Deep Learning Technology and Application

Ge Li

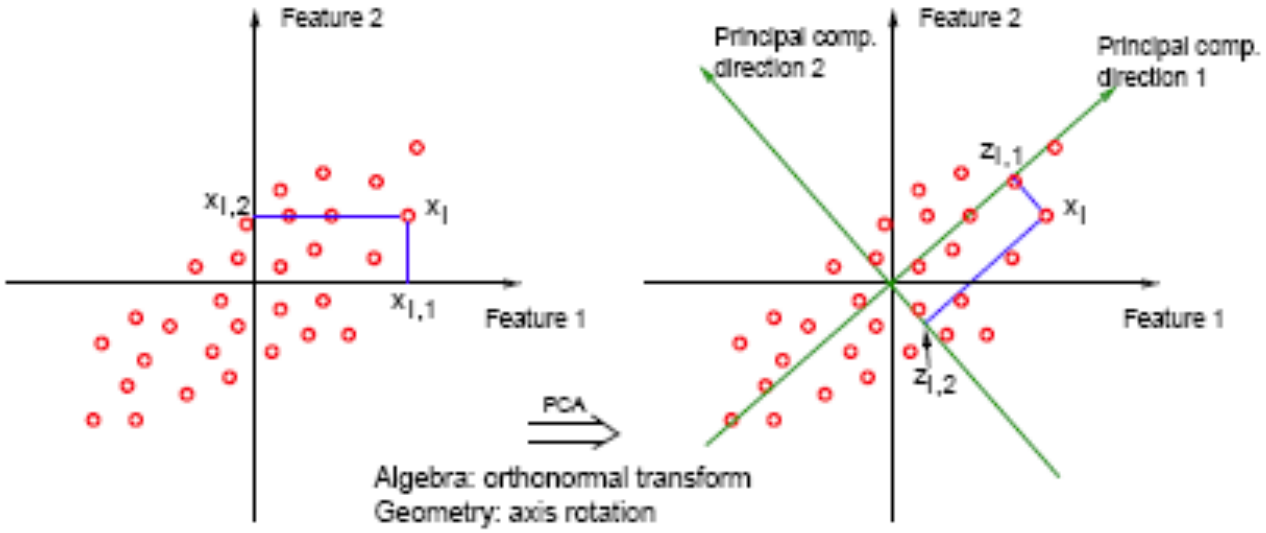
Peking University

主成分分析（PCA）

- 主成分分析（PCA）
  - Principal Component Analysis
  - 一种能够极大提升无监督特征学习速度的数据降维算法。
- 例如：
  - 在图像处理中，图像中相邻的像素之间具有较高的相关度，因此输入数据存在较大的冗余性。
  - 16x16的灰度值图像 对应 一个256维向量$x \in R^{256}$；其中特征值x_j对应每个像素的亮度值。
  - 由于相邻像素间的相关性，PCA算法可以将输入向量转换为一个维数低很多的近似向量，而且误差非常小。

主成分分析

- PCA 的思想
  - 通过线性变换是将n维特征映射到k维（k<n）全新的正交特征上。



PCA计算过程

- Step 1：
  - 分别求原始数据集中各个特征（各个列）的平均值，然后，对所有样例的各个特征，均减去该特征的均值。

Data =

x	y
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9

DataAdjust =

x	y
.69	.49
-1.31	-1.21
.39	.99
.09	.29
1.29	1.09
.49	.79
.19	-.31
-.81	-.81
-.31	-.31
-.71	-1.01

PCA计算过程

- Step 2：求各个特征的协方差矩阵；
  - 特征的协方差：
    - [两个（特征 相对于 其 数学期望 之差）的乘积]的数学期望
    - 协方差大于0表示x和y若有一个增，另一个也增；小于0表示若有一个增，另一个则减；协方差为0时，表示两者独立。
    - 协方差绝对值越大，两者对彼此的影响越大，反之越小。

Data =

x	y
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9

DataAdjust =

x	y
.69	.49
-1.31	-1.21
.39	.99
.09	.29
1.29	1.09
.49	.79
.19	-.31
-.81	-.81
-.31	-.31
-.71	-1.01

$$cov = \begin{pmatrix} .616555556 & .615444444 \\ .615444444 & .716555556 \end{pmatrix}$$

PCA计算过程

- Step 3：求协方差方阵的特征值和特征向量，得到

Data =

x	y
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9

DataAdjust =

x	y
.69	.49
-1.31	-1.21
.39	.99
.09	.29
1.29	1.09
.49	.79
.19	-.31
-.81	-.81
-.31	-.31
-.71	-1.01

$$cov = \begin{pmatrix} .616555556 & .615444444 \\ .615444444 & .716555556 \end{pmatrix}$$

$$eigenvalues = \begin{pmatrix} .0490833989 \\ 1.28402771 \end{pmatrix}$$

$$eigenvectors = \begin{pmatrix} -.735178656 & -.677873399 \\ .677873399 & -.735178656 \end{pmatrix}$$

PCA计算过程

Step 4：组建特征向量矩阵

- 将特征值按照从大到小的顺序排序，选择其中最大的k个，然后将其对应的k个特征向量分别作为列向量组成特征向量矩阵。

Data =

x	y
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9

DataAdjust =

x	y
.69	.49
-1.31	-1.21
.39	.99
.09	.29
1.29	1.09
.49	.79
.19	-.31
-.81	-.81
-.31	-.31
-.71	-1.01

$$cov = \begin{pmatrix} .616555556 & .615444444 \\ .615444444 & .716555556 \end{pmatrix}$$

$$eigenvalues = \begin{pmatrix} .0490833989 \\ 1.28402771 \end{pmatrix}$$

$$eigenvectors = \begin{pmatrix} -.735178656 & -.677873399 \\ .677873399 & -.735178656 \end{pmatrix}$$

选择最大的特征向量1.28402771；
其对应的特征向量是$(0.677873399, 0.735178656)^T$

PCA计算过程

- Step 5：将样本点投影到选取的特征向量上
  - 设数据样本数为m，特征数为n；
  - 减去均值后的样本矩阵为$DataAdjust(m*n)$；
  - 选取的k个特征向量组成的矩阵为$EigenVectors(n*k)$；
  - 则，投影后的数据$FinalData$为：

$$FinalData(m*k) = DataAdjust(m*n) \times EigenVectors(n*k)$$

Data =

x	y
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9

DataAdjust =

x	y
.69	.49
-1.31	-1.21
.39	.99
.09	.29
1.29	1.09
.49	.79
.19	-.31
-.81	-.81
-.31	-.31
-.71	-1.01

$$cov = \begin{pmatrix} .616555556 & .615444444 \\ .615444444 & .716555556 \end{pmatrix}$$

