbb{E}_{(s_t,a_t)\sim\pi_{\theta'}}\left[\frac{p_\theta\left(a_t\mid s_t\right)}{p_{\theta'}\left(a_t\mid s_t\right)}A^{\theta'}\left(s_t,a_t\right)-\beta\mathrm{KL}\left[\pi_{\theta'}\left(\cdot\mid s_t\right),\pi_\theta\left(\cdot\mid s_t\right)\right]\right]$$
$$\approx\sum_{(s_t,a_t)}\frac{p_\theta\left(a_t\mid s_t\right)}{p_{\theta'}\left(a_t\mid s_t\right)}A^{\theta'}\left(s_t,a_t\right)$$

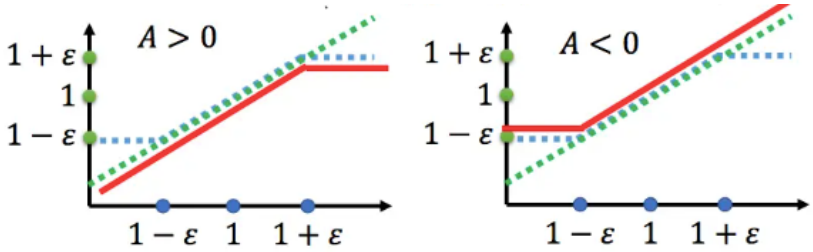
β是可以动态调整的，故称之为自适应KL惩罚(adaptive KL penalty)。具体而言$d=\hat{\mathbb{E}}_t\left[\mathrm{KL}\left[\pi_{\theta'}\left(\cdot\mid s_t\right),\pi_\theta\left(\cdot\mid s_t\right)\right]\right]$

- 如果$d>\mathrm{KL}_{max}$，意味着θ与θ'差距过大(即学习率/步长过大)，也就代表惩罚项的效果太弱，故增大惩罚即增大β
- 如果$d<\mathrm{KL}_{min}$，意味着θ与θ'差距过小，也就代表后面惩罚项的效果太强，故减小惩罚即减小β

近端策略优化裁剪PPO-clip：

$$J_{\mathrm{PPO2}}^{\theta'}(\theta)\approx\sum_{(s_t,a_t)}\min\left(\frac{p_\theta\left(a_t\mid s_t\right)}{p_{\theta'}\left(a_t\mid s_t\right)}A^{\theta'}\left(s_t,a_t\right),\mathrm{clip}\left(\frac{p_\theta\left(a_t\mid s_t\right)}{p_{\theta'}\left(a_t\mid s_t\right)},1-\varepsilon,1+\varepsilon\right)A^{\theta'}\left(s_t,a_t\right)\right)$$

- $clip$括号里的部分，用一句话简要阐述下其核心含义就是：如果$p_\theta(a_t|s_t)$和$p_{\theta'}(a_t|s_t)$之间的概率比落在范围$(1-\varepsilon)$和$(1+\varepsilon)$之外，$\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)}$将被剪裁，使得其值最小不小于$(1-\varepsilon)$，最大不大于$(1+\varepsilon)$
- 如果$A^{\theta'}(s_t,a_t)$大于0，则说明这是好动作，需要增大$p_\theta(a_t|s_t)$，但$\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)}$最大不能超过$(1+\varepsilon)$；如果$A^{\theta'}(s_t,a_t)$小于0，则说明该动作不是好动作，需要减小$p_\theta(a_t|s_t)$，但$\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)}$最小不能小过$(1-\varepsilon)$



这么做的本质目标就是为了让$p_\theta(a_t|s_t)$和$p_{\theta'}(a_t|s_t)$可以尽可能接近，不致差距太大。换言之，这个裁剪算法和KL散度约束所要做的事情本质上是一样的，都是为了让两个分布之间的差距不致过大，但裁剪算法相对好实现。

PPO的思路是：

0点时：我与环境进行互动，收集了很多数据。然后利用数据更新我的策略，此时我成为1点的我。当我被更新后，理论上，1点的我再次与环境互动，收集数据，然后把我更新到2点，然后这样往复迭代。

