第十四章　自编码器

自编码器（autoencoder）是神经网络的一种，经过训练后能尝试将输入复制到输出。**自编码器**（autoencoder）内部有一个隐藏层 $\boldsymbol{h}$，可以产生**编码**（code）表示输入。该网络可以看作由两部分组成：一个由函数 $\boldsymbol{h} = f(\boldsymbol{x})$ 表示的编码器和一个生成重构的解码器 $\boldsymbol{r} = g(\boldsymbol{h})$。图 14.1 展示了这种架构。如果一个自编码器只是简单地学会将处处设置为 $g(f(\boldsymbol{x})) = \boldsymbol{x}$，那么这个自编码器就没什么特别的用处。相反，我们不应该将自编码器设计成输入到输出完全相等。这通常需要向自编码器强加一些约束，使它只能近似地复制，并只能复制与训练数据相似的输入。这些约束强制模型考虑输入数据的哪些部分需要被优先复制，因此它往往能学习到数据的有用特性。

现代自编码器将编码器和解码器的概念推而广之，将其中的确定函数推广为随机映射 $p_{\text{encoder}}(\boldsymbol{h} \mid \boldsymbol{x})$ 和 $p_{\text{decoder}}(\boldsymbol{x} \mid \boldsymbol{h})$。

数十年间，自编码器的想法一直是神经网络历史景象的一部分 (LeCun, 1987; Bourlard and Kamp, 1988; Hinton and Zemel, 1994)。传统自编码器被用于降维或特征学习。近年来，自编码器与潜变量模型理论的联系将自编码器带到了生成式建模的前沿，我们将在第二十章揭示更多细节。自编码器可以被看作是前馈网络的一个特例，并且可以使用完全相同的技术进行训练，通常使用小批量梯度下降法（其中梯度基于反向传播计算）。不同于一般的前馈网络，自编码器也可以使用**再循环**（recirculation）训练 (Hinton and McClelland, 1988)，这种学习算法基于比较原始输入的激活和重构输入的激活。相比反向传播算法，再循环算法更具生物学意义，但很少用于机器学习应用。

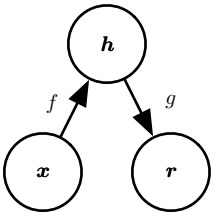

图 14.1: 自编码器的一般结构，通过内部表示或编码 $\boldsymbol{h}$ 将输入 $\boldsymbol{x}$ 映射到输出（称为重构）$\boldsymbol{r}$。自编码器具有两个组件：编码器 f（将 $\boldsymbol{x}$ 映射到 $\boldsymbol{h}$）和解码器 g（将 $\boldsymbol{h}$ 映射到 $\boldsymbol{r}$）。

14.1 欠完备自编码器

将输入复制到输出听起来没什么用，但我们通常不关心解码器的输出。相反，我们希望通过训练自编码器对输入进行复制而使 $\boldsymbol{h}$ 获得有用的特性。

从自编码器获得有用特征的一种方法是限制 $\boldsymbol{h}$ 的维度比 $\boldsymbol{x}$ 小，这种编码维度小于输入维度的自编码器称为**欠完备**（undercomplete）自编码器。学习欠完备的表示将强制自编码器捕捉训练数据中最显著的特征。

学习过程可以简单地描述为最小化一个损失函数

$$L(\boldsymbol{x}, g(f(\boldsymbol{x}))), \tag{14.1}$$

其中 L 是一个损失函数，惩罚 $g(f(\boldsymbol{x}))$ 与 $\boldsymbol{x}$ 的差异，如均方误差。

当解码器是线性的且 L 是均方误差，欠完备的自编码器会学习出与 PCA 相同的生成子空间。这种情况下，自编码器在训练来执行复制任务的同时学到了训练数据的主元子空间。

因此，拥有非线性编码器函数 f 和非线性解码器函数 g 的自编码器能够学习出更强大的 PCA 非线性推广。不幸的是，如果编码器和解码器被赋予过大的容量，自编码器会执行复制任务而捕捉不到任何有关数据分布的有用信息。从理论上说，我们可以设想这样一个自编码器，它只有一维编码，但它具有一个非常强大的非线性编码器，能够将每个训练数据 $\boldsymbol{x}^{(i)}$ 表示为编码 i。而解码器可以学习将这些整数索引映射回特定训练样本的值。这种特定情形不会在实际情况中发生，但它清楚地说明，如果自编码器的容量太大，那训练来执行复制任务的自编码器可能无法学习到数据集的任何有用信息。

14.2 正则自编码器

编码维数小于输入维数的欠完备自编码器可以学习数据分布最显著的特征。我们已经知道，如果赋予这类自编码器过大的容量，它就不能学到任何有用的信息。

如果隐藏编码的维数允许与输入相等，或隐藏编码维数大于输入的**过完备**（overcomplete）情况下，会发生类似的问题。在这些情况下，即使是线性编码器和线性解码器也可以学会将输入复制到输出，而学不到任何有关数据分布的有用信息。

理想情况下，根据要建模的数据分布的复杂性，选择合适的编码维数和编码器、解码器容量，就可以成功训练任意架构的自编码器。正则自编码器提供这样的能力。正则自编码器使用的损失函数可以鼓励模型学习其他特性（除了将输入复制到输出），而不必限制使用浅层的编码器和解码器以及小的编码维数来限制模型的容量。这些特性包括稀疏表示、表示的小导数、以及对噪声或输入缺失的鲁棒性。即使模型容量大到足以学习一个无意义的恒等函数，非线性且过完备的正则自编码器仍然能够从数据中学到一些关于数据分布的有用信息。

除了这里所描述的方法（正则化自编码器最自然的解释），几乎任何带有潜变量并配有一个推断过程（计算给定输入的潜在表示）的生成模型，都可以看作是自编码器的一种特殊形式。强调与自编码器联系的两个生成式建模方法是 Helmholtz 机 (Hinton *et al.*, 1995b) 的衍生模型，如变分自编码器（第 20.10.3 节）和生成随机网络（第 20.12 节）。这些变种（或衍生）自编码器能够学习出高容量且过完备的模型，进而发现输入数据中有用的结构信息，并且也无需对模型进行正则化。这些编码显然是有用的，因为这些模型被训练为近似训练数据的概率分布而不是将输入复制到输出。

14.2.1 稀疏自编码器

稀疏自编码器简单地在训练时结合编码层的稀疏惩罚 $\Omega(\boldsymbol{h})$ 和重构误差：

$$L(\boldsymbol{x}, g(f(\boldsymbol{x}))) + \Omega(\boldsymbol{h}), \tag{14.2}$$

其中 $g(\boldsymbol{h})$ 是解码器的输出，通常 $\boldsymbol{h}$ 是编码器的输出，即 $\boldsymbol{h} = f(\boldsymbol{x})$。

稀疏自编码器一般用来学习特征，以便用于像分类这样的任务。稀疏正则化的自编码器必须反映训练数据集的独特统计特征，而不是简单地充当恒等函数。以这种方式训练，执行附带稀疏惩罚的复制任务可以得到能学习有用特征的模型。

