重参数化技巧与强化学习

为什么需要重参数化技巧？随机函数的梯度计算问题

在机器学习问题中，目标函数常常写成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：

$$\begin{aligned} \nabla_\theta L(\theta) &= \nabla_\theta E_{q_\theta(\mathbf{z})}[l(\theta, \mathbf{z})] = \nabla_\theta \int l(\theta, \mathbf{z}) q_\theta(\mathbf{z}) \mathrm{d}\mathbf{z} \\ &= \underbrace{\int \nabla_\theta \ l(\theta, \mathbf{z}) q_\theta(\mathbf{z}) \mathrm{d}\mathbf{z}}_{A} + \underbrace{\int l(\theta, \mathbf{z}) \nabla_\theta \ q_\theta(\mathbf{z}) \mathrm{d}\mathbf{z}}_{B} \end{aligned}$$

上式中的第一项可以使用蒙特卡洛法近似求解：

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

如果随机变量 $\mathbf{z}$ 的分布 $q_\theta(\mathbf{z})$ 与优化的参数 θ 无关, 则 $B = 0$，那么上述的蒙特卡洛法近似是对梯度的无偏估计; 例如最常见的监督学习中，数据集中样本 $\mathbf{z} = (\mathbf{x}, \mathbf{y})$ 的分布 p_{data} 固定而不受模型参数 θ 的影响，那么，

$$\nabla_\theta \ L(\theta) = \ A = \ \mathbb{E}_{p_{\text{data}}(\mathbf{x}, \mathbf{y})}[\nabla_\theta \log p_\theta(\mathbf{y}|\mathbf{x})] \approx \frac{1}{S} \sum_{s=1}^{\mathcal{S}} \nabla_\theta \ \log p_\theta(\mathbf{y}|\mathbf{x})$$

但如果随机变量 $\mathbf{z}$ 的分布 $q_\theta(\mathbf{z})$ 与优化的参数 θ 有关(例如在强化学习中，轨迹变量 $\mathbf{z} = \boldsymbol{\tau}$ 的分布 $q_\theta(\boldsymbol{\tau})$ 与策略的参数 θ 相关; 又例如在变分自编码器中，隐变量 $\mathbf{z}$ 的分布 $q_\phi(\mathbf{z}|\ \mathbf{x})$ 与变分后验的参数 ϕ 相关), 则 $I_2 \ \neq \ 0$, 那么用蒙特卡洛法近似梯度显然就会忽视 I_2 对于梯度的贡献。

在这种情况下，一般有两种主流的办法来优化概率分布中的参数 θ ：

- **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:

$$\nabla_\theta\ q_\theta(\mathbf{z}) = q_\theta(\mathbf{z})\ \frac{\nabla_\theta\ q_\theta(\mathbf{z})}{q_\theta(\mathbf{z})}\ = q_\theta(\mathbf{z})\ \nabla_\theta\ \log q_\theta(\mathbf{z})$$

根据对数导数技巧，上文中所述的第二项 I_2 可以写成:

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

- **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。

重参数化技巧怎么做？具体原理详解

重参数化方法将随机变量从目标分布中 $\mathbf{z} \sim q_\theta(\mathbf{z})$ 的采样过程分解为两步:

1. 构造一个与 θ **无关**的辅助分布，从中采样一个随机噪声 $\boldsymbol{\epsilon} \sim q(\boldsymbol{\epsilon})$;
2. 通过一个与 θ **有关的可导变换** $\mathbf{z}\ =\ r(\theta,\ \boldsymbol{\epsilon})$，把随机噪声变换为目标分布中的随机变量。

例如一个多元随机变量 $\mathbf{z}$ 服从高斯分布:

$$\mathbf{z}\ \sim \mathcal{N}(\boldsymbol{\mu},\ \mathrm{diag}(\boldsymbol{\sigma}))\,,$$

