重参数化与强化学习

为什么需要重参数化？随机函数的梯度计算问题

在机器学习领域，目标函数常常需要计算如下形式的随机函数：

$$L(\theta) = \mathbb{E}_{q_\theta(\mathbf{z})}(l(\theta, \mathbf{z})) = \int l(\theta, \mathbf{z}) q_\theta(\mathbf{z}) \mathrm{d}\mathbf{z}$$

其梯度为：

$$\begin{aligned}\nabla_\theta L(\theta) &= \nabla_\theta E_{q_\theta(\mathbf{z})}(l(\theta, \mathbf{z})) = \nabla_\theta \int l(\theta, \mathbf{z}) q_\theta(\mathbf{z}) \mathrm{d}\mathbf{z} \\ &= \int \nabla_\theta \ l(\theta, \mathbf{z}) q_\theta(\mathbf{z}) \mathrm{d}\mathbf{z} + \int l(\theta, \mathbf{z}) \nabla_\theta \ q_\theta(\mathbf{z}) \mathrm{d}\mathbf{z} \\ &= I_1 + I_2\end{aligned}$$

上式中的第一项可以使用蒙特卡洛法近似求解：

$$I_1 = \int \nabla_\theta \ l(\theta, \mathbf{z}) q_\theta(\mathbf{z}) \mathrm{d}\mathbf{z} \approx \frac{1}{S} \sum_{s=1, \mathbf{z_s} \sim q_\theta}^{S} \nabla_\theta \ l(\theta, \mathbf{z_s})$$

但第二项则无法使用朴素蒙特卡洛法来近似，因为它是关于一个概率分布的梯度，而蒙特卡洛法只能计算某个函数关于采样的概率分布的期望。这里一般有两种主流的办法来处理它：

- **得分函数估计 Score function estimator**

> 得分函数(score function)是对数似然函数关于概率分布的参数的梯度，即
>
> $$s_\theta(\mathbf{x}) = \nabla_\theta \log q_\theta(\mathbf{x})$$
>
> 通过使用对数导数技巧(log-derivative trick)，我们可以将概率密度函数的梯度，转化为概率密度函数和 得分函数乘积:
>
> $$\nabla_\theta \ q_\theta(\mathbf{z}) = q_\theta(\mathbf{z}) \frac{\nabla_\theta \ q_\theta(\mathbf{z})}{q_\theta(\mathbf{z})} = q_\theta(\mathbf{z}) \ \nabla_\theta \ \log q_\theta(\mathbf{z})$$

上文中所述的第二项 I_2 可以写成:

$$\begin{aligned}I_2 &= \int l(\theta, \mathbf{z}) \nabla_\theta \ q_\theta(\mathbf{z}) \mathrm{d}\mathbf{z} = \int l(\theta, \mathbf{z}) q_\theta(\mathbf{z}) \ \nabla_\theta \log q_\theta(\mathbf{z}) \mathrm{d}\mathbf{z} \\ &= \mathbb{E}_{q_\theta(\mathbf{z})}(l(\theta, \mathbf{z}) \nabla_\theta \ \log q_\theta(\mathbf{z})) \\ &\approx \frac{1}{S} \sum_{s=1, \mathbf{z_s} \sim q_\theta}^{S} l(\theta, \mathbf{z_s}) \ \nabla_\theta \ \log q_\theta(\mathbf{z_s})\end{aligned}$$

- **重参数化技巧 Reparameterization trick**

但是实践中，往往得分函数估计带来的计算方差太大(下文中的分析会介绍具体原因)。为了解决这个问题，我们可以使用重参数化方法。该方法被广泛地应用到例如变分自动编码器(VAE)这样的经典机器学习模型中。

重参数化怎么做？具体原理详解

我们发现，使用高斯分布这类可参数化的标准分布，它的分布的参数与采样的过程是可以独立分离的。它可以被分离为**一组独立的参数**与**一个新的随机变量**。分离后各个参数与新的随机变量无关。因而新的目标函数可以表示为新随机变量分布的期望，这种分离方法称为重参数化技巧。

