

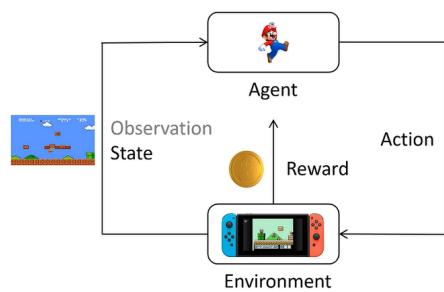
[https://www.bilibili.com/video/BV1iz421h7gb/?](https://www.bilibili.com/video/BV1iz421h7gb/)

spm_id_from=333.788&vd_source=962885f33dd6c9f036a5bb4e9afaaa01

P
参考视频。

概率分布：是描述一个“随机变量”在所有可能取值上出现的概率的函数。

比如：设投骰子的结果“X”， $P(X) = \frac{1}{6}$ ($X \in \{1, 2, 3, 4, 5, 6\}$) 就是一个概率分布。



Action Space: 可选择的动作，比如 {left, up, right}

Policy: 策略函数，输入State，输出Action的概率分布。一般用 π 表示。

$$\begin{aligned}\pi(left|s_t) &= 0.1 \\ \pi(up|s_t) &= 0.2 \\ \pi(right|s_t) &= 0.7\end{aligned}$$

(随机性策略)

Trajectory: 轨迹，用 τ 表示，一连串状态和动作的序列。Episode, Rollout。 $\{s_0, a_0, s_1, a_1, \dots\}$

$$\begin{aligned}s_{t+1} &= f(s_t, a_t) \text{ 确定} \\ s_{t+1} &= P(\cdot | s_t, a_t) \text{ 随机}\end{aligned}$$

Return: 报酬，从当前时间点到游戏结束的Reward的累积和。

一场成功的回报。
一次 state 的回报。

1. policy 的输入“向量” s_t ；输出“向量”
(环境描述) (动作维度，概率值)

目标：训练一个 Policy 神经网络 π ，在所有状态 s 下，给出相应的 Action，得到 Return 的期望最大。← “一次性输出”的均值

目标：训练一个 Policy 神经网络 π ，在所有的 Trajectory 中，得到 Return 的期望最大。← “一步步输出”的均值

$$\begin{aligned}E(R(\tau))_{\tau \sim P_\theta(\tau)} &= \sum_{\tau} R(\tau) P_\theta(\tau) \\ \text{乙代表一个策略 trajectory} &\quad \nearrow \\ \text{为求最大的 } E(R(\tau)) \text{ 对该} & \\ \text{参数进行梯度上升方法，多改} & \\ \theta &\end{aligned}$$

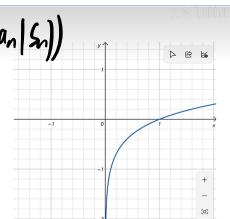
$$\begin{aligned} &= \nabla E(R(\tau))_{\tau \sim P_\theta(\tau)} \\ &= \sum_{\tau} R(\tau) \nabla P_\theta(\tau) \\ &= \sum_{\tau} P_\theta(\tau) R(\tau) \frac{\nabla P_\theta(\tau)}{P_\theta(\tau)} \\ &= \left(\sum_{\tau} P_\theta(\tau) R(\tau) \right) \frac{\nabla P_\theta(\tau)}{P_\theta(\tau)} \\ &\approx \left(\frac{1}{N} \sum_{n=1}^N R(\tau^n) \right) \frac{\nabla P_\theta(\tau^n)}{P_\theta(\tau^n)} \\ &= \frac{1}{N} \sum_{n=1}^N R(\tau^n) \nabla \log P_\theta(\tau^n) \quad \tau \sim P_\theta(\tau) \\ &= \frac{1}{N} \sum_{n=1}^N R(\tau^n) \nabla \log P_\theta(\tau^n) \quad \nearrow \\ &= \frac{1}{N} \sum_{n=1}^N R(\tau^n) \nabla \log \prod_{t=1}^{T_n} P_\theta(a_n^t | s_n^t) \\ &= \frac{1}{N} \sum_{n=1}^N R(\tau^n) \sum_{t=1}^{T_n} \nabla \log P_\theta(a_n^t | s_n^t) \\ &= \frac{1}{N} \sum_{n=1}^N \sum_{t=1}^{T_n} R(\tau^n) \nabla \log P_\theta(a_n^t | s_n^t)\end{aligned}$$

