# 第十讲　无监督网络模型

主讲教师：滕竹

北京交通大学　《深度学习》课程组

无监督学习（ Unsupervised Learning， UL） 是指从无标签的数据中学习出一些有用的模式。

无监督学习算法一般直接从原始数据中学习，不借助于任何人工给出标签或者反馈等指导信息。如果监督学习是建立输入-输出之间的映射关系，那么无监督学习就是发现隐藏的数据中的有价值信息，包括有效的特征、类别、结构以及概率分布等。

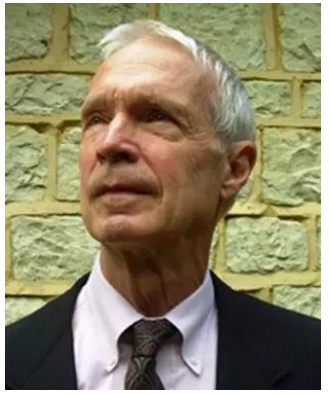
典型问题：无监督特征学习、概率密度估计、聚类等。

# 目录

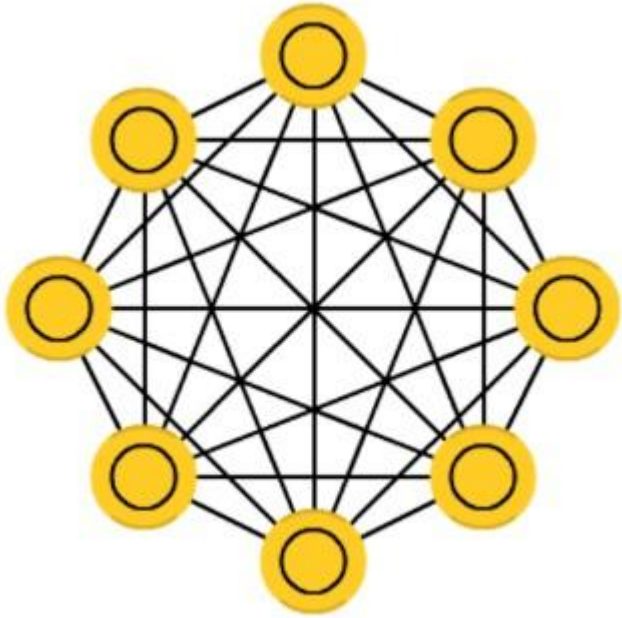
10.1

# Hopfield神经网络

# Hopfield神经网络



- Hopfield神经网络是一种单层互相全连接的反馈型神经网络。
- 每个神经元既是输入也是输出，网络中的每一个神经元都将自己的输出通过连接权传送给所有其它神经元，同时又都接收所有其它神经元传递过来的信息。即：网络中的神经元在t时刻的输出状态实际上间接地与自己t-1时刻的输出状态有关。
- 神经元之间互连接，所以得到的权重矩阵将是对称矩阵。



# Hopfield神经网络

假设有$n$个单元组成的Hopfield神经网络，第 $i$ 个单元在 $t$ 时刻的输入记作$u_i(t)$，输出记作$x_i(t)$，连接权重为$w_{ij}$，阈值为$b_i(t)$，则$t+1$时刻$i$单元的输出$x_i(t+1)$可表示为

$$x_i(t+1)=\begin{cases}1, & u_i(t+1)>0\\ x_i(t), & u_i(t+1)=0\\ 0, & u_i(t+1)<0\end{cases}$$

权重总和大于阈值，取1
权重总和小于阈值，取0

$$u_i(t+1)=\sum_{j=1}^{n} w_{ij}x_j(t)-b_i(t)$$

来自于其他单元的输入$x_j(t)$的权重综合

# Hopfield神经网络

- Hopfield网络的优点
  - ✓ 单元之间的连接权重对称（$w_{ij}=w_{ji}$）
  - ✓ 每个单元没有到自身的连接（ $w_{ii}=0$）
  - ✓ 单元的状态采用随机异步更新方式，每次只有一个单元改变状态
  - ✓ n个二值单元做成的二值神经网络，每个单元的输出只能是0或1的两个值

# Hopfield神经网络

- 在Hopfield神经网络中，每个时刻都只有一个随机选择的单元发生状态变化
- 对于一个由n个单元组成的网络，如果要完成全部单元的状态变化，至少需要n个时刻
- 单元的状态变化会一直进行下去，直到网络达到稳定状态。各单元的最终状态就是输出模式

# Hopfield神经网络

- 基于能量的模型是一种具有普适意义的模型，统计力学的结论表明，任何概率分布都可以转变成基于能量的模型(Energy Based Model, EBM)。EBM通过对变量的配置施加一个有范围限制的能量，即对每一个变量建立一个能量公式，来捕获这些变量之间的依赖关系

# Hopfield神经网络

❑根据输入模式联想输出模式时，需要事先确定连接权重$wij$，而连接权重$wij$要对输入模式的训练样本进行训练后才能确定。和多层神经网络一样，一次训练并不能确定连接权重，而是要不断重复这个过程，直到满足终止判断条件，而这个指标就是Hopfield神经网络的能量函数 $E$

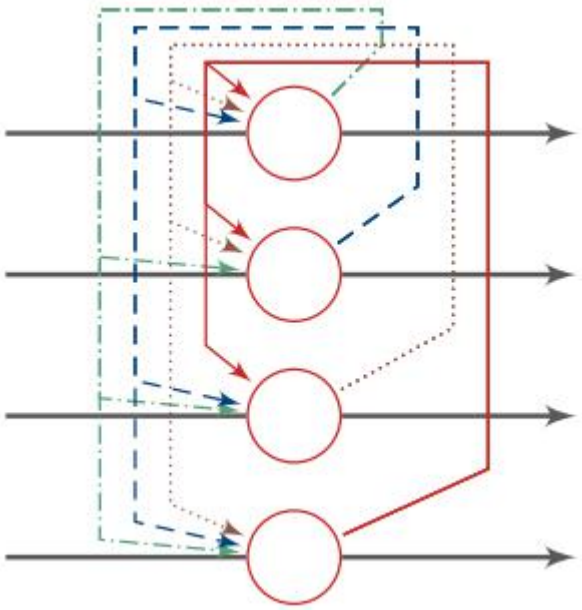
$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}x_i x_j + \sum_{i=1}^{n} b_i x_i$$

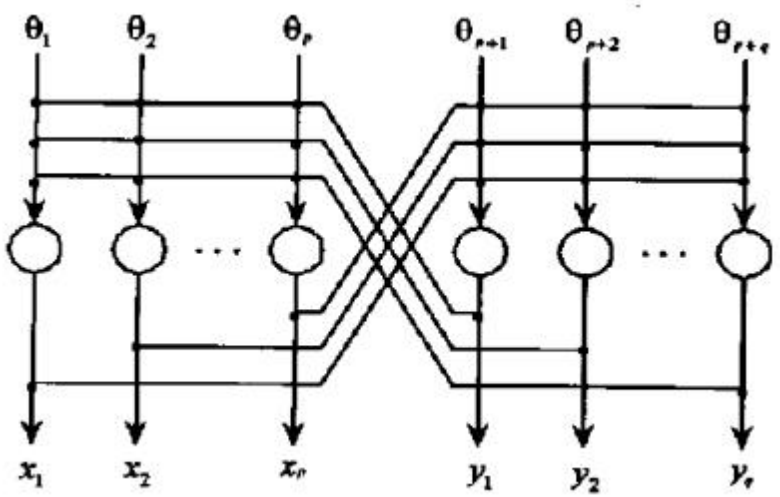
❑这种以能量函数作为网络计算的求解工具，被称为计算能量函数。

# Hopfield神经网络

联想记忆就是当输入模式为某种状态时，输出端要给出与之相应的输出模式

如果输入模式与输出模式一致，称为自联想记忆，否则称为异联想记忆






异联想记忆

输入：$x_1, x_2, \cdots, x_p$

输出：$y_1, y_2, \cdots, y_q$

# Hopfield神经网络

- 根据前面定义的状态变化规则改变网络状态时，上式中定义的能量函数 $E$ 总是非递增的，即随时间的不断增加而逐渐减小，直到网络达到稳定状态为止
  - 能量函数分解成单元$k$ 的能量函数和 $k$ 以外的单元的能量函数

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}x_i x_j + \sum_{i=1}^{n} b_i x_i$$

$$E = -\frac{1}{2}\left( x_k \sum_{j=1}^{n} w_{kj}x_j + \sum_{i\neq k}^{n}\sum_{j=1}^{n} w_{ij}x_i x_j \right) + b_k x_k + \sum_{i\neq k}^{n} b_k x_k$$

$$= -\frac{1}{2}\left( x_k \left( w_{kk}x_k + \sum_{j\neq k}^{n} w_{kj}x_j \right) + \sum_{i\neq k}^{n}\left( w_{ik}x_i x_k + \sum_{j\neq k}^{n} w_{ij}x_i x_j \right) \right) + b_k x_k + \sum_{i\neq k}^{n} b_i x_i$$

$$= -\frac{1}{2}\left( w_{kk}x_k^2 + x_k \sum_{j\neq k}^{n} w_{kj}x_j + \sum_{i\neq k}^{n}\left( w_{ik}x_i x_k + \sum_{j\neq k}^{n} w_{ij}x_i x_j \right) \right) + b_k x_k + \sum_{i\neq k}^{n} b_i x_i$$

$$= -\frac{1}{2}\sum_{i\neq k}^{n}\sum_{j\neq k}^{n} w_{ij}x_i x_j + \sum_{i\neq k}^{n} b_i x_i - \frac{1}{2}\left( \sum_{j\neq k}^{n} w_{kj}x_j + \sum_{i\neq k}^{n} w_{ik}x_i \right) x_k + b_k x_k$$