高斯分布的参数化

这里以常用的高斯分布为例解释重参数化。假设一个随机变量 $x \in R^D$ 服从高斯分布：

$$x \sim N(\mu, \Sigma)$$

我们可以将这个高斯分布进一步拆解为来自于一个标准高斯分布 $\epsilon \sim N(0, I)$ 的线性变换：

$$x = \mu + L\,\epsilon \sim N(\mu, LL^T)$$

式中 L 为一个下三角方阵，而当进一步简化假设，假设各维度变量独立分布时，则可以将 L 退化为一个对角矩阵：

$$L = \begin{bmatrix} \sigma_{11} & & & \\ \sigma_{21} & \sigma_{22} & & \\ \vdots & \vdots & \ddots & \\ \sigma_{D1} & \sigma_{D2} & \cdots & \sigma_{DD} \end{bmatrix} \approx \begin{bmatrix} \sigma_{11} & & & \\ & \sigma_{22} & \ddots & \\ & & & \sigma_{DD} \end{bmatrix} = \mathrm{diag}(\boldsymbol{\sigma})$$

通过这种参数化方法，我们可以使用**有限**个自然参数来近似表征**复杂连续**的概率分布。

在上述例子中，对于这个高斯分布，仅需要 $(D + D)$ 个参数来描述即可。

当然，我们可以根据数据分布和需要，对概率分布进行多种形式的分解，比如 Gamma 分布，Beta 分布， 以及 Student 分布等等。这样我们可以把概率分布簇的参数与概率分布自身的随机性拆分和剥离，从而有益于计算和求导。

具体的实用例子，比如在 VAE 中，假设算法中的隐变量服从高斯分布 $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$，那么这个隐变量可以被重参数化表达为 $\mathbf{z} = \mu_\phi(\mathbf{x}) + \epsilon \odot \sigma_\phi(\mathbf{x})$，从而使目标函数从原来的 $\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}(f(\mathbf{z}))$ 变为 $\mathbb{E}_{q(\epsilon)}(f(z))$。

使用条件

重参数化法的使用条件是

- 目标函数 $l(\theta, z)$ 对参数 z 和 θ 都可微分
- 分布 z 的采样可以从某个独立分布 ϵ 获得，然后使用一组参数 θ 将其转化为分布 z ，即 $z = r(\theta, \epsilon)$

重参数化梯度

将重参数化后的形式代入，我们原先的随机目标函数可以变为：

$$
\begin{aligned}
L(\theta) &= \mathbb{E}_{q_\theta(z)}(l(\theta, z)) = \int l(\theta, z) q_\theta(z) \mathrm{d}z \\
&= \int l(\theta, r(\theta, \epsilon)) q(\epsilon) \mathrm{d}\epsilon = \mathbb{E}_{q(\epsilon)}(l(\theta, r(\theta, \epsilon)))
\end{aligned}
$$

那么我们就可以计算其梯度：

$$
\begin{aligned}
\nabla L(\theta) &= \nabla \mathbb{E}_{q(\epsilon)}(l(\theta, r(\theta, \epsilon))) = \mathbb{E}_{q(\epsilon)}(\nabla l(\theta, r(\theta, \epsilon))) \\
&\approx \frac{1}{S} \sum_{s=1, \epsilon_s \sim q}^{S} (\nabla l(\theta, r(\theta, \epsilon_s)))
\end{aligned}
$$

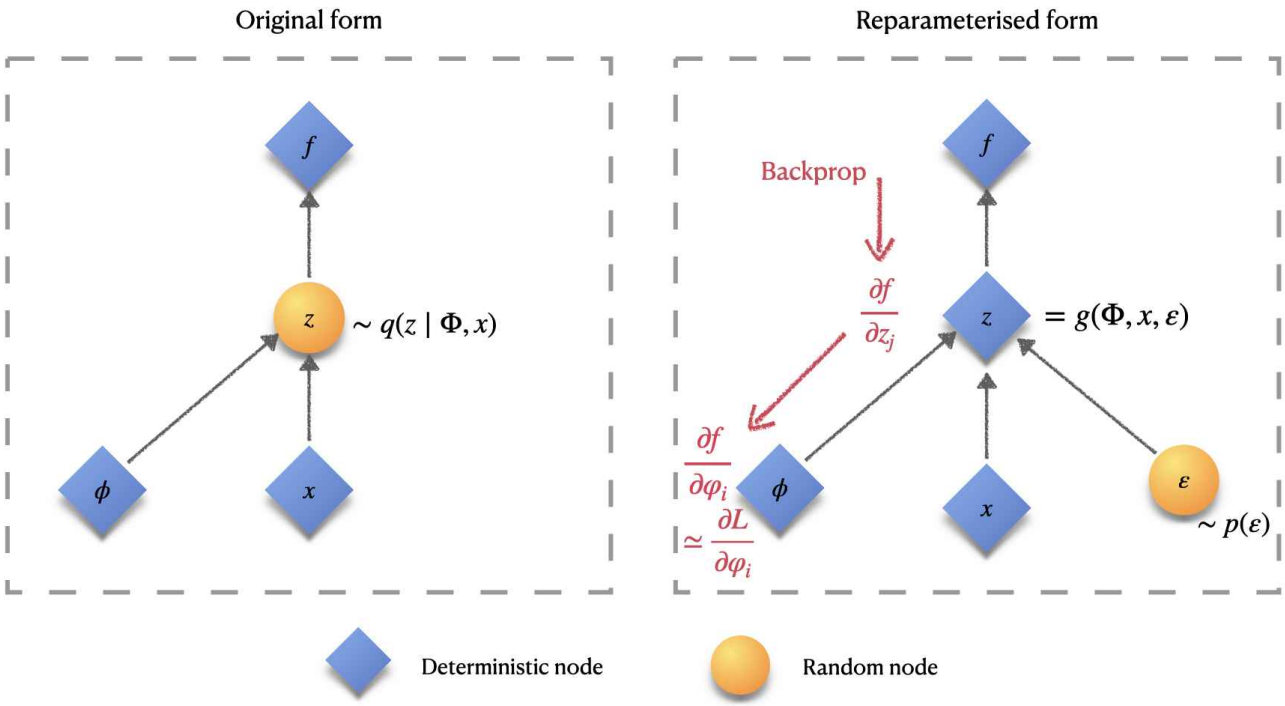
这个梯度称为重参数化梯度(reparameterization gradient / pathwise derivative) 。