我们可以简单地将惩罚项 $\Omega(\boldsymbol{h})$ 视为加到前馈网络的正则项，这个前馈网络的主要任务是将输入复制到输出（无监督学习的目标），并尽可能地根据这些稀疏特征执行一些监督学习任务（根据监督学习的目标）。不像其它正则项如权重衰减——没有直观的贝叶斯解释。如第 5.6.1 节描述，权重衰减和其他正则惩罚可以被解释为一个 MAP 近似贝叶斯推断，正则化的惩罚对应于模型参数的先验概率分布。这种观点认为，正则化的最大似然对应最大化 $p(\boldsymbol{\theta} \mid \boldsymbol{x})$，相当于最大化 $\log p(\boldsymbol{x} \mid \boldsymbol{\theta}) + \log p(\boldsymbol{\theta})$。$\log p(\boldsymbol{x} \mid \boldsymbol{\theta})$ 即通常的数据似然项，参数的对数先验项 $\log p(\boldsymbol{\theta})$ 则包含了对 $\boldsymbol{\theta}$ 特定值的偏好。这种观点在第 5.6 节有所描述。正则自编码器不适用这样的解释是因为正则项取决于数据，因此根据定义上（从文字的正式意义）来说，它不是一个先验。虽然如此，我们仍可以认为这些正则项隐式地表达了对函数的偏好。

我们可以认为整个稀疏自编码器框架是对带有潜变量的生成模型的近似最大似然训练，而不将稀疏惩罚视为复制任务的正则化。假如我们有一个带有可见变量 $\boldsymbol{x}$ 和潜变量 $\boldsymbol{h}$ 的模型，且具有明确的联合分布 $p_{\text{model}}(\boldsymbol{x}, \boldsymbol{h}) = p_{\text{model}}(\boldsymbol{h}) p_{\text{model}}(\boldsymbol{x} \mid \boldsymbol{h})$。我们将 $p_{\text{model}}(\boldsymbol{h})$ 视为模型关于潜变量的先验分布，表示模型看到 $\boldsymbol{x}$ 的信念先验。这与我们之前使用 ‘‘先验’’ 的方式不同，之前指分布 $p(\boldsymbol{\theta})$ 在我们看到数据前就对模型参数的先验进行编码。对数似然函数可分解为

$$\log p_{\text{model}}(\boldsymbol{x}) = \log \sum_{\boldsymbol{h}} p_{\text{model}}(\boldsymbol{h}, \boldsymbol{x}). \tag{14.3}$$

我们可以认为自编码器使用一个高似然值 $\boldsymbol{h}$ 的点估计近似这个总和。这类似于稀疏编码生成模型（第 13.4 节），但 $\boldsymbol{h}$ 是参数编码器的输出，而不是从优化结果推断出的最可能的 $\boldsymbol{h}$。从这个角度看，我们根据这个选择的 $\boldsymbol{h}$，最大化如下

$$\log p_{\text{model}}(\boldsymbol{h}, \boldsymbol{x}) = \log p_{\text{model}}(\boldsymbol{h}) + \log p_{\text{model}}(\boldsymbol{x} \mid \boldsymbol{h}). \tag{14.4}$$

$\log p_{\text{model}}(\boldsymbol{h})$ 项能被稀疏诱导。如Laplace先验，

$$p_{\text{model}}(h_i) = \frac{\lambda}{2} e^{-\lambda |h_i|}, \tag{14.5}$$

对应于绝对值稀疏惩罚。将对数先验表示为绝对值惩罚，我们得到

$$\Omega(\boldsymbol{h}) = \lambda \sum_i |h_i|, \tag{14.6}$$

$$-\log p_{\text{model}}(\boldsymbol{h}) = \sum_i (\lambda |h_i| - \log \frac{\lambda}{2}) = \Omega(\boldsymbol{h}) + \text{const}, \tag{14.7}$$

这里的常数项只跟 λ 有关。通常我们将 λ 视为超参数，因此可以丢弃不影响参数学习的常数项。其他如 Student-t 先验也能诱导稀疏性。从稀疏性导致 $p_{\text{model}}(\boldsymbol{h})$ 学习成近似最大似然的结果看，稀疏惩罚完全不是一个正则项。这仅仅影响模型关于潜变量的分布。这个观点提供了训练自编码器的另一个动机：这是近似训练生成模型的一种途径。这也给出了为什么自编码器学到的特征是有用的另一个解释：它们描述的潜变量可以解释输入。

稀疏自编码器的早期工作 (Ranzato *et al.*, 2007a, 2008) 探讨了各种形式的稀疏性，并提出了稀疏惩罚和 $\log Z$ 项（将最大似然应用到无向概率模型 $p(\boldsymbol{x}) = \frac{1}{Z}\tilde{p}(\boldsymbol{x})$ 时产生）之间的联系。这个想法是最小化 $\log Z$ 防止概率模型处处具有高概率，同理强制稀疏可以防止自编码器处处具有低的重构误差 。这种情况下，这种联系是对通用机制的直观理解而不是数学上的对应。在数学上更容易解释稀疏惩罚对应于有向模型 $p_{\text{model}}(\boldsymbol{h})p_{\text{model}}(\boldsymbol{x} \mid \boldsymbol{h})$ 中的 $\log p_{\text{model}}(\boldsymbol{h})$。

Glorot *et al.* (2011b) 提出了一种在稀疏（和去噪）自编码器的 $\boldsymbol{h}$ 中实现真正为零的方式。该想法是使用整流线性单元产生编码层。基于将表示真正推向零（如绝对值惩罚）的先验，可以间接控制表示中零的平均数量。

14.2.2 去噪自编码器

除了向代价函数增加一个惩罚项，我们也可以通过改变重构误差项来获得一个能学到有用信息的自编码器。

传统的自编码器最小化以下目标

$$L(\boldsymbol{x}, g(f(\boldsymbol{x}))), \tag{14.8}$$

其中 L 是一个损失函数，惩罚 $g(f(\boldsymbol{x}))$ 与 $\boldsymbol{x}$ 的差异，如它们彼此差异的 L^2 范数。如果模型被赋予过大的容量，L 仅仅使得 $g \circ f$ 学成一个恒等函数。

相反，**去噪自编码器**（denoising autoencoder, DAE）最小化

$$L(\boldsymbol{x}, g(f(\tilde{\boldsymbol{x}}))), \tag{14.9}$$

其中 $\tilde{\boldsymbol{x}}$ 是被某种噪声损坏的 $\boldsymbol{x}$ 的副本。因此去噪自编码器必须撤消这些损坏，而不是简单地复制输入。

Alain and Bengio (2013) 和 Bengio *et al.* (2013c) 指出去噪训练过程强制 f 和 g 隐式地学习 $p_{\text{data}}(\boldsymbol{x})$ 的结构。因此去噪自编码器也是一个通过最小化重构误差获

取有用特性的例子。这也是将过完备、高容量的模型用作自编码器的一个例子——只要小心防止这些模型仅仅学习一个恒等函数。去噪自编码器将在第 14.5 节给出更多细节。

14.2.3 惩罚导数作为正则

另一正则化自编码器的策略是使用一个类似稀疏自编码器中的惩罚项 Ω，

$$L(\boldsymbol{x}, g(f(\boldsymbol{x}))) + \Omega(\boldsymbol{h}, \boldsymbol{x}), \tag{14.10}$$

