PPO 算法

本章我们开始讲解强化学习中比较重要的 PPO 算法，它在相关应用中有着非常重要的地位，是一个里程碑式的算法。不同于 DDPG 算法，PPO 算法是一类典型的 Actor-Critic 算法，既适用于连续动作空间，也适用于离散动作空间。

PPO 算法是一种基于策略梯度的强化学习算法，由 OpenAI 的研究人员 Schulman 等人在 2017 年提出。PPO 算法的主要思想是通过在策略梯度的优化过程中引入一个重要性权重来限制策略更新的幅度，从而提高算法的稳定性和收敛性。PPO 算法的优点在于简单、易于实现、易于调参，应用十分广泛，正可谓“遇事不决 PPO ”。

PPO 的前身是 TRPO 算法，旨在克服 TRPO 算法中的一些计算上的困难和训练上的不稳定性。TRPO 是一种基于策略梯度的算法，它通过定义策略更新的信赖域来保证每次更新的策略不会太远离当前的策略，以避免过大的更新引起性能下降。然而，TRPO 算法需要解决一个复杂的约束优化问题，计算上较为繁琐。本书主要出于实践考虑，这种太复杂且几乎已经被淘汰的 TRPO 算法就不再赘述了，需要深入研究或者工作面试的读者可以自行查阅相关资料。 接下来将详细讲解 PPO 算法的原理和实现，希望能够帮助读者更好地理解和掌握这个算法。

重要性采样

在展开 PPO 算法之前，我们先铺垫一个概念，即重要性采样（ importance sampling ）。重要性采样是一种估计随机变量的期望或者概率分布的统计方法。它的原理也很简单，假设有一个函数 $f(x)$ ，需要从分布 $p(x)$ 中采样来计算其期望值，但是在某些情况下我们可能很难从 $p(x)$ 中采样，这个时候我们可以从另一个比较容易采样的分布 $q(x)$ 中采样，来间接地达到从 $p(x)$ 中采样的效果。这个过程的数学表达式如式 (1) 所示。

$$E_{p(x)}[f(x)] = \int_a^b f(x)\frac{p(x)}{q(x)}q(x)dx = E_{q(x)}\left[f(x)\frac{p(x)}{q(x)}\right] \tag{1}$$

对于离散分布的情况，可以表达为式 (2) 。

$$E_{p(x)}[f(x)] = \frac{1}{N}\sum f\left(x_i\right)\frac{p\left(x_i\right)}{q\left(x_i\right)} \tag{2}$$

这样一来原问题就变成了只需要从 $q(x)$ 中采样，然后计算两个分布之间的比例 $\frac{p(x)}{q(x)}$ 即可，这个比例称之为**重要性权重**。换句话说，每次从 $q(x)$ 中采样的时候，都需要乘上对应的重要性权重来修正采样的偏差，即两个分布之间的差异。当然这里可能会有一个问题，就是当 $p(x)$ 不为 0 的时候，$q(x)$ 也不能为 0，但是他们可以同时为 0 ，这样 $\frac{p(x)}{q(x)}$ 依然有定义，具体的原理由于并不是很重要，因此就不展开讲解了。

通常来讲，我们把这个 $p(x)$ 叫做目标分布， $q(x)$ 叫做提议分布（ Proposal Distribution ），那么重要性采样对于提议分布有什么要求呢? 其实理论上 $q(x)$ 可以是任何比较好采样的分布，比如高斯分布等等，但在实际训练的过程中，聪明的读者也不难想到我们还是希望 $q(x)$ 尽可能 $p(x)$，即重要性权重尽可能接近于 1 。我们可以从方差的角度来具体展开讲讲为什么需要重要性权重尽可能等于 1 ， 回忆一下方差公式，如式 (3) 所示。

$$Var_{x\sim p}[f(x)] = E_{x\sim p}\left[f(x)^2\right] - \left(E_{x\sim p}[f(x)]\right)^2 \tag{3}$$

结合重要性采样公式，我们可以得到式 (4) 。

$$\begin{aligned} Var_{x\sim q}\left[f(x)\frac{p(x)}{q(x)}\right] &= E_{x\sim q}\left[\left(f(x)\frac{p(x)}{q(x)}\right)^2\right] - \left(E_{x\sim q}\left[f(x)\frac{p(x)}{q(x)}\right]\right)^2 \\ &= E_{x\sim p}\left[f(x)^2\frac{p(x)}{q(x)}\right] - \left(E_{x\sim p}[f(x)]\right)^2 \end{aligned} \tag{4}$$