为什么要用重参数化？两类优势

优势一：穿透随机性的求导

我们可以把概率分布重参数化后的参数，理解为概率分布的“坐标”，因而我们就可以把目标函数对于概率分布的梯度，转移到目标函数对这些坐标分量的梯度计算上去。对这些自然参数的求导有着很明确的数学含义，比如 $\frac{\partial}{\partial \mu}$ 是指概率分布的均值变化单位数值所带来的目标函数的变化量。

通过这种方法，我们可以近似认为：**计算某个函数对该分布的导数，等价于计算该函数对该概率分布自然参数的导数（如下图所示）**。而在很多机器学习算法的实际使用中，假如最终的目标函数与某个概率分布有关，那么对目标函数的梯度就可以一直向后传递至对该概率分布的自然参数中，而不受随机性带来的影响，即该这种计算导数的方式在关于这种随机性的统计意义上成立。由此，我们可以便捷地构建基于梯度下降优化的各种机器学习模型（例如 VAE），从而解决一系列实用性问题。



(图1：重参数化求导的原理解释图)

优势二：降低方差

通过重参数化，我们可以解决使用Score Function Method进行求取梯度时，可能导致的高方差问题[1]。具体来说，分别展开 Score Function Method 与重参数化两种方法的梯度公式，在Score Function Method中为：

$$\begin{aligned}\nabla L_{\mathrm{SF}}(\theta) &= \int \nabla\, l(\theta, z) q_\theta(z) \mathrm{d}z + \int l(\theta, z) \nabla\, q_\theta(z) \mathrm{d}z \\ &= \mathbb{E}_{q_\theta(z)}(\nabla\, l(\theta, z) + l(\theta, z) \nabla\, \log q_\theta(z)) \\ &= \mathbb{E}_{q_\theta(z)}(\frac{\partial\, l(\theta, z)}{\partial\, \theta} + \frac{\partial\, l(\theta, z)}{\partial\, z} \frac{\partial\, z}{\partial\, \theta} + l(\theta, z) \frac{\partial\, \log q_\theta(z)}{\partial \theta})\end{aligned}$$

在Reparameterization Method中为：

$$\begin{aligned}\nabla L_{\mathrm{Reparam}}(\theta) &= \nabla \int l(\theta, z) q_\theta(z) \mathrm{d}z = \nabla \int l(\theta, r(\theta, \epsilon)) q(\epsilon) \mathrm{d}\epsilon \\ &= \mathbb{E}_{q(\epsilon)}(\nabla\, l(\theta, r(\theta, \epsilon))) \\ &= \mathbb{E}_{q(\epsilon)}(\frac{\partial\, l(\theta, r)}{\partial\, \theta} + \frac{\partial\, l(\theta, r)}{\partial\, r} \frac{\partial\, r(\theta, \epsilon)}{\partial\, \theta})\end{aligned}$$