其中 $\theta\ =\ (\boldsymbol{\mu},\ \boldsymbol{\sigma})$; 那么我们就可以把随机变量 $\mathbf{z}$ 重参数化为:

$$\mathbf{z} = r((\boldsymbol{\mu},\ \boldsymbol{\sigma}),\ \boldsymbol{\epsilon})\ =\ \boldsymbol{\mu}\ +\ \boldsymbol{\epsilon}\ \odot\ \boldsymbol{\sigma},\ \boldsymbol{\epsilon} \sim \mathcal{N}(0,\ I)$$

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如下所示，部分细节需要参考本文附录:

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

那么我们就可以计算其梯度:

$$\begin{aligned} \nabla_\theta L(\theta) = \nabla_\theta \mathbb{E}_{q(\boldsymbol{\epsilon})}[l(\theta, r(\theta, \boldsymbol{\epsilon}))] &= \mathbb{E}_{q(\boldsymbol{\epsilon})}[\nabla_\theta l(\theta, r(\theta, \boldsymbol{\epsilon}))] \\ &\approx \frac{1}{S} \sum_{s=1, \boldsymbol{\epsilon}_s \sim q}^{\mathcal{S}} \nabla_\theta l(\theta, r(\theta, \boldsymbol{\epsilon}_s)) \end{aligned}$$

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

使用条件

重参数化法的使用条件是

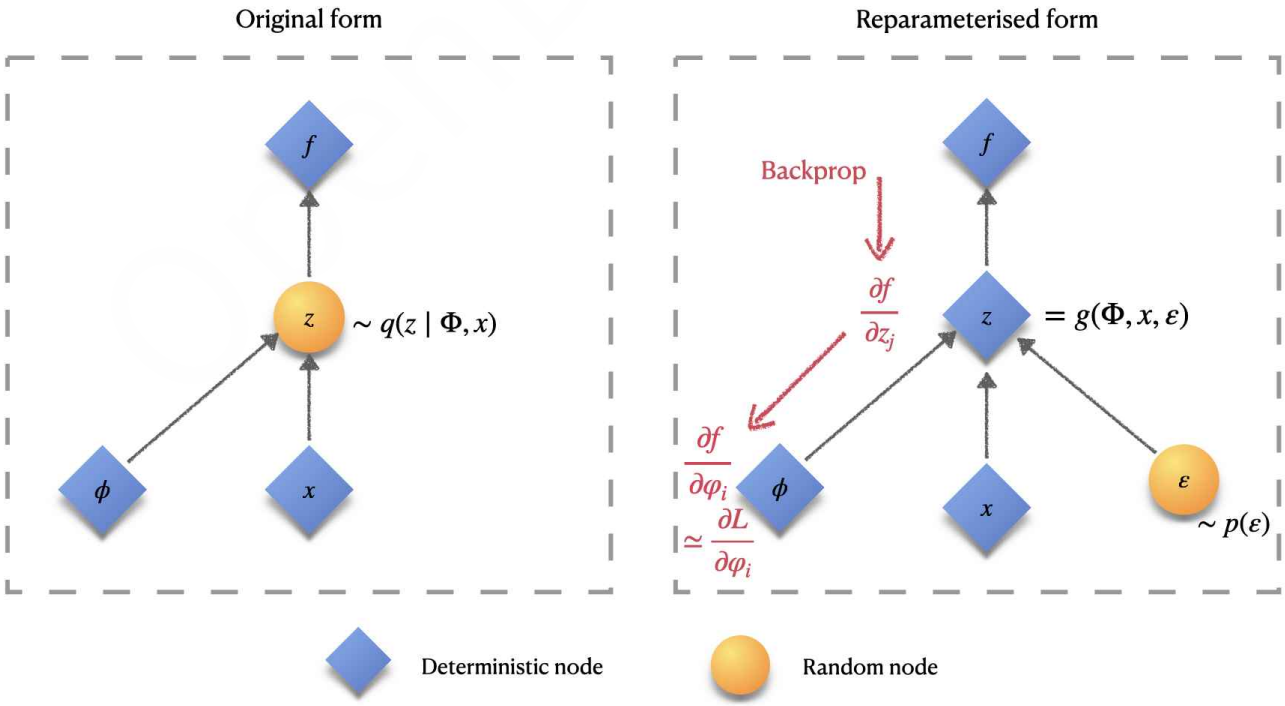
- 目标函数 $l(\theta, \mathbf{z})$ 对 $\mathbf{z}$ 可微分
- 分布 z 的采样可以从某个独立分布 ϵ 获得，然后使用一组参数 θ 将其转化为分布 $\mathbf{z}$ ，即 $\mathbf{z} = r(\theta, \epsilon)$