目标是 $\max(E(R(\tau)))$ 的
参数为目标梯度。

$$\frac{1}{N} \sum_{n=1}^N \sum_{t=1}^{T_n} R(\tau^n) \log P_\theta(a_n^t | s_n^t)$$

τ : trajectory (τ) 服从多级为
日的策略网络 P 的输出分布。
 $R(\tau)$: trajectory (τ) 输入 Reward
model 得到的“奖励分配”

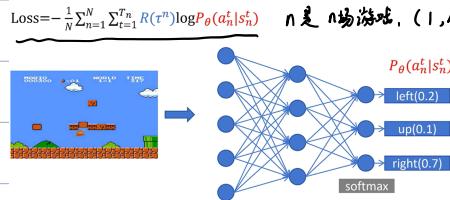
τ 代表什么？第几条轨迹



Policy gradient

在整个梯度上升算法的 Policy 改进包含的自修改更新过程中，
值得当大游戏场 ~~big~~
如果为上，则这场游戏
每个步的值下 ~~值~~ 都被弱化。

7 是一场游戏中的一步 (1, t)



$R(\tau^n)$
 $\tau^1 \rightarrow R(\tau^1)$
 $\tau^2 \rightarrow R(\tau^2)$
 \vdots
 $\tau^n \rightarrow R(\tau^n)$



$$\frac{1}{N} \sum_{n=1}^N \sum_{t=1}^{T_n} (R_t^n - B(s_t^n)) \nabla \log P_\theta(a_t^n | s_t^n)$$

例子： $A_\theta, Q_\theta, V_\theta$ 是用 R_t^n 从 $(cause, R_t^n)$ 方差大。

Action-Value Function

R_t^n 每次都是一次随机采样，方差很大，训练不稳定。
 $Q_\theta(s, a)$ 在 state s 下，做出 Action a ，期望的回报。动作价值函数。

State-Value Function

$V_\theta(s)$ 在 state s 下，期望的回报。状态价值函数。

Advantage Function

优势函数。

$A_\theta(s, a) = Q_\theta(s, a) - V_\theta(s)$ 在 state s 下，做出 Action a ，比其他动作能带来多少优势。

$$\frac{1}{N} \sum_{n=1}^N \sum_{t=1}^{T_n} A_\theta(s_t^n, a_t^n) \nabla \log P_\theta(a_t^n | s_t^n)$$

目的是：平滑 梯度下降过程 ✓

要想再往下，确是需要知道。对于 Policy 改进的更新过程。

$$\theta \rightarrow \theta \rightarrow \Delta \theta$$

输入 (s) 、输出 $P(a)$ \Rightarrow Loss_Func. $\theta \Rightarrow \theta \uparrow \theta \downarrow \Rightarrow$ 输入 (s) 、输出 $P(a)$
Change

梯度上升算法的梯度更新。

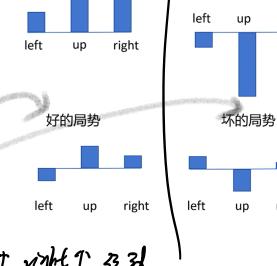
r_t^n 在第 t 步游戏中，第 t 步所获回报。

$$\begin{aligned} &= \frac{1}{N} \sum_{n=1}^N \sum_{t=1}^{T_n} R(\tau^n) \nabla \log P_\theta(a_t^n | s_t^n) \\ &= \frac{1}{N} \sum_{n=1}^N \sum_{t=1}^{T_n} R_t^n \nabla \log P_\theta(a_t^n | s_t^n) \\ &= \frac{1}{N} \sum_{n=1}^N \sum_{t=1}^{T_n} (R_t^n - B(s_t^n)) \nabla \log P_\theta(a_t^n | s_t^n) \end{aligned}$$