# Hopfield神经网络

- 根据前面定义的状态变化规则改变网络状态时，上式中定义的能量函数 $E$ 总是非递增的，即随时间的不断增加而逐渐减小，直到网络达到稳定状态为止
  - 能量函数分解成单元$k$ 的能量函数和 $k$ 以外的单元的能量函数

$$E = -\frac{1}{2}\sum_{i\neq k}^{n}\sum_{j\neq k}^{n} w_{ij}x_i x_j + \sum_{i\neq k}^{n} b_i x_i - \frac{1}{2}\left(\sum_{j\neq k}^{n} w_{kj}x_j + \sum_{i\neq k}^{n} w_{ik}x_i\right)x_k + b_k x_k$$

$$\Delta E_k = -\frac{1}{2}\left(\sum_{j\neq k}^{n} w_{kj}x_j + \sum_{i\neq k}^{n} w_{ik}x_i\right)\Delta x_k + b_k \Delta x_k \qquad \Delta x_k = x_k(t+1) - x_k(t)$$

$$\Delta E_k = -\left(\sum_{j\neq k}^{n} w_{kj}x_j - b_k\right)\Delta x_k = -u_k(t+1)\Delta x_k$$

当$\Delta x_k > 0$时，$x_k$由0变成1，即 $u_k(t+1) > 0$

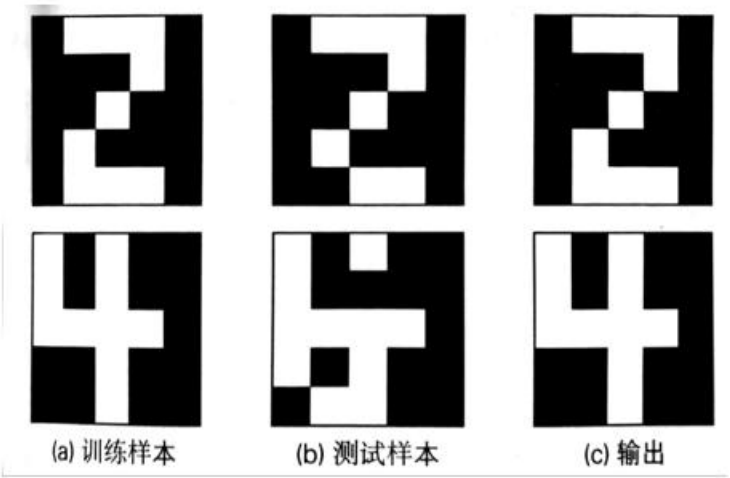
$$\Delta E_k(t) < 0$$

当$\Delta x_k < 0$时，$x_k$由1变成0，即 $u_k(t+1) < 0$

$$\Delta E_k(t) < 0$$

$$x_i(t+1) = \begin{cases} 1, & u_i(t+1) > 0 \\ x_i(t), & u_i(t+1) = 0 \\ 0, & u_i(t+1) < 0 \end{cases}$$

# Hopfield神经网络




Hopfield网络的应用

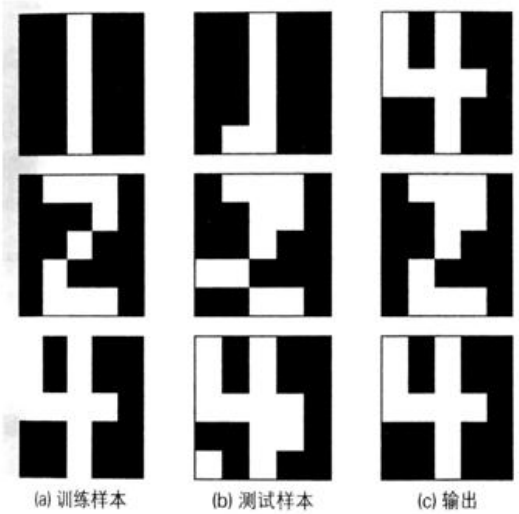
记住训练样本的输入

↓

自联想记忆

↓

对测试样本去噪




Hopfield网络的串扰

- 由于“4”中包含了“1”中的全部竖线，Hopfield网络错把带有“横线”噪音的测试样本“1”识别为“4”，这种情况即串扰

当需要记忆的模式之间较为相似，或者需要记忆的模式太多时，Hopfield神经网络就不能正确地辨别模式。这种相互干扰、不能准确记忆的情况成为串扰(crosstalk)。

# Hopfield神经网络

- Hopfield神经网络能够记忆的模式数量有限，大约是网络单元数的15%左右，为了防止串扰，可以采用先把模式的正交化再进行记忆等方法
- 但是正交化方法并不能完全解决问题，玻尔兹曼机可以解决这一问题

10.2

# 玻尔兹曼机

# 玻尔兹曼机

- 玻尔兹曼机也是相互连接型网络

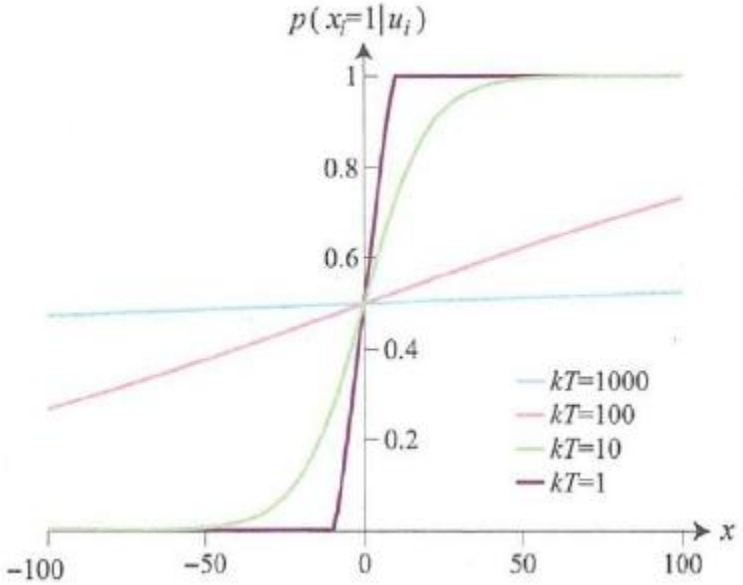
Hopfield神经网络的输出是按照某种确定性决定的，如果发生串扰或陷入局部最优解，Hopfield神经网络就不能正确的辨别模式。而玻尔兹曼机(Boltzmann Machine)通过让输出按照一定的概率分布发生状态变化（下页输出公式），来避免陷入局部最优解。

- 玻尔兹曼机保持了Hopfield神经网络的假设：
  - 权重对称
  - 自身无连接
  - 二值输出

# 玻尔兹曼机

❑ 玻尔兹曼机的输出是按照某种概率分布决定的

$$\begin{cases} p(x_i = 1|u_i) = \dfrac{\exp(\frac{x}{kT})}{1+\exp(\frac{x}{kT})} \\ p(x_i = 0|u_i) = \dfrac{1}{1+\exp(\frac{x}{kT})} \end{cases}$$




温度系数引起的概率变化

- $T(>0)$表示温度系数，当 $T$ 趋近于无穷时，无论$u_i$取值如何，$x_i$等于1 或 0 的概率都是1/2，这种状态称为稳定状态
- $k$是玻尔兹曼常数

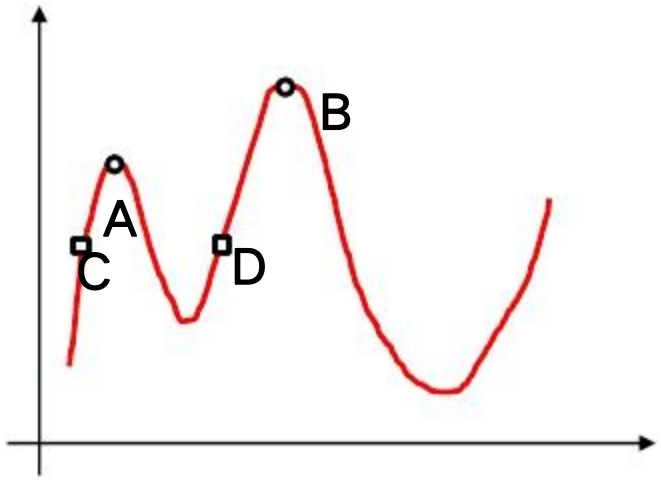
# 玻尔兹曼机

玻尔兹曼机选择模拟退火算法，可以先采用较大的温度系数进行粗调，然后逐渐减小温度系数进行微调。

温度系数越大，跳出局部最优解的概率越高。但是温度系数增大时，获得能量函数极小值的概率就会降低。反之，温度系数减小时，虽然获得能量函数极小值的概率增加了，但是玻尔兹曼机需要经历较长时间才能达到稳定状态

# 玻尔兹曼机

- 模拟退火算法




模拟退火是一种贪心算法，但是它的搜索过程引入了随机因素,以一定的概率来接受一个比当前解要差的解，因此有可能会跳出这个局部的最优解，达到全局的最优解,而且这个概率随着时间推移逐渐降低(逐渐降低才能趋向稳定)

