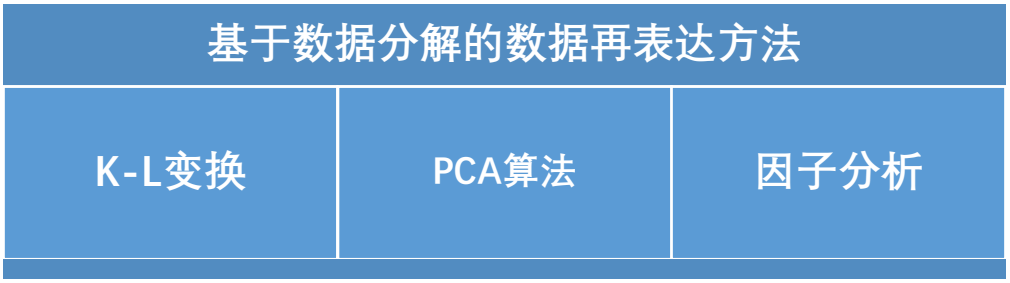
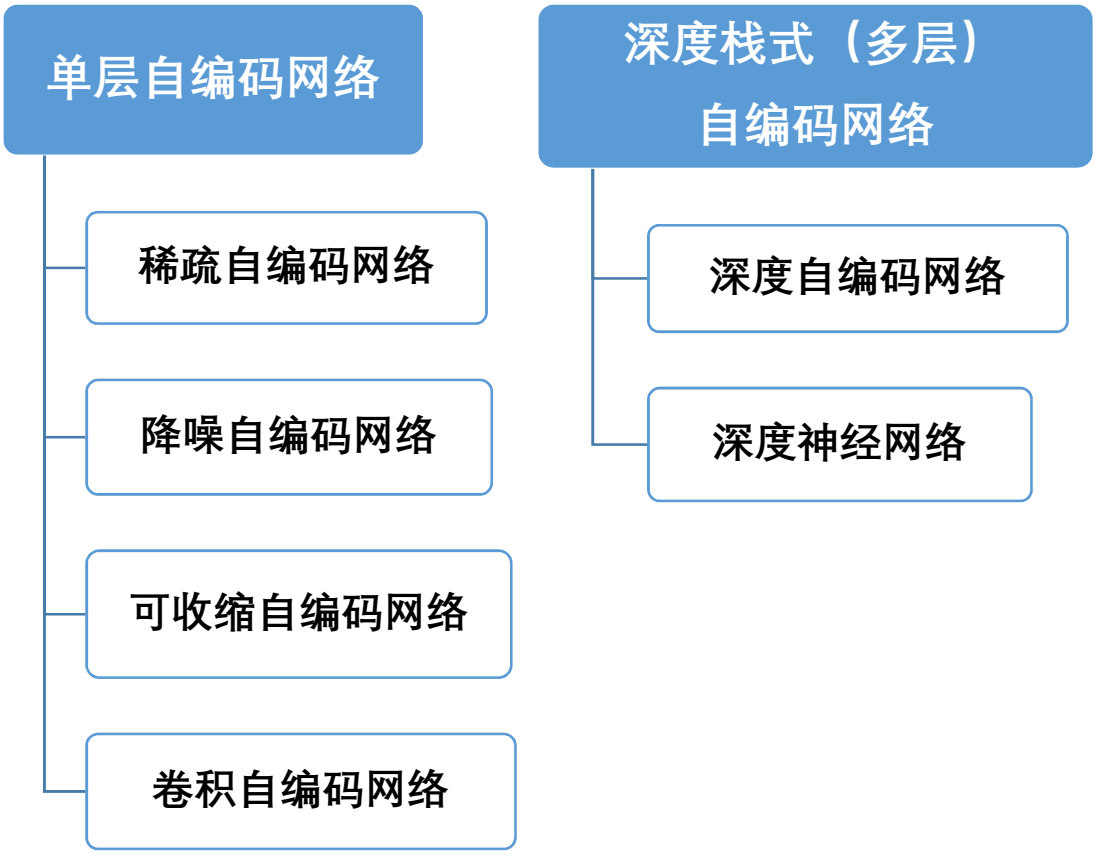
2.2 单层和多层数据再表达的非概率学习模型

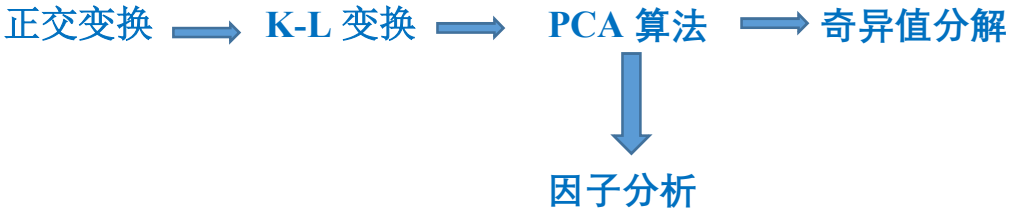




在非概率模型中，数据再表达学习方法可以分为线性方法和非线性方法，两类代表性方法分别为主成分分析和编码。数据驱动的编码类方法包括自编码、稀疏编码等。**主成分分析、自编码和稀疏编码分别都对应各自的概率形式，叫做概率 PCA (Probabilistic PCA) 、变分自编码器（Variational AE， VAE）和概率稀疏编码。**本节我们介绍它们的非概率形式，下一节我们将介绍它们的概率形式。

2.2.1 数据再表达中的主成分分析（PCA）方法

下面对 PCA 方法的介绍中包括其方法、数学原理到改进算法。



一、PCA 方法

1. 正交变换

在介绍 PCA 方法之前，首先从 Karhunen-Loeve (K-L)变换说起。设 X,Y 为两个希尔伯特空间， x,y 分别是其中的向量，对于线性变换 $y=A^Tx$，若 A 将 x 变换为 y 后，其 2 范数保持不变，即

$$< x,x >=< y,y >\Leftrightarrow < x,x >=< A^Tx,A^Tx >\Leftrightarrow A^TA=I$$

则称 A 是正交变换。

正交变换的好处有以下几点：

（1）它是线性变换；

（2）它的逆变换唯一存在： $x=Ay=y_1a_1+y_2a_2+\cdots y_na_n$；

（3）它的正交变换的正反变换用硬件容易实现，不需要求逆；

（4）利用正交变换可以把实对称矩阵对角化，如果这个实对称矩阵是数据的协方差矩阵，则对数据进行正交变换则意味着数据之间的去相关性，便于数据压缩。

2. K-L 变换

K-L 变换是一种特殊的正交变换，对于一个给定的随机向量 $x=(x_1,x_2,...,x_n)^T$，经 K-L 变换后变为 n 维随机向量 y，y 的各分量之间完全**去除了相关性**，且 y 对 x **截断近似的均方误差最小**。K-L 变换的思想即是寻求正交矩阵 U，使得 U 对 x 的变换 $y=U^Tx$ 的自协方差矩阵 $R_y=E\{yy^T\}=U^TR_xU$ 为对角阵。（去除了各维度之间的相关性）。K-L 变换步骤如下。