注 left, up, right 就是梯。
因为 left, up, right 值
up 值。

$$W_\theta = \nabla \theta_{m+1} + \Delta$$

∴ 改变一直朝 left ↑ up ↑ right ↑ 进步。



即 R_t^n 的方差大，即 trajectory 的修改太多。

为什么方差就大？它的值大又为什么不稳定？

$Q_\theta(s, a), V_\theta(s)$ 是什么？输入输出是啥？ ✓

1. 状态价值函数 (State-Value Function)

$$V_\theta(s) = \mathbb{E}_{P_\theta}[R_t | s_t = s]$$

2. 动作价值函数 (Action-Value Function)

$$Q_\theta(s, a) = \mathbb{E}_{P_\theta}[R_t | s_t = s, a_t = a]$$

在这两个公式中， P_θ 表示基于参数 θ 的策略的概率分布，用于计算期望回报。

关于梯度下降：

$$y = x^2 + 2x - 3$$

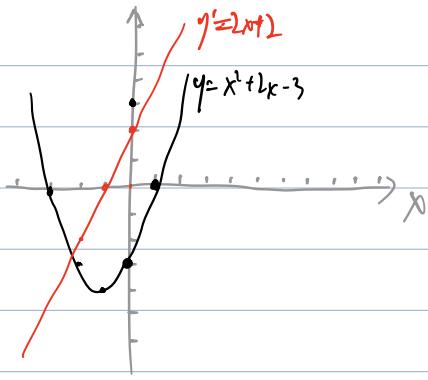
$$y' = 2x + 1$$

$$x_0 = 3$$

If, Aim, Find minimizer of "f(x)"

$$x_1 = x_0 - \alpha \cdot \nabla f(x)$$

$$\text{若 } x_0 = 3 \rightarrow \alpha = 0.1 \quad 0.2 \quad \nabla f(x) = 8 \quad -4$$



$$x_1 = 2.2 - 2 \cdot 0.1 = 2.0 \quad \nabla f(x) = 6.4 \rightarrow -4 \quad y_1 - y_0 = f(x_1) - f(x_0)$$

$$x_2 = 1.66 \quad \alpha = 0.1 \quad \nabla f(x) = 4.12$$

$$x_3 = 1 \quad \alpha = 0.1 \quad \nabla f(x) = 3 \quad \text{为什么梯度下降法可以找到最小值.}$$

$$x_4 = 0.2 \quad \alpha = 0.1 \quad \nabla f(x) = 2.7 \quad \text{答, 梯度是该点处一元函数的导数.}$$

$$x_5 = 0.4 \quad \alpha = 0.1 \quad \nabla f(x) = 2.3 \quad \text{梯度} > 0 \quad x_5 > x_4 \quad f(x_5) > f(x_4)$$

$$x_6 = 0.2 \quad \alpha = 0.1 \quad \nabla f(x) = 2.14 \quad \text{梯度} < 0 \quad x_6 > x_5 \quad f(x_6) < f(x_5)$$

$$x_7 = 0$$

目标函数：为什么要用目标函数，而不是直接一个网络去训练。

" x " 是梯度下降的输入；但对于一个网络， $y = f_\theta(x)$ 为观察值-预测。

real- $x \rightarrow \text{network}(\theta) \rightarrow \text{real-}y$

$$\text{expected-}y \quad [\text{expected-}y - \text{network}(\theta)(\text{real-}x)] \rightarrow \min$$

$$\text{LOSS} = \text{梯度二乘, 有 } \frac{1}{n} \sum [(\text{expected-}y - \text{network}(\theta)(\text{real-}x))^2] \rightarrow \min$$

$$\therefore \nabla \text{LOSS}(\theta)$$

$$W_1 = W_0 - \alpha \cdot \nabla \text{LOSS}(W_0, \text{real-}x, \text{expected-}y)$$