但是如果我仍然想继续0点的我收集的数据来进行更新。因为这些数据是0点的我（而不是1点的我）所收集的。所以，我要对这些数据做一些重要性重采样，让这些数据看起来像是1点的我所收集的。当然这里仅仅是看起来像而已，所以我们要对这个“不像”的程度加以更新时的惩罚（KL）。

更新的方式是：我收集到的每个数据序列，对序列中每个(s_t,a_t)的优势程度做评估，评估越好的动作，将来就又在s_t状态时，让a_t出现的概率加大。这里评估优势程度的方法，可以用数据后面的总折扣奖励来表示。另外，考虑引入基线的Tip，我们就又引入一个评价者小明，让他跟我们一起学习，他只学习每个状态的期望折扣奖励的平均期望。这样，我们评估(s_t,a_t)时，我们就可以把小明对s_t的评估结果（s_t状态后续能获得的折扣期望）作为我们的基线。注意：优势函数中，前一半是实际数据中的折扣期望，后一半是估计的折扣期望（小明心中认为s_t应该得到的分数，即小明对s_t的期望奖励），如果你选取的动作得到的实际奖励比这个小明心中的奖励高，那小明为你打正分，认为可以提高这个动作的出现概率；如果选取的动作的实际得到的奖励比小明心中的期望还低，那小明为这个动作打负分，你应该减小这个动作的出现概率。这样，小明就成为了一个评判官。

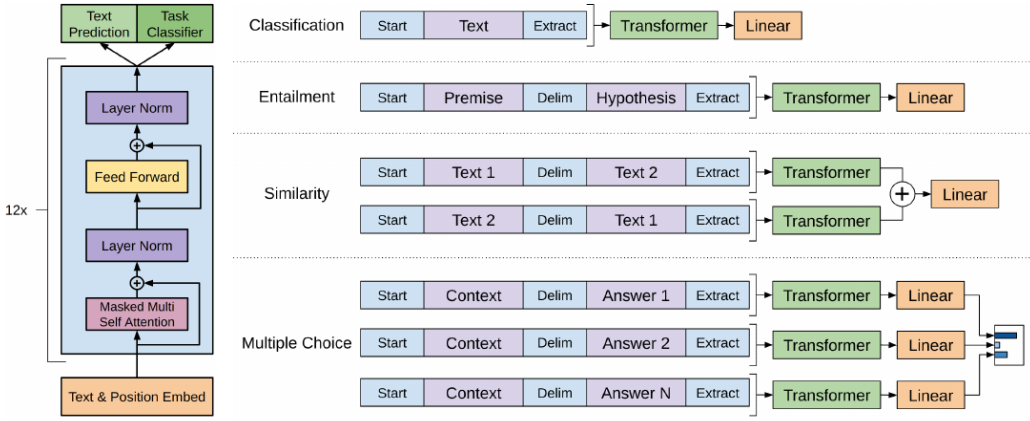
2 GPT

2.1 GPT1

基于 Transformer Decoder 预训练+微调。Generative Pre-Training GPT，主要指 NLP 生成式的预训练模型。训练模式分为2阶段：

1. 第1阶段预训练：利用语言模型LLM进行预训练学习
2. 第2阶微调：通过微调Fine tuning 解决下游任务

微调的时候根据下游任务输入的个数(一个、两个或者多个)采用不同的方案，如下图所示：



2.2 GPT2

虽然GPT1的预训练加微调的范式仅需要少量的微调和些许的架构改动，但能不能有一种模型完全不需要对下游任务进行适配就可以表现优异？GPT2便是在往这个方向努力：不微调但给模型一定的参考样例以帮助模型推断如何根据任务输入生成相应的任务输出。最终，针对小样本/零样本的N-shot Learning应运而生，分为如下三种：

1. Zero-shot Learning(零样本学习）：没有任何训练样本进行微调训练的情况下，让预训练语言模型完成特定任务。相当于不再使用二阶段训练模式(预训练+微调)，而是彻底放弃了微调阶段，仅通过大规模多领域的数据预训练，让模型在Zero-shot Learming的设置下自己学会解决多任务的问题
2. One shot Learning(单样本学习）：在一个训练样本进行微调训练的情况下，预训练语言模型完成特定任务
3. Few-shot Learning（少样本或小样本学习）：在只有少量样本进行微调训练的情况下，预训练语言模型完成特定任务