但 Ω 的形式不同：

$$\Omega(\boldsymbol{h}, \boldsymbol{x}) = \lambda \sum_i \|\nabla_{\boldsymbol{x}} h_i\|^2. \tag{14.11}$$

这迫使模型学习一个在 $\boldsymbol{x}$ 变化小时目标也没有太大变化的函数。因为这个惩罚只对训练数据适用，它迫使自编码器学习可以反映训练数据分布信息的特征。

这样正则化的自编码器被称为 **收缩自编码器**（contractive autoencoder, CAE）。这种方法与去噪自编码器、流形学习和概率模型存在一定理论联系。收缩自编码器将在第 14.7 节更详细地描述。

14.3 表示能力、层的大小和深度

自编码器通常只有单层的编码器和解码器，但这不是必然的。实际上深度编码器和解码器能提供更多优势。

回忆第 6.4.1 节，其中提到加深前馈网络有很多优势。这些优势也同样适用于自编码器，因为它也属于前馈网络。此外，编码器和解码器各自都是一个前馈网络，因此这两个部分也能各自从深度结构中获得好处。

万能近似定理保证至少有一层隐藏层且隐藏单元足够多的前馈神经网络能以任意精度近似任意函数（在很大范围里），这是非平凡深度（至少有一层隐藏层）的一个主要优点。这意味着具有单隐藏层的自编码器在数据域内能表示任意近似数据的恒等函数。但是，从输入到编码的映射是浅层的。这意味这我们不能任意添加约束，比如约束编码稀疏。深度自编码器（编码器至少包含一层额外隐藏层）在给定足够多的隐藏单元的情况下，能以任意精度近似任何从输入到编码的映射。

深度可以指数地降低表示某些函数的计算成本。深度也能指数地减少学习一些函数所需的训练数据量。读者可以参考第 6.4.1 节巩固深度在前馈网络中的优势。

实验中，深度自编码器能比相应的浅层或线性自编码器产生更好的压缩效率 (Hinton and Salakhutdinov, 2006)。

训练深度自编码器的普遍策略是训练一堆浅层的自编码器来贪心地预训练相应的深度架构。所以即使最终目标是训练深度自编码器，我们也经常会遇到浅层自编码器。

14.4 随机编码器和解码器

自编码器本质上是一个前馈网络，可以使用与传统前馈网络相同的损失函数和输出单元。

如第 6.2.2.4 节中描述，设计前馈网络的输出单元和损失函数普遍策略是定义一个输出分布 $p(\boldsymbol{y} \mid \boldsymbol{x})$ 并最小化负对数似然 $-\log p(\boldsymbol{y} \mid \boldsymbol{x})$。在这种情况下，$\boldsymbol{y}$ 是关于目标的向量（如类标）。

在自编码器中，$\boldsymbol{x}$ 既是输入也是目标。然而，我们仍然可以使用与之前相同的架构。给定一个隐藏编码 $\boldsymbol{h}$，我们可以认为解码器提供了一个条件分布 $p_{\text{model}}(\boldsymbol{x} \mid \boldsymbol{h})$。接着我们根据最小化 $-\log p_{\text{decoder}}(\boldsymbol{x} \mid \boldsymbol{h})$ 来训练自编码器。损失函数的具体形式视 p_{decoder} 的形式而定。就传统的前馈网络来说，如果 $\boldsymbol{x}$ 是实值的，那么我们通常使用线性输出单元参数化高斯分布的均值。在这种情况下，负对数似然对应均方误差准则。类似地，二值 $\boldsymbol{x}$ 对应于一个 Bernoulli 分布，其参数由 sigmoid 输出单元确定的。而离散的 $\boldsymbol{x}$ 对应 softmax 分布，以此类推。在给定 $\boldsymbol{h}$ 的情况下，为了便于计算概率分布，输出变量通常被视为是条件独立的，但一些技术（如混合密度输出）可以解决输出相关的建模。

为了更彻底地与我们之前了解到的前馈网络相区别，我们也可以将**编码函数** (encoding function) $f(\boldsymbol{x})$ 的概念推广为**编码分布** (encoding distribution) $p_{\text{encoder}}(\boldsymbol{h} \mid \boldsymbol{x})$，如图 14.2 中所示。

任何潜变量模型 $p_{\text{model}}(\boldsymbol{h}, \boldsymbol{x})$ 定义一个随机编码器

$$p_{\text{encoder}}(\boldsymbol{h} \mid \boldsymbol{x}) = p_{\text{model}}(\boldsymbol{h} \mid \boldsymbol{x}) \tag{14.12}$$

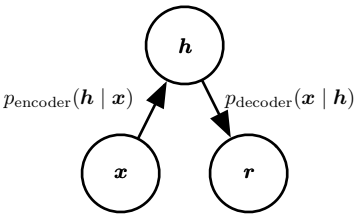

图 14.2: 随机自编码器的结构，其中编码器和解码器包括一些噪声注入，而不是简单的函数。这意味着可以将它们的输出视为来自分布的采样（对于编码器是 $p_{\text{encoder}}(\boldsymbol{h} \mid \boldsymbol{x})$，对于解码器是 $p_{\text{decoder}}(\boldsymbol{x} \mid \boldsymbol{h})$）。

以及一个随机解码器

$$p_{\text{decoder}}(\boldsymbol{x} \mid \boldsymbol{h}) = p_{\text{model}}(\boldsymbol{x} \mid \boldsymbol{h}). \tag{14.13}$$

通常情况下，编码器和解码器的分布没有必要是与唯一一个联合分布 $p_{\text{model}}(\boldsymbol{x}, \boldsymbol{h})$ 相容的条件分布。Alain *et al.* (2015) 指出，在保证足够的容量和样本的情况下，将编码器和解码器作为去噪自编码器训练，能使它们渐近地相容。

14.5 去噪自编码器

去噪自编码器（denoising autoencoder, DAE）是一类接受损坏数据作为输入，并训练来预测原始未被损坏数据作为输出的自编码器。

DAE 的训练过程如图 14.3 中所示。我们引入一个损坏过程 $C(\tilde{\mathbf{x}} \mid \mathbf{x})$，这个条件分布代表给定数据样本 $\mathbf{x}$ 产生损坏样本 $\tilde{\mathbf{x}}$ 的概率。自编码器则根据以下过程，从训练数据对 $(\boldsymbol{x}, \tilde{\boldsymbol{x}})$ 中学习**重构分布** (reconstruction distribution) $p_{\text{reconstruct}}(\mathbf{x} \mid \tilde{\mathbf{x}})$：

1. 从训练数据中采一个训练样本 $\boldsymbol{x}$。
2. 从 $C(\tilde{\mathbf{x}} \mid \mathbf{x} = \boldsymbol{x})$ 采一个损坏样本 $\tilde{\boldsymbol{x}}$。
3. 将 $(\boldsymbol{x}, \tilde{\boldsymbol{x}})$ 作为训练样本来估计自编码器的重构分布 $p_{\text{reconstruct}}(\boldsymbol{x} \mid \tilde{\boldsymbol{x}}) = p_{\text{decoder}}(\boldsymbol{x} \mid \boldsymbol{h})$，其中 $\boldsymbol{h}$ 是编码器 $f(\tilde{\boldsymbol{x}})$ 的输出，p_{decoder} 根据解码函数 $g(\boldsymbol{h})$ 定义。