$$\{\nabla \}_{\theta} \Rightarrow W^\theta$$

就是 LOSS 函数 使得 " $y = f_\theta(x) + \theta$ 不变量. 0 情况" $\Rightarrow z = \text{expected-}y - \text{real-}y$

$$= \text{expected-}y - f_\theta(x)$$

中 θ 成为了变量

$$A_\theta(s, a) = Q_\theta(s, a) - V_\theta(s)$$

$Q_\theta(s, a)$ 在 state s 下，做出 Action a ，期望的回报。动作价值函数。

$$\text{这里有个假定 } Q_\theta(s, a) = V_\theta(s_t) + \gamma * V_\theta(s_{t+1})$$

$$Q_\theta(s_t, a) = r_t + \gamma * V_\theta(s_{t+1})$$

根据时间轴“累积奖励”

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

$$A_\theta^1(s_t, a) = r_t + \gamma * V_\theta(s_{t+1}) - V_\theta(s_t)$$

$$V_\theta(s_{t+1}) \approx r_{t+1} + \gamma * V_\theta(s_{t+2})$$

$$A_\theta^2(s_t, a) = r_t + \gamma * r_{t+1} + \gamma^2 * V_\theta(s_{t+2}) - V_\theta(s_t)$$

$$A_\theta^3(s_t, a) = r_t + \gamma * r_{t+1} + \gamma^2 * r_{t+2} + \gamma^3 * V_\theta(s_{t+3}) - V_\theta(s_t)$$

⋮

$$A_\theta^T(s_t, a) = r_t + \gamma * r_{t+1} + \gamma^2 * r_{t+2} + \gamma^3 * r_{t+3} + \dots + \gamma^T * r_T - V_\theta(s_t)$$



$$\text{偏差} = E[\hat{V}(s)] - \hat{V}(s)$$

$$\text{方差} = E[(\hat{V}(s) - E[\hat{V}(s)])^2]$$

估计

estimation

$$A_\theta^1(s_t, a) = r_t + \gamma * V_\theta(s_{t+1}) - V_\theta(s_t)$$



$$A_\theta^2(s_t, a) = r_t + \gamma * r_{t+1} + \gamma^2 * V_\theta(s_{t+2}) - V_\theta(s_t)$$

“偏差过大”，意味着存在“系统性误差”

$$A_\theta^3(s_t, a) = r_t + \gamma * r_{t+1} + \gamma^2 * r_{t+2} + \gamma^3 * V_\theta(s_{t+3}) - V_\theta(s_t)$$

⋮

$$A_\theta^T(s_t, a) = r_t + \gamma * r_{t+1} + \gamma^2 * r_{t+2} + \gamma^3 * r_{t+3} + \dots + \gamma^T * r_T - V_\theta(s_t)$$

导致策略朝错误方向学习。

$$\delta_t^V = r_t + \gamma * V_\theta(s_{t+1}) - V_\theta(s_t)$$

$$\delta_{t+1}^V = r_{t+1} + \gamma * V_\theta(s_{t+2}) - V_\theta(s_{t+1})$$

“方差过大”意味着学习过程不稳定。

$$A_\theta^1(s_t, a) = \delta_t^V$$

$$A_\theta^2(s_t, a) = \delta_t^V + \gamma \delta_{t+1}^V$$

$$A_\theta^3(s_t, a) = \delta_t^V + \gamma \delta_{t+1}^V + \gamma^2 \delta_{t+2}^V$$

⋮

$$\delta_t^V = r_t + \gamma * V_\theta(s_{t+1}) - V_\theta(s_t)$$

$$A_\theta^1(s_t, a) = \delta_t^V$$

$$A_\theta^2(s_t, a) = \delta_t^V + \gamma \delta_{t+1}^V$$

$$A_\theta^3(s_t, a) = \delta_t^V + \gamma \delta_{t+1}^V + \gamma^2 \delta_{t+2}^V$$