单样本也好、小样本也好，更多只是作为例子去提示模型，模型不利用样本做训练，即不做模型参数的任何更新。基于prompt即给预训练语言模型的一个线索/提示，帮助它可以更好的理解人类的问题。

2.3 GPT3

GPT3简单来说，就是参数规模大、训练数据规模大、效果出奇好，具体而言：

- 最大规模版本的参数规模达到了1750亿，层数达到了96层，输入维度则达到了12888维
- 使用45TB数据进行训练
- 其预训练任务就是“句子接龙”，给定前文持续预测下一个字，而且更为关键的是，当模型参数规模和训练数据的规模都很大的时候，面对小样本时，其性能表现一度超越SOTA模型

2.4 GPT系列总结

Fine-tuning：预训练语言模型通边微调适配到各种不同的下游任务。具体体现就是通过引入各种辅助任务loss，将其添加到预训练模型中，然后继续pre-training，以便让其更加适配下游任务。

Prompting：各种下游任务能力上移到预训练语言模型。需要对不同任务进行重构，使得它达到适配预训练语言模型的效果。这个过程中，是下游任务做出了更多的牺牲，但是能够更好地统一语言模型范式。

其实GPT-2(包括后面的GPT-3)模型通过大量的文本，已经学到了或者说具备了翻译的能力，只不过它并不知道(或者我们没有办法告诉它)这个任务是翻译的任务。如果我们能够通过一种方法告诉它我们要它翻译，它就有能力翻译。

样本的多少并不会影响GPT-3模型的参数，所以不能说它通过几个例子”学到”了这个任务。而只能说它”理解”了这个任务是什么。从后面的InstructGPT的文章来看，我觉得理解的作用更大一些，因为InstructGPT的小模型也比GPT-3的大模型的效果好。而我们知道InstructGPT只是通过了人类反馈让它理解问题，所以从这个角度来看大模型的理解问题的能力明显要超过小模型。

2.5 指令微调技术(IFT)

OpenAI的GPT3虽然不再微调模型(pre-training + prompt)，但Google依然坚持预训练 + 微调的模式。谷歌FLAN大模型提出了基于Instruction Fine-Tuning(指令微调，简称IFT)，极大地提升了大语言模型的理解能力与多任务能力，且其在评估的25个数据集中有20个数据集的零样本学习能力超过175B版本的GPT3(毕竟指令微调的目标之一即是致力于improving zero-shot generalization to tasks that were not seen in training)，最终达到的效果就是：遵循人类指令，举一反三地完成任务。

IFT的数据通常是由人工手写指令和语言模型引导的指令实例的集合，这些指令数据由三个主要组成部分组成：指令、输入和输出，对于给定的指令，可以有多个输入和输出实例。FLAN的核心思想是，当面对给定的任务A时，首先将模型在大量的其他不同类型的任务比如B、C、D...上进行微调，微调的方式是将任务的指令与数据进行拼接(可以理解为一种Prompt)，随后给出任务A的指令，直接进行推断。相当于通过指令微调之后，模型可以更好的做之前预训练时没见过的新任务且降低了对prompt的敏感度。

2.6 思维链技术(CoT)

为让大语言模型进一步具备解决数学推理问题的能力，谷歌提出了最新的Prompting机制——Chain of Thought(简称CoT)，简言之就是给模型推理步骤的prompt，让其学习人类如何一步步思考/推理，从而让模型具备基本的推理能力，最终可以求解一些简单甚至相对复杂的数学推理能力。

不论是few-shot还是zero-shot，在加入Cot技术之后，都能回答此前不能回答的某些数学推理问题，甚至出现了风靡一时的“let's think step by step”的梗。

3 ChatGPT

基于GPT3的发展路线：

- 为了具备代码/推理能力：GPT3 + 代码训练 = Codex
- 为了更好理解人类：GPT3 + 指令学习 + RLHF = instructGPT

基于GPT3.5的发展路线：增强代码/推理能力 + RLHF = ChatGPT

ChatGPT初版与InstructGPT的差别是基于GPT3还是GPT3.5微调，通过OpenAI公布的ChatGPT训练图可知，ChatGPT的训练流程与InstructGPT是一致的。

