机器学习与深度学习面试系列十五（可微松弛和重参数化）

为什么需要可微松弛和重参数化?

在深度学习中，我们几乎都借助于梯度下降来进行优化，正如前面的文章说的，大部分深度学习模型都属于可微编程。可微编程要求各个模块都是可微的，或“几乎处处可微”的。对于“几乎处处可微”的情况，我们需要对个别不可微的情况额外处理。这种情况在深度学习中很常见，处理方式包括可微松弛(differentiable relaxations)和重参数化(reparameterisations)。可微松弛就是用一个连续处处可微的函数去近似目标函数，重参数化就是重写函数以提取随机变量，后面具体举例来说明。

softmax解读?

在分类问题中，我们常常在网络的最后一层对logits转为类别的概率分布。实际上，从名字上我们大概就可以猜到，它其实是对argmax函数的一种松弛。

例如：我们正在做一个四分类的神经网络，在最后一层有4个节点，输出分别是 $[1.1, 4.0, -0.1, 2.3]$ ，我们要想得到它属于哪个分类，符合直觉的做法应该是直接做argmax函数。但是argmax函数我们很难求其梯度，于是采用softmax这样一种松弛形式来代替argmax，方便基于梯度的优化器对网络进行优化。

考虑带有temperature parameter T的softmax函数： $softmax(x/T)_i = \frac{e^{x_i/T}}{\sum_{j=1}^{K} e^{x_j/T}}$ 。

$\boldsymbol{x} =$	1.1	4.0	-0.1	2.3
$\text{softmax}(\boldsymbol{x}/1.0) =$	0.044	0.797	0.013	0.146
$\text{softmax}(\boldsymbol{x}/0.8) =$	0.023	0.868	0.005	0.104
$\text{softmax}(\boldsymbol{x}/0.6) =$	0.008	0.937	0.001	0.055
$\text{softmax}(\boldsymbol{x}/0.4) =$	6.997e-04	9.852e-01	3.484e-05	1.405e-02
$\text{softmax}(\boldsymbol{x}/0.2) =$	5.042e-07	9.998e-01	1.250e-09	2.034e-04

可以看到，T越小，得到的向量越接近于 $[0, 1, 0, 0]$ 。如果我们希望直接得到分类，可以再乘上类别的索引向量，即 $softmax(x/0.2)[0, 1, 2, 3] = 1$ ，这样我们得到这个样本的类别应该是1。
$T \to 0 \quad softmax(x/T)[0, 1, , 2 \ldots N] \approx argmax(x)$ 。

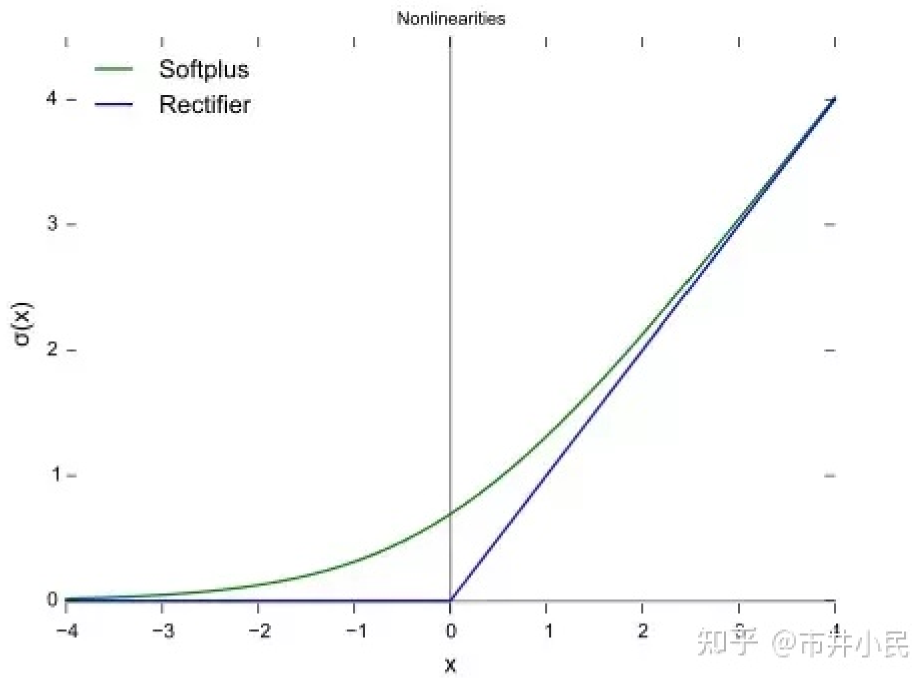
ReLU松弛?

ReLU函数: $f(x)=max(0,x)$ ，它是连续的并几乎处处可导。但是在x=0处，它是不可导的，通常我们的处理手段是取其次梯度:

$$f'(x)=\begin{cases}1, & if \quad x>0 \\ [0,1], & if \quad x=0 \\ 0, & if \quad x<0\end{cases}$$

在x=0处，f'(x)取[0, 1]都可以，工程上为了方便直接取 f'(0) = 0。

实际上，ReLU有一个松弛函数: $Softplus(x)=ln(1+e^x)$ ，它的导函数是sigmoid函数。



可以看到它于ReLU的图像非常接近，并且在x=0处也是可微的，这使它可以很好的作为ReLU的松弛函数。实际中我们并不常用它，因为它的计算和求导比ReLU复杂，在深度学习训练中，需要大量前向计算和反向求导，这将带来一笔不小的开销。

L1正则化松弛?