不难看出，当 $q(x)$ 越接近 $p(x)$ 的时候，方差就越小，也就是说重要性权重越接近于 1 的时候，反之越大。

其实重要性采样也是蒙特卡洛估计的一部分，只不过它是一种比较特殊的蒙特卡洛估计，允许我们在复杂问题中利用已知的简单分布进行采样，从而避免了直接采样困难分布的问题，同时通过适当的权重调整，可以使得蒙特卡洛估计更接近真实结果。

PPO 算法

既然重要性采样本质上是一种在某些情况下更优的蒙特卡洛估计，再结合前面 Actor-Critic 章节中我们讲到策略梯度算法的高方差主要来源于 Actor 的策略梯度采样估计，读者应该不难猜出 PPO 算法具体是优化在什么地方了。没错，PPO 算法的核心思想就是通过重要性采样来优化原来的策略梯度估计，其目标函数表示如式 (5) 所示。

$$
\begin{aligned}
J^{\mathrm{TRPO}}(\theta) &= \mathbb{E}\left[r(\theta)\hat{A}_{\theta_{\text{old}}}(s,a)\right] \\
r(\theta) &= \frac{\pi_\theta(a \mid s)}{\pi_{\theta_{\text{old}}}(a \mid s)}
\end{aligned}
\tag{5}
$$

这个损失就是置信区间的部分，一般称作 TRPO 损失。这里旧策略分布 $\pi_{\theta_{\text{old}}}(a \mid s)$ 就是重要性权重部分的目标分布 $p(x)$ ，目标分布是很难采样的，所以在计算重要性权重的时候这部分通常用上一次与环境交互采样中的概率分布来近似。相应地， $\pi_\theta(a \mid s)$ 则是提议分布，即通过当前网络输出的 `probs` 形成的类别分布 Catagorical 分布（离散动作）或者 Gaussian 分布（连续动作）。

读者们可能对这个写法感到陌生，似乎少了 Actor-Critic 算法中的 `logit_p`，但其实这个公式等价于式 (6) 。

$$
J^{\mathrm{TRPO}}(\theta) = E_{(s_t,a_t)\sim\pi_{\theta'}}\left[\frac{p_\theta(a_t \mid s_t)}{p_{\theta'}(a_t \mid s_t)} A^{\theta'}(s_t,a_t)\nabla \log p_\theta\left(a_t^n \mid s_t^n\right)\right] \tag{6}
$$

换句话说，本质上 PPO 算法就是在 Actor-Critic 算法的基础上增加了重要性采样的约束而已，从而确保每次的策略梯度估计都不会过分偏离当前的策略，也就是减少了策略梯度估计的方差，从而提高算法的稳定性和收敛性。

前面我们提到过，重要性权重最好尽可能地等于 1 ，而在训练过程中这个权重它是不会自动地约束到 1 附近的，因此我们需要在损失函数中加入一个约束项或者说正则项，保证重要性权重不会偏离 1 太远。具体的约束方法有很多种，比如 KL 散度、JS 散度等等，但通常我们会使用两种约束方法，一种是 clip 约束 ，另一种是 KL 散度。clip 约束定义如式 (7) 所示。

$$
J_{\text{clip}}(\theta) = \hat{\mathbb{E}}_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right] \tag{7}
$$

其中 ϵ 是一个较小的超参，一般取 0.1 左右。这个 clip 约束的意思就是始终将重要性权重 $r(\theta)$ 裁剪在 1 的邻域范围内，实现起来非常简单。

另一种 KL 约束定义如式 (8) 所示。

$$
J^{KL}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t - \beta \mathrm{KL}\left[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)\right]\right] \tag{8}
$$

KL 约束一般也叫 KL-penalty，它的意思是在 TRPO 损失的基础上，加上一个 KL 散度的惩罚项，这个惩罚项的系数 β 一般取 0.01 左右。这个惩罚项的作用也是保证每次更新的策略分布都不会偏离上一次的策略分布太远，从而保证重要性权重不会偏离 1 太远。在实践中，我们一般用 clip 约束，因为它更简单，计算成本较低，而且效果也更好。