⋮

Generalized Advantage Estimation (GAE)

$$A_\theta^{GAE}(s_t, a) = (1 - \lambda)(A_\theta^1 + \lambda * A_\theta^2 + \lambda^2 * A_\theta^3 + \dots) \quad \lambda = 0.9: \quad A_\theta^{GAE} = 0.1A_\theta^1 + 0.09A_\theta^2 + 0.081A_\theta^3 + \dots$$

$$= (1 - \lambda)(\delta_t^V + \lambda * (\delta_t^V + \gamma \delta_{t+1}^V) + \lambda^2 * (\delta_t^V + \gamma \delta_{t+1}^V + \gamma^2 \delta_{t+2}^V) + \dots)$$

$$= (1 - \lambda)(\delta_t^V(1 + \lambda + \lambda^2 + \dots) + \gamma \delta_{t+1}^V * (\lambda + \lambda^2 + \dots) + \dots)$$

$$= (1 - \lambda)(\delta_t^V \frac{1}{1 - \lambda} + \gamma \delta_{t+1}^V \frac{\lambda}{1 - \lambda} + \dots)$$

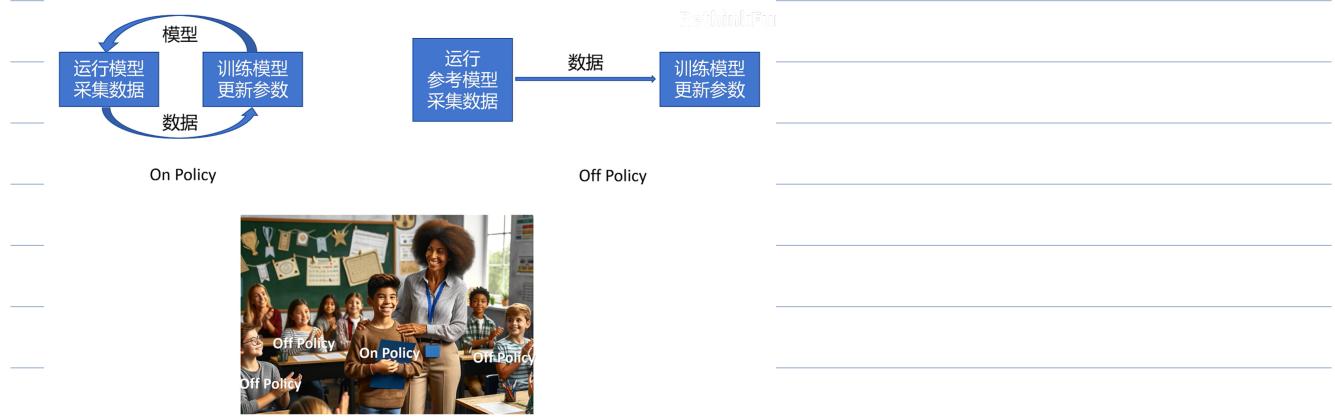
$$= \sum_{b=0}^{\infty} (\gamma \lambda)^b \delta_{t+b}^V$$

A_θ^{GAE} 变成了基于“优势价值函数” $V_\theta(s)$ 的改进。

$$\delta_t^V = r_t + \gamma * V_\theta(s_{t+1}) - V_\theta(s_t)$$

$$A_\theta^{GAE}(s_t, a) = \sum_{b=0}^{\infty} (\gamma \lambda)^b \delta_{t+b}^V$$

$$\frac{1}{N} \sum_{n=1}^N \sum_{t=1}^{T_n} A_\theta^{GAE}(s_n^t, a_n^t) \nabla \log P_\theta(a_n^t | s_n^t)$$



重要性采样

目标是求 $E(f(x))_{x \sim p(x)}$ 即 $p(x)$ 分布下的 $E(f(x))$.