对于含L1正则化的线性回归（又称Lasso回归）问题来说， $f(x)=g(x)+||w||_1$ ， $||w||_1$ 在w=0处是不可导的，所以不能被大部分优化器直接优化。西瓜书中介绍过可以用近端梯度下降(Proximal Gradient Descent, 简称 PGD)的方法来进行优化。

其实我们还可以用L1的松弛函数Huber来解决: $z_i=\begin{cases}0.5(x_i-y_i)^2, & if \quad |x_i-y_i|<1 \\ |x_i-y_i|-0.5 & otherwise\end{cases}$ ，即在w在接近0的区间使用L2正则化，而在两边的区间使用L1正则化。

重参数化如何解决随机操作的不可微问题?

假设我们要对下式进行优化：

$$E_{z\sim P_\theta(z)}[f(z)]$$

由于Sample操作是不可微的，我们无法通过梯度传播实现对 P_θ 的优化，我们要将关于 Z 的期望转化为另一个随机变量的期望：

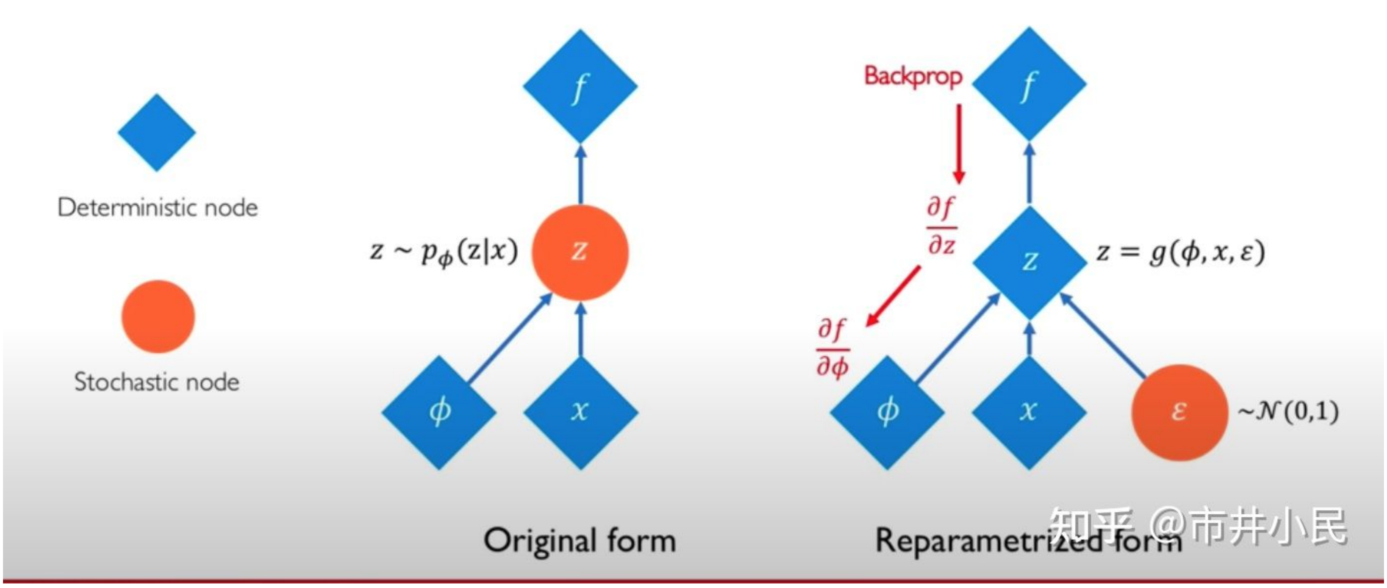
$$E_{z\sim P_\theta(z)}[f(z)] = E_{\epsilon\sim P(\varepsilon)}[f(g_\theta(\epsilon))]$$

其中， g_θ 是转化函数，该函数能将 ϵ 映射到 z ，从而通过对 ϵ 的采样实现对 z 的采样。利用链式求导法则：

$$\frac{\partial}{\partial\theta}E_{z\sim P_\theta(z)}[f(z)] = \frac{\partial}{\partial\theta}E_{\epsilon\sim P(\varepsilon)}[f(g_\theta(\epsilon))] = E_{\epsilon\sim P(\varepsilon)}[\frac{\partial f}{\partial g}\cdot\frac{\partial g}{\partial\theta}]$$

$g_\theta(\epsilon)$ 为一个确定性函数，就是说 $g_\theta(\epsilon)$ 是可以求对 θ 的梯度的。这样 z 和参数 θ 的关系从采样关系变为确定性关系，使得 $z\sim P_\theta(z)$ 的随机性独立于参数 θ，从而可以求 z 关于 θ 的导数。Reparameterization的关键在于如何确定转换函数 $g_\theta(\epsilon)$ 。

我们以变分自编码器为例，来说明这个问题：



左边是朴素的VAE结构，z是由从推断网络采样得到的隐变量，由于采样操作是不可微的，导致反向传播算法到z节点时，无法继续传播，优化参数 ϕ 。假设 $q_\phi(z|x)$ 为正态分布 $N(\mu_I I,\sigma_I^2 I)$ ，其中 $\{\mu_I,\sigma_I\}$ 是推断网络 $f_\phi(x)$ 的输出，它依赖于参数 ϕ，我们可以重参数化来保持整个网络可微： $z=\mu_I+\sigma_I\odot\epsilon$， 其中 $\epsilon\in N(0,I)$ 。如图右侧所示。