# 玻尔兹曼机

- 玻尔兹曼机的训练过程

  - 1. 训练准备：初始化连接权重$w_{ij}$和偏置$b_i$
  - 2. 调整参数

    2.1 选取一个单元 $i$ ,求$u_i$

    2.2 根据$u_i$的值，计算输出$x_i$

    2.3 根据输出$x_i$和$x_j$的值，调整连接权重$w_{ij}$和偏置$b_i$

  - 重复步骤2，直到满足终止判断条件

# 玻尔兹曼机

- 具体而言，当初始化连接权重后，选取一个单元 $i$
  - 1.计算单元激活值$u_i$

$$x_i(t+1)=\begin{cases}1, & u_i(t+1)>0\\ x_i(t), & u_i(t+1)=0\\ 0, & u_i(t+1)<0\end{cases}$$

权重总和大于阈值，取1
权重总和小于阈值，取0

$$u_i(t+1)=\sum_{j=1}^{n} w_{ij}x_j(t)-b_i(t)$$

2.计算$x_i$等于 1 或 0 的概率

$$\begin{cases} p(xi=1|ui)=\dfrac{\exp(\frac{x}{kT})}{1+\exp(\frac{x}{kT})} \\ p(xi=0|ui)=\dfrac{1}{1+\exp(\frac{x}{kT})} \end{cases}$$

根据概率$x_i$等于1或0的概率，调整$x_i$的取值。一般是随机产生一个(0,1)之间的随机数λ，如果$p$>λ，确认状态改变，否则不改变

不能将计算所得概率直接作为$x_i$的值，而是作为概率来决定$x_i$的值

# 玻尔兹曼机

❑ 调整连接权重$w_{ij}$和偏置$b_i$，这里用似然函数$L(\theta)$导出调整值，$\theta$表示所有的连接权重和偏置

$$L(\theta) = \prod_{k=1}^{K} p(s_k|\theta)$$

- 其中，概率分布的定义如下，$E$表示能量函数，$Z(\theta)$是归一化常数
- 训练样本S={$s_1$,$s_2$,⋯,$s_k$,⋯,$s_K$}

$$p(s_k|\theta) = \frac{1}{Z(\theta)} exp\{-E(s_k, \theta)\}$$

$$Z(\theta) = \sum_{k} exp\{-E(s_k, \theta)\}$$

# 玻尔兹曼机

❑ 通常，使用对数似然函数求解

$$\ln L(\theta) = \sum_{k=1}^{K} \ln p(s_k|\theta)$$

➢ 当对数似然函数的梯度为0时，就可以得到最大似然估计量，即通过求连接权重$w_{ij}$和偏置$b_i$的相关梯度，可以求出调整值

# 玻尔兹曼机

❑ 求解困难

➢ 似然函数是基于所有单元组合来计算的，所以单元数过多将导致组合数异常庞大，无法进行实时计算。为了解决这个问题，人们提出了一种近似算法，对比散度算法

$$p(s_k|\theta) = \frac{1}{Z(\theta)} exp\{-E(s_k,\theta)\}$$

$$\ln L(\theta) = \sum_{k=1}^{K} \ln p(s_k|\theta)$$

$$\ln L(\theta) = -\ln Z(\theta) + \sum_{k=1}^{K} \left(-E(s_k,\theta)\right)$$

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}x_i x_j - \sum_{i=1}^{n} b_i x_i$$

$$\ln L(\theta) = -\ln Z(\theta) + \sum_{k=1}^{K}\left(\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}x_i^k x_j^k + \sum_{i=1}^{n} b_i x_i^k\right)$$

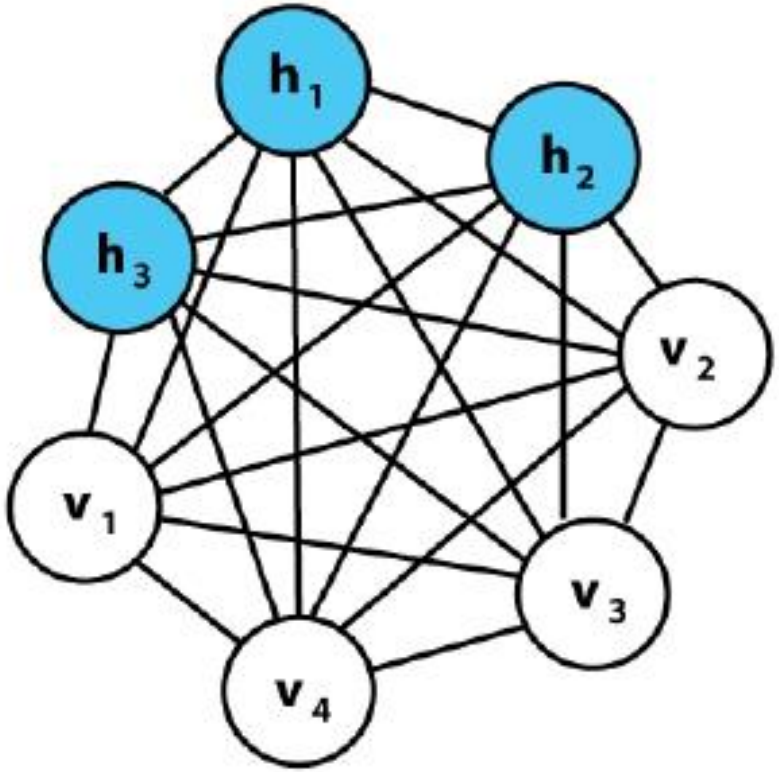
10.3

# 受限玻尔兹曼机

# 受限玻尔兹曼机

❑ 前面介绍的玻尔兹曼机默认了所有单元都为可见单元，在实际应用上，玻尔兹曼机还可以由可见单元和隐藏单元共同构成

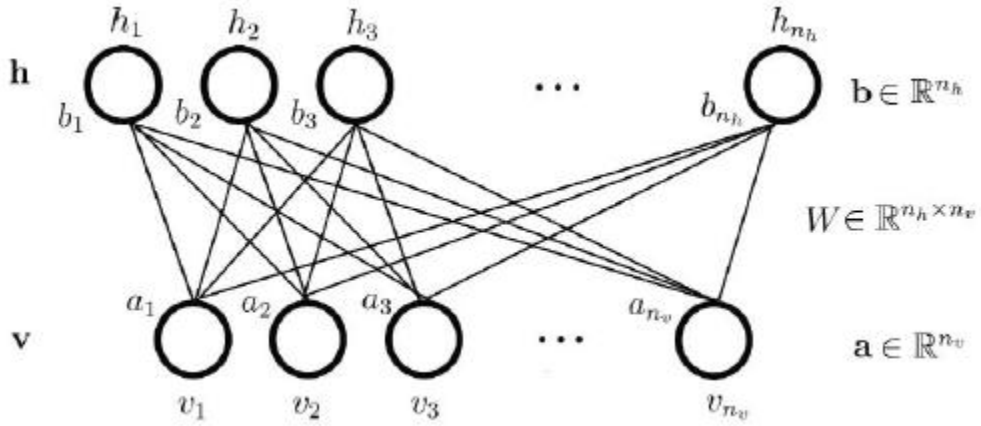
– 隐藏单元与输入数据没有直接联系，但会影响可见单元的概率。假设可见单元为可见变量 $v$ ，隐藏单元为隐藏变量 $h$ 。玻尔兹曼机含有隐藏变量时，概率分布仍然与前面计算的结果相同



# 受限玻尔兹曼机

❑ 含有隐藏变量的玻尔兹曼机训练非常困难，所以Hinton等人提出了受限玻尔兹曼机（Restricted Boltzmann Machine）

- 由可见层和隐藏层构成
- 层内单元之间无连接




- $n_v, n_h$分别表示可见层和隐藏层中包含的神经元数目，下标$v$和$h$代表visible和hidden
- $\mathbf{v}=(v_1;v_2;\cdots;v_{nv})$:可见层的状态向量，$v_i$表示可见层中第$i$个神经元的状态
- $\mathbf{h}=(h_1;h_2;\cdots;h_{nh})$:隐藏层的状态向量，$h_j$表示隐藏层中第$j$个神经元的状态
- $\mathbf{a}=(a_1;a_2;\cdots;a_{nv})$:可见层的偏置向量，$a_i$表示可见层中第$i$个神经元的偏置
- $\mathbf{b}=(b_1;b_2;\cdots;b_{nh})$:隐藏层的偏置向量，$b_j$表

示隐藏层中第$j$个神经元的偏置

- $W=(w_{ij})$:隐藏层和可见层之间的权值矩阵，

$w_{ij}$表示隐藏层中第$i$个神经元和可见层中第$j$

# 受限玻尔兹曼机

- RBM是基于能量的概率分布模型

  第一部分是能量函数，第二部分是基于能量函数的概率分布函数。

- 受限玻尔兹曼机的能量函数为:

$$E(\mathbf{v},\mathbf{h},\theta) = -\sum_{i=1}^{n_v} a_i v_i - \sum_{j=1}^{n_h} b_j h_j - \sum_{i=1}^{n_v}\sum_{j=1}^{n_h} w_{ij} v_i h_j$$

$$E(\mathbf{v},\mathbf{h},\theta) = -\mathbf{a}^T\mathbf{v} - \mathbf{b}^T\mathbf{h} - \mathbf{h}^T W\mathbf{v}$$

  其中，$a_i$是可见变量的偏置，$b_j$是隐藏变量的偏置，$w_{ij}$是连接权重，$\theta = (W, \mathbf{a}, \mathbf{b})$是表示所有连接权重和偏置的参数集合。