通常我们可以简单地对负对数似然 $-\log p_{\text{decoder}}(\boldsymbol{x} \mid \boldsymbol{h})$ 进行基于梯度法（如小批

量梯度下降）的近似最小化。只要编码器是确定性的，去噪自编码器就是一个前馈网络，并且可以使用与其他前馈网络完全相同的方式进行训练。

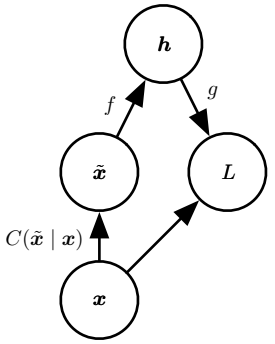

图 14.3: 去噪自编码器代价函数的计算图。去噪自编码器被训练为从损坏的版本 $\tilde{\boldsymbol{x}}$ 重构干净数据点 $\boldsymbol{x}$。这可以通过最小化损失 $L = -\log p_{\text{decoder}}(\boldsymbol{x} \mid \boldsymbol{h} = f(\tilde{\boldsymbol{x}}))$ 实现，其中 $\tilde{\boldsymbol{x}}$ 是样本 $\boldsymbol{x}$ 经过损坏过程 $C(\tilde{\boldsymbol{x}} \mid \boldsymbol{x})$ 后得到的损坏版本。通常，分布 p_{decoder} 是因子的分布（平均参数由前馈网络 g 给出）。

因此我们可以认为 DAE 是在以下期望下进行随机梯度下降：

$$-\mathbb{E}_{\mathbf{x}\sim\hat{p}_{\text{data}}(\mathbf{x})}\mathbb{E}_{\tilde{\mathbf{x}}\sim C(\tilde{\mathbf{x}}|\boldsymbol{x})} \log p_{\text{decoder}}(\boldsymbol{x} \mid \boldsymbol{h} = f(\tilde{\boldsymbol{x}})), \tag{14.14}$$

其中 $\hat{p}_{\text{data}}(\boldsymbol{x})$ 是训练数据的分布。

14.5.1 得分估计

得分匹配 (Hyvärinen, 2005a) 是最大似然的代替。它提供了概率分布的一致估计，促使模型在各个数据点 $\boldsymbol{x}$ 上获得与数据分布相同的**得分**（score）。在这种情况下，得分是一个特定的梯度场：

$$\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x}). \tag{14.15}$$

我们将在第 18.4 节中更详细地讨论得分匹配。对于现在讨论的自编码器，理解学习 $\log p_{\text{data}}$ 的梯度场是学习 p_{data} 结构的一种方式就足够了。

DAE 的训练准则（条件高斯 $p(\boldsymbol{x} \mid \boldsymbol{h})$）能让自编码器学到能估计数据分布得分的向量场 $(g(f(\boldsymbol{x})) - \boldsymbol{x})$，这是 DAE 的一个重要特性。具体如图 14.4 所示。

对一类采用高斯噪声和均方误差作为重构误差的特定去噪自编码器（具有 sigmoid 隐藏单元和线性重构单元）的去噪训练过程，与训练一类特定的被称为 RBM 的

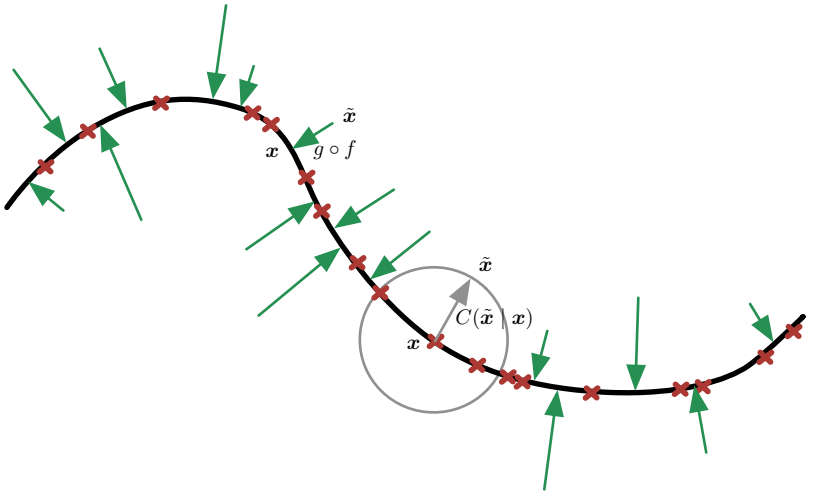

图 14.4: 去噪自编码器被训练为将损坏的数据点 $\tilde{\boldsymbol{x}}$ 映射回原始数据点 $\boldsymbol{x}$。我们将训练样本 $\boldsymbol{x}$ 表示为位于低维流形（粗黑线）附近的红叉。我们用灰色圆圈表示等概率的损坏过程 $C(\tilde{\boldsymbol{x}} \mid \boldsymbol{x})$。灰色箭头演示了如何将一个训练样本转换为经过此损坏过程的样本。当训练去噪自编码器最小化平方误差 $\|g(f(\tilde{\boldsymbol{x}})) - \boldsymbol{x}\|^2$ 的平均值时，重构 $g(f(\tilde{\boldsymbol{x}}))$ 估计 $\mathbb{E}_{\mathbf{x},\tilde{\mathbf{x}} \sim p_{\text{data}}(\mathbf{x})C(\tilde{\mathbf{x}}|\mathbf{x})}[\mathbf{x} \mid \tilde{\boldsymbol{x}}]$。$g(f(\tilde{\boldsymbol{x}}))$ 对可能产生 $\tilde{\boldsymbol{x}}$ 的原始点 $\boldsymbol{x}$ 的质心进行估计，所以向量 $g(f(\tilde{\boldsymbol{x}})) - \tilde{\boldsymbol{x}}$ 近似指向流形上最近的点。因此自编码器可以学习由绿色箭头表示的向量场 $g(f(\boldsymbol{x})) - \boldsymbol{x}$。该向量场将得分 $\nabla_{\boldsymbol{x}} \log p_{\text{data}}(\boldsymbol{x})$ 估计为一个乘性因子，即重构误差均方根的平均。

无向概率模型是等价的 (Vincent, 2011)。这类模型将在第 20.5.1 节给出更详细的介绍；对于现在的讨论，我们只需知道这个模型能显式的给出 $p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta})$。当 RBM 使用 **去噪得分匹配**（denoising score matching）算法 (Kingma and LeCun, 2010a) 训练时，它的学习算法与训练对应的去噪自编码器是等价的。在一个确定的噪声水平下，正则化的得分匹配不是一致估计量；相反它会恢复分布的一个模糊版本。然而，当噪声水平趋向于 0 且训练样本数趋向于无穷时，一致性就会恢复。我们将会在第 18.5 节更详细地讨论去噪得分匹配。

自编码器和 RBM 还存在其他联系。在 RBM 上应用得分匹配后，其代价函数将等价于重构误差结合类似 CAE 惩罚的正则项 (Swersky *et al.*, 2011)。Bengio and Delalleau (2009) 指出自编码器的梯度是对 RBM 对比散度训练的近似。