算法 2.1（K-L 变换）

第 1 步：求 x 的自协方差矩阵 $R_x=E\{xx^T\}$ 的特征值 $\{\lambda_1,\lambda_2,...\lambda_n\}$；

第 2 步：由 $R_xu_i=\lambda_iu_i$ 求 R_x 的特征向量 u_i（已经相互正交）；

第 3 步：将特征向量 $u_1,u_2,...,u_n$ 归一化；

第 4 步：将归一化后的构成归一化正交矩阵 U；

第 5 步：由矩阵 U 实现对信号 x 的 K-L 变换：$y=U^Tx$。

此时，$R_x=U\cdot diag(\lambda_1,\lambda_2,...\lambda_n)\cdot U^T$，而 y 的协方差矩阵：

$$\begin{aligned}R_y&=E\{yy^T\}=E\{[U^Tx][U^Tx]^T\}=U^TE\{xx^T\}U=U^TR_xU\\&=diag\{\lambda_1,\lambda_2,...,\lambda_n\}\end{aligned}\qquad(2.1)$$

这样我们就将y的协方差矩阵R_y变成了一个以R_x的特征值构成的对角矩阵。

K-L 变换可以看成是将向量x关于基向量$u_1,u_2,...,u_n$的展开：

$$x = Uy = [u_1,...,u_n]y = y_1u_1 + ... + y_nu_n \qquad (2.2)$$

事实上，对于向量x所在的空间中的任意一组正交基底$u_1,u_2,...,u_n$，x都可以写成这组正交基的线性组合$x = \sum_{i=1}^{n} k_i u_i$。现在我们希望对$x$的这个表达式中只取前$m$项$m < n$，作为$x$的一个近似（也就是对$x$进行了降维再表达），令$\hat{x} = \sum_{i=1}^{m} k_i u_i$，则$\hat{x}$对$x$的均方误差为$\varepsilon = E\{|x - \hat{x}|^2\} = \sum_{i=m+1}^{n} u_i^T R_x u_i$，那么在基底向量满足正交归一化的前提下我们自然希望找出使得截断误差最小的那一组基底，因而使用拉格朗日条件极值法，令

$$\frac{\partial[\sum_{i=m+1}^{n} u_i^T R_x u_i - \sum_{i=m+1}^{n} \lambda_i (u_i^T u_i - 1)]}{\partial u_i} = 0 \qquad (2.3)$$

由此解得$R_x u_i = \lambda_i u_i, i = 1,\ldots,n$，说明应该选取$R_x$的特征向量做基向量。所以截断误差$\varepsilon = \sum_{i=m+1}^{n} \lambda_i$，为了使之最小，应将$u_i$按照$\lambda_i$的降序排列，这时得到的截断误差$\varepsilon$最小。

3. PCA 算法

下面介绍 PCA 算法，**PCA 算法是 K-L 变换的一个应用**，它的原理就是将一个高维向量 x 通过某个特征向量构成的矩阵 U 投影到一个低维向量空间中，用 y 来表示，并且损失的信息量最少。PCA 的算法步骤为。

算法 2.2（PCA 算法）

设有 m 个 n 维数据 $x_i(i=1,2,\cdots,m)$：

$$X=\begin{pmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_m^T \end{pmatrix}=(p_1,p_2,\cdots p_n)$$

其中

$x_i \in R^n(i=1,2,\cdots,m), p_j \in R^m(j=1,2,\cdots,n)$，矩阵形式为：

$$X=\begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{pmatrix} \in R^{m\times n}$$

第 1 步：将原始数据组成一个 $m\times n$ 的矩阵 X（第 i 行 x_i^T 代表第 i 个样本）；

第 2 步：将矩阵 X 的每一列 $p_j=(x_{1j},x_{2j},\cdots,x_{mj})^T$ 进行零均值化 $p_j:=p_j-\overline{p}_j$，其中 $\overline{p}_j=\left(\sum_{i=1}^{m}x_{ij}\right)e$，得到新的样本矩阵，不妨仍记为 X；

第 3 步：求出这组样本的协方差矩阵 $R_x=\frac{1}{m}\sum_{i=1}^{m}(x_i-\mu_i e)(x_i-\mu_i e)^T$，其中 μ_i 为第 i 个样本的均值；

第 4 步：求出该协方差矩阵 R_x 的特征值 $\lambda_i(i=1,2,\cdots,n)$ 以及对应的特征向量 $u_i(i=1,2,\cdots,n)$；

第 5 步：将特征值按照降序排列，并以此顺序将对应的特征向量排成一个矩阵，取前 k 个特征向量构成的矩阵 $U\in R^{m\times k}$；

第 6 步：计算 $Y=U^TX$，即将 m 个 n 维数据 $x_i\in R^n(i=1,2,\cdots,m)$ 降为 m 个 k 维的新数据 $y_i\in R^k(i=1,2,\cdots,m)$：

$$Y=\begin{pmatrix}y_1^T\\y_2^T\\\vdots\\y_m^T\end{pmatrix}=\begin{pmatrix}y_{11}&y_{12}&\cdots&y_{1k}\\y_{21}&y_{22}&\cdots&y_{2k}\\\vdots&\vdots&\cdots&\vdots\\y_{m1}&y_{m2}&\cdots&y_{mk}\end{pmatrix}\in R^{m\times k}$$

由 $R_xu_i=\lambda_iu_i$ 可得 $R_xU=U\Lambda$，其中 $\Lambda=diag(\lambda_1,\lambda_2,...,\lambda_k)$，又因为 U 是正交矩阵，从而 $U^TR_xU=\Lambda$。在 K-L 变换中我们已经说明了在这样的变换规则下新变量的自协方差矩阵 $R_y=U^TR_xU$，所以 $R_y=\Lambda$，

从而我们知道降维后的数据 y 各个分量不相关，且各分量 y_i 的方差是 λ_i。

由 K-L 变换的分析我们还可以知道这样规则下得出的新表达的向量 y 是原向量 x 的线性最小均方误差估计。

4. 奇异值分解（SVD）

PCA 可以和数据矩阵 X 的奇异值分解（SVD）相联系，$X = U_M \Sigma V_N^T$，其中 U_M 与 V_N 分别是 $M \times M, N \times N$ 的正交归一化矩阵，其中 $\sigma_i = \sqrt{\lambda_i}$，

$$\Sigma = \begin{pmatrix} \sigma_1 & & & & \\ & \sigma_2 & & & \\ & & \ldots & & \\ & & & \sigma_n & \\ & & & & 0 \end{pmatrix} \qquad (2.4)$$

事实上，对于任何的一个矩阵，我们都能找到一组坐标轴，它是由原来的坐标轴通过旋转和缩放得到的。矩阵 Σ 的作用是将一个向量从 V_N 这组正交基的向量的空间旋转到 U_M 这组正交基的向量的空间，并且按照 Σ 在各个方向做了缩放，缩放的倍数就是奇异值。

二、因子分析（Factor Analysis）

对数据分析的另一个切入点是从假设出发，它是假设所有的自变量 x 出现的原因是因为背后存在一个潜变量（latent variable）Z，也就是我们所说的因子，在这个因子的作用下，X 可以被观察到。

因子分析假设自变量x背后存在影响它的因子Z，既然是Z影响X，那么x就可以写成Z的函数如下：$x-\mu=Az+\varepsilon$。（我们只假设它们之间是线性关系，或者说主要部分是线性关系，从而我们忽略掉非线性部分）