- InstructGPT(有1.3B 6B 175B参数的版本)，是在GPT-3(原始的GPT3有1.3B 2.7B 6.7B 13B 175B等8个参数大小的版本)上做Fine-Tune
- ChatGPT是在GPT-3.5上做Fine-Tune

3.1 以用户为中心

大模型其实通过预训练已经学到了很多技能，但是我们要有一种办法告诉它我们需要这种技能。之前的方法就是Prompt Learning，在GPT-3的论文里最简单的就是直接的Instruct和间接的Few-shot的示例。但是这两种方法都有问题。

- Instruct是比较tricky的，不同的人表达相似的要求会差异很多
- 示例有的时候也很难，比如写一篇关于熊和兔子的作文

GPT-3采用的Prompt是非常简单的，它并没有把重点放到怎么探索魔法咒语，只是证明了GPT-3可以通过Instruct和Few-Shot的例子学习到任务，而OpenAI的研究也从来没有关注过Prompt Learning。

用户还是用最习惯的自然语言来表达它们的Instruct，用他们最自然的方式来表达需求。但是机器怎么知道呢？标注数据让它学习就行了。我们并不是要教模型怎么写作文，而是要教它怎么理解任务。大模型可能会生成一些不真实的、有害的和无用的内容，也就是说大模型和用户的需求并不对齐(aligned)。通过人类反馈来微调模型从而在大量任务上对齐大语言模型和用户意图的方法。比如理解"写一篇关于熊和兔子的故事"。我们要教模型的是理解任务，还需要教给模型用合适的方式生成答案，这就是所谓的从人类反馈学习。

3.2 为什么使用强化学习

GPT 模型本质就是一个语言模型，不管是什么 Instruct，对于它来说都是一个条件生成的问题。对于一些简单的 task，比如问答，让人给出 Demonstration 还比较容易。但是另外一些复杂的 task，比如"写一篇关于青蛙王子的故事"，这就费劲了，而且写出来质量也不见得好。那怎么办呢？批评比解决问题容易。你让一个人写一篇好的作文当然很难，但是给你两篇文章让你判断哪篇写得更好，这就容易的多了。这就是RM模型，其实就是对两个结果进行比较。

给定一个prompt，我穷举所有的response，然后用RM模型打分，选出最好的那些来训练。但是这里有一个问题，response的空间太大了！我们可以发现，我们的目标其实就是给定一个prompt，我们需要探索出更好的生成策略。我们可以把生成一个response的过程看成一个与环境互动的过程：首先生成第一个词，得到reward，再生成第二个词，又得到reward，.....。不过环境给的reward是整个response的奖励，而不是分解到每一步的reward。如果知道每一步的好坏，那就可以回到监督学习了。

比如我们的prompt是"你叫什么名字？"，然后有两个response:"我叫张三"和"你叫李四"。奖励函数告诉第一个response要比第二个好。但是RM模型不会反馈到每个词的粒度。如果要到词的粒度，应该反馈"你"是不好的回复，而"叫张三"和"叫李四"都没有问题。如果有这样的反馈，我们直接使用监督学习就可以了。现在的问题是：虽然"你叫李四"这个回复不好，但是我们要把主要责任算到"你"这个词的头上，而不应该算在"叫李四"的头上，这样模型才会正确的学到应该怎么回复。而怎么根据最终的reward来判断每一步的好坏，这正是强化学习要解决的问题。一个理想的强化学习Agent应该可以通过探索学习到Reward函数到底要什么。

虽然RL理论上虽不需要大量标注数据，但实际上它所需求的reward不好制定会存在缺陷。RLHF试图解决的问题是，在奖励函数不够明确的情况下，通过基于人类对事物比较的偏好而非绝对奖励值训练奖励函数。通过人类标注数据训练得到Reward Model（相当于有了人类标注数据，则相信它是不错的，然后反推人类因为什么样的奖励函数才会采取这些行为）。有了奖励函数之后，就可以使用一般的强化学习的方法去找出最优策略/动作。