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，我们可以解决使用得分函数估计进行求取梯度时，可能导致的高方差问题 [1]。具体来说，分别展开 得分函数估计与重参数化两种方法的梯度公式，在得分函数估计中为：

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

在重参数化技巧中为：

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

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：

$$\pi^* = \arg\max V_\pi(s)$$

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

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：

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

对于一个batch为 B 的数据，这一批次数据产生的梯度为：

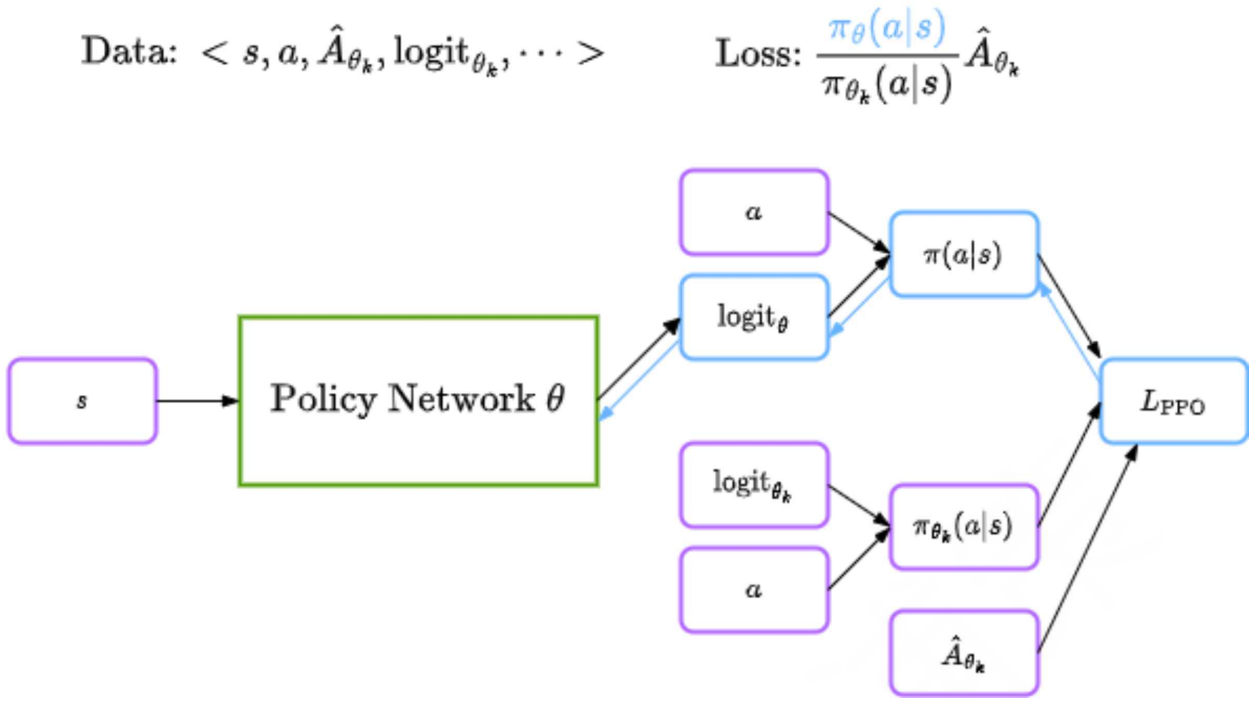
$$\begin{aligned}\nabla_\theta J(\theta) &= \nabla_\theta \ \frac{1}{|B|}\sum_{s\ \sim\ \mathcal{B}}(\min_{j=1,2} Q_{\phi_j}(s, a_\theta(s,\epsilon)) - \alpha\log(\pi_\theta(a_\theta(s,\epsilon)|s))) \\ &= \frac{1}{|B|}\sum_{s\ \in\ \mathcal{B}}(\nabla_\theta[\min_{j=1,2} Q_{\phi_j}(s, a_\theta(s,\epsilon))]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