对于连续的 $\boldsymbol{x}$，高斯损坏和重构分布的去噪准则得到的得分估计适用于一般编

码器和解码器的参数化 (Alain and Bengio, 2013)。这意味着一个使用平方误差准则

$$\|g(f(\tilde{\boldsymbol{x}})) - \boldsymbol{x}\|^2 \tag{14.16}$$

和噪声方差为 σ^2 的损坏

$$C(\tilde{x} = \tilde{\boldsymbol{x}} \mid \boldsymbol{x}) = N(\tilde{\boldsymbol{x}}; \mu = \boldsymbol{x}, \Sigma = \sigma^2 I) \tag{14.17}$$

的通用编码器-解码器架构可以用来训练估计得分。图 14.5 展示其中的工作原理。

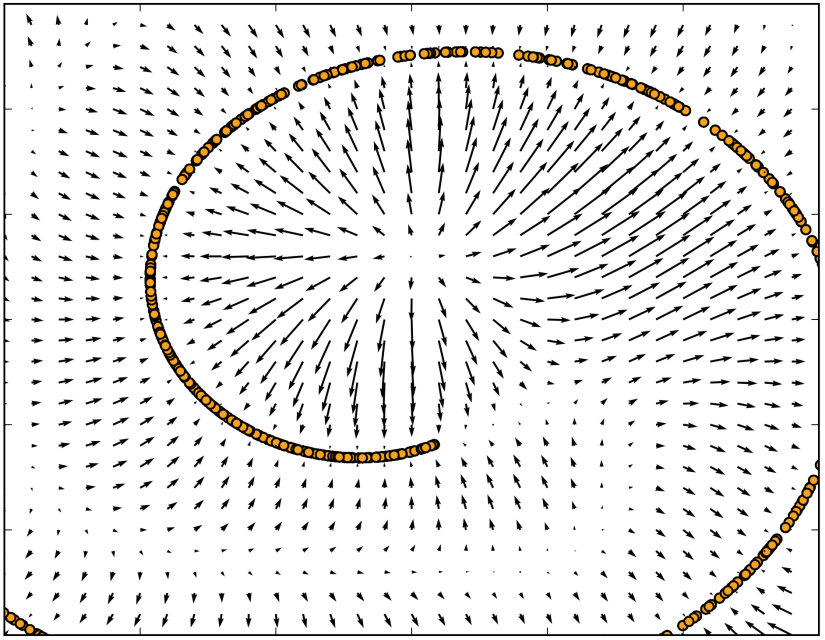

图 14.5: 由去噪自编码器围绕 1 维弯曲流形学习的向量场，其中数据集中在 2 维空间中。每个箭头与重构向量减去自编码器的输入向量后的向量成比例，并且根据隐式估计的概率分布指向较高的概率。向量场在估计的密度函数的最大值处（在数据流形上）和密度函数的最小值处都为零。例如，螺旋臂形成局部最大值彼此连接的 1 维流形。局部最小值出现在两个臂间隙的中间附近。当重构误差的范数（由箭头的长度示出）很大时，在箭头的方向上移动可以显著增加概率，并且在低概率的地方大多也是如此。自编码器将这些低概率点映射到较高的概率重构。在概率最大的情况下，重构变得更准确，因此箭头会收缩。经 Alain and Bengio (2013) 许可转载此图。

一般情况下，不能保证重构函数 $g(f(\boldsymbol{x}))$ 减去输入 $\boldsymbol{x}$ 后对应于某个函数的梯度，更不用说得分 。这是早期工作 (Vincent, 2011) 专用于特定参数化的原因（其中 $g(f(\boldsymbol{x})) - \boldsymbol{x}$ 能通过另一个函数的导数获得）。Kamyshanska and Memisevic (2015)

通过标识一类特殊的浅层自编码器家族，使 $g(f(\boldsymbol{x})) - \boldsymbol{x}$ 对应于这个家族所有成员的一个得分，以此推广 Vincent (2011) 的结果。

目前为止我们所讨论的仅限于去噪自编码器如何学习表示一个概率分布。更一般的，我们可能希望使用自编码器作为生成模型，并从其分布中进行采样。这将在第 20.11 节中讨论。

14.5.2 历史展望

采用 MLP 去噪的想法可以追溯到 LeCun (1987) 和 Gallinari *et al.* (1987) 的工作。Behnke (2001) 也曾使用循环网络对图像去噪。在某种意义上，去噪自编码器仅仅是被训练去噪的 MLP。然而，“去噪自编码器” 的命名指的不仅仅是学习去噪，而且可以学到一个好的内部表示（作为学习去噪的副效用）。这个想法提出较晚 (Vincent *et al.*, 2008b, 2010)。学习到的表示可以被用来预训练更深的无监督网络或监督网络。与稀疏自编码器、稀疏编码、收缩自编码器等正则化的自编码器类似，DAE 的动机是允许学习容量很高的编码器，同时防止在编码器和解码器学习一个无用的恒等函数。

在引入现代 DAE 之前，Inayoshi and Kurita (2005) 探索了其中一些相同的方法和目标。他们除了在监督目标的情况下最小化重构误差之外，还在监督 MLP 的隐藏层注入噪声，通过引入重构误差和注入噪声提升泛化能力。然而，他们的方法基于线性编码器，因此无法学习到现代 DAE 能学习的强大函数族。

14.6 使用自编码器学习流形

如第 5.11.3 节描述，自编码器跟其他很多机器学习算法一样，也利用了数据集中在一个低维流形或者一小组这样的流形的思想。其中一些机器学习算法仅能学习到在流形上表现良好但给定不在流形上的输入会导致异常的函数。自编码器进一步借此想法，旨在学习流形的结构。

要了解自编码器如何做到这一点，我们必须介绍流形的一些重要特性。

流形的一个重要特征是 **切平面**（tangent plane）的集合。d 维流形上的一点 $\boldsymbol{x}$，切平面由能张成流形上允许变动的局部方向的 d 维基向量给出。如图 14.6 所示，这些局部方向决定了我们能如何微小地变动 $\boldsymbol{x}$ 而保持于流形上。