3.3 数据集

因为GPT-3模型收集的prompt并不是非常自然语言描述的。要求标注者写如下三种prompt：

1. 普通的prompt：我们随意给定一些任务，让它们写prompt
2. Few-shot：我们要求他们写一个指令(Instruct)，以及遵循这个指令的多个query/repsone对
3. 用户要求：OpenAI的API会收集一些用户的需求，我们让他们根据这些需求写prompt

通过这些prompt，我们构造了3个不同的数据集

1. 有监督微调(SFT)数据集，需要标注者根据prompt写demonstration
2. RM数据集，给定prompt，让模型的输出多个completion，标注者对这些输出的好坏进行排序
3. PPO数据集，用于RLHF微调。这个数据集只有prompt，没有标注

标注任务有两个来源：标注人员写的prompt 和 早期InstructGPT模型通过API手机的prompt。这些prompt分成多样，包括生成、问答、对话、摘要、抽取和其它自然语言任务。对于每一个自然语言的prompt，任务通常直接用自然语言的指令来制定(比如"写一个关于聪明的青蛙的故事")，但是也有间接通过few-shot示例来描述任务的(比如给定两个青蛙的故事，让模型在生成一个故事)，或者通过隐式的续写(比如提供一个关于青蛙故事的开头部分)。在每种情况下，我们都要求标注者尽他们最大的努力通过prompt来推测用户的意图，如果实在看不懂就跳过。

3.4 阶段1：监督学习微调

利用人类的问答数据去对GPT3进行有监督训练(作为baseline)。这个微调好的GPT3我们称之为SFT模型(SFT的全称Supervised fine-tuning)，它作为baseline具备了最基本的预测能力(该基线模型的大小为175B)。

1. 先设计了一个 prompt dataset，里面有大量提示样本，给出了各种各样的任务描述
2. 其次，标注团队对 prompt dataset进行标注（本质就是人工回答问题）
3. 用标注后的数据集微调 GPT3（可允许过拟合），微调后模型称为 SFT 模型（Supervised fine-tuning，SFT），具备了最基本的文本生成能力

3.5 阶段2：训练Reward模型

通过将上一阶段SFT模型去掉最后的嵌入层初始化出我们的RM模型，且考虑到175B计算量大且不稳定不适合作为奖励函数，故最后用的6B版本的SFT初始化RM模型。它的输入是prompt和Reponse，输出是奖励值，它可以看做一个回归模型。GPT-3最大的模型时175B的，但是发现这种规模的模型训练不稳定，最后用了6B的版本。核心是由人对SFT生成的多个输出进行排序，再用来训练RM。

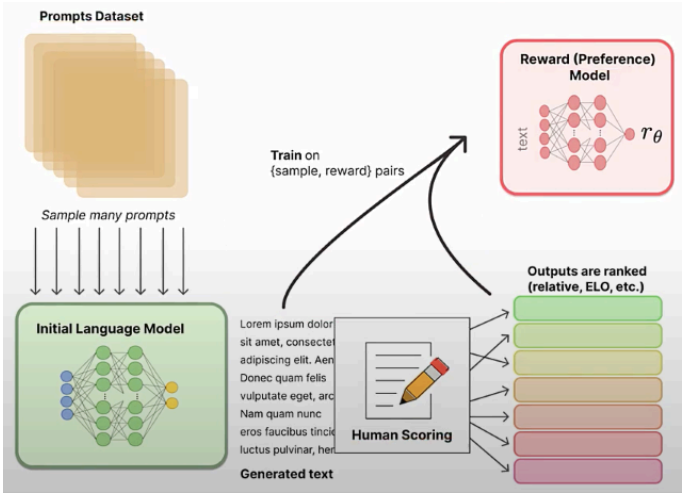
1. 微调后的 SFT 模型去回答 prompt dataset 问题，收集 4 ~ 9 个不同的输出
2. 接着人工对SFT模型生成的 4 ~ 9 个回答的好坏进行标注且排序
3. 排序结果用来训练奖励模型 RM（Reward Model），即学习排序结果从而理解人类的偏好