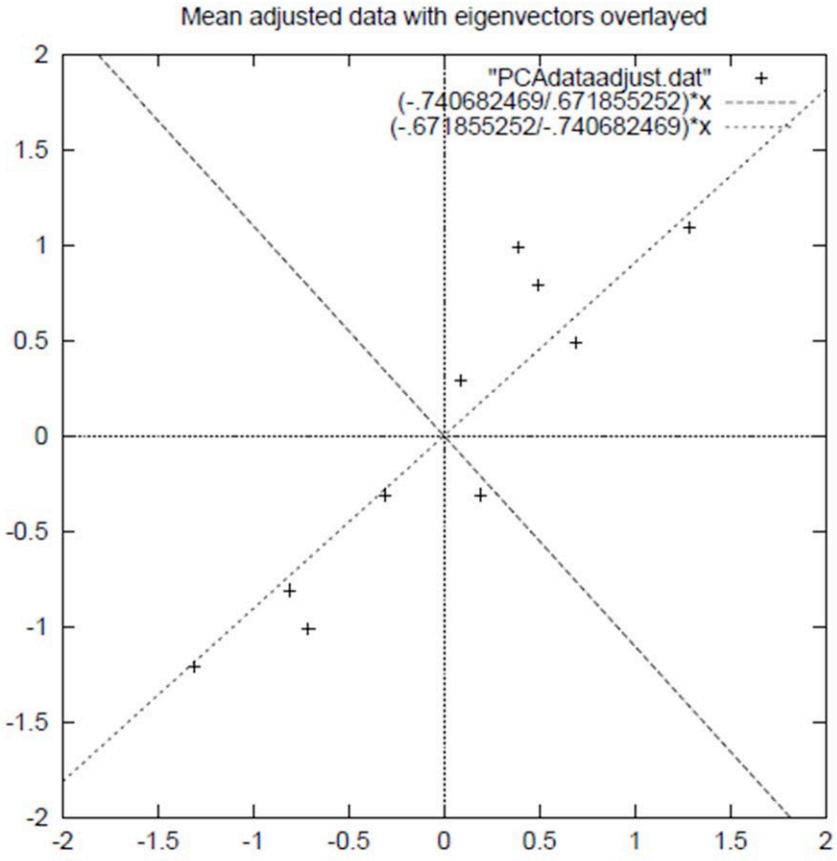
$$eigenvalues = \begin{pmatrix} .0490833989 \\ 1.28402771 \end{pmatrix}$$

$$eigenvectors = \begin{pmatrix} -.735178656 & -.677873399 \\ .677873399 & -.735178656 \end{pmatrix}$$

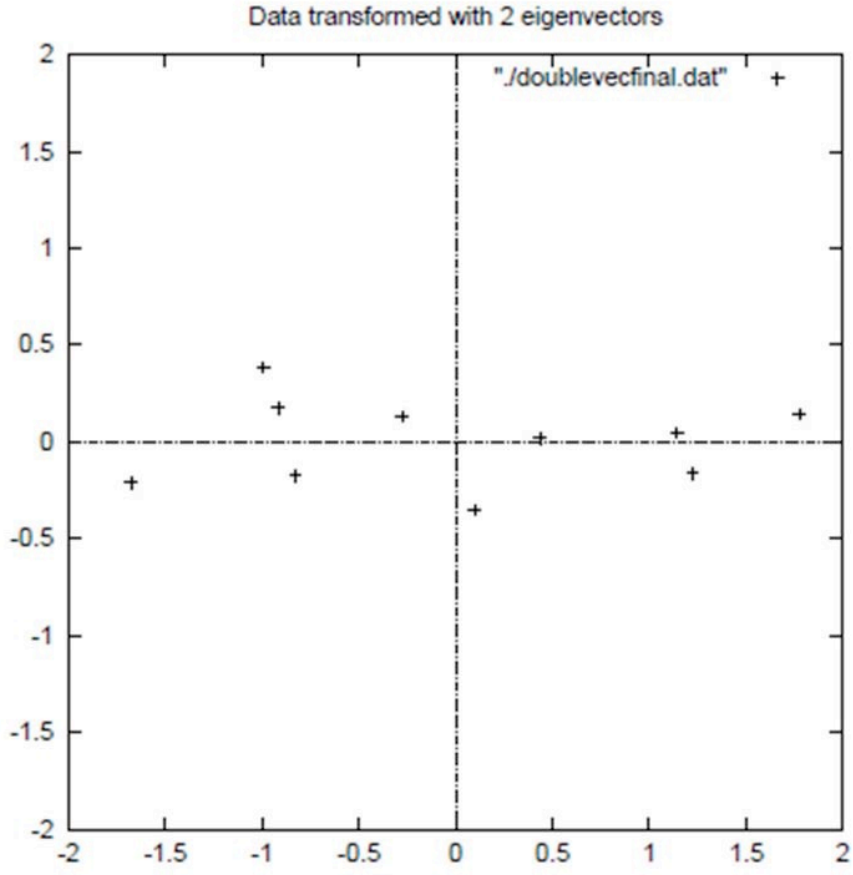
Transformed Data (Single eigenvector)

x
-.827970186
1.77758033
-.992197494
-.274210416
-1.67580142
-.912949103
.0991094375
1.14457216
.438046137
1.22382056