- 受启发于伊辛模型（Ising model）

  https://www.physik.tu-dresden.de/itp/members/kobe/isingconf.html

# 受限玻尔兹曼机

- 利用能量函数，可以给出状态(v,h)的联合概率分布

$$P(\mathbf{v},\mathbf{h}|\theta)=\frac{1}{Z_\theta}exp\{-E(\mathbf{v},\mathbf{h},\theta)\}$$

$$Z_\theta=\sum_{\mathbf{v},\mathbf{h}}exp\{-E(\mathbf{v},\mathbf{h},\theta)\}$$

# 受限玻尔兹曼机

- 观测数据 $\mathbf{S}=\{\mathbf{v}^1, \mathbf{v}^2, \ldots, \mathbf{v}^{n_s}\}$ 的概率分布，也就是可见层状态变量的概率分布

$$P(\mathbf{v}|\theta) = \sum_{\mathbf{h}} P(\mathbf{v}, \mathbf{h}|\theta) = \frac{1}{Z_\theta} \sum_{\mathbf{h}} exp\{-E(\mathbf{v}, \mathbf{h}, \theta)\}$$

- 类似地，有关于隐藏层状态变量的概率分布

$$P(\mathbf{h}|\theta) = \sum_{\mathbf{v}} P(\mathbf{v}, \mathbf{h}|\theta) = \frac{1}{Z_\theta} \sum_{\mathbf{v}} exp\{-E(\mathbf{v}, \mathbf{h}, \theta\}$$

# 受限玻尔兹曼机

❑ 给定可见层(或隐藏层)上所有神经元的状态时，隐藏层(或可见层)上的某个神经元被激活(即取值为1)的概率

$$P(h_k = 1|\mathbf{v}) \quad 或 \quad P(v_k = 1|\mathbf{h})$$

❑ 中间变量符号

$$\mathbf{h}_{-k} = (h_1, h_2, \cdots, h_{k-1}, h_{k+1}, \cdots, h_{n_h})^T$$

$$\alpha_k(\mathbf{v}) = b_k + \sum_{i=1}^{n_v} w_{ki} v_i$$

$$\beta(\mathbf{v}, \mathbf{h}_{-k}) = \sum_{i=1}^{n_v} a_i v_i + \sum_{\substack{j=1 \\ j\neq k}}^{n_h} b_j h_j + \sum_{i=1}^{n_v} \sum_{\substack{j=1 \\ j\neq k}}^{n_h} h_j w_{ji} v_i$$

# 受限玻尔兹曼机

- 能量函数可以表达为

$$E(\mathbf{v},\mathbf{h}) = -\beta(\mathbf{v},\mathbf{h}_{-k}) - \boldsymbol{h_k}\alpha_k(\mathbf{v})$$

- $P(h_k=1|\mathbf{v})$的推导

$$P(h_k=1|\mathbf{v})$$

$$= P(h_k=1|h_{-k},\mathbf{v})$$

$$= \frac{P(h_k=1,h_{-k},\mathbf{v})}{P(h_{-k},\mathbf{v})}$$

$$= \frac{P(h_k=1,h_{-k},\mathbf{v})}{P(h_k=1,h_{-k},\mathbf{v})+P(h_k=0,h_{-k},\mathbf{v})}$$

$$= \frac{\frac{1}{Z}e^{-E(h_k=1,\mathbf{h}_{-k},\mathbf{v})}}{\frac{1}{Z}e^{-E(h_k=1,\mathbf{h}_{-k},\mathbf{v})}+\frac{1}{Z}e^{-E(h_k=0,\mathbf{h}_{-k},\mathbf{v})}}$$

$$= \frac{e^{-E(h_k=1,\mathbf{h}_{-k},\mathbf{v})}}{e^{-E(h_k=1,\mathbf{h}_{-k},\mathbf{v})}+e^{-E(h_k=0,\mathbf{h}_{-k},\mathbf{v})}}$$

$$= \frac{1}{1+e^{-E(h_k=0,\mathbf{h}_{-k},\mathbf{v})+E(h_k=1,\mathbf{h}_{-k},\mathbf{v})}}$$

$$= \frac{1}{1+e^{[\beta(\mathbf{v},h_{-k})+0\cdot\alpha_k(\mathbf{v})]+[-\beta(\mathbf{v},h_{-k})-1\cdot\alpha_k(\mathbf{v})]}}$$

$$= \frac{1}{1+e^{-\alpha_k(\mathbf{v})}}$$

$$P(h_k=1|\mathbf{v}) = \frac{1}{1+e^{-\alpha_k(\mathbf{v})}} = sigmoid\left(b_k + \sum_{i=1}^{n_v} w_{ki}v_i\right)$$

$$P(v_k=1|\mathbf{h}) = \frac{1}{1+e^{-\alpha_k(\mathbf{v})}} = sigmoid\left(a_k + \sum_{j=1}^{n_h} w_{jk}h_j\right)$$

# 受限玻尔兹曼机

- 给定训练样本 $\mathbf{S}=\{\mathbf{v}^1, \mathbf{v}^2, \ldots, \mathbf{v}^{n_s}\}$ ，调整参数$\theta$训练RBM

$\mathbf{v}^i=(v_1^i, v_2^i, \cdots, v_{n_v}^i)^T, i=1,2,\ldots,n_s$ ，它们是独立同分布的，训练RBM是最大化似然函数

$$L_{\theta,S} = \prod_{i=1}^{n_s} P(\mathbf{v}^i|\theta)$$

- 和玻尔兹曼机一样，计算时通常使用对数似然函数

$$\ln L_{\theta,S} = \ln \prod_{i=1}^{n_s} P(\mathbf{v}^i|\theta) = \sum_{i=1}^{n_s} \ln P(\mathbf{v}^i)$$

# 受限玻尔兹曼机

- 最大化似然，采用梯度正方向更新

$$w_{ij} \leftarrow w_{ij} + \lambda \frac{\partial \ln L_{\theta,S}}{\partial w_{ij}}$$

$$a_i \leftarrow a_i + \lambda \frac{\partial \ln L_{\theta,S}}{\partial a_i}$$

$$b_i \leftarrow b_i + \lambda \frac{\partial \ln L_{\theta,S}}{\partial b_i}$$

# 受限玻尔兹曼机

❑ 梯度求解

- 由于是求和形式，所以仅以一个样本为例

$$L_{\theta,S}^{i} = \ln\left(P(\mathbf{v}^i)\right) = \ln\left(\frac{1}{Z}\sum_{\mathbf{h}} e^{-E(\mathbf{v}^i,\mathbf{h})}\right)$$

$$= \ln\left(\sum_{\mathbf{h}} e^{-E(\mathbf{v}^i,\mathbf{h})}\right) - \ln Z$$

$$= \ln\left(\sum_{\mathbf{h}} e^{-E(\mathbf{v}^i,\mathbf{h})}\right) - \ln \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}$$

$\mathbf{v}_i$表示一个特定的样本，$\mathbf{v}$表示任意的,Z去掉下标为了简洁表示

# 受限玻尔兹曼机

❑ 梯度求解

$$\frac{\partial \ln\left(P(\mathbf{v}^i)\right)}{\partial\theta} = \frac{\partial}{\partial\theta}\left(\ln\sum_{\mathbf{h}} e^{-E(\mathbf{v}^i,\mathbf{h})}\right) - \frac{\partial}{\partial\theta}\left(\ln\sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}\right)$$

$$= -\frac{1}{\sum_{\mathbf{h}} e^{-E(\mathbf{v}^i,\mathbf{h})}}\left(\sum_{\mathbf{h}} e^{-E(\mathbf{v}^i,\mathbf{h})}\frac{\partial E(\mathbf{v}^i,\mathbf{h})}{\partial\theta}\right) + \frac{1}{\sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}}\left(\sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial\theta}\right)$$

$$= -\sum_{\mathbf{h}}\left(P(\mathbf{h}|\mathbf{v}^i)\frac{\partial E(\mathbf{v}^i,\mathbf{h})}{\partial\theta}\right) + \sum_{\mathbf{v},\mathbf{h}}\left(P(\mathbf{v},\mathbf{h})\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial\theta}\right)$$

$$\frac{e^{-E(\mathbf{v}^i,\mathbf{h})}}{\sum_{\mathbf{h}} e^{-E(\mathbf{v}^i,\mathbf{h})}} = \frac{\frac{1}{Z}e^{-E(\mathbf{v}^i,\mathbf{h})}}{\frac{1}{Z}\sum_{\mathbf{h}} e^{-E(\mathbf{v}^i,\mathbf{h})}} = \frac{P(\mathbf{v}^i,\mathbf{h})}{P(\mathbf{v}^i)} = P(\mathbf{h}|\mathbf{v}^i)$$

# 受限玻尔兹曼机

- 梯度求解

$$\frac{\partial \ln\left(P(\mathbf{v}^i)\right)}{\partial\theta} = -\sum_{\mathbf{h}}\left(P(\mathbf{h}|\mathbf{v}^i)\frac{\partial E(\mathbf{v}^i,\mathbf{h})}{\partial\theta}\right) + \sum_{\mathbf{v},\mathbf{h}}\left(P(\mathbf{v},\mathbf{h})\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial\theta}\right)$$

$$\sum_{\mathbf{v},\mathbf{h}}\left(P(\mathbf{v},\mathbf{h})\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial\theta}\right) = \sum_{\mathbf{v}}\sum_{\mathbf{h}}P(\mathbf{v})P(\mathbf{h}|\mathbf{v})\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial\theta} = \sum_{\mathbf{v}}P(\mathbf{v})\sum_{\mathbf{h}}P(\mathbf{h}|\mathbf{v})\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial\theta}$$

$$\sum_{\mathbf{h}}P(\mathbf{h}|\mathbf{v})\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial\theta} = ?$$

