13.2.5 再参数化

再参数化（Reparameterization）是将一个函数 $f(\theta)$ 的参数 θ 用另外一组参数表示 $\theta = g(\vartheta)$，这样函数 $f(\theta)$ 就转换成参数为 ϑ 的函数 $\hat{f}(\vartheta) = f\big(g(\vartheta)\big)$. 再参数化通常用来将原始参数转换为另外一组具有特殊属性的参数. 比如当 θ 为一个很大的矩阵时，可以使用两个低秩矩阵的乘积来再参数化，从而减少参数量.

再参数化的另一个例子是逐层归一化，参见第7.5节.

在公式(13.21)中，期望 $\mathbb{E}_{\boldsymbol{z}\sim q(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\phi})}\Big[\log p(\boldsymbol{x}|\boldsymbol{z};\theta)\Big]$ 依赖于分布 q 的参数 $\boldsymbol{\phi}$. 但是，由于随机变量 $\boldsymbol{z}$ 采样自后验分布 $q(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\phi})$，它们之间不是确定性关系，因此无法直接求解 $\boldsymbol{z}$ 关于参数 $\boldsymbol{\phi}$ 的导数. 这时，我们可以通过再参数化方法来将 $\boldsymbol{z}$ 和 $\boldsymbol{\phi}$ 之间随机性的采样关系转变为确定性函数关系.

我们引入一个分布为 $p(\epsilon)$ 的随机变量 ϵ，期望 $\mathbb{E}_{\boldsymbol{z}\sim q(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\phi})}\Big[\log p(\boldsymbol{x}|\boldsymbol{z};\theta)\Big]$ 可以重写为

$$\mathbb{E}_{\boldsymbol{z}\sim q(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\phi})}\Big[\log p(\boldsymbol{x}|\boldsymbol{z};\theta)\Big] = \mathbb{E}_{\epsilon\sim p(\epsilon)}\Big[\log p(\boldsymbol{x}|g(\boldsymbol{\phi},\epsilon);\theta)\Big], \tag{13.25}$$

其中 $\boldsymbol{z} \triangleq g(\boldsymbol{\phi},\epsilon)$ 为一个确定性函数.

假设 $q(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\phi})$ 为正态分布 $N(\boldsymbol{\mu}_I, \boldsymbol{\sigma}_I^2\boldsymbol{I})$，其中 $\{\boldsymbol{\mu}_I, \sigma_I\}$ 是推断网络 $f_I(\boldsymbol{x};\boldsymbol{\phi})$ 的输出，依赖于参数 $\boldsymbol{\phi}$，我们可以通过下面方式来再参数化：

$$\boldsymbol{z} = \boldsymbol{\mu}_I + \boldsymbol{\sigma}_I \odot \epsilon, \tag{13.26}$$

参见习题13-1.

其中 $\epsilon \sim \mathcal{N}(\boldsymbol{0},\boldsymbol{I})$. 这样 $\boldsymbol{z}$ 和参数 $\boldsymbol{\phi}$ 的关系从采样关系变为确定性关系，使得 $\boldsymbol{z}\sim q(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\phi})$ 的随机性独立于参数 $\boldsymbol{\phi}$，从而可以求 $\boldsymbol{z}$ 关于 $\boldsymbol{\phi}$ 的导数.

13.2.6 训练

通过再参数化，变分自编码器可以通过梯度下降法来学习参数，从而提高变分自编码器的训练效率.