对上述矩阵形式的关系式需要做一点说明：其中μ是用来对x做中心化（即零均值化），ε是随机误差，A就是载荷矩阵。当x本身就是零均值化的时候，有$x=Az+\varepsilon$。对此关系式取转置后得到x^T的表达式，用x左乘x^T后对等式两端同时取数学期望，为了使得我们能够解出这个等式，我们对假设的数据还需要继续做假设，就是误差项ε均值为零，且协方差矩阵为对角矩阵，即

$$\sum_{\varepsilon}=diag(\sigma_1^2,\sigma_2^2,...,\sigma_l^2)$$

且假设因子Z与误差项ε是独立的，从而原等式可简化为

$$E[xx^T]=AE[zz^T]A^T+E[\varepsilon\varepsilon^T], \qquad (2.5)$$

即$\sum_x=AA^T+\sum_{\varepsilon}$，那么这样事实上消去了假设因子$z$，我们先解出载荷矩阵$A$，很显然$A$是矩阵$\sum_x-\sum_{\varepsilon}$的一个分解，这个分解显然是不是唯一的，因为对于任意正交矩阵U，我们有

$$AA^T=AU(AU)^T$$

如果A计算出来了，则$z=A^T\sum_x^{-1}x$也就计算出来了。

当我们假设误差项ε是服从高斯分布$N(0,\overline{\Psi})$，再令z是服从高斯分布$N(0,I)$（其实也就是选择一组类似于标准基来做z，重点还是载荷矩阵A），那么这个时候我们可以通过高斯分布的边缘分布仍然是高斯分布这一特性以及一些简单的计算就可以得出x也服从高斯分布$N(\mu,WW^T+\overline{\Psi})$。

这个算法对比 PCA 的优势在于A不是唯一的，我们可以通过右乘正交阵U来对之进行变换，也就是对A进行因子轴旋转，这个操作可以使A原始变量在公因子（主成分）上的载荷重新分布，从而使原始变量在公因子上的载荷两级分化，这样公因子（主成分）就能够用哪些载荷大的原始变量来解释。因为矩阵形式$x=Az+\varepsilon$的线性方程组的形式为

$$x_i-\mu_i=\sum_{j=1}^{m}a_{ij}z_{ij}+\varepsilon_i \qquad i=1,2,...,j\ , \qquad (2.6)$$

所以A的不唯一性很重要。

2.2.2 数据再表达中的栈式自编码网络方法

数据再表达中的非线性学习方法主要包括自编码学习方法、矩阵分解方法、稀疏编码方法和流形上的学习方法等四类算法。这四类方法也体现出了不同角度的“非线性”。流形方法的非线性体现在它认为再表达的数据一般分布在一个低维流形上，而流形本身就是非线性的。这四类方法对模型和数据的使用上有很大的区别，其中自编码学

习方法、稀疏编码方法和流形上学习方法主要依靠训练数据进行学习，所以属于数据驱动的前端学习方法，而**矩阵分解方法则主要依赖矩阵分解模型，并辅以一定的训练数据，所以属于数据与模型混合驱动的数据再表达方法。**下面对这三类数据驱动的数据再表达算法分别加以介绍。

自编码网络就是在保持输入与输出尽可能一致（通过信息损失判定）的前提下，实现无监督方式下的隐层特征提取的数据再表达和参数学习。自编码网络是一种无监督学习算法，它在以迁移学习为代表的机器学习任务中起着重要的作用。其实，自编码网络的概念在上个世纪八十年代就被 Hinton 和他的 PDP 团队[16]引入，它可以对数据进行压缩，从而实现对数据的降维。它的实现方式开始一般通过浅层神经网络实现，使用隐层特征的维数进行刻画，其中升维对应稀疏性，降维则对应压缩。根据特征表示数据再表达的衡标准，自编码网络可以分为两种类型。**一种是从编码特征可以较好重构出输入数据来划分，如稀疏自编码、卷积自编码等；另外一种是对输入数据具有一定的抗噪声性来划分，如降噪自编码、可收缩自编码。**

一般来说，自编码网络都具备三层网络结构，分别为输入层，隐藏层和输出层，其中输入层的输入是原始数据，隐藏层的输出是数据的一种再表达，输出层的输出是为了恢复原始数据，使丢失的信息尽可能地少。2006 年，Hinton 等人[8]提出的带“深度结构”自编码网络的方法取得了巨大的成功。该文章中的自编码网络是以受限玻尔兹曼机（RBMs）的形式堆积而成，自下而上逐层进行训练，最后再微调

整个网络，使得深度网络的训练变得可行。从此开启了深度学习时代。

所以说，深度学习的本质是一种逐层数据再表达的自编码方法。

通常来说，自编码网络的结构图 2.5 如下所示：

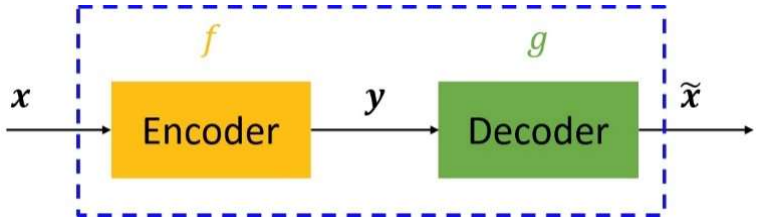


图 2.5 自编码网络的结构图

图 2.5 中，虚线框中是一个自编码网络的模型，它由编码网络和解码网络两部分构成。编码网络将输入数据 x 变换成编码数据 y，解码网络将编码数据 y 变换成输出数据 $\tilde{x}$。

因此，自编码网络的优化模型如下：

$$\min\ J\left(x,\tilde{x}\right)$$
$$s.t.\begin{cases} y=f\left(x\right) \\ \tilde{x}=g\left(y\right) \end{cases} \qquad (2.7)$$

其中，$J:R^m\times R^m\to R$ 为一个衡量二者之间误差的函数，m 是自变量的维度。

对于简单的自编码网络（只有一个隐藏层），如果采用神经网络的语言可以表述为：

$$y=f\left(x\right)\coloneqq\sigma_f\left(Wx+b_1\right) \qquad (2.8)$$

$$\tilde{x}=g\left(y\right)\coloneqq\sigma_g\left(W^Ty+b_2\right) \qquad (2.9)$$

其中，W 为输入层和隐藏层之间的权值矩阵，W^T（W 的转置）为隐藏层和输出层之间的权值矩阵（当然这个权值矩阵可以是任意的，

为了简单起见取成W的转置）；b_1和b_2分别为隐藏层和输出层的偏置向量；σ_f和σ_g一般可取为 sigmoid 函数，即

$$\sigma_f(z)=\sigma_g(z)=\frac{1}{1+e^{-z}} \qquad (2.10)$$

对于损失函数$L(x,y)$的选取，可以取为平方误差函数

$$L(x,y)=\|x-y\|^2 \qquad (2.11)$$

或者交叉熵函数

$$L(x,y)=-\sum_{i=1}^{n}\left[x_i\log(y_i)+(1-x_i)\log(1-y_i)\right] \qquad (2.12)$$

等等。因为最终是要让输出重构输入（使丢失的信息尽可能地少），所以目标函数可以归结为如下形式：

$$J(W,b_1,b_2)=\sum_{i=1}^{n}L\left(W,b_1,b_2;x^{(i)},\tilde{x}^{(i)}\right) \qquad (2.13)$$