对比上述两式，我们发现 Score Function Method 会需要额外引入：

$$\mathbb{E}_{q_\theta(z)}(l(\theta, z) \nabla\, \log q_\theta(z))$$

当 $l(\theta, z)$ 和分布无关时，该项为零。

这项的存在会带来额外的梯度方差。因此一般来说：

$$\mathbb{V}_{q(\epsilon)}(\frac{\partial\, l(\theta, r)}{\partial\, \theta} + \frac{\partial\, l(\theta, r)}{\partial\, r} \frac{\partial\, r(\theta, \epsilon)}{\partial\, \theta}) \leq \mathbb{V}_{q_\theta(z)}(\frac{\partial\, l(\theta, z)}{\partial\, \theta} + \frac{\partial\, l(\theta, z)}{\partial\, z} \frac{\partial\, z}{\partial\, \theta} + l(\theta, z) \frac{\partial\, \log q_\theta(z)}{\partial \theta})$$

而重参数化之所以可以避免这一项，是因为使用了独立于分布参数的采样，故此项梯度为0。

（不过，我们可以构造某些特殊的函数，让上式中的各项产生相关性，从而使得，重参数化法的梯度方差比 Score Function Method 更小这个命题不成立，但是在统计意义上，即实际应用中的绝大多数情况，使用重参数化可以有效降低方差）

此外，在实践中，较低的采样样本数会带来较大的方差，但较高的采样样本数就会降低计算的效率。因此，如果使用重参数化技巧**直接**对目标函数进行求导，降低梯度的方差，就可以让我们使用更少的采样样本，从而也对提升训练效率也有帮助。

强化学习中的重参数化

一个实例：SAC

在强化学习 SAC 算法中[2]，策略通过 SAC 中定义的 soft value function（即引入最大熵的价值函数）来指导策略优化，具体来说，SAC 算法希望找到可以最大化 soft value function 的策略，即：

$$
\begin{aligned}
\pi^* &= \arg\max V_\pi(s) \\
V_\pi(s) &= \mathbb{E}_{a\sim\pi}(Q_\pi(s,a)) + \alpha H(\pi(\cdot|s)) \\
&= \mathbb{E}_{a\sim\pi}(Q_\pi(s,a) - \alpha\log(\pi(a|s)))
\end{aligned}
$$

在实现中，首先需要策略部分（Actor）输出当前状态对应的动作，然后交给价值函数（Critic）判断好坏得到目标函数并用梯度下降进行优化，梯度将会从 Critic 沿着动作一路回传到 Actor。

在连续动作空间上，我们需要对策略使用高斯分布进行重参数化，并使用 $\tanh$ 函数压缩最终动作的输出范围到 [-1, 1]，即采样到的动作通过下式获得：

$$
a_\theta(s,\epsilon) = \tanh\left(\mu_\theta(s) + \epsilon\cdot\ \sigma_\theta(s)\right), \epsilon\ \sim\ \mathcal{N}(0,I)
$$

> 需要注意的是，这里需要一个额外的矫正操作，因为使用 $\tanh$ 函数压缩最终动作 a 会改变原本无界动作 a_u 的概率密度函数，即：
>
> $$
> \pi(a|s) = p(a_u|s)|\det\frac{\mathrm{d}a}{\mathrm{d}a_u}|^{-1}
> $$
>
> 其中，
>
> $$
> \frac{\mathrm{d}a}{\mathrm{d}a_u} = \mathrm{diag}(1-\tanh^2(a_u))
> $$
>
> 所以代码实现中，我们需要修改压缩后的概率密度函数的 log prob 为：
>
> $$
> \log\pi(a|s) = \log p(a_u|s) - \sum_{d=1}^{D}\log\left(1-\tanh^2(a_{u,d})\right)
> $$

考虑 SAC 算法中使用的 double Q-trick，最终重参数化后的策略学习的目标函数为：

$$
\begin{aligned}
\theta^* &= \arg\max\ J(\theta) \\
&= \arg\max \mathop{\mathbb{E}}_{\substack{s\sim\mathcal{D} \\ \epsilon\sim\mathcal{N}(0,I)}}\left(\min_{j=1,2} Q_{\phi_j}(s,a_\theta(s,\epsilon)) - \alpha\log(\pi_\theta(a_\theta(s,\epsilon)|s))\right)
\end{aligned}
$$

对于一个batch为 B 的数据，这一批次数据产生的梯度为：

$$\nabla_\theta J(\theta) = \nabla_\theta \frac{1}{|B|} \sum_{s \in B} (\min_{j=1,2} Q_{\phi_j}(s, a_\theta(s,\epsilon)) - \alpha \log(\pi_\theta(a_\theta(s,\epsilon)|s)))$$