但 $p(x)$ 下的值不知道；只知道 $q(x)$ 下的 $E(f(x))$

于是用公式估计。

要确定 $p(x)$ ，是否知道？

$$\begin{aligned} E(f(x))_{x \sim p(x)} &= \sum_x f(x) * p(x) \\ &= \sum_x f(x) * p(x) \frac{q(x)}{q(x)} \\ &= \sum_x f(x) \frac{p(x)}{q(x)} * q(x) \\ &= E(f(x) \frac{p(x)}{q(x)})_{x \sim q(x)} \\ &\approx \frac{1}{N} \sum_{n=1}^N f(x) \frac{p(x)}{q(x)}_{x \sim q(x)} \end{aligned}$$

~~重要性采样~~ 变化。

$$\begin{aligned} &\frac{1}{N} \sum_{n=1}^N \sum_{t=1}^{T_n} A_\theta^{GAE}(s_n^t, a_n^t) \nabla \log P_\theta(a_n^t | s_n^t) \\ &= \frac{1}{N} \sum_{n=1}^N \sum_{t=1}^{T_n} A_\theta^{GAE}(s_n^t, a_n^t) \frac{P_\theta(a_n^t | s_n^t)}{P_\theta(a_n^t | s_n^t)} \nabla \log P_\theta(a_n^t | s_n^t) \\ &= \frac{1}{N} \sum_{n=1}^N \sum_{t=1}^{T_n} A_\theta^{GAE}(s_n^t, a_n^t) \frac{P_\theta(a_n^t | s_n^t)}{P_\theta(a_n^t | s_n^t)} \frac{\nabla P_\theta(a_n^t | s_n^t)}{P_\theta(a_n^t | s_n^t)} \\ &= \frac{1}{N} \sum_{n=1}^N \sum_{t=1}^{T_n} A_\theta^{GAE}(s_n^t, a_n^t) \frac{\nabla P_\theta(a_n^t | s_n^t)}{P_\theta(a_n^t | s_n^t)} \end{aligned}$$

$$\nabla \log f(x) = \frac{\nabla f(x)}{f(x)}$$

$$\text{Loss} = -\frac{1}{N} \sum_{n=1}^N \sum_{t=1}^{T_n} A_\theta^{GAE}(s_n^t, a_n^t) \frac{P_\theta(a_n^t | s_n^t)}{P_\theta(a_n^t | s_n^t)}$$

← 实现“数据集-01_参考模型-Pθ”与

“参数更新-01_训练模型-Pθ”

分离。

$$Loss_{ppo} = -\frac{1}{N} \sum_{n=1}^N \sum_{t=1}^{T_n} A_{\theta}^{GAE}(s_n^t, a_n^t) \frac{P_{\theta}(a_n^t | s_n^t)}{P_{\theta'}(a_n^t | s_n^t)} + \beta KL(P_{\theta}, P_{\theta'})$$

KL散度? ✓

在 LOSS 中加入 KL 散度会有什么用?

$$Loss_{ppo2} = -\frac{1}{N} \sum_{n=1}^N \sum_{t=1}^{T_n} \min(A_{\theta}^{GAE}(s_n^t, a_n^t) \frac{P_{\theta}(a_n^t | s_n^t)}{P_{\theta'}(a_n^t | s_n^t)}, \text{clip}(\frac{P_{\theta}(a_n^t | s_n^t)}{P_{\theta'}(a_n^t | s_n^t)}, 1 - \varepsilon, 1 + \varepsilon) A_{\theta}^{GAE}(s_n^t, a_n^t))$$

截断函数? ✓

训练效果?

使用的训练中，目标函数

熵 (entropy)

想要考察信息量的损失，就要先确定一个描述信息量的纲领。
在信息论这门学科中，一个很重要的目标就是量化描述数据中含有多少信息。
为此，提出了熵的概念，记作 H
一个概率分布所对应的熵表达如下：

$$H = -\sum_{i=1}^N p(x_i) \cdot \log p(x_i)$$

KL散度(KL divergence)

全称：Kullback-Leibler Divergence
用途：比较两个概率分布的接近程度
在统计应用中，我们经常需要用一个简单的，近似的概率分布 f^* 来描述观察数据 D 或者另一个复杂的概率分布 f