其中，W,b_1,b_2为需要优化的参数；$x^{(i)},\tilde{x}^{(i)}$分别为第i个输入数据和第i个输入数据的输出，$i=1,2,...,n$，n为数据集的大小。

如果输入层神经元的个数n小于隐层神经元个数m，相当于把数据从n维降到了m维；

然后利用这m维的特征向量，进行重构原始的数据。类似于 PCA 降维，只不过 PCA 是通过求解特征向量，进行降维，是一种线性的降维方式，而自编码是一种非线性降维。如果每两层之间的变换是线性变换并且损失函数选取平方误差，自编码网络等价于主成分分析[17]。自编码隐藏层可以比输入层的神经元个数还多，然后在神经网络的损失函数构造上，加入正则化约束项进行稀疏约束，这就是稀疏自编码。

以上是自编码网络最基本的形式，下面介绍几种特殊的自编码网络。

一、单层自编码网络

单层指的是只有一个隐藏层。除有某种需要，本节不提及由某种具体的单层自编码网络堆叠而成的深度网络。

1. 稀疏自编码网络

介绍自编码网络之前，先对稀疏性作一个解释：如果当神经元的输出接近于 1 的时候认为是被激活，输出接近于 0 认为是被抑制，使得神经元大部分时间被抑制的限制称作稀疏性限制（这里假设神经元的激活函数是 sigmoid 函数）。如果使用 tanh 函数作为激活函数的函数的话，则输出接近于-1 认为是抑制的。

稀疏自编码网络背后的思想是认为高维而稀疏的特征是有利于解决学习任务的。在这个网络中不会特意去指定哪些隐藏节点被抑制了，而是指定了一个稀疏性常数ρ，表示的是隐藏层神经元的平均活跃程度。为了实现这一目标，需要引入一个度量来衡量实际的激活度与期望的激活度之间的差别，并且将之作为正则项。在下面的模型介绍中，不妨引入 KL 散度作为这个度量。其实，KL 散度是信息论中的一个概念，它的计算公式如下述：

如果P和Q是两个离散随机变量，则它们之间的散度为

$$D_{KL}\left(P\|Q\right)=\sum_{i}P(i)\log\frac{P(i)}{Q(i)} \qquad (2.14)$$

如果P和Q是两个连续随机变量，则它们之间的散度为

$$D_{KL}\left(P\|Q\right)=\int_{-\infty}^{\infty}P(x)\log\frac{P(x)}{Q(x)}dx \qquad (2.15)$$

为了叙述方便，设样本集合的大小为m，隐藏层的节点数目为s_2。令$a_j^{(2)}(x)$表示在给定输入是x的情况下，隐藏层神经元j的激活度，则在样本集上的隐藏层神经元j的平均激活度为$\overline{\rho}_j=\frac{1}{m}\sum_{i=1}^{m}a_j^{(2)}\left(x^{(i)}\right)$，因此根据 KL 散度的定义，稀疏自编码网络模型中目标函数中的正则项为

$$\sum_{j=1}^{s_2}\rho\log\frac{\rho}{\overline{\rho}_j}+(1-\rho)\log\frac{1-\rho}{1-\overline{\rho}_j} \qquad (2.16)$$

这样，整个网络的损失函数为：

$$J_{sparse}\left(W,b_1,b_2\right)=J\left(W,b_1,b_2\right)+\lambda\sum_{j=1}^{s_2}D_{KL}\left(\rho\|\overline{\rho}_j\right) \qquad (2.17)$$

其中，λ是一个超参数，且

$$D_{KL}\left(\rho\|\overline{\rho}_j\right)=\rho\log\frac{\rho}{\overline{\rho}_j}+(1-\rho)\log\frac{1-\rho}{1-\overline{\rho}_j}。 \qquad (2.18)$$

除了稀疏约束项之外，为了防止过拟合，还加入了权重衰减项，所以最后的损失函数就是：

$$J_{SAE}(W,b)=J_E(W,b)+\beta J_{KL}(p\|\overline{p})+\frac{\lambda}{2}\sum_{l=1}^{2}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}\left(w_{ij}^{(l)}\right)^2 \qquad (2.19)$$

类似于一般的神经网络，稀疏自编码网络的训练也是基于反向传播算法。

2. 降噪自编码网络

降噪自编码网络[18][19]的思想是能够恢复出原始数据的特征未

必是最好的，能够对加了噪声的数据进行编码、解码，最后还能恢复出原始数据的鲁棒的特征才是好的。具体来说，将原始数据x通过一个随机映射$\tilde{x} \sim q_D\left(\tilde{x}|x\right)$“污染”成$\tilde{x}$，然后通过基本的自编码网络得到隐藏层的表示$y \sim f_\theta\left(\tilde{x}\right)=s\left(W\tilde{x}+b_1\right)$，并从中重构出$z=g_{\theta'}\left(y\right)$。一个降噪自编码网络的结构如图 2.6 所示。

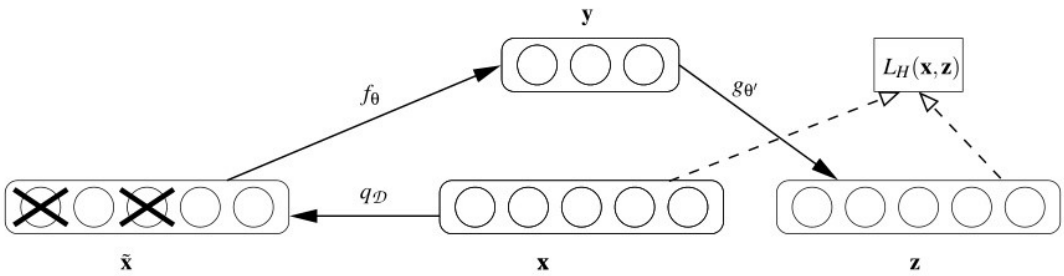


图 2.6 降噪自编码网络结构

图 2.6 中$\tilde{x}$是通过q_D随机“污染”x而来。自编码网络把$\tilde{x}$通过f_θ映射成y，然后通过解码网络$g_{\theta'}$试图重构x得到z。重构误差由损失函数$L_H\left(x,z\right)$来度量。

对于降噪（将一个“污染”过的数据映射到未被污染的数据）的过程，在几何上基于流形假设（高维数据集中在一个非线性的低维流形上）有个直观的解释。这可以通过下图 2.7 来阐释，假设原始数据集中分布在图中所示的流形上，被“污染”的数据会远离该流形（相比原始数据），通过学习的过程，是要将$\tilde{x}$靠近流形。所以如果学习出的降噪过程是成功的话都可以使得很远的点（相对流形来说）映射到靠近流形的小领域中。这样一来，降噪自编码网络也可以看作是在定义和学习一个流形。特殊地，如果我们对隐藏层的表示y的维度作一个限