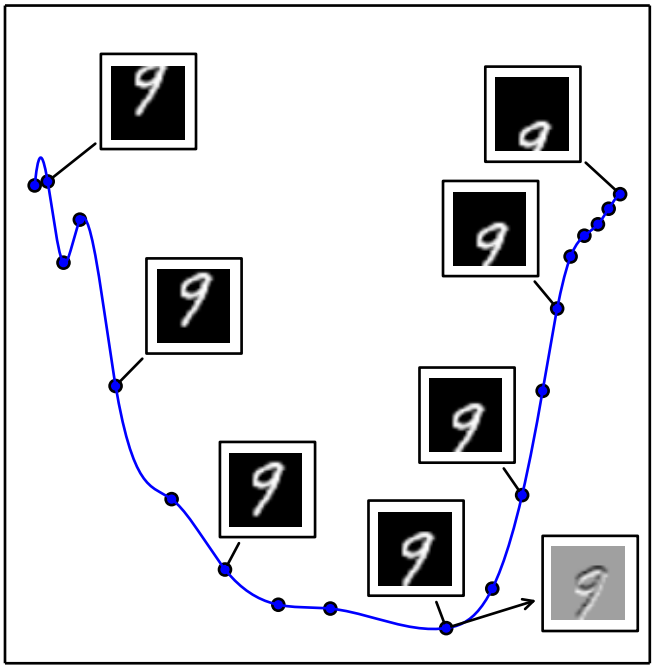

图 14.6: 正切超平面概念的图示。我们在 784 维空间中创建了 1 维流形。我们使用一张 784 像素的 MNIST 图像，并通过垂直平移来转换它。垂直平移的量定义沿着 1 维流形的坐标，轨迹为通过图像空间的弯曲路径。该图显示了沿着该流形的几个点。为了可视化，我们使用 PCA 将流形投影到 2 维空间中。n 维流形在每个点处都具有 n 维切平面。该切平面恰好在该点接触流形，并且在该点处平行于流形表面。它定义了为保持在流形上可以移动的方向空间。该 1 维流形具有单个切线。我们在图中示出了一个点处的示例切线，其中图像表示该切线方向在图像空间中是怎样的。灰色像素表示沿着切线移动时不改变的像素，白色像素表示变亮的像素，黑色像素表示变暗的像素。

所有自编码器的训练过程涉及两种推动力的折衷：

1. 学习训练样本 $\boldsymbol{x}$ 的表示 $\boldsymbol{h}$ 使得 $\boldsymbol{x}$ 能通过解码器近似地从 $\boldsymbol{h}$ 中恢复。$\boldsymbol{x}$ 是从训练数据挑出的这一事实很关键，因为这意味着自编码器不需要成功重构不属于数据生成分布下的输入。

2. 满足约束或正则惩罚。这可以是限制自编码器容量的架构约束，也可以是加入

到重构代价的一个正则项。这些技术一般倾向那些对输入较不敏感的解。

显然，单一的推动力是无用的——从它本身将输入复制到输出是无用的，同样忽略输入也是没用的。相反，两种推动力结合是有用的，因为它们驱使隐藏的表示能捕获有关数据分布结构的信息。重要的原则是，自编码器必须有能力表示重构训练实例所需的变化。如果该数据生成分布集中靠近一个低维流形，自编码器能隐式产生捕捉这个流形局部坐标系的表示：仅在 $\boldsymbol{x}$ 周围关于流形的相切变化需要对应于 $\boldsymbol{h} = f(\boldsymbol{x})$ 中的变化。因此，编码器学习从输入空间 $\boldsymbol{x}$ 到表示空间的映射，映射仅对沿着流形方向的变化敏感，并且对流形正交方向的变化不敏感。

图 14.7 中一维的例子说明，我们可以通过构建对数据点周围的输入扰动不敏感的重构函数，使得自编码器恢复流形结构。

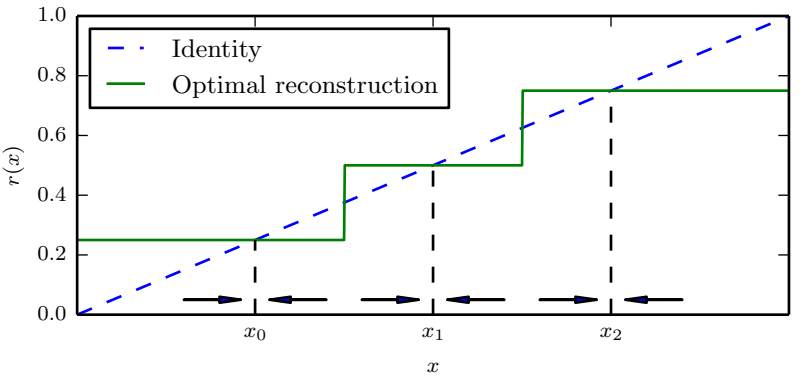

图 14.7: 如果自编码器学习到对数据点附近的小扰动不变的重构函数，它就能捕获数据的流形结构。这里，流形结构是 0 维流形的集合。虚线对角线表示重构的恒等函数目标。最佳重构函数会在存在数据点的任意处穿过恒等函数。图底部的水平箭头表示在输入空间中基于箭头的 $r(\boldsymbol{x}) - \boldsymbol{x}$ 重建方向向量，总是指向最近的“流形”（1 维情况下的单个数据点）。在数据点周围，去噪自编码器明确地尝试将重构函数 $r(\boldsymbol{x})$ 的导数限制为很小。收缩自编码器的编码器执行相同操作。虽然在数据点周围，$r(\boldsymbol{x})$ 的导数被要求很小，但在数据点之间它可能会很大。数据点之间的空间对应于流形之间的区域，为将损坏点映射回流形，重构函数必须具有大的导数。

为了理解自编码器可用于流形学习的原因，我们可以将自编码器和其他方法进行对比。学习表征流形最常见的是流形上（或附近）数据点的 **表示**（representation）。对于特定的实例，这样的表示也被称为嵌入。它通常由一个低维向量给出，具有比这个流形的“外围”空间更少的维数。有些算法（下面讨论的非参数流形学习算法）直

接学习每个训练样例的嵌入，而其他算法学习更一般的映射（有时被称为编码器或表示函数），将周围空间（输入空间）的任意点映射到它的嵌入。

流形学习大多专注于试图捕捉到这些流形的无监督学习过程。最初始的学习非线性流形的机器学习研究专注基于**最近邻图**（nearest neighbor graph）的**非参数**（non-parametric）方法。该图中每个训练样例对应一个节点，它的边连接近邻点对。如图 14.8 所示，这些方法 (Schölkopf *et al.*, 1998b; Roweis and Saul, 2000; Tenenbaum *et al.*, 2000; Brand, 2003b; Belkin and Niyogi, 2003a; Donoho and Grimes, 2003; Weinberger and Saul, 2004b; Hinton and Roweis, 2003; van der Maaten and Hinton, 2008) 将每个节点与张成实例和近邻之间的差向量变化方向的切平面相关联。

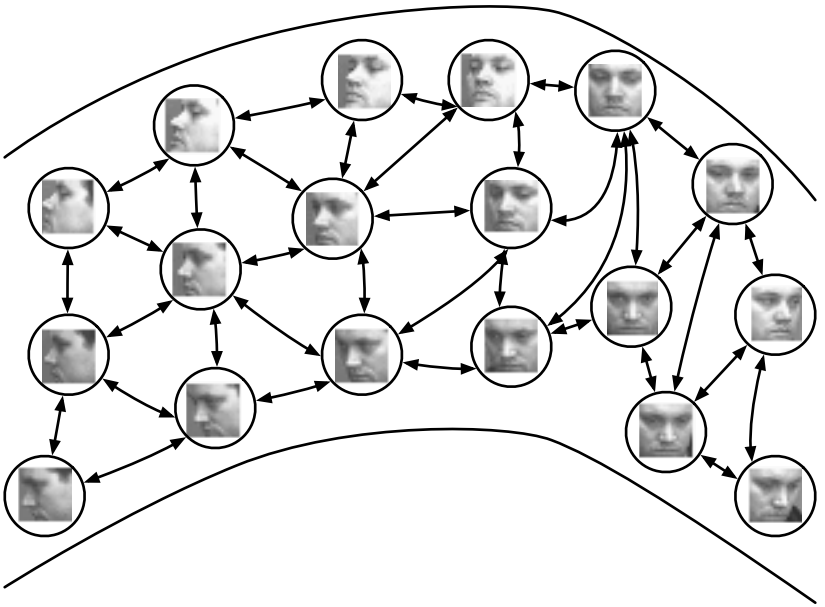

图 14.8: 非参数流形学习过程构建的最近邻图，其中节点表示训练样本，有向边指示最近邻关系。因此，各种过程可以获得与图的邻域相关联的切平面以及将每个训练样本与实值向量位置或**嵌入**（embedding）相关联的坐标系。我们可以通过插值将这种表示概括为新的样本。只要样本的数量大到足以覆盖流形的弯曲和扭转，这些方法工作良好。图片来自 QMUL 多角度人脸数据集 (Gong *et al.*, 2000)。