# 受限玻尔兹曼机

❑ 梯度求解

$$\sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{v}) \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial \theta} = ?$$

$$E(\mathbf{v},\mathbf{h},\theta) = -\sum_{i=1}^{n_v} a_i v_i - \sum_{j=1}^{n_h} b_j h_j - \sum_{i=1}^{n_v}\sum_{j=1}^{n_h} w_{ij} v_i h_j$$

$$\sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{v}) \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial w_{ij}} = -\sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{v})\, v_i h_j$$

$$= -\sum_{\mathbf{h}} \prod_{k=1}^{n_h} P(h_k|\mathbf{v})\, v_i h_j$$

$$= -\sum_{\mathbf{h}} P(h_j|\mathbf{v}) P(h_{-j}|\mathbf{v})\, v_i h_j$$

$$= -\sum_{h_j}\sum_{h_{-j}} P(h_j|\mathbf{v}) P(h_{-j}|\mathbf{v})\, v_i h_j$$

$$= -\sum_{h_j} P(h_j|\mathbf{v})\, v_i h_j \sum_{h_{-j}} P(h_{-j}|\mathbf{v})$$

$$= -\sum_{h_j} P(h_j|\mathbf{v})\, v_i h_j$$

$$= -\left(P(h_j = 0|\mathbf{v}) \cdot v_i \cdot 0 + P(h_j = 1|\mathbf{v}) \cdot v_i \cdot 1\right)$$

$$= -P(h_j = 1|\mathbf{v}) \cdot v_i$$

# 受限玻尔兹曼机

❑ 梯度求解

$$\frac{\partial \ln\left(P(\mathbf{v}^i)\right)}{\partial \theta} = -\sum_{\mathbf{h}}\left(P(\mathbf{h}|\mathbf{v}^i)\frac{\partial E(\mathbf{v}^i,\mathbf{h})}{\partial \theta}\right) + \sum_{\mathbf{v},\mathbf{h}}\left(P(\mathbf{v},\mathbf{h})\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial \theta}\right)$$

$$\frac{\partial \ln\left(P(\mathbf{v}^i)\right)}{\partial w_{ij}} = P(h_j = 1|\mathbf{v}^i)v_i^i - \sum_{\mathbf{v}} P(\mathbf{v})P(h_j = 1|\mathbf{v})\, v_i$$

$$\frac{\partial \ln\left(P(\mathbf{v}^i)\right)}{\partial a_i} = v_i - \sum_{\mathbf{v}} P(\mathbf{v})\, v_i$$

多个样本再求和即可

$$\frac{\partial \ln\left(P(\mathbf{v}^i)\right)}{\partial b_i} = P(h_i = 1|\mathbf{v}^i) + \sum_{\mathbf{v}} P(\mathbf{v})P(h_i = 1|\mathbf{v})$$

# 受限玻尔兹曼机

- 改良后的受限玻尔兹曼机依然在计算上存在着问题：$\sum_v P(v)$ 是所有输入模式的总和，不可避免会产生庞大的计算量

- 要想解决这个问题，可以使用常规蒙特卡罗MCMC采样估计，算法进行迭代计算求近似解，但即使这样处理，迭代次数也仍然非常多。于是，人们提出了对比散度算法

# 对比散度算法

- 对比散度法
  - 2002年Hinton提出, MCMC的状态以训练样本为起点，这样只需很少的状态转移就可以得到RBM的分布
- 对比散度算法的训练过程
  - 1. 训练准备：初始化连接权重和偏置
  - 2. 调整参数
    - 2.1 在可见层$v(0)$设置输入模式
    - 2.2 调整隐藏层中单元$h(0)$的值
    - 2.3根据输出$x_i$和$x_j$的值，调整连接权重$w_{ij}$、偏置$a_i$、偏置$b_j$
  - 重复步骤2，直到满足终止判断条件

Hinton, Geoffrey E. "Training products of experts by minimizing contrastive divergence." Neural computation 14, no. 8 (2002): 1771-1800.

# 对比散度算法

- 玻尔兹曼机中调整参数时以Mini-Batch为单位进行计算时效果更佳
  - Stochastic GD (SGD)
    - 每次一个样本，更新参数
  - Batch GD
    - 每次迭代的梯度方向计算由所有训练样本共同决定
  - Mini-batch GD
    - 每个mini-batch(训练集的一个子集)更新参数，Batch size
  - Epoch, iteration

Epoch：当一个完整的数据集通过了神经网络称为一个epoch
Iteration： 处理一个Batch称为一次迭代

10.4

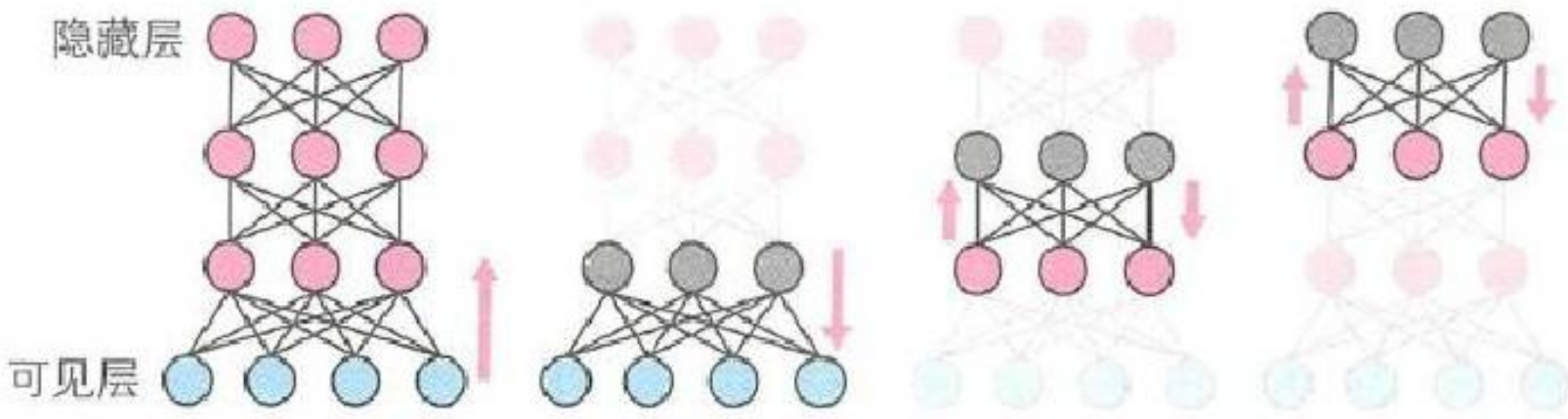
# 深度玻尔兹曼机

# 深度玻尔兹曼机

- 由受限玻尔兹曼机堆叠组成

- 深度玻尔兹曼机采用与多层神经网络不同的训练方法，在训练时采用对比散度算法，逐层来调整连接权重和偏置

- 具体做法：
  - 首先训练输入层和隐藏层之间的参数，把训练后得到的参数作为下一层的输入
  - 再调整该层与下一个隐藏层之间的参数
  - 然后逐次迭代，完成多层网络的训练



# 深度玻尔兹曼机

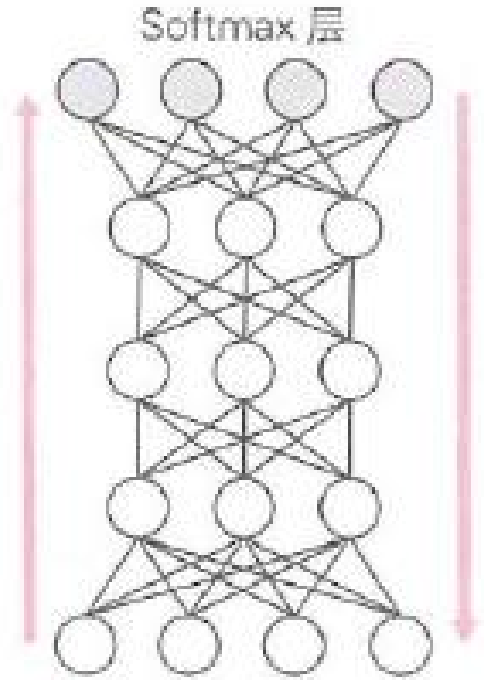
- 深度玻尔兹曼机既可以当作生成模型，也可以当作判别模型

  - 作为生成模型使用时，网络会按照某种概率分布生成训练数据。概率分布可根据训练样本导出，但是覆盖全部数据模式的概率分布很难导出，所以通常选择最大似然估计法训练参数，得到最能覆盖训练样本的概率分布

  - 这种生成模型能够：去除输入数据中含有的噪声，得到新的数据，对输入数据压缩和特征表达

# 深度玻尔兹曼机

- 作为判别模型使用时，需要在模型顶层添加一层Softmax实现分类
  - 进行分类时，需要同时提供训练样本和期望输出，在最顶层级联一个$Softmax$层
- 训练方法
  - 除最顶层外，其他各层都可以使用无监督学习进行训练。
  - 把训练得到的参数作为初始值，使用误差反向传播算法对包含最顶层的神经网络进行训练
  - 最顶层的参数使用随机数进行初始化