不似则
过大。

这个时候，我们需要一个量来衡量我们选择的近似分布

f^* 相比原分布 f 究竟损失了多少信息量，这就是KL散度起作用的地方

KL散度的计算

现在，我们能够量化数据中的信息量了，就可以来衡量近似分布带来的信息损失了。
KL散度的计算公式其实是熵计算公式的简单变形。在原有概率分布 p 上，加入我们的近似概率分布 q ，计算他们的每个取值对对数的差：

$$D_KL(p||q) = \sum_{i=1}^N p(x_i) \cdot (\log p(x_i) - \log q(x_i))$$

KL散度的取值范围是 $[0, +\infty)$

值越大，说明两分布越近，差异越大。

熵：最小值 0，表示完全确定

最大值 $\log N$ ， N 是样本数量，表示均匀分布下的不确定性最大。

截断函数:

1. 截断函数的基本定义

截断函数通常具有以下形式：

- **下截断：**将函数值限制在某个最小值 a 以上。例如，给定一个函数 $f(x)$ ，其下截断版本可以定义为：

$$f_{\text{trunc}}(x) = \max(f(x), a)$$

这意味着当 $f(x) < a$ 时，函数值被截断为 a 。

- **上截断：**将函数值限制在某个最大值 b 以下。例如，给定一个函数 $f(x)$ ，其上截断版本可以定义为：

$$f_{\text{trunc}}(x) = \min(f(x), b)$$

这意味着当 $f(x) > b$ 时，函数值被截断为 b 。

- **双边截断：**同时对函数值设置上下限 a 和 b ，通常定义为：

$$f_{\text{trunc}}(x) = \begin{cases} a & \text{if } f(x) < a \\ f(x) & \text{if } a \leq f(x) \leq b \\ b & \text{if } f(x) > b \end{cases}$$

Lee2024(07): PPO 算法

<https://medium.com/analytics-vidhya/coding-ppo-from-scratch-with-pytorch-part-1-4-613dfc1b14c8>

代码解读 ↗

<https://github.com/ericyangyu/PPO-for-Beginners>

代码地址 ↗

1. 对于 simulator 的输出 acts_dimention，比如三个动作，输出的形式是 [0, 1, 2]，三个动作的概率嘛？

? actor 的输入是 obs_dim，输出是 action_dim 例：输入 [1.5] 的向量，输出 [1, 3] 的向量。

? 还是说 输入是 5 个神经元，输出是 1 个神经元，这一个神经元的输出范围是 [-2, 1, 0]

2. episode：一个回合，一场游戏从开始到结束。

timestep：一个回合中的多次时间切片，在这个时间片下，action_dim, observation_dim, reward.

iteration：一次对于模型的训练。

max_timesteps_per_episode：一个 episode 例，若游戏未成功，都停止。

3. 为什么调用已有奖励函数？ ✓ 用 NLLoss，与状态值网络逼近呢

自己设计“actor” 输入 obs_dim. 输出 1 维 action.

自己设计“critic” 输入 obs_dim，输出 1 维 reward.

actor $\xrightarrow{\text{action}}$ "env" $\xrightarrow{\text{obs}}$ critic $\xrightarrow{\text{V}}$ \oplus $\xrightarrow{\text{reward} - \text{V}}$

Pseudocode

policy function θ value function ϕ

Algorithm 1 PPO-Clip
1: Input: initial policy parameters θ_0 , initial value function parameters ϕ_0
2: for $k = 0, 1, 2, \dots$ do
3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards-to-go \hat{R}_t .
5: Compute advantage estimates, \hat{A}_t (using any method of advantage estimation) based on the current value function V_{ϕ_k} .
6: Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg \max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \min \left(\frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_k}(a_t | s_t)} A^{\pi_{\theta_k}}(s_t, a_t), g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.
7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg \min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T (V_{\phi}(s_t) - \hat{R}_t)^2,$$

typically via some gradient descent algorithm.
8: end for

<https://spinningup.openai.com/en/latest/algorithms/ppo.html>

Rewards-to-go 是干嘛模型。是激励函数。
变大，正反馈，变小，负反馈。

这个 $\pi_0(\alpha(t))$ 在 $t=0$ 是 $\{t\}$, \checkmark 是一个根群值.