全局坐标系则可以通过优化或求解线性系统获得。图 14.9 展示了如何通过大量局部线性的类高斯样平铺（或“薄煎饼”，因为高斯块在切平面方向是扁平的）得到一个流形。

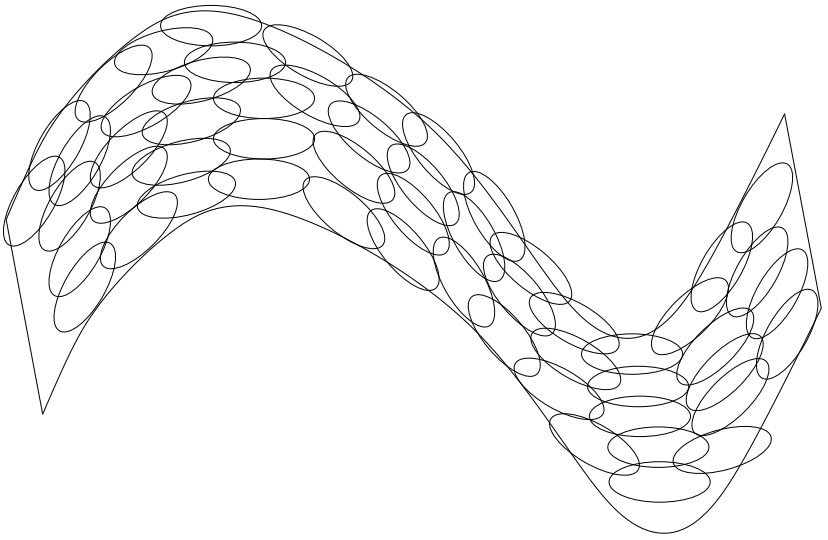

图 14.9: 如果每个位置处的切平面（见图 14.6）是已知的，则它们可以平铺后形成全局坐标系或密度函数。每个局部块可以被认为是局部欧几里德坐标系或者是局部平面高斯或 ‘‘薄饼’’，在与薄饼正交的方向上具有非常小的方差而在定义坐标系的方向上具有非常大的方差。这些高斯的混合提供了估计的密度函数，如流形中的 Parzen 窗口算法 (Vincent and Bengio, 2003) 或其非局部的基于神经网络的变体 (Bengio *et al.*, 2006b)。

然而，Bengio and Monperrus (2005) 指出了这些局部非参数方法应用于流形学习的根本困难：如果流形不是很光滑（它们有许多波峰、波谷和曲折），为覆盖其中的每一个变化，我们可能需要非常多的训练样本，导致没有能力泛化到没见过的变化。实际上，这些方法只能通过内插，概括相邻实例之间流形的形状。不幸的是，AI 问题中涉及的流形可能具有非常复杂的结构，难以仅从局部插值捕获特征。考虑图 14.6 转换所得的流形样例。如果我们只观察输入向量内的一个坐标 x_i，当平移图像，我们可以观察到当这个坐标遇到波峰或波谷时，图像的亮度也会经历一个波峰或波谷。换句话说，底层图像模板亮度的模式复杂性决定执行简单的图像变换所产生的流形的复杂性。这是采用分布式表示和深度学习捕获流形结构的动机。

14.7 收缩自编码器

收缩自编码器 (Rifai *et al.*, 2011a,b) 在编码 $\boldsymbol{h} = f(\boldsymbol{x})$ 的基础上添加了显式的正则项，鼓励 f 的导数尽可能小：

$$\Omega(\boldsymbol{h}) = \lambda \left\| \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} \right\|_F^2 . \tag{14.18}$$

惩罚项 $\Omega(\boldsymbol{h})$ 为平方 Frobenius 范数（元素平方之和），作用于与编码器的函数相关偏导数的 Jacobian 矩阵。

去噪自编码器和收缩自编码器之间存在一定联系：Alain and Bengio (2013) 指出在小高斯噪声的限制下，当重构函数将 $\boldsymbol{x}$ 映射到 $\boldsymbol{r} = g(f(\boldsymbol{x}))$ 时，去噪重构误差与收缩惩罚项是等价的。换句话说，去噪自编码器能抵抗小且有限的输入扰动，而收缩自编码器使特征提取函数能抵抗极小的输入扰动。

分类任务中，基于 Jacobian 的收缩惩罚预训练特征函数 $f(\boldsymbol{x})$，将收缩惩罚应用在 $f(\boldsymbol{x})$ 而不是 $g(f(\boldsymbol{x}))$ 可以产生最好的分类精度。如第 14.5.1 节所讨论，应用于 $f(\boldsymbol{x})$ 的收缩惩罚与得分匹配也有紧密的联系。

收缩（contractive）源于 CAE 弯曲空间的方式。具体来说，由于 CAE 训练为抵抗输入扰动，鼓励将输入点邻域映射到输出点处更小的邻域。我们能认为这是将输入的邻域收缩到更小的输出邻域。

说得更清楚一点，CAE 只在局部收缩——一个训练样本 $\boldsymbol{x}$ 的所有扰动都映射到 $f(\boldsymbol{x})$ 的附近。全局来看，两个不同的点 $\boldsymbol{x}$ 和 $\boldsymbol{x}'$ 会分别被映射到远离原点的两个点 $f(\boldsymbol{x})$ 和 $f(\boldsymbol{x}')$。f 扩展到数据流形的中间或远处是合理的（见图 14.7 中小例子的情况）。当 $\Omega(\boldsymbol{h})$ 惩罚应用于 sigmoid 单元时，收缩 Jacobian 的简单方式是令 sigmoid 趋向饱和的 0 或 1。这鼓励 CAE 使用 sigmoid 的极值编码输入点，或许可以解释为二进制编码。它也保证了 CAE 可以穿过大部分 sigmoid 隐藏单元能张成的超立方体，进而扩散其编码值。

我们可以认为点 $\boldsymbol{x}$ 处的 Jacobian 矩阵 $\boldsymbol{J}$ 能将非线性编码器近似为线性算子。这允许我们更形式地使用 “收缩” 这个词。在线性理论中，当 $\boldsymbol{Jx}$ 的范数对于所有单位 $\boldsymbol{x}$ 都小于等于 1 时，$\boldsymbol{J}$ 被称为收缩的。换句话说，如果 $\boldsymbol{J}$ 收缩了单位球，他就是收缩的。我们可以认为 CAE 为鼓励每个局部线性算子具有收缩性，而在每个训练数据点处将 Frobenius 范数作为 $f(\boldsymbol{x})$ 的局部线性近似的惩罚。