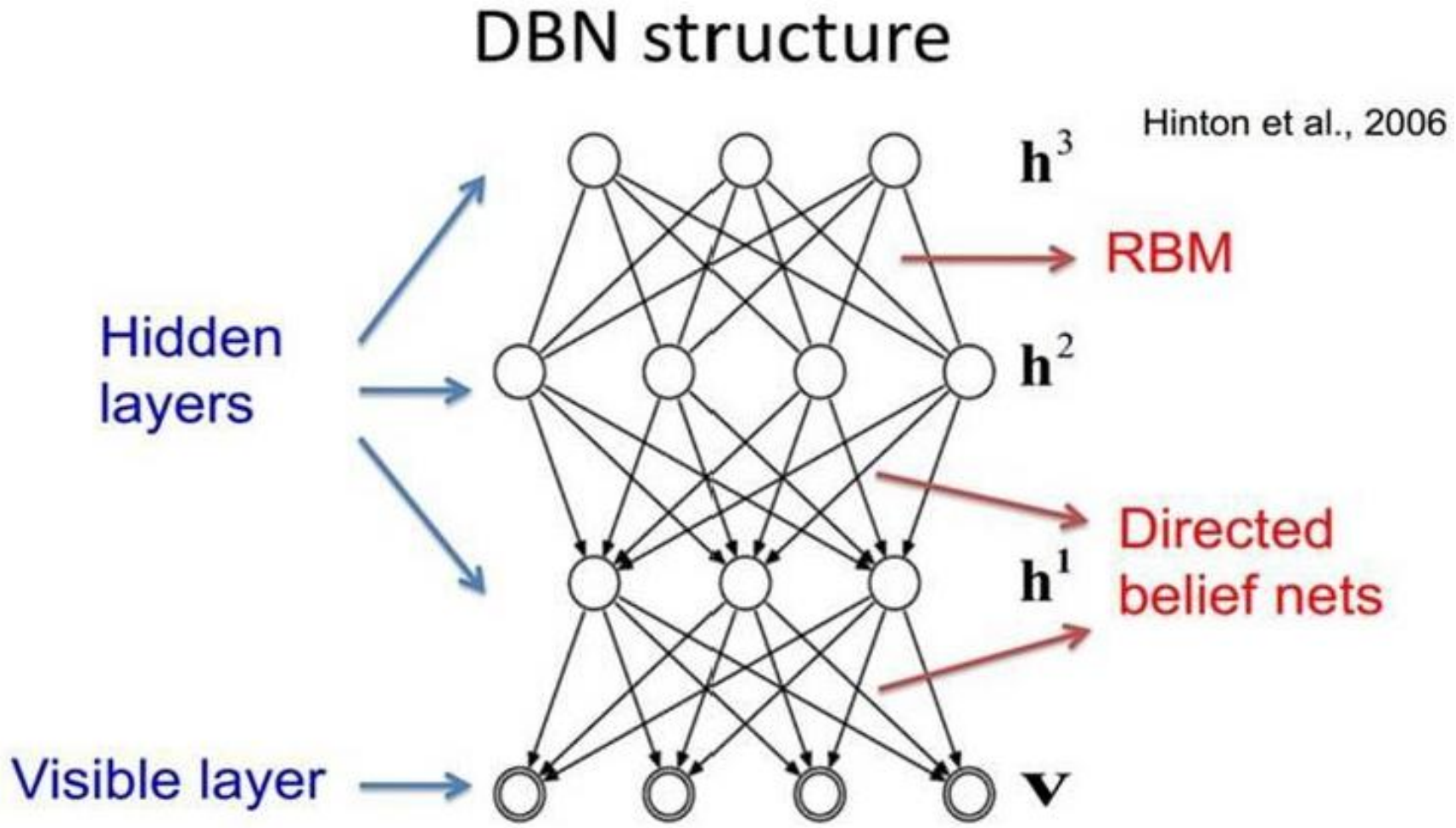
10.5

# 深度信念网络

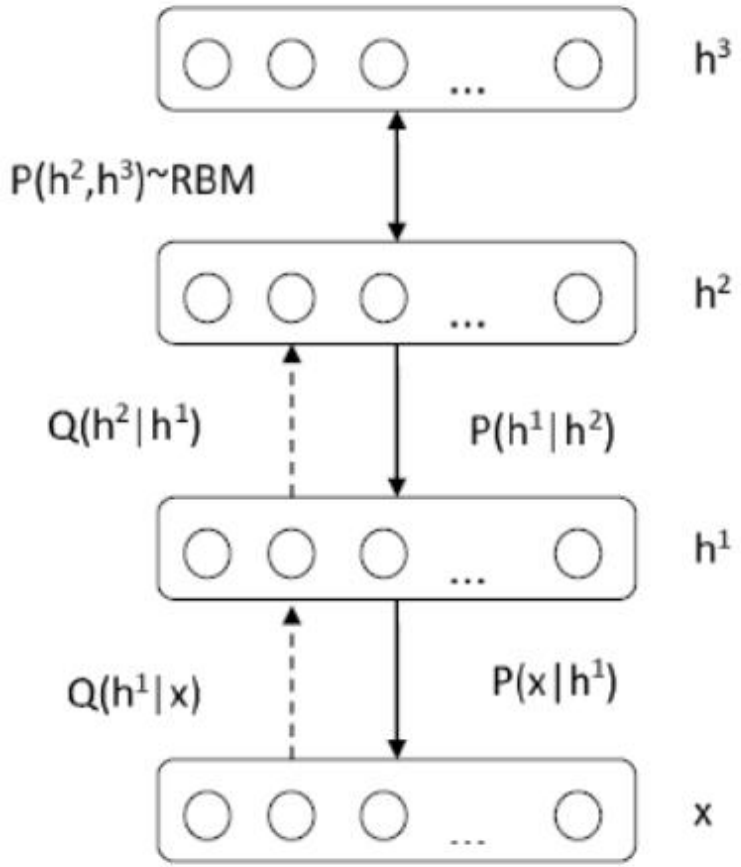
# 深度信念网络

深度信念网络是一种深层的概率有向图模型，其图结构由多层的节点构成。

每层节点的内部没有连接，相邻两层的节点之间为全连接．网络的最底层为可观测变量，其他层节点都为隐变量．最顶部的两层间的连接是无向的，其他层之间的连接是有向的。



# 深度信念网络




算法 1：完全非监督预训练 DBN

输入：DBN 网络结构$(x, h^1, \cdots, h^l)$，训练样本 S，大小为 M

输出：权重 $W^1,\ldots,W^l$，偏置 $a^1,\ldots,a^l$ 和 $b^1,\ldots,b^l$

for 所有 S 中的训练样本 $\mathbf{v}^i$ do,

/*训练第一个 RBM*/

将 $x$ 初始化为 $\mathbf{v}^i$

$h^0 = x$

调用 RBM$(h^0, h^1)$, 得到 $W^1$, $a^1$, $b^1$

/*逐层堆叠的 RBM*/

For k=1 : $l$-1

将符合 $Q(h^k|h^{k-1})$分布的样本作为 $h^k$ 的初始值

调用 RBM$(h^k, h^{k+1})$, 得到 $W^{k+1}$, $a^{k+1}$, $b^{k+1}$

end

end

10.6

# 自编码器

# 简介

❑ 自编码器(Autoencoder)：是一种有效的数据维度压缩算法，主要应用在以下两个方面：

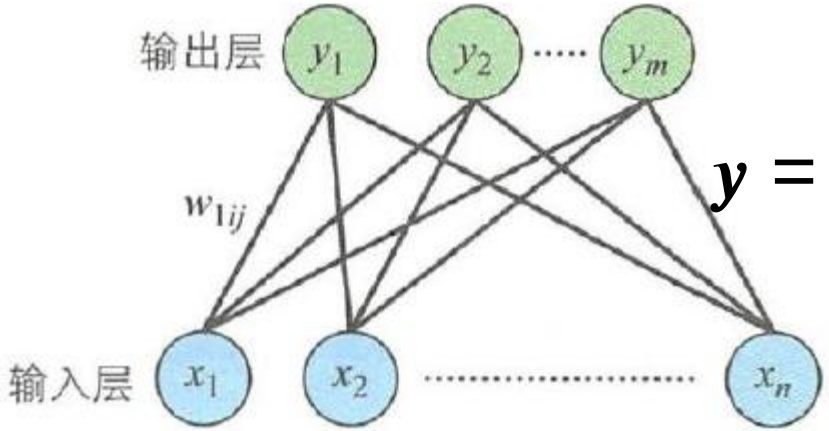
– 构建一种能够重构输入样本并进行特征表达的神经网络

• 特征表达：是指对于分类会发生变动的不稳定模式，例如手写字体识别中由于不同人的书写习惯和风格的不同造成字符模式不稳定，或者输入样本中包含噪声等情况，神经网络也能将其转换成可以准确识别的特征