PCA计算过程



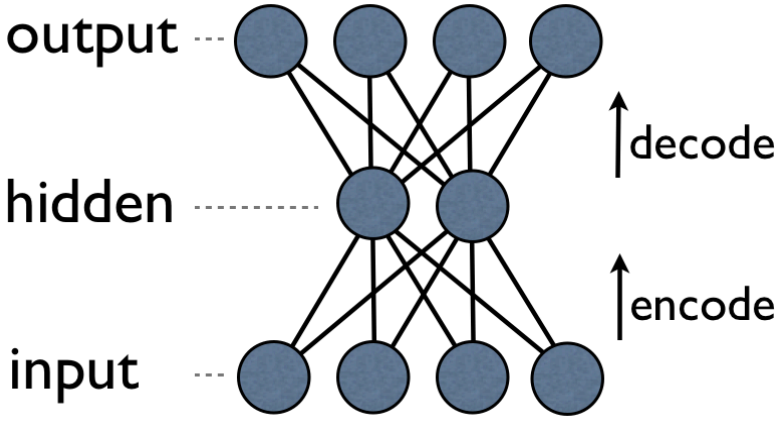
K=1 情况下结果



K=2 情况下结果

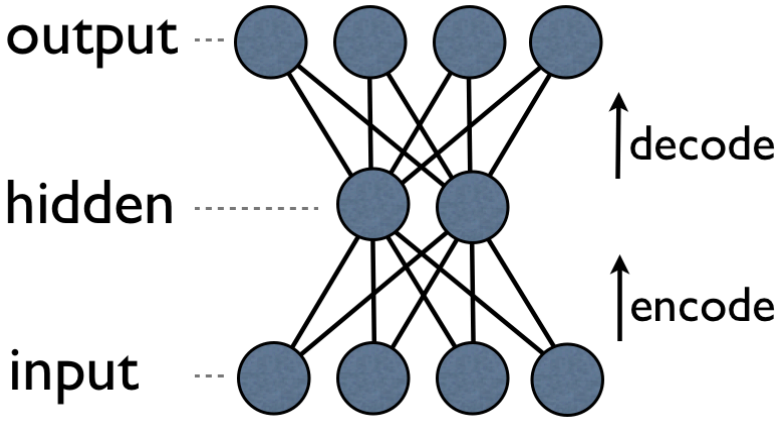
AutoEncoder

- Auto-Encoder（AE）
  - 自编码器，80年代晚期出现
  - 主要用于降维，后用于主成分分析



n 输入输出层神经元个数
m 隐藏层神经元个数
x, y, h 各层神经元上的向量
p , q 各层神经元的偏置
w 输入层与隐藏层之间的权值
$\tilde{w}$ 隐藏层与输出层之间的权值

Basic AutoEncoder



n 输入输出层神经元个数
m 隐藏层神经元个数
x, y, h 各层神经元上的向量
p，q 各层神经元的偏置
w 输入层与隐藏层之间的权值
$\tilde{w}$ 隐藏层与输出层之间的权值

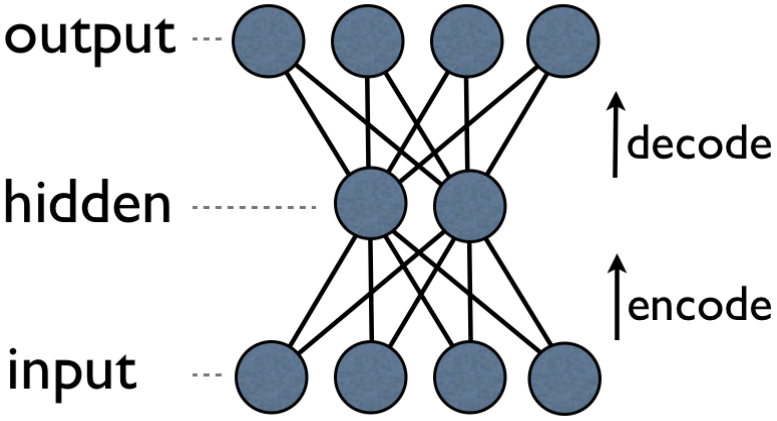
$$\mathbf{h} = f(\mathbf{x}) := s_f(W\mathbf{x} + \mathbf{p});$$
$$\mathbf{y} = g(\mathbf{h}) := s_g(\widetilde{W}\mathbf{h} + \mathbf{q}),$$

$$s_f(z) = \frac{1}{1+e^{-z}};$$

$$s_g(z) = \frac{1}{1+e^{-z}} \text{ 或 } s_g(z) = z.$$

Sometimes : $\widetilde{W} = W^T$

Basic AutoEncoder



$$\mathbf{h} = f(\mathbf{x}) := s_f(W\mathbf{x} + \mathbf{p});$$
$$\mathbf{y} = g(\mathbf{h}) := s_g(\widetilde{W}\mathbf{h} + \mathbf{q}),$$

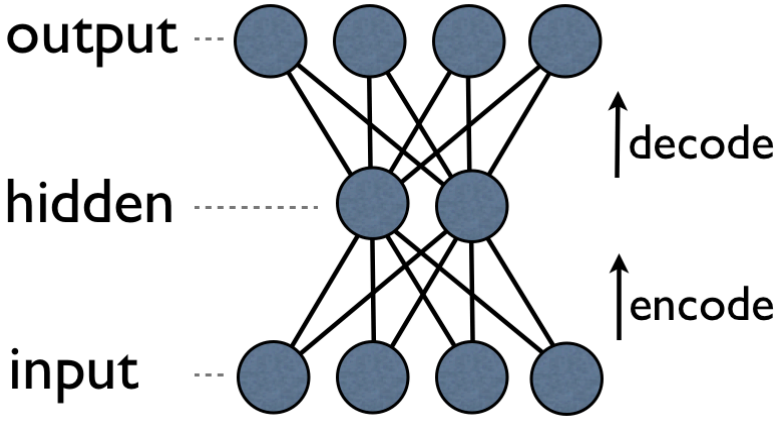
- 若 $s_g(z) = z.$ 通常取 重构误差 函数为 平方误差：

$$L(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|^2.$$

- 若 $s_g(z) = \frac{1}{1+e^{-z}}$ 通常取 重构误差 函数为 交叉熵：

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

Basic AutoEncoder



$$L(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|^2.$$

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

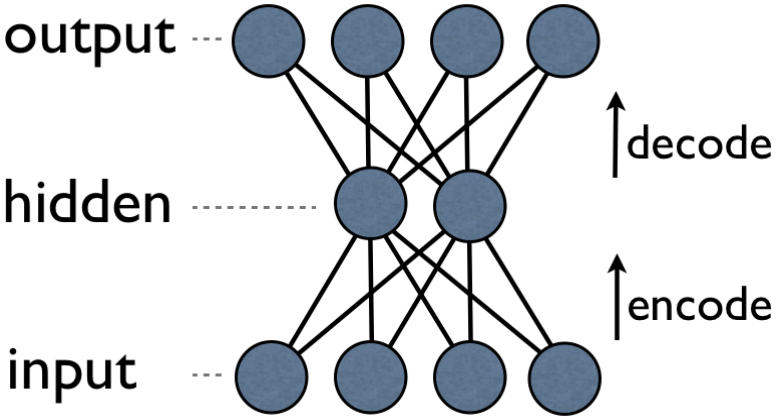
- 整体 损失函数 为：

$$\mathcal{J}_{AE}(\theta) = \sum_{\mathbf{x} \in S} L(\mathbf{x}, g(f(\mathbf{x}))),$$