制（比原始数据的维度小），那么这个表示可以看成是流形上的坐标系统。更一般地，这个表示可以捕获数据中的主要变化因素。

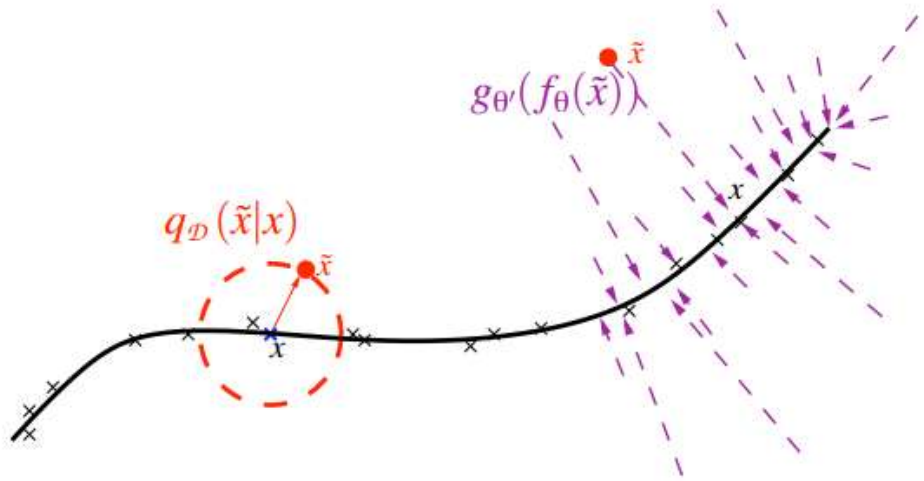


图 2.7 低维流形上降噪自编码

图 2.7 中，假设训练数据(×)集中在流形附近，被“污染”的数据(•)（通过 $q_D\left(\tilde{x}\middle|x\right)$ 获得）通常远离该流形，自编码网络 $g_\theta^{'}\left(f_\theta\left(\cdot\right)\right)$ 将 $\tilde{x}$ 映射到流形上。一般地，“污染”原始数据的方式有以下几种：

（1）各向同性的高斯噪声：$\tilde{x}\middle|x\sim N\left(x,\sigma^2\mathrm{I}\right)$；

（2）掩饰的噪声：x 的百分之 ν 的元素（在每个样本中随机挑选）设为 0；

（3）椒盐噪声：通过投掷硬币的方式将 x 的百分之 ν 的元素（在每个样本中随机挑选）设为它们的最小或者最大值（通常是 0 或者 1）。

对于损失函数的选取，可以引入两个超参数，以使得更适应具体的机器学习任务。对于平方损失函数可以修正为：

$$L_{2,\alpha}\left(x,z\right)=\alpha\left(\sum_{j\in\varsigma\left(\tilde{x}\right)}\left(x_j-z_j\right)^2\right)$$
$$+\beta\left(\sum_{j\notin\varsigma\left(\tilde{x}\right)}\left(x_j-z_j\right)^2\right) \qquad (2.20)$$

其中，$\varsigma\left(\tilde{x}\right)$表示被“污染”数据的分量的索引。对于交叉熵损失函数可以修正为：

$$L_{H,\alpha}\left(x,z\right)=\alpha\left(-\sum_{j\in\varsigma\left(\tilde{x}\right)}\left[x_j\log z_j+\left(1-x_j\right)\log\left(1-z_j\right)\right]\right)$$
$$+\beta\left(-\sum_{j\notin\varsigma\left(\tilde{x}\right)}\left[x_j\log z_j+\left(1-x_j\right)\log\left(1-z_j\right)\right]\right) \qquad (2.21)$$

类似于一般的神经网络，降噪自编码网络的训练也是基于反向传播算法。

通过实验可以发现降噪自编码网络到底学到了什么类型的特征，这可以通过可视化隐藏层的方式来观察。由于每个隐藏层神经元y_j是通过一个权重向量w_j和输入数据做内积运算之后通过非线性变换而得到的，并且这些权重向量w_j和输入数据同维度，因此可以展现出每个神经元对什么样的图像最敏感。以下是一些实验结果：

（1）含有不同隐藏层神经元数目的自编码网络的对比

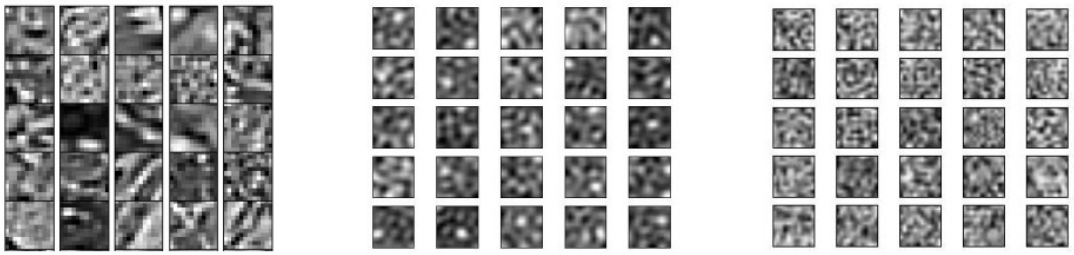


图 2.8 基本的自编码网络在自然图像块上训练的结果

图 2.8 中是基本的自编码网络在自然图像块上训练的结果，左边的是原始的图像，中间的是 50 个隐藏层神经元的自编码网络的结果，右边的是 200 个隐藏层神经元的自编码网络的结果。从实验结果中可以看出，中间的自编码网络学到的特征和局部 blob 检测器类似，而右边的自编码网络学到的特征无明显可识别的结构。

（2）含有相同隐藏层神经元数目而不同类型自编码网络的对比

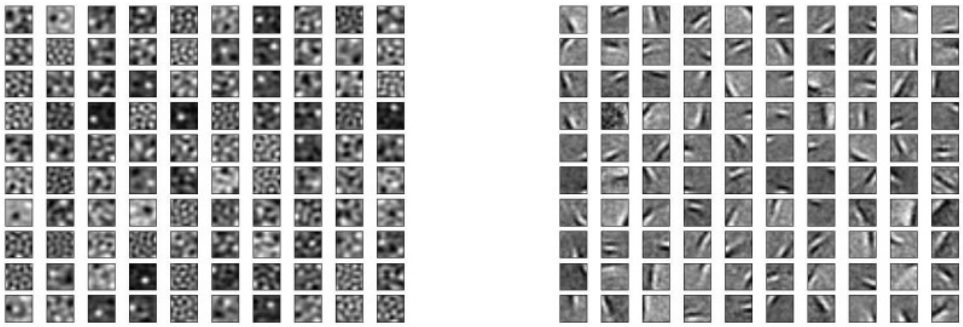


图 2.9 基本的自编码网络与降噪自编码网络结果比较

图 2.9 中左边的是基本的自编码网络（加了 L2 正则项），右边的是降噪自编码网络（未加 L2 正则项），从实验中可以看出，左边的自编码网络学到的特征和局部 blob 检测器类似，右边的自编码网络学到的特征则和 Gabor 局部边缘检测器类似。

3. 可收缩自编码网络

可收缩自编码网络[19]也是一种正则化的自编码网络，即它的损失函数是由经典的重构误差函数和一个惩罚项组合而成。这种网络希望得到的特征数据再表达对训练样本周围数据的小扰动是鲁棒的，这可以通过使惩罚项是非线性映射的雅克比矩阵的 Frobenius 范数实现。具体来说，如果原始数据 x 编码函数 f 映射到隐藏层的表示 h，则惩罚