为了让 RM 学到人类偏好（即排序），可以 4 ~ 9 个语句两两组合分别计算 loss 再相加取均值

$$\text{loss}(\theta) = -\frac{1}{\binom{K}{2}} E_{(x,y_w,y_l)\sim D}\left[\log\left(\sigma\left(r_\theta\left(x,y_w\right)-r_\theta\left(x,y_l\right)\right)\right)\right]$$

- $r_\theta(x,y)$ 是奖励模型对输入 x,y 的打分。对于输入 x,y_w 是更好的 completion, 而 y_l 是较差的那个。如果 $r_\theta\left(x,y_w\right)$ 与 $r_\theta\left(x,y_l\right)$ 差越大, 则 $\sigma\left(r_\theta\left(x,y_w\right)-r_\theta\left(x,y_l\right)\right)$ 也越大。所以模型的优化目标就是要让好的completion的得分大于差的

RM 逐渐学会了给 D 这类语句打高分，给 A/B/C 这类语句打低分，从而模仿人类偏好。可以这么简单理解RLHF：某种意义上来说，由人类打分来充当 Reward。



3.6 阶段3：强化学习微调

在交互环境(environment)使用 PPO 算法来微调SFT模型。

1. 让 SFT 模型去回答 prompt dataset 问题，得到一个输出，即生成的回答
2. 此时不再让人工评估好坏，而是让 阶段2 RM 模型去给 SFT 模型的预测结果进行打分排序
3. 使用 PPO 算法对 SFT 模型进行反馈更新，更新后的模型称为 PPO-ptx。

目标函数如下：

$$\begin{aligned}
&\text{objective}\left(\phi\right) = E_{(x,y)\sim D_{\pi_\phi^{RL}}}\left[r_\theta(x,y)-\beta\log\left(\pi_\phi^{RL}(y\mid x)/\pi^{SFT}(y\mid x)\right)\right]+\gamma E_{x\sim D_{\text{pretrain}}}\left[\log\left(\pi_\phi^{RL}\right)\right]\\
&=E_{(x,y)\sim D_{\pi^{RL'}}}\left[\frac{\pi_\phi^{RL}(y\mid x)}{\pi^{RL'}(y\mid x)}r_{\theta'}(x,y)-\beta\log\left(\pi_\phi^{RL}(y\mid x)/\pi^{SFT}(y\mid x)\right)\right]+\gamma E_{x\sim D_{\text{pretrain}}}\left[\log\left(\pi_\phi^{RL}\right)\right]\\
&=E_{(x,y)\sim D_{\pi^{RL'}}}\left[\min\left(\frac{\pi_\phi^{RL}(y\mid x)}{\pi^{RL'}(y\mid x)}r_{\theta'}(x,y),\text{clip}\left(\frac{\pi_\phi^{RL}(y\mid x)}{\pi^{RL'}(y\mid x)},1-\varepsilon,1+\varepsilon\right)r_{\theta'}(x,y)\right)-\beta\log\left(\pi_\phi^{RL}(y\mid x)/\pi^{SFT}(y\mid x)\right)\right]+\gamma E_{x\sim D_{\text{pretrain}}}\left[\log\right.
\end{aligned}$$

- $r_{\theta'}(x,y)$：这里的参数 θ 并不是强化学习 Agent 的参数，而是前面奖励模型的参数，在强化学习时它是不变的
- π^{SFT}：基线策略
- $\pi^{RL'}$：新策略 π_ϕ^{RL} 更新之前的旧策略

已经掌握人类偏好的RM模型一旦判定现有回答的不够好，便更新 π_ϕ^{RL}，但如果 π_ϕ^{RL}一旦变化，会导致后续 $\bar{R}_\theta = \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[R(\tau)\nabla\log p_\theta(\tau)\right]$ 计算一系列问答评分时中的 $p_\theta(\tau)$ 发生变化(策略一变轨迹必变)，进而已采样的问答数据 $(x_{n+2},\{y_{n+2}^1,y_{n+2}^2,y_{n+2}^3,y_{n+2}^4\})(x_{n+3},\cdots)(x_{n+4},\cdots)(x_{n+5},\cdots)$ 便没法继续使用。而只能不断采样一批新的问答数据(更新一次 π_ϕ^{RL} 后，得采样新一批数据；再更新一次 π_ϕ^{RL}后，再采样新一批数据..)