– 训练多层神经网络时，通过自编码器训练样本得到参数初始值

# 自编码器

- 自编码器的基本形式如下图所示




$$y = f(Wx + b)$$

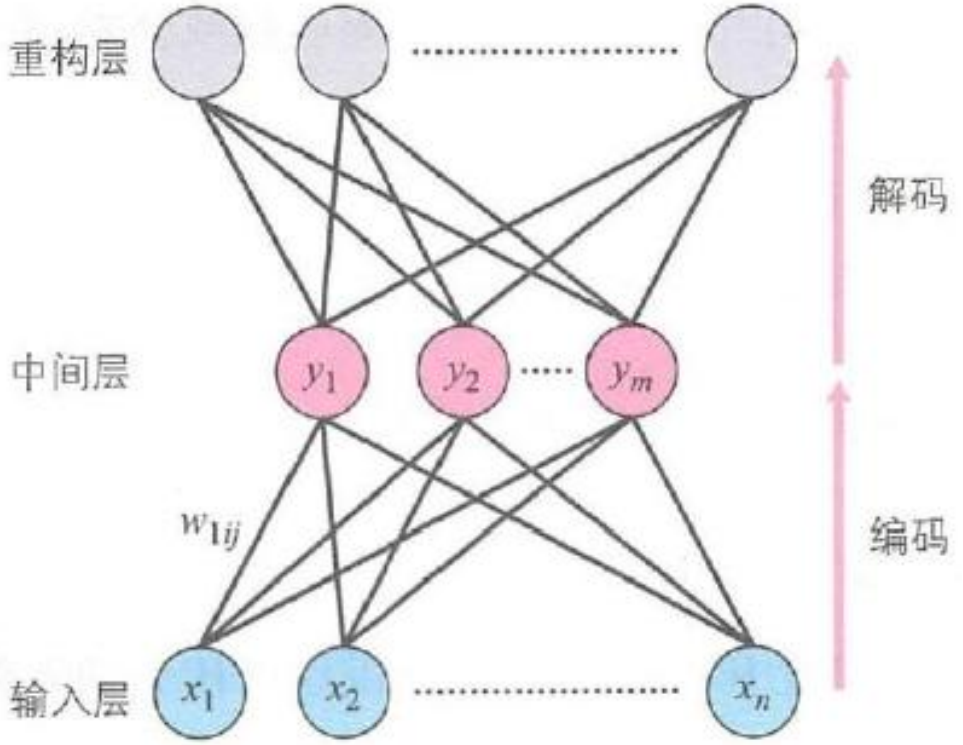
自编码器(两层结构)

和RBM类似，由输入层和输出层组成。输入数据$x$与对应的连接权重$W$相乘，再加上偏置$b$，并经过激活函数$f(\cdot)$变换后，就可以得到输出$y$

# 自编码器

- 自编码器是一种基于无监督学习的神经网络，目的在于通过不断调整参数，重构经过维度压缩的输入样本

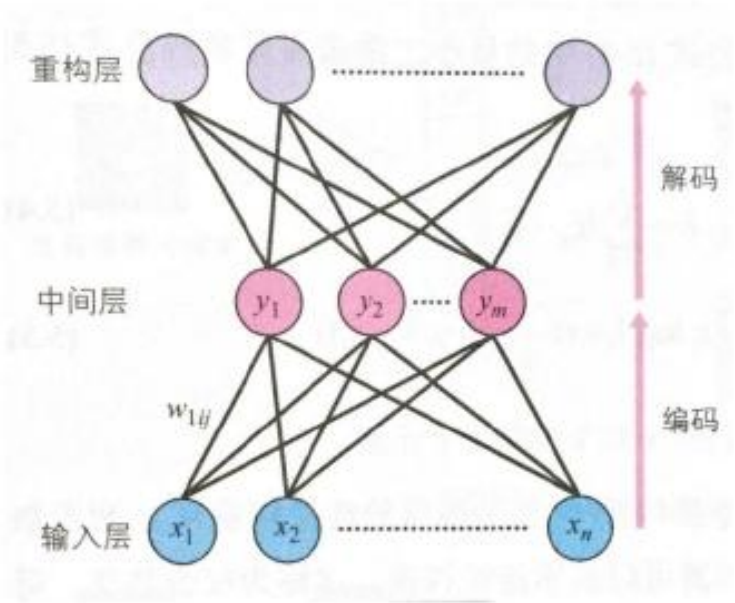
- 一种能够重构输入样本的三层神经网络如右图




自编码器(三层结构)

# 自编码器

❑ 这里，$f(\cdot)$表示编码器的激活函数，$\tilde{f}(\cdot)$ 表示解码器的激活函数。中间层和重构层之间的连接权重及偏置分别记为 $\tilde{W}$ 和 $\tilde{b}$ ,重构值记作$\tilde{x}$




$$\tilde{x} = \tilde{f}\left(\tilde{W}y + \tilde{b}\right)$$

$$y = f(Wx + b)$$

$$\tilde{x} = \tilde{f}\left(\tilde{W}f(Wx + b) + \tilde{b}\right)$$

# 自编码器的训练

- 自编码器的训练就是确定编码器和解码器的参数$W$, $\tilde{W}$, $b$, $\tilde{b}$ 的过程。其中$W$和 $\tilde{W}$ 可以相同，这称为权值共享。训练的目的是尽可能重构其原始输入

- 参数的训练使用误差反向传播算法，误差函数可以使用最小二乘误差函数或交叉熵代价函数

$$E = \sum_{n=1}^{N} \| x_n - \tilde{x}_n \|^2$$

$$E = -\sum_{n=1}^{N} (x_n \log \tilde{x}_n + (1 - x_n) \log(1 - \tilde{x}_n))$$

# 自编码器

- 当样本中包含噪声时，如果神经网络能够消除噪声，则被称为降噪自编码器（Denoising Autoencoder）

- 还有一种称为稀疏自编码器( Sparse Autoencoder )的网络，它在自编码器中引入了正则化项，以去除冗余信息

10.7

# 自编码器变种、预训练

# 降噪自编码器

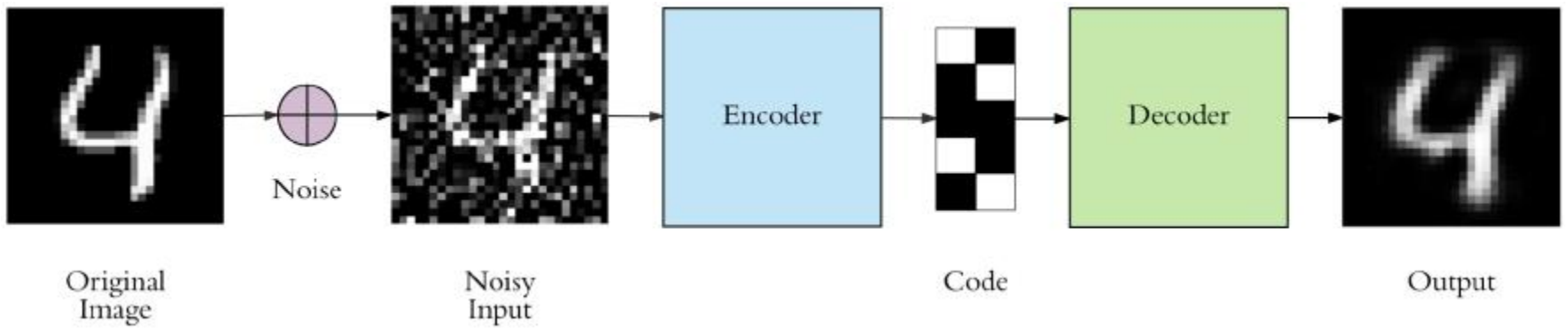
❑ 降噪自编码器（Denoising Autoencoder）的网络结构和自编码器一样，只是对训练方法进行了改进

– 自编码器是把训练样本直接输入给输入层，而降噪自编码器则是把通过向训练样本中加入随机噪声得到的样本输入给输入层

$$\tilde{x} = x + \varepsilon$$

# 降噪自编码器

- 假设随机噪声$\varepsilon$服从均值为0，方差为$\sigma_2$的正态分布，我们需要训练神经网络，使得重构结果和不含噪声的样本之间的误差收敛于极小值
- 误差函数会对不含噪声的输入样本进行计算，故降噪自编码器可以完成以下两项训练
  - 保持输入样本不变的条件下，能够更好地反映样本属性的特征
  - 消除输入样本中包含的噪声