项是提取到的特征关于输入的偏导数的平方求和，即：

$$\left\|J_f\left(x\right)\right\|_F^2=\sum_{ij}\left(\frac{\partial h_j\left(x\right)}{\partial x_i}\right)^2 \qquad (2.22)$$

从几何上看，惩罚项$\left\|J_f\left(x\right)\right\|_F^2$使得到特征空间的映射在训练数据附近是收缩的。直观上看，一阶导数的值较低，这导致隐藏层的表示对输入数据小变动的鲁棒性或者不变性。所以可收缩自编码网络中需要优化的目标函数是：

$$J_{CAE}\left(\theta\right)=\sum_{x\in D_n}\left(L\left(x,g\left(f\left(x\right)\right)\right)+\lambda\left\|J_f\left(x\right)\right\|_F^2\right) \qquad (2.23)$$

其中，D_n是训练数据集，f是编码函数，g是解码函数，L是损失函数，λ是一个超参数。其实，可收缩自编码网络和其它形式的自编码网络是有一定的关系的。

（1）和权重衰减的自编码网络的关系：权重衰减的自编码网络中需要优化的目标函数是：

$$J_{AE+wd}\left(\theta\right)=\sum_{x\in D_n}\left(L\left(x,g\left(f\left(x\right)\right)\right)+\lambda\sum_{ij}W_{ij}^2\right) \qquad (2.24)$$

可以看出，当编码函数$f(x)$是一个线性函数时，$\left\|J_f\left(x\right)\right\|_F^2=\sum_{ij}W_{ij}^2$。也就是说，在线性编码的情况下，保持小的权值是一种收缩的方式。在非线性的情况下，收缩和鲁棒性可以通过将隐藏层神经元驱使到饱和的区域即可。

（2）和稀疏自编码网络的关系：稀疏的表示意味着表示中大部分的元素接近于 0。对于这些接近于 0 的特征，它们的值必定是在

sigmoid 函数左边的饱和区域，一阶导数的值较小，这也对应$J_f(x)$中小的元素。因此稀疏自编码网络输出很多接近于 0 的特征，对应一种高度收缩的映射。

（3） 和降噪自编码网络的关系：降噪自编码网络的动机之一是对输入数据扰动的鲁棒性。在可收缩自编码网络中是鼓励表示$f(x)$的鲁棒性，而降噪自编码网络是鼓励重构$(g\circ f)(x)$的鲁棒性（可能会部分或者非直接地鼓励表示的鲁棒性）。如果降噪自编码网络的输入数据是通过很小的高斯噪声“污染”过，其中的随机鲁棒准则近似和$\|J_{g\circ f}(x)\|_F^2$等价。表 2.1 是可收缩自编码网络在分类任务上的实验结果（数据集：CIFAR-bw（CIFAR-10 的灰度版本）/ MNIST）：

表 2.1 可收缩自编码网络在分类任务上的实验结果

	Model	Test error	Average $\|J_f(x)\|_F$	SAT
MNIST	CAE	**1.14**	0.73 10^{-4}	86.36%
	DAE-g	**1.18**	0.86 10^{-4}	17.77%
	RBM-binary	1.30	2.50 10^{-4}	78.59%
	DAE-b	1.57	7.87 10^{-4}	68.19%
	AE+wd	1.68	5.00 10^{-4}	12.97%
	AE	1.78	17.5 10^{-4}	49.90%
CIFAR-bw	CAE	**47.86**	2.40 10^{-5}	85,65%
	DAE-b	**49.03**	4.85 10^{-5}	80,66%
	DAE-g	54.81	4.94 10^{-5}	19,90%
	AE+wd	55.03	34.9 10^{-5}	23,04%
	AE	55.47	44.9 10^{-5}	22,57%

表中的 Test error 指的是在测试集上的分类错误率，Average $\|J_f(x)\|_F$ 指的是编码网络在验证集上收缩量 $\|J_f(x)\|_F$ 的平均值（用无监督的预训练之后，微调之前的参数计算），SAT 指的是隐藏层饱和神经元的平均比例。可以看出，预训练后模型的 Average $\|J_f(x)\|_F$ 和最后的 Test error 高度相关，并且可收缩自编码在极小化 $\|J_f(x)\|_F$ 的同时也在得到一个好的重构。

4. 卷积自编码网络

卷积自编码器利用了传统自编码器的无监督的学习方式，改变了一般自编码网络中的全连接模式，改用卷积神经网络的卷积操作，选择局部连接，实现特征提取（数据再表达）。

在叙述卷积自编码网络[21]之前，先简单介绍下卷积神经网络[22-25]的主要结构。卷积神经网络模型主要是卷积层和下采样层交替使用，这和大脑初级视觉皮层的简单与复杂细胞接受外界刺激的处理方式类似，它的一种较常见的结构是由卷积层，最大-池化层（也是一

种下采样层）和分类层堆叠而成。

因为图像数据有着特殊的结构，一般的全连接自编码网络（如前面提到的稀疏自编码网络，降噪自编码网络和可收缩自编码网络）会忽视这种结构。除了这个问题，还会引入参数上的冗余等问题。像卷积神经网络（CNN）那样，卷积自编码网络在输入图像中的所有局部位置共享参数，保持空间的局部性。

对于单通道的输入x，隐藏层第k个特征图的表示为：

$$h^k = \sigma\left(x * W^k + b^k\right) \qquad (2.25)$$

其中，偏置b^k对于第k个特征图来说是相同的，W^k 是参数，σ是激活函数（这里一般采取双曲正切函数），$*$表示二维的卷积。重构（解码过程）可以如下计算：

$$y = \sigma\left(\sum_{k \in H} h^k * \tilde{W}^k + c\right) \qquad (2.26)$$

其中，每个输入通道对应一个偏置c，$\tilde{W}^k$ 是参数，H表示特征图的集合。损失函数可以选择平方误差函数（MSE）：

$$E(\theta) = \frac{1}{2n}\sum_{i=1}^{n}\left(x_i - y_i\right)^2 \qquad (2.27)$$

类似于一般的神经网络，卷积自编码网络的训练也是基于反向传播算法。

其实，在解码的过程中可以采用最大池化的操作。实验表明，采取了这种操作后就没有必要对隐藏神经元或者权重施加 L1 或者 L2 正则化。因此从某种意义上说这也可以视为正则化的作用。

二、深度栈式（多层）自编码网络

上一节介绍的是单层的自编码网络，这一节介绍的是多层的自编码网络，也称作堆叠的自编码网络或者深度的自编码网络。当然，深度的自编码网络可以由上一节提到的那些特殊的**单层自编码网络堆叠而成**。其实，在单层的自编码网络中，隐藏层是最为重要的，因为这一层的输出是编码的特征，也是最终适合后端学习任务的特征。所以真正关心的是输入层到隐藏层的变换，如图 2.10 所示：

图 2.10 自编码器结构

在图 2.10 中，x 是输入层的数据，h 是隐藏层的输出。在深度学习中，多层神经网络实际上是在逐层地学习对原始数据的逐层再表达。类似地，既然有单层的自编码网络，很自然就可以将多个单层的自编码网络堆叠成深度的自编码网络：