- 或：

$$\mathcal{J}_{AE}(\theta) = \frac{1}{N} \sum_{\mathbf{x} \in S} L(\mathbf{x}, g(f(\mathbf{x}))),$$

Basic AutoEncoder

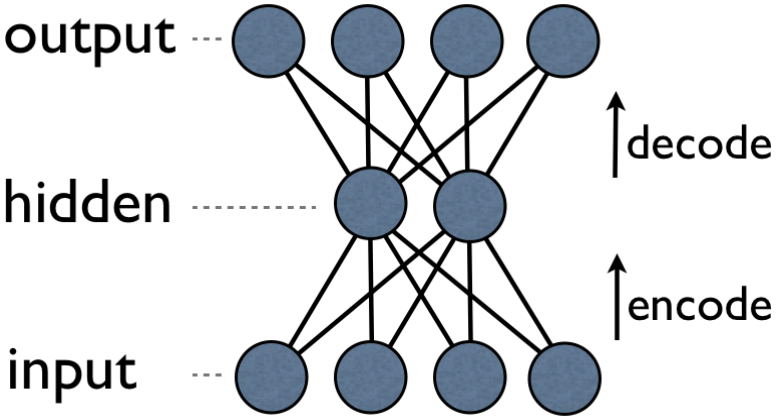


$$\mathcal{J}_{AE}(\theta) = \sum_{\mathbf{x} \in S} L(\mathbf{x}, g(f(\mathbf{x}))),$$

$$\mathcal{J}_{AE}(\theta) = \frac{1}{N} \sum_{\mathbf{x} \in S} L(\mathbf{x}, g(f(\mathbf{x}))),$$

- 问题：
  - 训练过程中，如果不对 重构函数 进行 限制，而直接进行最小化求解
  - 则，训练结果 极有可能 成为一个恒等函数
  - 因此，必须对 损失函数 进行正则化处理，以限制其表现

Regularized AutoEncoder



$$\mathcal{J}_{AE}(\theta) = \sum_{\mathbf{x} \in S} L(\mathbf{x}, g(f(\mathbf{x}))),$$

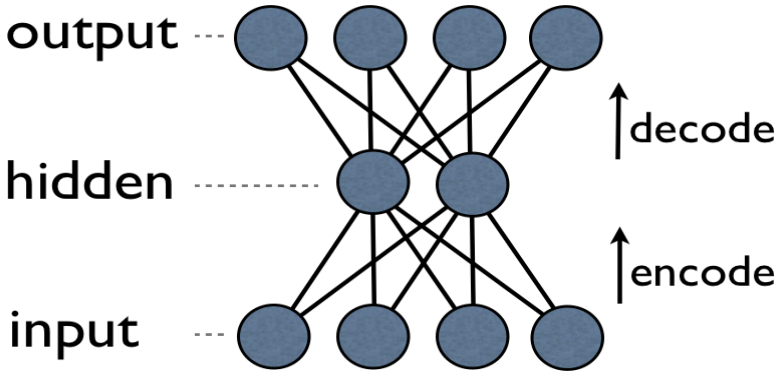
$$\mathcal{J}_{AE}(\theta) = \frac{1}{N} \sum_{\mathbf{x} \in S} L(\mathbf{x}, g(f(\mathbf{x}))),$$

- 增加权重衰减项的 损失函数：

$$\mathcal{J}_{AE+wd}(\theta) = \sum_{\mathbf{x} \in S} L(\mathbf{x}, g(f(\mathbf{x}))) + \lambda \sum_{i,j} W_{i,j}^2,$$

权重衰减项

Sparse AutoEncoder



n 输入输出层神经元个数
m 隐藏层神经元个数
x, y, h 各层神经元上的向量
p，q 各层神经元的偏置
w 输入层与隐藏层之间的权值
$\tilde{w}$ 隐藏层与输出层之间的权值

- 隐藏层上第 j 号神经元在训练集 $S = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$ 上的平均激活度.

$$\widehat{\rho}_j = \frac{1}{N}\sum_{i=1}^{N} h_j(\mathbf{x}^{(i)}), \quad \text{令} \quad \widehat{\rho}_j = \rho,\ j = 1, 2, \cdots, m,$$