为避免 π_ϕ^{RL} 只要一更新便只能一次次去采样一批批新问答数据，提高数据利用率，我们改让 $\pi^{RL'}$ 去和环境交互（$\pi^{RL'}$ 也被 π^{SFT} 初始化），在策略 $\pi^{RL'}$ 下模型回答的好不好始终由RM模型评判。然后通过最大化奖励而不断迭代 π^{RL} ，迭代过程中可一定程度的重复使用旧策略 $\pi^{RL'}$ 生成的已有数据

由于在上文的instructGPT训练阶段2中，我们已经得到了根据人类偏好学习出来的RM模型，便可基于“最大化奖励”这个目标下通过PPO算法不断优化RL模型(或也可以叫PPO模型)的策略 π^{RL}

$$E_{(x,y)\sim D_{\pi^{RL}}}\left[\min\left(\frac{\pi_\phi^{RL}(y\mid x)}{\pi^{RL'}(y\mid x)}r_{\theta'}(x,y),\text{clip}\left(\frac{\pi_\phi^{RL}(y\mid x)}{\pi^{RL'}(y\mid x)},1-\varepsilon,1+\varepsilon\right)r_{\theta'}(x,y)\right)\right]$$

具体PPO流程如下：

1. 使用旧策略 $\pi^{RL'}$ 生成一批数据，包括状态、动作和奖励等信息，存储在一个经验回放缓冲区(Experience Replay Buffer)中
2. 在训练新策略 π^{RL} 时，从经验回放缓冲区中随机抽取一批数据
3. 对于旧策略 $\pi^{RL'}$ 采样到的每个数据样本 (x,y)，计算重要性采样权重：$w(x,y)=\frac{\pi_\phi^{RL}(y|x)}{\pi^{RL'}(y|x)}$
4. 使用这些加权样本来更新新策略，即将原始目标函数中的期望部分替换为加权样本的期望：$\text{objective}(\phi)=E_{(x,y)\sim D_\pi^{RL'}}\left[w(x,y)*r_{\theta'}(x,y)\right]$
5. PPO中的截断重要性采样比率 $J_{\text{PPO2}}^{\theta'}(\theta)\approx\sum_{(s_t,a_t)}\min\left(\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)}A^{\theta'}\left(s_t,a_t\right),\text{clip}\left(\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)},1-c,1+c\right)A^{\theta'}\left(s_t,a_t\right)\right)$ 相当于为了对重要性比值做约束，限制新旧策略的比值大小(相当于限制新策略的更新范围)。新旧策略之间的差异过大时，重要性采样权重可能变得不稳定，从而影响训练的稳定性。当然也可以改成根据一个KL散度去约束，毕竟截断和KL散度约束都是实现PPO算法本身的方式

6. 按照更新后的目标函数进行梯度计算和参数更新

由于在当前(旧)策略下的生成/预测结果由RM评判(当前策略优化的目的就是为了让RM最大化)，所以对优化策略的过程中加了一个惩罚项，防止一切RM说了算，进而过于绝对变成独裁，相当于避免不断优化的当前策略与基线策略偏离太远。

$\pi^{RL'}$ 的初始化值就是 π^{SFT}，最终希望 $\pi^{RL'}$ 最终迭代结束后，它俩之间的差距不至于太大。通过KL散度衡量两个策略的概率分布之间的差距，从而使得在优化策略时限制参数更新的范围。注意，这个KL散度和PPO已经没有关系了，只是一个KL散度约束的普通应用

$$\log\left(\pi_\phi^{\mathrm{RL}}(y \mid x)/\pi^{\mathrm{SFT}}(y \mid x)\right)$$

防止ChatGPT在训练过程中过度优化，从而避免过于放飞自我，通过某种刁钻的方式取悦人类，而不是老老实实地根据人类的问题给出正确答案。通俗点说，以保持GPT3.5原有的核心性能加入修正偏置项，避免大模型为了过拟合强化学习任务而遗忘了其它预训练的能力。

$$E_{x\sim D_{\mathrm{pretrain}}}\left[\log\left(\pi_\phi^{\mathrm{RL}}(x)\right)\right.$$