如第 14.6 节中描述，正则自编码器基于两种相反的推动力学习流形。在 CAE 的

情况下，这两种推动力是重构误差和收缩惩罚 $\Omega(\boldsymbol{h})$。单独的重构误差鼓励 CAE 学习一个恒等函数。单独的收缩惩罚将鼓励 CAE 学习关于 $\boldsymbol{x}$ 是恒定的特征。这两种推动力的折衷产生导数 $\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}}$ 大多是微小的自编码器。只有少数隐藏单元，对应于一小部分输入数据的方向，可能有显著的导数。

CAE 的目标是学习数据的流形结构。使 $\boldsymbol{J}\boldsymbol{x}$ 很大的方向 $\boldsymbol{x}$，会快速改变 $\boldsymbol{h}$，因此很可能是近似流形切平面的方向。Rifai *et al.* (2011a,b) 的实验显示训练 CAE 会导致 $\boldsymbol{J}$ 中大部分奇异值（幅值）比 1 小，因此是收缩的。然而，有些奇异值仍然比 1 大，因为重构误差的惩罚鼓励 CAE 对最大局部变化的方向进行编码。对应于最大奇异值的方向被解释为收缩自编码器学到的切方向。理想情况下，这些切方向应对应于数据的真实变化。比如，一个应用于图像的 CAE 应该能学到显示图像改变的切向量，如图 14.6 图中物体渐渐改变状态。如图 14.10 所示，实验获得的奇异向量的可视化似乎真的对应于输入图象有意义的变换。

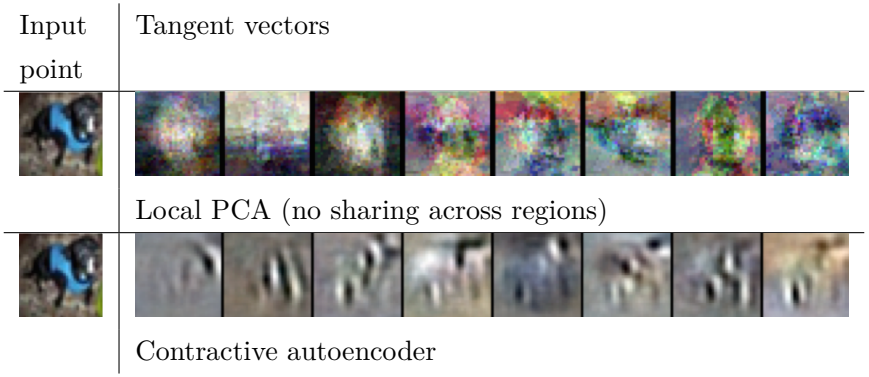

图 14.10: 通过局部 PCA 和收缩自编码器估计的流形切向量的图示。流形的位置由来自 CIFAR-10 数据集中狗的输入图像定义。切向量通过输入到代码映射的 Jacobian 矩阵 $\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}}$ 的前导奇异向量估计。虽然局部 PCA 和 CAE 都可以捕获局部切方向，但 CAE 能够从有限训练数据形成更准确的估计，因为它利用了不同位置的参数共享（共享激活的隐藏单元子集）。CAE切方向通常对应于物体的移动或改变部分（例如头或腿）。经 Rifai *et al.* (2011c) 许可转载此图。

收缩自编码器正则化准则的一个实际问题是，尽管它在单一隐藏层的自编码器情况下是容易计算的，但在更深的自编码器情况下会变的难以计算。根据 Rifai *et al.* (2011a) 的策略，分别训练一系列单层的自编码器，并且每个被训练为重构前一个自编码器的隐藏层。这些自编码器的组合就组成了一个深度自编码器。因为每个层分别训练成局部收缩，深度自编码器自然也是收缩的。这个结果与联合训练深度模型完整架构（带有关于Jacobian的惩罚项）获得的结果是不同的，但它抓住了

许多理想的定性特征。

另一个实际问题是，如果我们不对解码器强加一些约束，收缩惩罚可能导致无用的结果。例如，编码器将输入乘一个小常数 ϵ，解码器将编码除以一个小常数 ϵ。随着 ϵ 趋向于 0，编码器会使收缩惩罚项 $\Omega(\boldsymbol{h})$ 趋向于 0 而学不到任何关于分布的信息。同时，解码器保持完美的重构。Rifai *et al.* (2011a) 通过绑定 f 和 g 的权重来防止这种情况。f 和 g 都是由线性仿射变换后进行逐元素非线性变换的标准神经网络层组成，因此将 g 的权重矩阵设成 f 权重矩阵的转置是很直观的。

14.8 预测稀疏分解

预测稀疏分解（predictive sparse decomposition, PSD）是稀疏编码和参数化自编码器 (Kavukcuoglu *et al.*, 2008) 的混合模型。参数化编码器被训练为能预测迭代推断的输出。PSD 被应用于图片和视频中对象识别的无监督特征学习 (Kavukcuoglu *et al.*, 2009, 2010; Jarrett *et al.*, 2009b; Farabet *et al.*, 2011)，在音频中也有所应用 (Henaff *et al.*, 2011)。这个模型由一个编码器 $f(\boldsymbol{x})$ 和一个解码器 $g(\boldsymbol{h})$ 组成，并且都是参数化的。在训练过程中，$\boldsymbol{h}$ 由优化算法控制。优化过程是最小化

$$\|\boldsymbol{x} - g(\boldsymbol{h})\|^2 + \lambda|\boldsymbol{h}|_1 + \gamma\|\boldsymbol{h} - f(\boldsymbol{x})\|^2. \tag{14.19}$$

就像稀疏编码，训练算法交替地相对 $\boldsymbol{h}$ 和模型的参数最小化上述目标。相对 $\boldsymbol{h}$ 最小化较快，因为 $f(\boldsymbol{x})$ 提供 $\boldsymbol{h}$ 的良好初始值以及损失函数将 $\boldsymbol{h}$ 约束在 $f(\boldsymbol{x})$ 附近。简单的梯度下降算法只需 10 步左右就能获得理想的 $\boldsymbol{h}$。

PSD 所使用的训练程序不是先训练稀疏编码模型，然后训练 $f(\boldsymbol{x})$ 来预测稀疏编码的特征。PSD 训练过程正则化解码器，使用 $f(\boldsymbol{x})$ 可以推断出良好编码的参数。

预测稀疏分解是**学习近似推断**（learned approximate inference）的一个例子。在第 19.5 节中，这个话题将会进一步展开。第十九章中展示的工具能让我们了解到，PSD 能够被解释为通过最大化模型的对数似然下界训练有向稀疏编码的概率模型。

在 PSD 的实际应用中，迭代优化仅在训练过程中使用。模型被部署后，参数编码器 f 用于计算已经习得的特征。相比通过梯度下降推断 $\boldsymbol{h}$，计算 f 是很容易的。因为 f 是一个可微带参函数，PSD 模型可堆叠，并用于初始化其他训练准则的深度网络。