# 降噪自编码器

- 在MNIST数据集上应用降噪自编码器

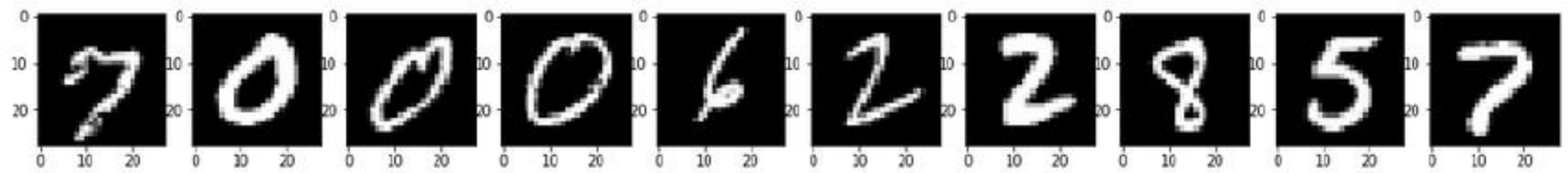


图1：原始图像

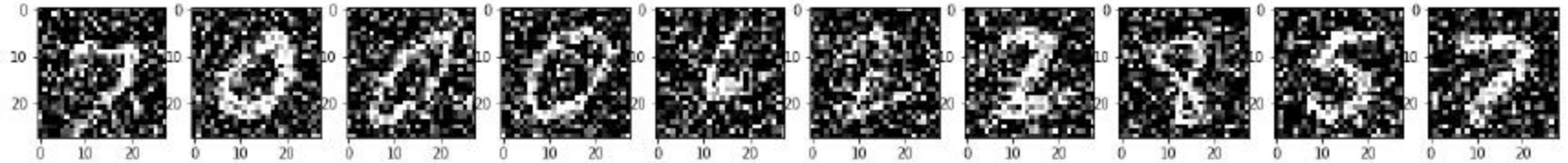


图2：加入噪声后的图像

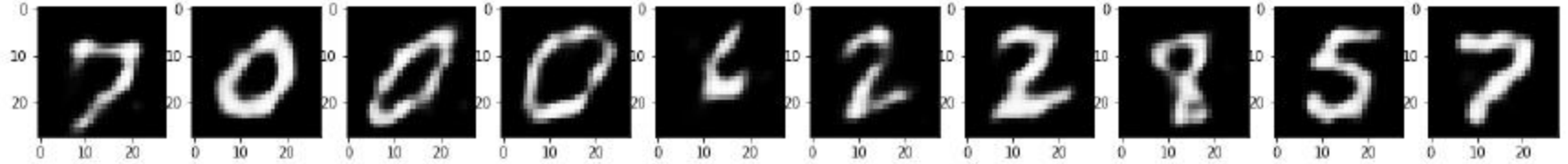


图3：去噪后的图像

# 稀疏自编码器

- 在多层自编码器中，中间层的单元数太少会导致神经网络很难重构输入样本，而单元数太多又会产生单元冗余，降低压缩效率

- 为了解决这个问题，人们将稀疏正则化引入到自编码器中，提出了稀疏自编码器（Sparse Autoencoder）

  - 通过增加正则化项，大部分单元的输出都变成了0，就能利用少数单元完成压缩或重构

# 稀疏自编码器

- 加入正则化后的误差函数$E$如下所示

$$E = \sum_{n=1}^{N} \|\boldsymbol{x}_n - \tilde{\boldsymbol{x}}_n\|^2 + \beta \sum_{j=1}^{M} KL(\rho||\hat{\rho}_j)$$

  - $\rho$表示平均激活度的目标值，$\beta$用于控制稀疏性的权重，$\hat{\rho}_j$表示中间层第$j$个单元的平均激活度

$$\hat{\rho}_j = \frac{1}{N}\sum_{n=1}^{N} f(W_j \boldsymbol{x}_n + b_j)$$

  - $KL(\rho||\hat{\rho}_j)$表示KL距离（Kullback-Leibler divergence）

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

# 稀疏自编码器的训练

- KL距离反映了平均激活度和目标值的差异。$\rho$值越接近于0，中间层的平均激活度 $\hat{\rho}_j$ 就越小
- 稀疏自编码器的训练也需要用到误差反向传播算法，对误差函数求导时须考虑 $KL(\rho||\hat{\rho}_j)$ 的导数

$$KL(\rho||\hat{\rho}_j) = \rho \log\frac{\rho}{\hat{\rho}_j} + (1-\rho)\log\frac{1-\rho}{1-\hat{\rho}_j} \qquad \hat{\rho}_j = \frac{1}{N}\sum_{n=1}^{N} f\left(W_j \boldsymbol{x}_n + b_j\right)$$

$$u_i = f\left(W_i \boldsymbol{x}_n + b_j\right)$$

$$\frac{\partial KL(\rho||\hat{\rho}_i)}{\partial u_i} = \left(-\frac{\rho}{\hat{\rho}_i} + \frac{1-\rho}{1-\hat{\rho}_i}\right)\frac{\partial \hat{\rho}_i}{\partial u_i}$$

# 稀疏自编码器的训练

- 平均激活度是根据所有样本计算出来的，所以在计算任何单元的反向传播之前，需要对所有样本计算一遍正向传播，从而获取平均激活度，所以使用小批量梯度下降法进行训练时的效率很低

- 为了解决此问题，可以只计算Mini-Batch中包含的样本的平均激活度，然后在Mini-Batch之间计算加权平均并求近似值

# 稀疏自编码器的训练

- 假设时刻($t$−1)的Mini-Batch的平均激活度为 $\hat{\rho}_j^{(t-1)}$ 当前t时刻Mini-Batch的平均激活度为

$$\hat{\rho}_j^{(t)} = \lambda \hat{\rho}_j^{(t-1)} + (1-\lambda)\hat{\rho}_j^{(t)}$$

这里的$\lambda$是权重，$\lambda$越大，则时刻($t$ − 1)的Mini-Batch所占的比重也越高

# 栈式自编码器

- 自编码器、降噪自编码器、稀疏自编码器都是包括编码器和解码器的三层结构。

- 但是在进行维度压缩时，可以只包括输入层和中间层。输入层和中间层多层堆叠后，就可以得到栈式自编码器(Stacked Autoencoder)

# 栈式自编码器的训练

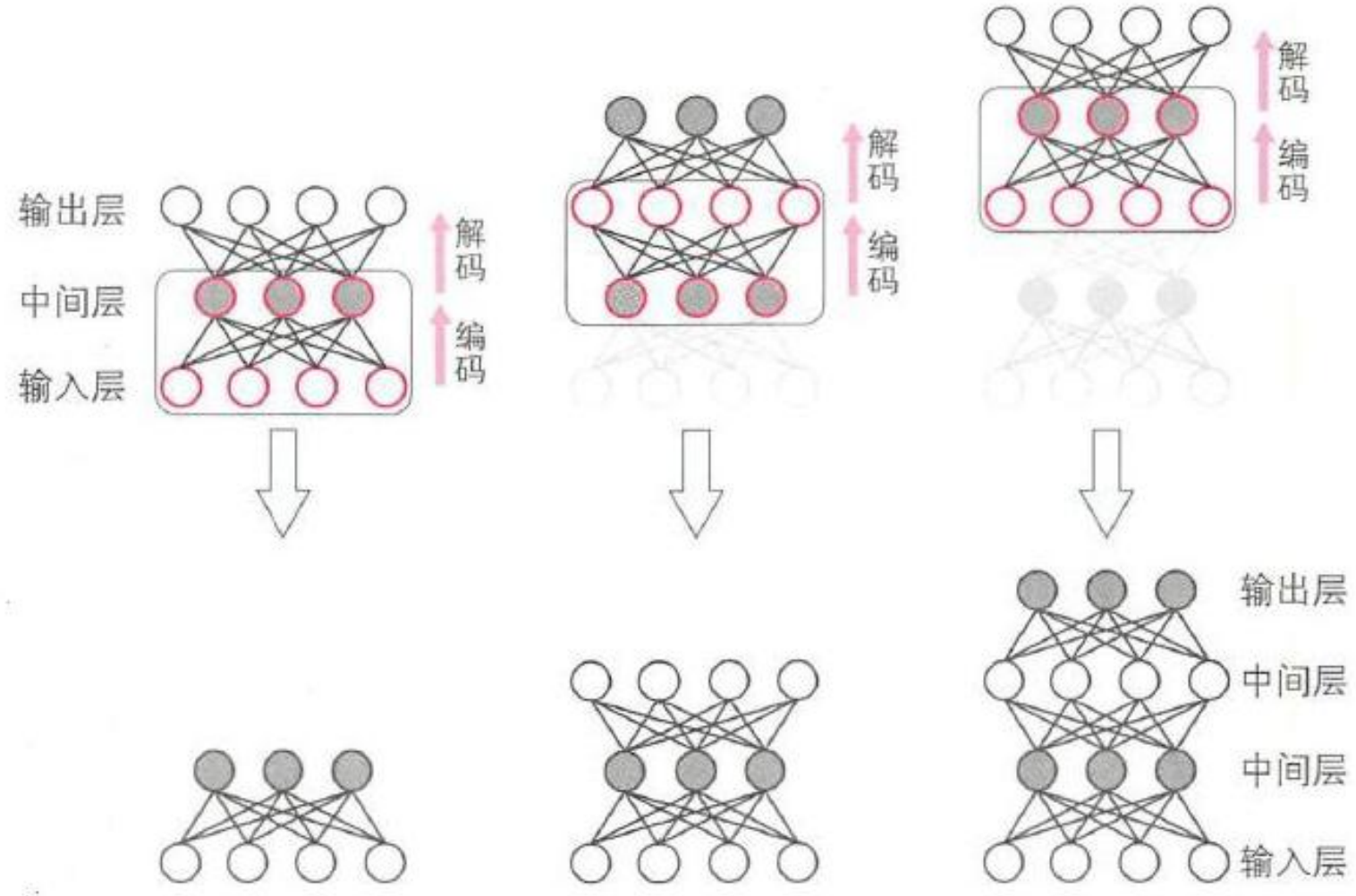
❑ 栈式自编码器和深度信念网络一样，都是逐层训练。但两种网络的训练方法不同，深度信念网络是利用对比散度算法逐层训练两层之间的参数。而栈式自编码器的训练过程如下

➢ 首先训练第一个自编码器，然后保留第一个自编码器的编码器部分

➢ 把第一个自编码器的中间层作为第二个自编码器的输入层进行训练

➢ 反复地把前一个自编码器的中间层作为后一个编码器的输入层，进行迭代训练

# 栈式自编码器的训练



# 在预训练中的应用

- 栈式自编码器每层都能得到有效的参数，所以我们可以把训练后的参数作为神经网络或卷积神经网络的参数初始值,这种方法叫作预训练

- 预训练属于无监督学习，接下来需要使用有监督学习来调整整个网络的参数，这也叫作微调( fine tuning )

# 北京交通大学《深度学习》课程组成员


景丽萍：http://faculty.bjtu.edu.cn/8249/

桑基韬：http://faculty.bjtu.edu.cn/9129/

张淳杰：http://faculty.bjtu.edu.cn/9371/

万怀宇：http://faculty.bjtu.edu.cn/8793/

滕　竹：http://faculty.bjtu.edu.cn/8902/

原继东：http://faculty.bjtu.edu.cn/9076/

丛润民：http://faculty.bjtu.edu.cn/9374/

夏佳楠：http://faculty.bjtu.edu.cn/9430/

许万茹

杨　扩