给定一个数据集 $\mathcal{D} = \{\boldsymbol{x}^{(n)}\}_{n=1}^N$，对于每个样本 $\boldsymbol{x}^{(n)}$，随机采样 M 个变量 $\epsilon^{(n,m)}, 1 \leq m \leq M$，并通过公式(13.26)计算 $\boldsymbol{z}^{(n,m)}$. 变分自编码器的目标函数近似为

$$\mathcal{J}(\boldsymbol{\phi},\theta|\mathcal{D}) = \sum_{n=1}^{N}\left(\frac{1}{M}\sum_{m=1}^{M}\log p(\boldsymbol{x}^{(n)}|\boldsymbol{z}^{(n,m)};\theta) - \mathrm{KL}\Big(q(\boldsymbol{z}|\boldsymbol{x}^{(n)};\boldsymbol{\phi}),\mathcal{N}(\boldsymbol{z};\boldsymbol{0},\boldsymbol{I})\Big)\right). \tag{13.27}$$

如果采用随机梯度方法，每次从数据集中采集一个样本 $\boldsymbol{x}$ 和一个对应的随机变量 ϵ，并进一步假设 $p(\boldsymbol{x}|\boldsymbol{z};\theta)$ 服从高斯分布 $\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_G,\lambda\boldsymbol{I})$，其中 $\boldsymbol{\mu}_G = f_G(\boldsymbol{z};\theta)$ 是生成网络的输出，λ 为控制方差的超参数，则目标函数可以简化为

$$\mathcal{J}(\boldsymbol{\phi},\theta|\boldsymbol{x}) = -\frac{1}{2}\|\boldsymbol{x}-\boldsymbol{\mu}_G\|^2 - \lambda\,\mathrm{KL}\Big(\mathcal{N}(\boldsymbol{\mu}_I,\boldsymbol{\sigma}_I),\mathcal{N}(\boldsymbol{0},\boldsymbol{I})\Big), \tag{13.28}$$

参见习题13-2.

其中第一项可以近似看作输入 $\boldsymbol{x}$ 的重构正确性，第二项可以看作正则化项，λ 可以看作正则化系数. 这和自编码器在形式上非常类似，但它们的内在机理是完全不同的.

参见习题13-3.

变分自编码器的训练过程如图13.5所示，其中空心矩形表示“目标函数”.

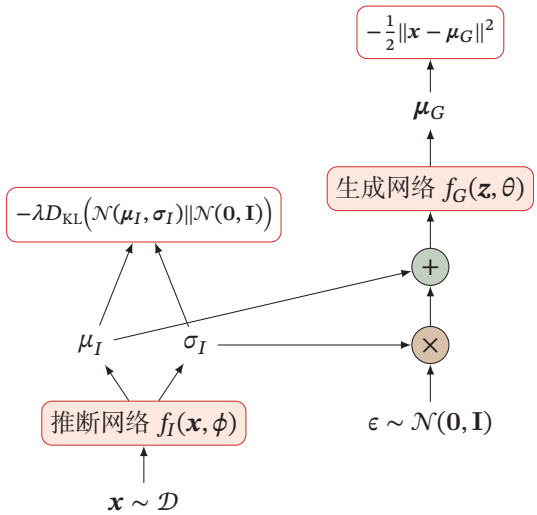
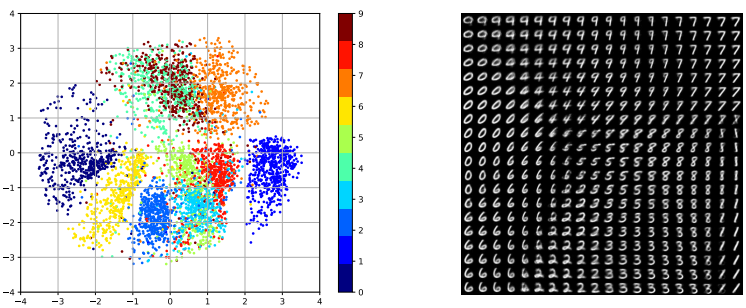


图 13.5 变分自编码器的训练过程

图13.6给出了在 MNIST 数据集上变分自编码器学习到的隐变量流形的可视化示例. 图13.6a是将训练集上每个样本 $\boldsymbol{x}$ 通过推断网络映射到 2 维的隐变量空间，图中的每个点表示 $\mathbb{E}[\boldsymbol{z}|\boldsymbol{x}]$，不同颜色表示不同的数字. 图13.6b是对 2 维的标准高斯分布上进行均匀采样得到不同的隐变量 $\boldsymbol{z}$，然后通过生成网络产生 $\mathbb{E}[\boldsymbol{x}|\boldsymbol{z}]$.