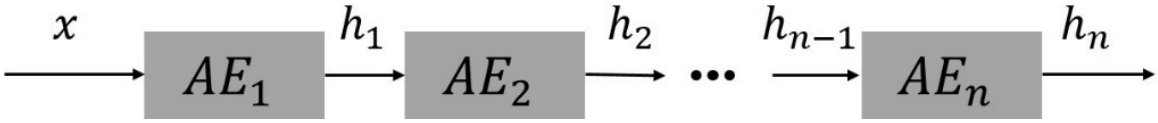


图 2.11 深度的自编码网络结构

图 2.11 中，x 是输入层的数据，h_1 是第一个隐藏层的输出，h_2 是第一个隐藏层的输出，以此类推，h_n 是第 n 个隐藏层的输出，也就是最终适合后端学习任务的特征。

但是直接用反向传播的算法训练这样的深度栈式自编码网络有个很严峻的问题：容易出现“**梯度消失**”或者“**梯度爆炸**”等现象。为了

解决这样的问题，Hinton 等人[8][26]提出了一种新的训练方式，这种训练方式分为两个阶段：逐层预训练和微调。也正是这种训练方式的提出，才导致深度学习第三次兴起。

对于深度自编码网络来说，逐层预训练就是：

把每一层的输出都用一个单层自编码网络来训练，然后这层的输出作为下一个单层自编码网络的输入得到下一层的输出，以此类推直到得到最后一层的输出。

而微调就是：

在预训练的基础上调整各层之间的参数。

详细来说，如上图所示，将 x 作为第一个单层自编码网络的输入，再训练该网络直至收敛，最后把输出层以及隐藏层和输出层之间的连接去掉，保留剩下的网络结构；将隐藏层的输出 h_1 作为第二个单层自编码网络的输入，再训练这个网络，和之前的做法一样，将隐藏层的输出 h_2 作为下一个自编码网络的输入。以此类推，得到隐藏层的输出 h_n，这样就完成了预训练的过程。微调则要视具体任务而定。对于分类任务，可以直接把编码后的特征 h_n 作为分类层（如 Softmax 层）的输入，利用原始数据的标签进行有监督的学习，调整预训练过程中得到的参数。其实，训练神经网络的反向传播算法是在优化一个高度非凸的函数，预训练阶段得到的参数值可以为优化损失函数提供一个很好的初始点（优化算法一般采用的是数值迭代的方法，需要初始化），然后微调阶段中的迭代则可以很快地找到一个不错的局部极小点，整个训练过程也就终止。因此，通过预训练和微调这两个阶段的训练方

法，可以得到适合机器学习任务（比如说是分类任务）的原始数据的很好的一种表示。

例如，假设要训练一个 4 层的神经网络模型用于分类任务，网络结构如下：

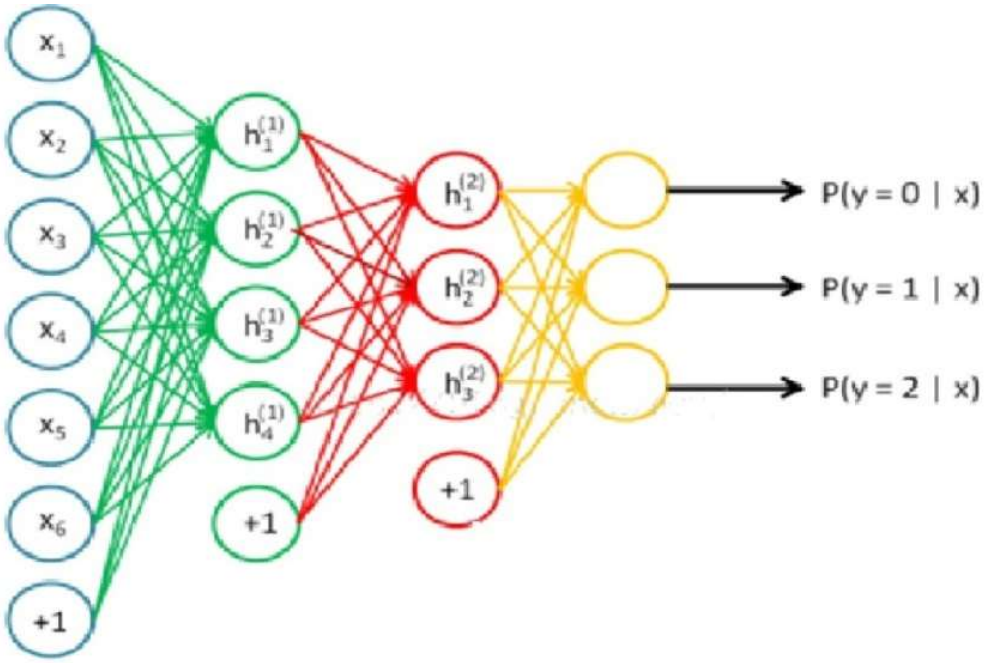


图 2.12 一个 4 层的神经网络模型

那么，数据再表达的前端学习就包括输入层和两个隐藏层的自编码过程，最后一个隐藏层就是用于分类任务的数据再表达（特征）。栈式自编码的训练上面这个网络的方法就是无监督预训练。