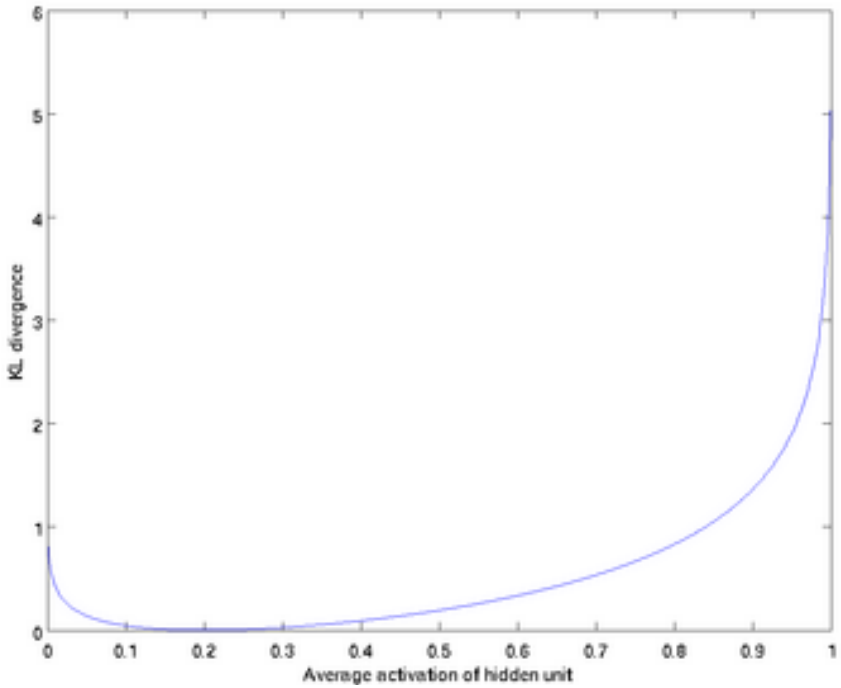
其中ρ为一个很小的数，例如$\rho < 0.05$

目的：当某隐藏层神经元的平均激活度超过ρ时，对其进行惩罚处理

如何进行惩罚处理？

- 引入 相对熵值 函数

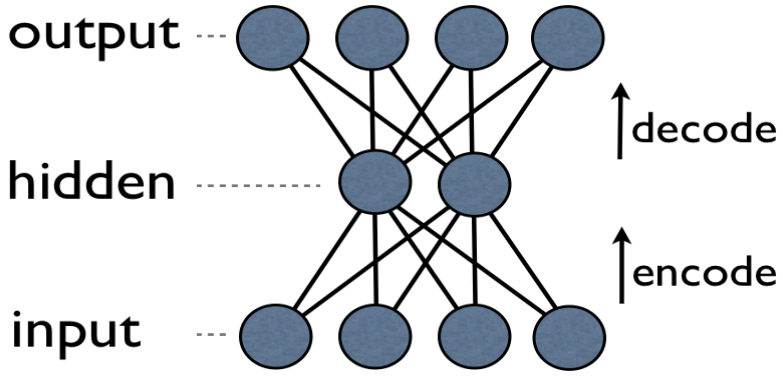
$$KL(\rho||\hat{\rho}_j) = \rho * \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) * \log \frac{1-\rho}{1-\hat{\rho}_j}.$$



特点：

- 在 $\hat{\rho}_j = \rho$ 时达到最小值 $\hat{\rho}_j$
- 当 靠近0或者1的时候，相对熵则变得非常大；

Sparse AutoEncoder



$\boldsymbol{n}$ 输入输出层神经元个数
m 隐藏层神经元个数
x, y, h 各层神经元上的向量
p，q 各层神经元的偏置
w 输入层与隐藏层之间的权值
$\tilde{w}$ 隐藏层与输出层之间的权值

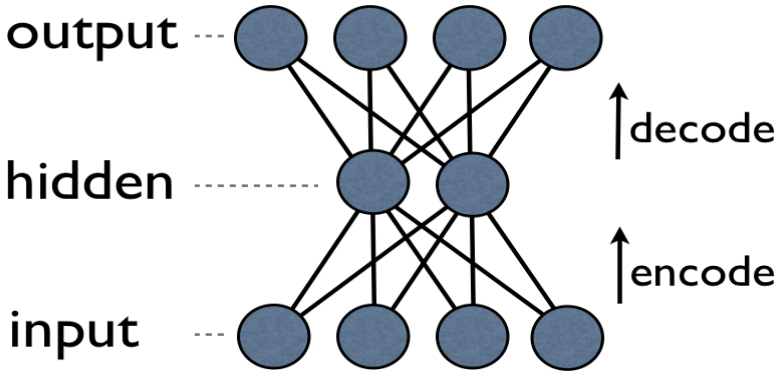
- 整体 损失函数 为：

$$\mathcal{J}_{AE+sp}(\theta) = \sum_{\mathbf{x}\in S} L(\mathbf{x}, g(f(\mathbf{x}))) + \beta \sum_{j=1}^{m} KL(\rho||\hat{\rho}_j),$$

- 结合正则化的 损失函数 为：

$$\mathcal{J}_{AE+wd+sp}(\theta) = \sum_{\mathbf{x}\in S} L(\mathbf{x}, g(f(\mathbf{x}))) + \lambda \sum_{i,j} W_{i,j}^2 + \beta \sum_{j=1}^{m} KL(\rho||\hat{\rho}_j).$$

Sparse AutoEncoder



$$\mathcal{J}_{AE+sp}(\theta) = \sum_{\mathbf{x}\in S} L(\mathbf{x}, g(f(\mathbf{x}))) + \beta \sum_{j=1}^{m} KL(\rho||\hat{\rho}_j),$$

- 反向传播的 残差 计算：

$$\delta_i^{(2)} = \left(\sum_{j=1}^{s_2} W_{ji}^{(2)} \delta_j^{(3)} \right) f'(z_i^{(2)}),$$

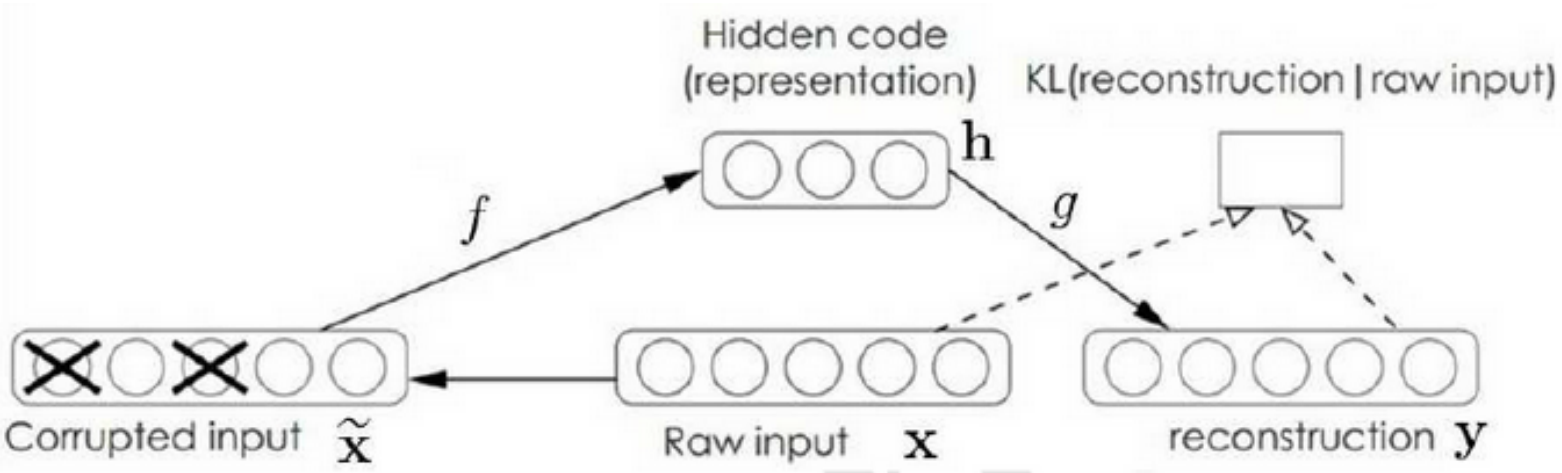
现在我们将其换成

$$\delta_i^{(2)} = \left(\left(\sum_{j=1}^{s_2} W_{ji}^{(2)} \delta_j^{(3)} \right) + \beta \left(-\frac{\rho}{\hat{\rho}_i} + \frac{1-\rho}{1-\hat{\rho}_i} \right) \right) f'(z_i^{(2)}).$$