到这里，我们就基本讲完了 PPO 算法的核心内容，其实在熟练掌握 Actor-Critic 算法的基础上，去学习这一类的其他算法是不难的，读者只需要注意每个算法在 Actor-Critic 框架上做了哪些改进，取得了什么效果即可。

一个常见的误区

在之前的章节中，我们讲过 on-policy 和 off-policy 算法，前者使用当前策略生成样本，并基于这些样本来更新该策略，后者则可以使用过去的策略采集样本来更新当前的策略。on-policy 算法的数据利用效率较低，因为每次策略更新后，旧的样本或经验可能就不再适用，通常需要重新采样。而 off-policy 算法由于可以利用历史经验，一般使用经验回放来存储和重复利用之前的经验，数据利用效率则较高，因为同一批数据可以被用于多次更新。但由于经验的再利用，可能会引入一定的偏见，但这也有助于稳定学习。但在需要即时学习和适应的环境中，on-policy 算法可能更为适合，因为它们直接在当前策略下操作。

那么 PPO 算法究竟是 on-policy 还是 off-policy 的呢？有读者可能会因为 PPO 算法在更新时重要性采样的部分中利用了旧的 Actor 采样的样本，就觉得 PPO 算法会是 off-policy 的。实际上虽然这批样本是从旧的策略中采样得到的，但我们并没有直接使用这些样本去更新我们的策略，而是使用重要性采样先将数据分布不同导致的误差进行了修正，即是两者样本分布之间的差异尽可能地缩小。换句话说，就可以理解为重要性采样之后的样本虽然是由旧策略采样得到的，但可以近似为从更新后的策略中得到的，即我们要优化的 Actor 和采样的 Actor 是同一个，因此 **PPO 算法是 on-policy 的。**

思考

为什么 DQN 和 DDPG 算法不使用重要性采样技巧呢?

DQN 和 DDPG 是 off-policy 算法，它们通常不需要重要性采样来处理不同策略下的采样数据。相反，它们使用目标网络和优势估计等技巧来提高训练的稳定性和性能。

PPO 算法原理上是 on-policy 的，但它可以是 off-policy 的吗，或者说可以用经验回放来提高训练速度吗?为什么？（提示：是可以的，但条件比较严格）

答：跟 A2C 一样，可以将经验回放与 PPO 结合，创建一个 PPO with Experience Replay (PPO-ER) 算法。在 PPO-ER 中，智能体使用经验回放缓冲区中的数据来训练策略网络，这样可以提高训练效率和稳定性。这种方法通常需要调整PPO的损失函数和采样策略，以适应 off-policy 训练的要求，需要谨慎调整。

PPO 算法更新过程中在将轨迹样本切分个多个小批量的时候，可以将这些样本顺序打乱吗？为什么?

将轨迹样本切分成多个小批量时，通常是可以将这些样本顺序打乱的，这个过程通常称为样本随机化（sample shuffling），这样做的好处有降低样本相关性、减小过拟合风险以及增加训练多样性（更全面地提高探索空间）。

为什么说重要性采样是一种特殊的蒙特卡洛采样?

原因有：**估计期望值**：蒙特卡洛方法的核心目标之一是估计一个随机变量的期望值。蒙特卡洛采样通过从分布中生成大量的样本，并求取这些样本的平均值来估计期望值。重要性采样也是通过从一个分布中生成样本，但不是均匀地生成样本，而是按照另一个分布的权重生成样本，然后使用这些带权重的样本来估计期望值。**改进采样效率**：重要性采样的主要目的是改进采样效率。当我们有一个难以从中采样的分布时，可以使用重要性采样来重新调整样本的权重，以使估计更准确。这类似于在蒙特卡洛采样中调整样本大小以提高估计的精确性。**权重分布**：在重要性采样中，我们引入了一个额外的权重分布，用于指导采样过程。这个权重分布决定了每个样本的相对贡献，以确保估计是无偏的。在蒙特卡洛采样中，权重通常是均匀分布，而在重要性采样中，权重由分布的比率（要估计的分布和采样分布之间的比例）决定。