(a) 训练集上所有样本在隐空间上的投影 (b) 隐变量 $\boldsymbol{z}$ 在图像空间的投影

图 13.6 在 MNIST 数据集上变分自编码器学习到的隐变量流形的可视化示例

13.3 生成对抗网络

13.3.1 显式密度模型和隐式密度模型

之前介绍的深度生成模型，比如变分自编码器、深度信念网络等，都是显示地构建出样本的密度函数 $p(\boldsymbol{x};\theta)$，并通过最大似然估计来求解参数，称为显式密度模型（Explicit Density Model）. 比如，变分自编码器的密度函数为 $p(\boldsymbol{x},\boldsymbol{z};\theta) = p(\boldsymbol{x}|\boldsymbol{z};\theta)p(\boldsymbol{z};\theta)$. 虽然使用了神经网络来估计 $p(\boldsymbol{x}|\boldsymbol{z};\theta)$，但是我们依然假设 $p(\boldsymbol{x}|\boldsymbol{z};\theta)$ 为一个参数分布族，而神经网络只是用来预测这个参数分布族的参数. 这在某种程度上限制了神经网络的能力.

如果只是希望有一个模型能生成符合数据分布 $p_r(\boldsymbol{x})$ 的样本，那么可以不显示地估计出数据分布的密度函数. 假设在低维空间 $\mathcal{Z}$ 中有一个简单容易采样的分布 $p(\boldsymbol{z})$，$p(\boldsymbol{z})$ 通常为标准多元正态分布 $\mathcal{N}(\boldsymbol{0},\boldsymbol{I})$. 我们用神经网络构建一个映射函数 $G:\mathcal{Z}\to\mathcal{X}$，称为生成网络. 利用神经网络强大的拟合能力，使得 $G(\boldsymbol{z})$ 服从数据分布 $p_r(\boldsymbol{x})$. 这种模型就称为隐式密度模型（Implicit Density Model）. 所谓隐式模型就是指并不显式地建模 $p_r(\boldsymbol{x})$，而是建模生成过程. 图13.7给出了隐式模型生成样本的过程.

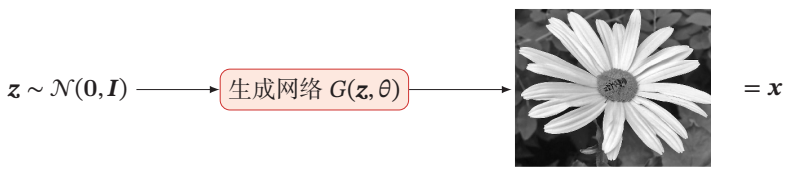


图 13.7 隐式模型生成样本的过程

13.3.2 网络分解

隐式密度模型的一个关键是如何确保生成网络产生的样本一定是服从真实的数据分布. 既然我们不构建显式密度函数，就无法通过最大似然估计等方法来训练. 生成对抗网络（Generative Adversarial Networks，GAN）[Goodfellow et al., 2014] 是通过对抗训练的方式来使得生成网络产生的样本服从真实数据分布. 在生成对抗网络中，有两个网络进行对抗训练. 一个是判别网络，目标是尽量准确地判断一个样本是来自于真实数据还是由生成网络产生；另一个是生成网络，目标是尽量生成判别网络无法区分来源的样本. 这两个目标相反的网络不断地进行交替训练. 当最后收敛时，如果判别网络再也无法判断出一个样本的来源，那么也就等价于生成网络可以生成符合真实数据分布的样本. 生成对抗网络的流程图如图13.8所示.