Denoising AutoEncoder

- Sparse AutoEncoder
  - 优点：隐藏层利于保持输入层的特征
  - 缺点：容易受到 输入数据中的扰动 的干扰
- Denoising AutoEncoder
  - P. Vincent等人于2008年提出的一种抵御输入数据扰动的AutoEncoder
  - 对输入数据进行 corruption 以提供网络的 健壮性

Denoising AutoEncoder



$$\mathbf{h} = f(\tilde{\mathbf{x}}) := s_f(W\tilde{\mathbf{x}} + \mathbf{p});$$

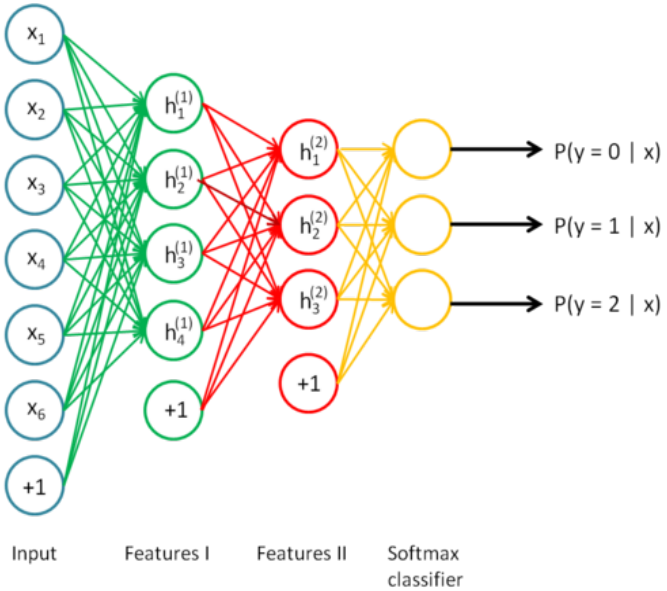
$$\mathbf{y} = g(\mathbf{h}) := s_g(\widetilde{W}\mathbf{h} + \mathbf{q}),$$

$$L(\mathbf{x}, \mathbf{y}) = L(\mathbf{x}, g(f(\tilde{\mathbf{x}}))$$

多层自编码网络

栈式自编码神经网络

- 一个由多层稀疏自编码器组成的神经网络；
- 前一层自编码器的输出作为其后一层自编码器的输入；



按照从前向后的顺序，执行每一层自编码器的编码步骤：

$$a^{(l)} = f(z^{(l)})$$

$$z^{(l+1)} = W^{(l,1)}a^{(l)} + b^{(l,1)}$$

按照从后向前的顺序，执行每一层自编码器的解码步骤：

$$a^{(n+l)} = f(z^{(n+l)})$$

$$z^{(n+l+1)} = W^{(n-l,2)}a^{(n+l)} + b^{(n-l,2)}$$

多层自编码网络

- 训练方法

  逐层贪婪训练法

  ① 先利用原始输入来训练网络的第一层，得到参数 $W^{(1,1)}, W^{(1,2)}, b^{(1,1)}, b^{(1,2)}$；

  ② 网络第一层将原始输入转化成为由隐藏单元激活值组成的向量；

  ③ 该向量作为第二层的输入，继续训练得到第二层的参数 $W^{(2,1)}, W^{(2,2)}, b^{(2,1)}, b^{(2,2)}$；

  ④ 对后面的各层同样采用的策略，依次训练。

  思想：在训练每一层参数的时候，固定其它各层参数保持不变。

Fine-tuning

- Fine-tuning
  - 在预训练过程完成之后，通过反向传播算法同时调整所有层的参数以改善结果，这个过程一般被称作“微调（fine-tuning）”。
- 基本做法
  - 使用逐层贪婪训练方法将参数训练到快要收敛时，再使用微调。
  - 反之，如果直接在随机化的初始权重上使用微调，容易使参数收敛到局部最优。
- 如果只对以分类为目的，那么惯用的做法是
  - 丢掉栈式自编码网络的“解码”层，直接把最后一个隐藏层的输出作为特征输入到softmax分类器进行分类，这样，分类器的分类错误的梯度值就可以直接反向传播给编码层。

MNIST Database

- The MNIST database
  - Mixed National Institute of Standards and Technology database
  - A large database of handwritten digits that is commonly used for training various image processing systems.
- Origin
  - It was created by "re-mixing" the samples from NIST's original datasets.
    - NIST's training dataset was taken from American Census Bureau employees, while the testing dataset was taken from American high school students, NIST's complete dataset was too hard.[5]
    - Furthermore, the black and white images from NIST were normalized to fit into a 20x20 pixel bounding box and anti-aliased, which introduced grayscale levels.