这些多根李解簇

是剩下做 α 的根群.

Loc2024/10/09 : PPO代码2. PPO讲解视频的原配代码

https://github.com/RethinkFun/trian_ppo/tree/main/train_ppo

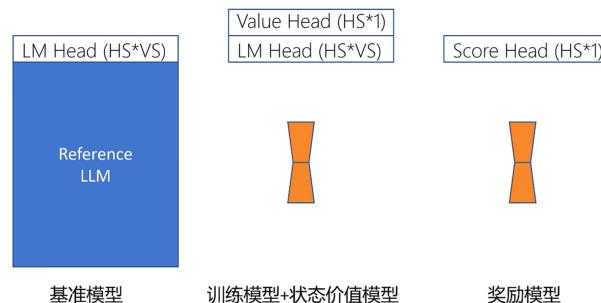
使用人工数据训练了一个 Reward model.

HS:hidden_size
VS:vocab_size

评估一个模型输出的好坏；
若是比输出一个好的回答容易。



对于标准的大模型，每个 token 在大模型的输出层，
会通过 LM-Head 层，LM-Head 层的输入维度
是 hidden_size，输出的维度是 vocab_size.



训练模型+状态价值模型 奖励模型

state：是截止到目前为止已经输出的序列。（for LLM base）

action：是下一个要生成的 token.

Reward：针对完整输出，给出一个得分。

优势函数：
我们还是训两个模型，一个生成 token AS

$A_\theta \rightarrow V_\theta$

1. 我现在已经有 Reference LLM 与 Reward LLM，要训练 Active LLM 与 State Value LLM 对吗？

2. 在 LLM 场景下，我是不是可以将 Active LLM 理解为 Transformer 的最后一个线性层？

3. Active LLM 的输出维度是 vocab_size，我是否可以理解为，
输出的向量维度还是 hidden_size，只是有 vocab_size 大小的多种可能输出。
(比如说：vocab_size = 2，我，你。 hidden_size = 3 {1, 0, 0} 或 [0, 10])

即 Active LLM 的输入神经元个数是 hidden_size，输出神经元个数是 3。

4. 没有看到 Reference LLM 与 Active LLM 的对比

PPPO 第2.

Active LLM 表示出

Active LLM

Active LLM 表示出

Active LLM
与
Active LLM
是“同一件事”

For batch_prompt in prompt_dataset:
batch_response = active_model.generate(batch_prompt)
batch_data = concat(batch_prompt, batch_response)
batch_scores = reward_model(batch_data)

Active LLM

与

Active LLM

是“同一件事”

Active LLM , Gate Value

与

Active LLM

是“同一件事”

Reference LLM

```
batch_all_probs, batch_probs, batch_all_values = active_model.forward_pass(batch_data)
ref_all_probs, ref_probs, ref_all_values = ref_model.forward_pass(batch_data)

klis = compute_KL(batch_all_probs, ref_all_probs)
rewards = compute_rewards(batch_scores, klis)
advantages = compute_advantages(batch_all_values, rewards)
returns = advantages + batch_all_values

for i in range(epoch):
    active_all_probs, active_probs, active_all_values = active_model.forward_pass(batch_data)

    loss_state_value = torch.mean((returns - active_all_values) ** 2)
    ratio = active_probs / batch_probs
    loss_ppo = torch.mean(-advantages * ratio)
    loss = loss_ppo + value_loss_rate * loss_state_value
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()
```

batch-all-probs : 对于每个 token 的所有可能的 token，预测概率。

batch-probs : token 真实的 prob.

batch-all-values : 每个 token 的真实值。

Lec20241009, 理清思路自己写个伪代码试一试。



是否应该这样？或许不用，我们自己是找工作，跑完上面流程OK。