（i）首先采用稀疏自编码网络（也可以是上述任何一种自编码方法），训练从输入层到h_1层的参数。训练完毕后，去除解码层，只留下从输入层到隐藏层的编码阶段。

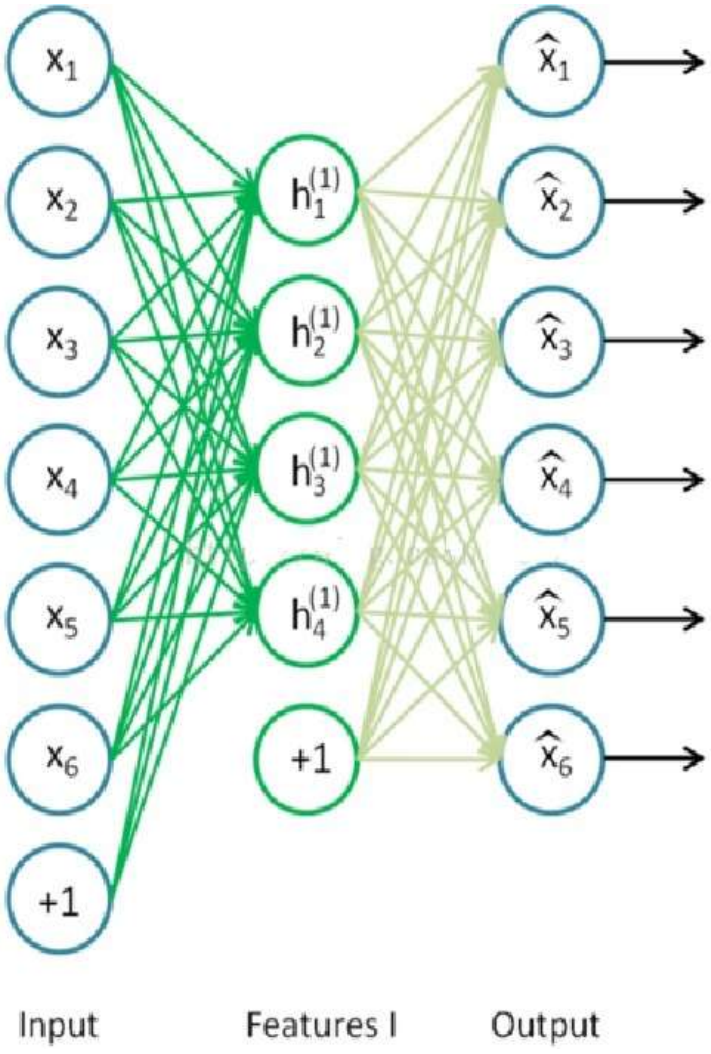


图 2.13 第一层数据再表达的训练

（ii）接着训练从h_1到h_2的参数。把无标签数据的 h1 层神经元的激活值，作为h_2层的输入层，然后再进行自编码训练；

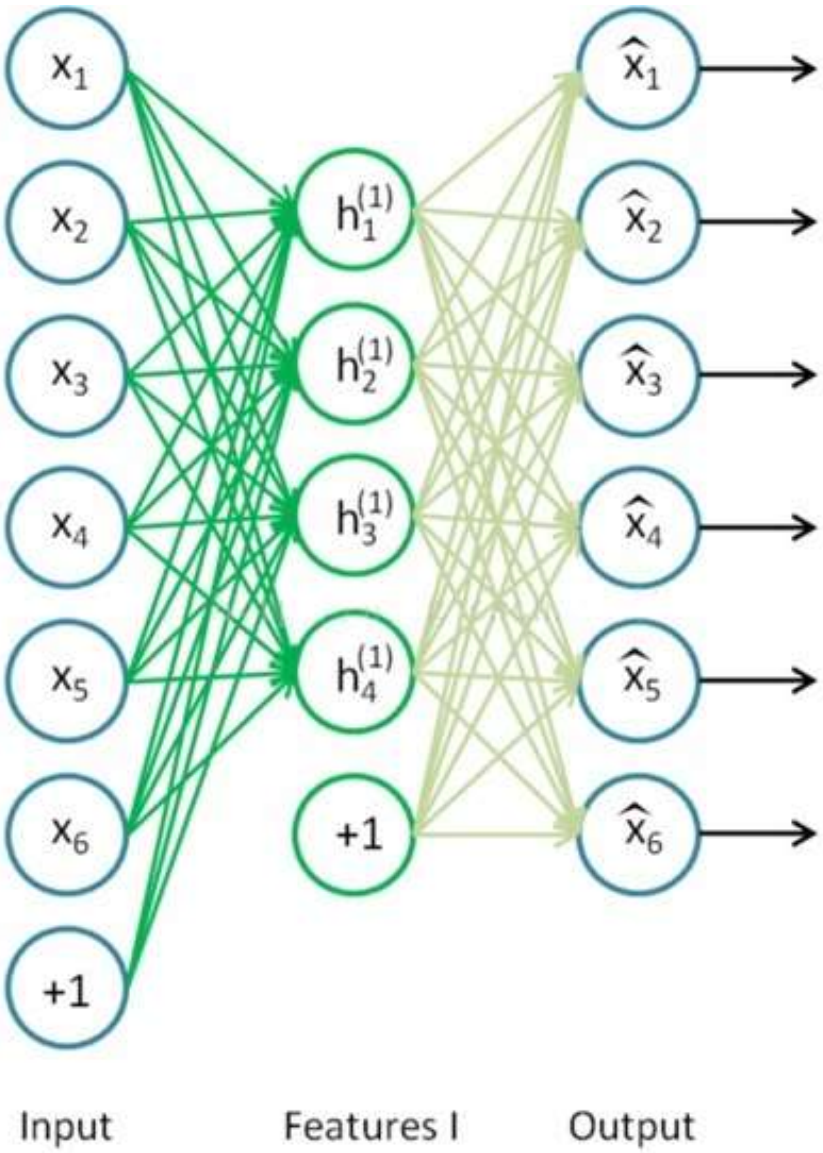


图 2.14 第二层数据再表达的训练

最后训练完毕后，再去除h_2层的解码层。如此重复，可以训练更高层的网络，这就是逐层贪婪训练的思想。这个过程就是所谓的无监督预训练的前端学习过程。

（iii）训练完h_2后，就可以接分类层 softmax，用于多分类任务。

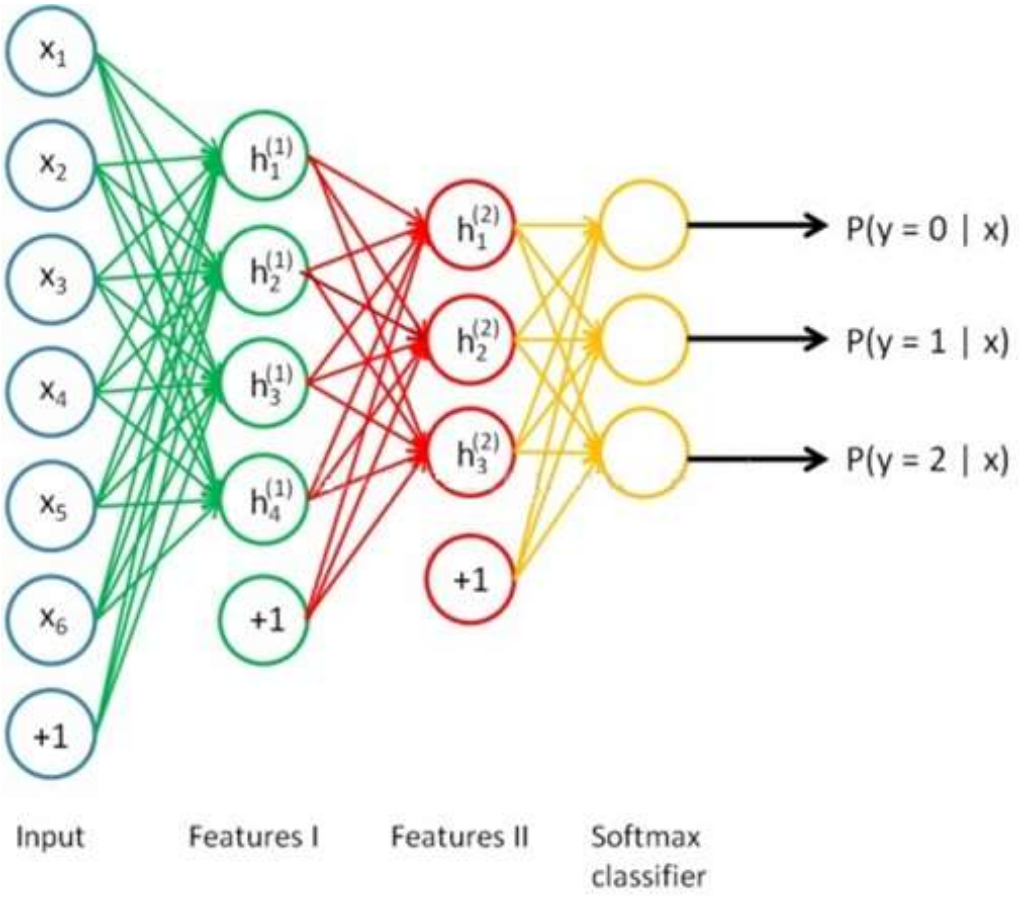


图 2.15 前端学习（数据再表达）的自编码网络连接后端学习的分类任务的网络结构