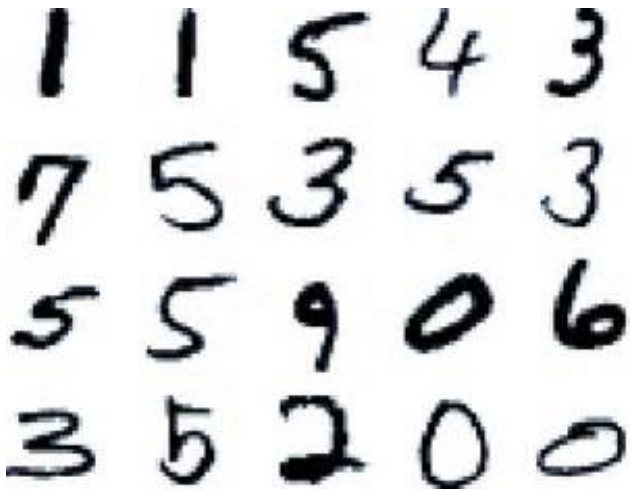
MNIST Database

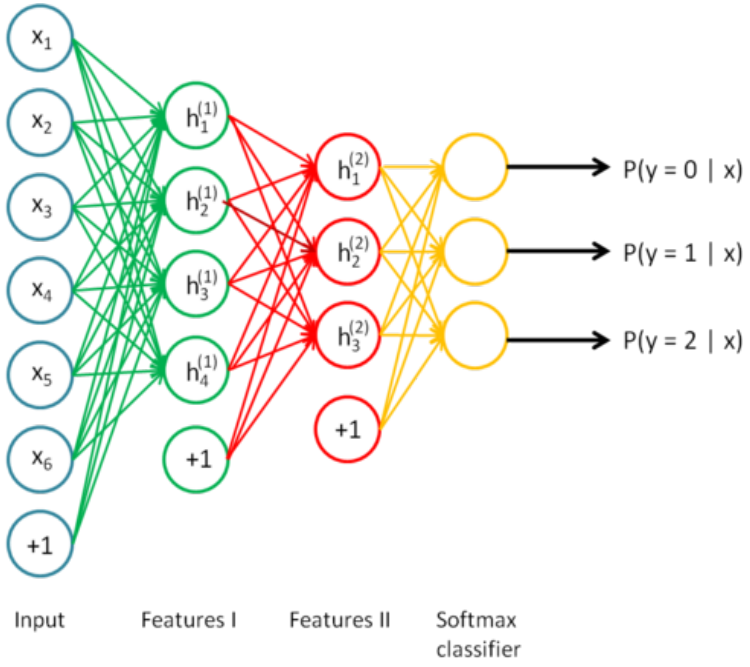
- Data
  - The database contains 60,000 training images and 10,000 testing images.
  - Half of the training set and half of the test set were taken from NIST's training dataset, while the other half of the training set and the other half of the test set were taken from NIST's testing dataset.

Type	Classifier	Preprocessing	Error rate (%)
Linear classifier	Pairwise linear classifier	Deskewing	7.6[9]
K-Nearest Neighbors	K-NN with non-linear deformation (P2DHMDM)	Shiftable edges	0.52[14]
Boosted Stumps	Product of stumps on Haar features	Haar features	0.87[15]
Non-Linear Classifier	40 PCA + quadratic classifier	None	3.3[9]
Support vector machine	Virtual SVM, deg-9 poly, 2-pixel jittered	Deskewing	0.56[16]
Neural network	6-layer NN 784-2500-2000-1500-1000-500-10 (on GPU), with elastic distortions	None	0.35[17]
Convolutional neural network	Committee of 35 conv. net, 1-20-P-40-P-150-10, with elastic distortions	Width normalizations	0.23[8]

Handwritten Digits Classifiers

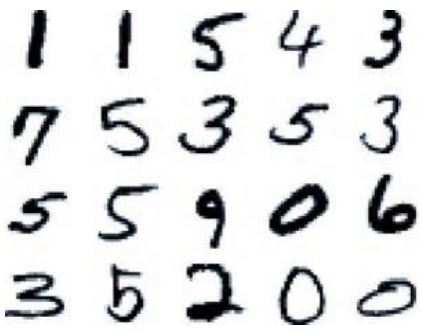
- 最终目标
  - 训练一个包含两个隐藏层的自编码器，用于MNIST手写数字识别。

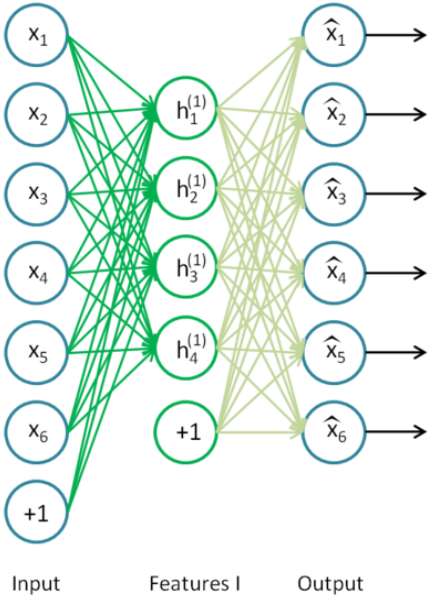




Handwritten Digits Classifiers

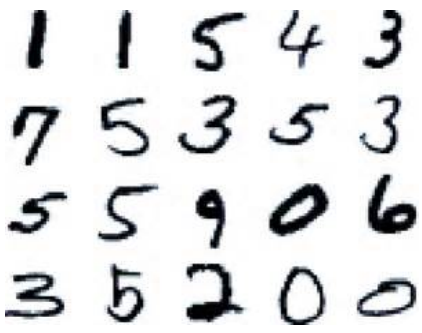
- Step 1：用原始输入$x^{(k)}$训练第一个自编码器，学习得到原始输入的一阶特征表示$h^{(1)(k)}$;
  - Step 1.5：把原始数据输入到Step1训练好的稀疏自编码器中，对于每一个原始输入$x^{(k)}$，都可以得到它对应的一阶特征表示$h^{(1)(k)}$