PPO 与重参数化的关系

参考和回顾PPO的目标函数：

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

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)

附录

A 概率分布与记法

一般，我们将符号 p 与 q 用于标记关于某个变量的概率分布函数。对于连续变量，其为概率密度函数(probability density function, PDF)，而对于离散变量，其为概率质量函数(probability mass function, PMF)。

不过需要注意的是，与一般意义上的函数记法里， $f(x)$ 与 $f(y)$ 代表的是同一个函数所不同， $p(x)$ 与 $p(y)$ 代表的是两个不一样的概率分布函数。这其实是因为大部分教材和文献，出于简洁化符号的考虑，将 $p_X(x)$ 与 $p_Y(y)$ 进行缩写导致的。本文中对概率密度函数的表示也进行了缩写处理，大家需要注意 $q(\epsilon)$ 与 $q(z)$ 并非同一个概率分布，而是关于各自变量的分布。

此外，在机器学习和变分推断(variational inference)理论中，对于同一个变量，使用 p 或 q 作为概率分布的符号，其含义约定成俗上往往存在不同。在文献中，使用 q 作为变量 x 的概率分布的符号，即 $q(x)$ ，一般用于表示该概率分布是人为构造的变分概率分布(variational distribution)。它们往往是由模型生成的，用于优化或近似。使用 p 作为变量 x 的概率分布的符号，即 $p(x)$ ，一般用于表示该概率分布是数据本身的真实的概率分布，它们往往难以直接获得相应的概率分布函数，可能仅能获取它的采样结果，也可能仅能获得它的少部分的样本。
本文中，与大众习惯保持一致，概率分布 q_θ 中的 θ 用于表示构造这个概率分布的模型参数。对应的， $q(\epsilon)$ 的含义是，构造了一个关于 ϵ 的分布，它没有构造参数，因为它是一个高斯分布， $\mathcal{N}(0, I)$ 。

B 重参数化证明细节说明

为了证明改变随机变量前后，以下公式仍然成立，需要使用以下的一些概率论与测度的知识：

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

概率空间 $(\Omega, \mathcal{F}, \mathcal{P})$ [3] 是一个测度为1的空间，其中样本空间 Ω，事件集合 $\mathcal{F}$，和测度函数 $\mathcal{P}$，有：

$$\mathcal{P}(\Omega) = 1$$

对于任意的事件 A，或是这些事件的并集，$A \in \mathcal{F}$，有 $\mathcal{P}(A) \in [0, 1]$。

对于随机变量 x 和 y，如果有 $y = f(x)$ 始终成立，那么对于所有 $x = c$ 的事件 $A_{x=c}$，显然与 $y = f(x) = f(c)$ 是同一类事件 $A_{y=f(c)}$，所以其概率测度 [4] 相同。对于这些事件的并集也相同。

那么对于连续的随机变量的任意此类事件的概率测度，则有如下公式成立：

$$\mathcal{P}(A_{x=c}) = \int_{A_{x=c}} p_X(x) \mathrm{d}x = \int dp = \int_{A_{y=f(c)}} p_Y(y) \mathrm{d}y = \mathcal{P}(A_{y=f(c)})$$

其中 $p_X(x)$ 和 $p_Y(y)$ 是对应的连续变量的概率密度，$\mathrm{d}p$ 为对应的概率质量。

因为上式对于任意基本事件范围的求和或积分都成立，那么将等式两边添加任意相同的权重系数 $l(y)$ 之后，其求和或积分也成立，所以有：

$$\int l(y) p_Y(y) \mathrm{d}y = \int l(f(x)) p_X(x) \mathrm{d}x$$

于是对应着所需证明的等式成立。

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.

[3] https://en.wikipedia.org/wiki/Probability_space

[4] https://en.wikipedia.org/wiki/Probability_measure