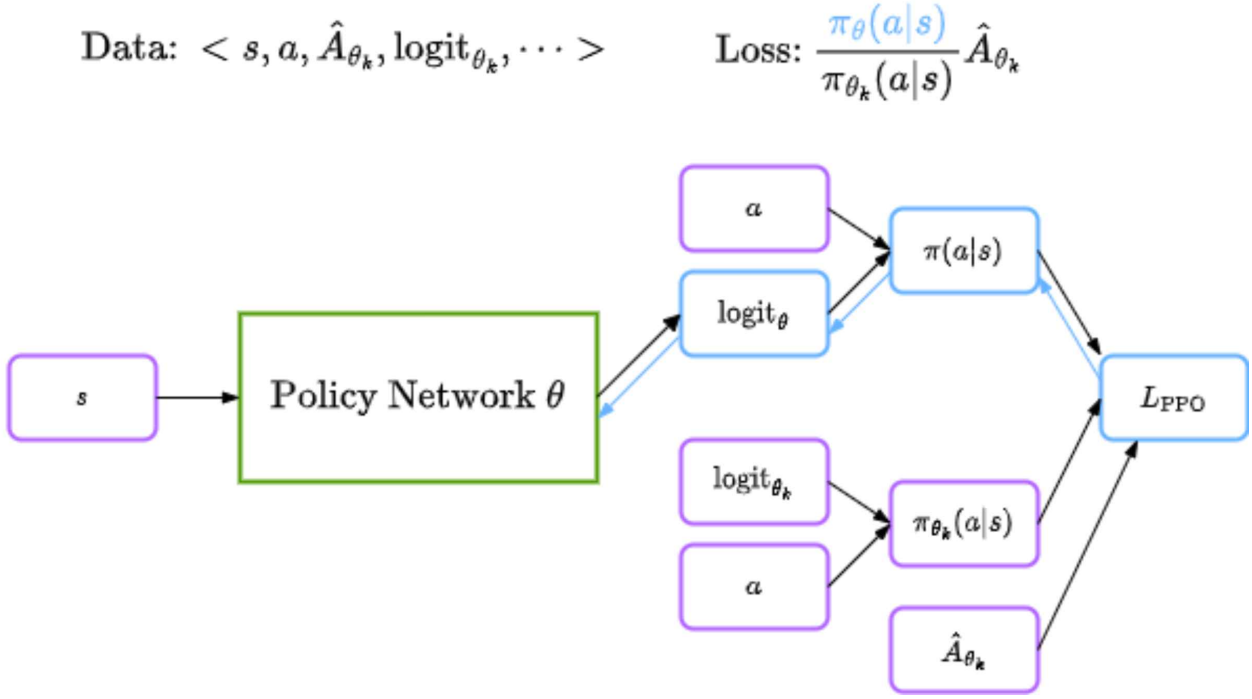
$$= \frac{1}{|B|} \sum_{s \in B} (\nabla_\theta [\min_{j=1,2} Q_{\phi_j}(s, a_\theta(s,\epsilon))] - \alpha \nabla_\theta \log(\pi_\theta(a_\theta(s,\epsilon)|s)))$$

PPO 与重参数化的关系

参考和回顾PPO的目标函数：

$$\mathbb{E}_t[\min(\frac{p_\theta(a_t|s_t)}{p_{\theta_k}(a_t|s_t)} \hat{A}^{\theta_k}(s_t,a_t), \text{clip}(\frac{p_\theta(a_t|s_t)}{p_{\theta_k}(a_t|s_t)}, 1-\epsilon, 1+\epsilon)\hat{A}^{\theta_k}(s_t,a_t))]$$

我们可以发现虽然公式中存在着类似于 $p_\theta(a|s)$ 的形式，但是与 SAC 中不同，训练时用到的动作 a_t 并不由将要更新的策略函数的参数 θ 生成，而是由旧策略（收集数据的策略）函数的参数 θ_k 生成。因此，在 PPO 中仅是借用重参数化的形式，将策略函数为参数化某个概率函数（比如连续动作空间使用高斯分布），即可让梯度通过概率函数的定义式直接回传。另外，公式中的动作 a_t ，状态 s_t ，优势函数估计 A^{θ_k} 只是一些为了更新参数 θ 而存在的数据，它们本身并不回传任何梯度。



(图2：PPO 策略部分优化计算图，只有蓝色部分回传梯度，紫色部分只是参与计算)

参考文献

[1] Xu M, Quiroz M, Kohn R, et al. Variance reduction properties of the reparameterization trick[C]//The 22nd International Conference on Artificial Intelligence and Statistics. PMLR, 2019: 2711-2720.

[2] Haarnoja T, Zhou A, Abbeel P, et al. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor[C]//International conference on machine learning. PMLR, 2018: 1861-1870.