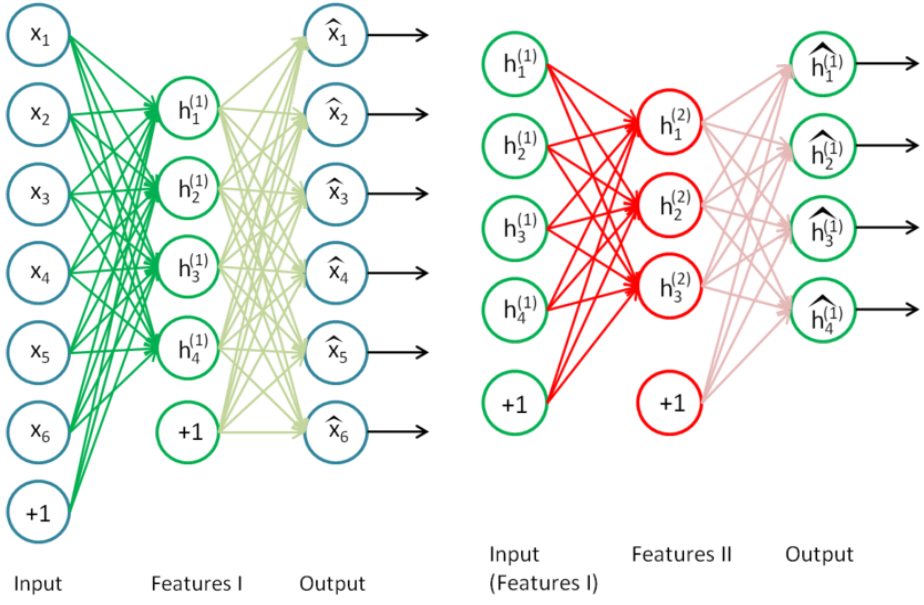




Handwritten Digits Classifiers

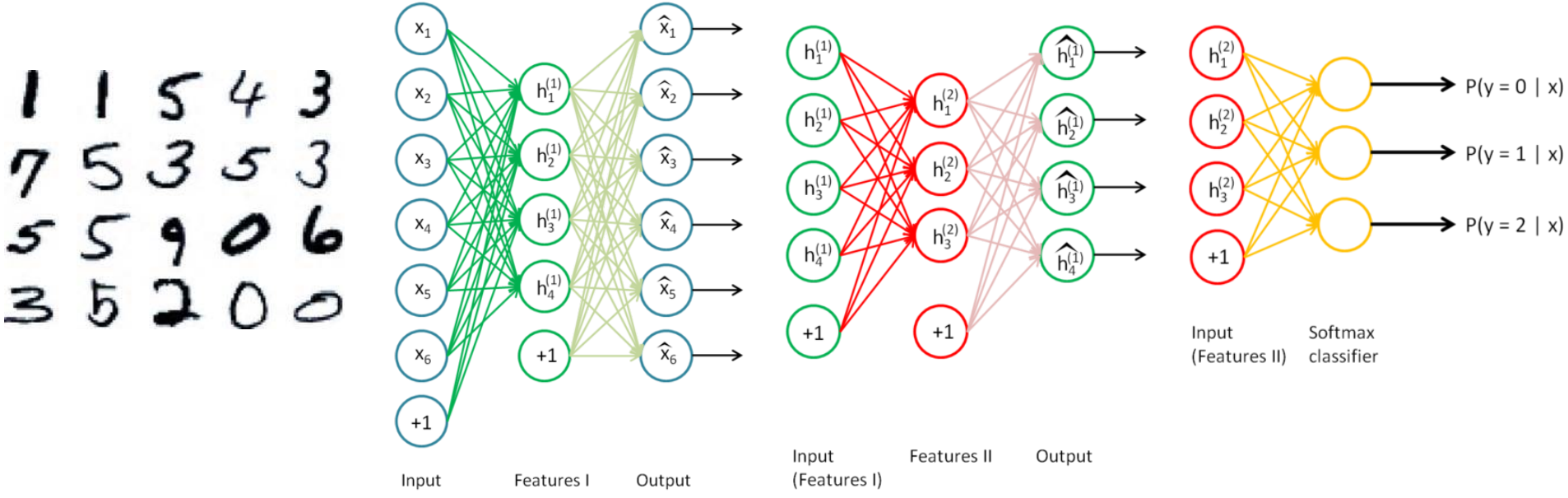
- Step 2：然后再用这些一阶特征作为另一个稀疏自编码器的输入，使用它们来学习二阶特征$h^{(2)(k)}$。
  - Step 2.5：把一阶特征输入到Step2训练好的第二层稀疏自编码器中，得到每个$h^{(1)(k)}$对应的二阶特征激活值$h^{(2)(k)}$。





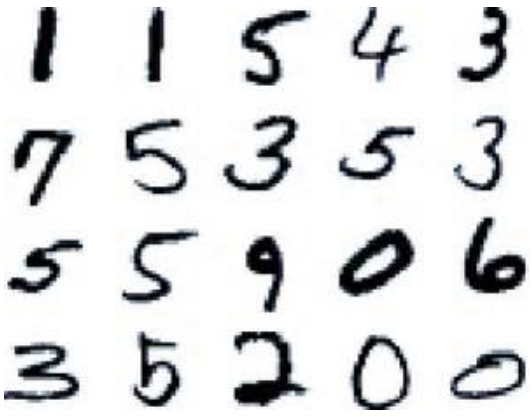
Handwritten Digits Classifiers

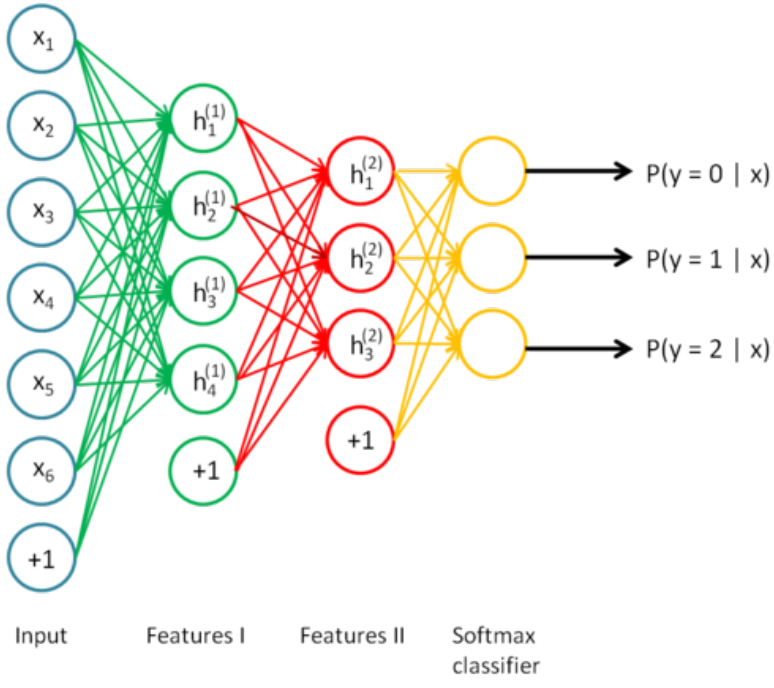
- Step 3：把这些二阶特征作为softmax分类器的输入，训练得到一个能将二阶特征映射到数字标签的模型。



Handwritten Digits Classifiers

- Step 4：将上述三层结合起来构建一个包含两个隐藏层和一个最终softmax分类器层的栈式自编码网络，对MNIST数字进行分类。





Thanks.