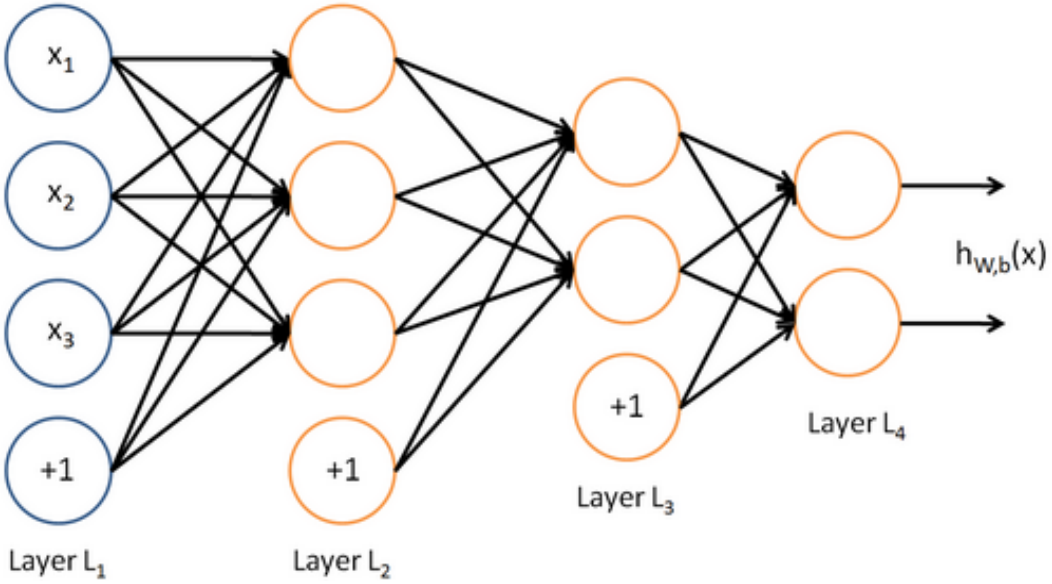
深度学习技术与应用（2）

Deep Learning: Techniques and Applications (2)

Ge Li

Peking University

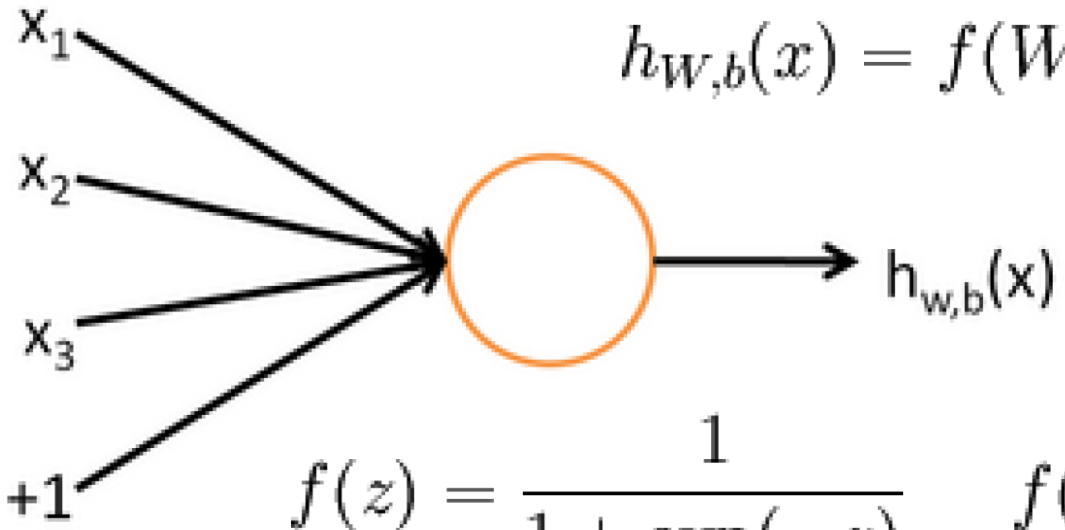


$\boldsymbol{w_{ij}^{(l)}}$ *表示第l层第j个神经元与第l+1层第i个神经元之间的连接权重；*

$\boldsymbol{z_i^{(l)}}$ *表示第l层第i个神经元的输入（加权和）*

$\boldsymbol{a_i^{(l)}}$ *表示第l层第i个神经元的输出（激活值）*

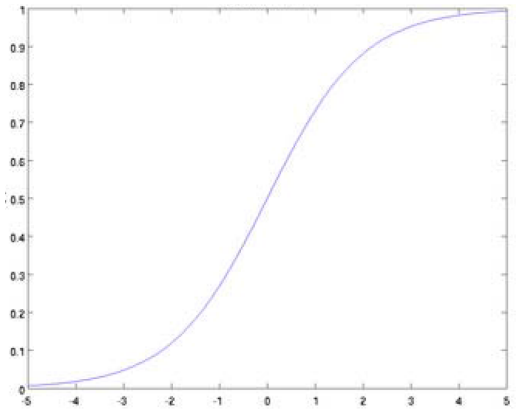
$\boldsymbol{b_i^{(l)}}$ *表示第 l+1层第i个神经元的偏置项*

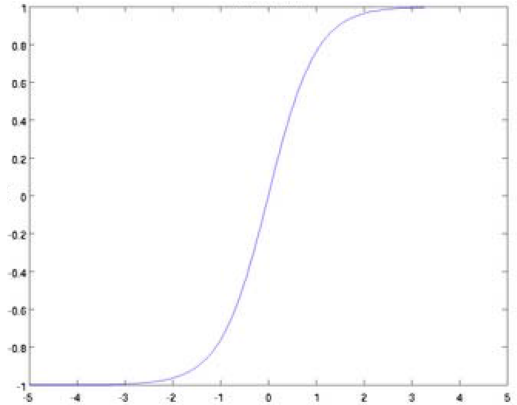


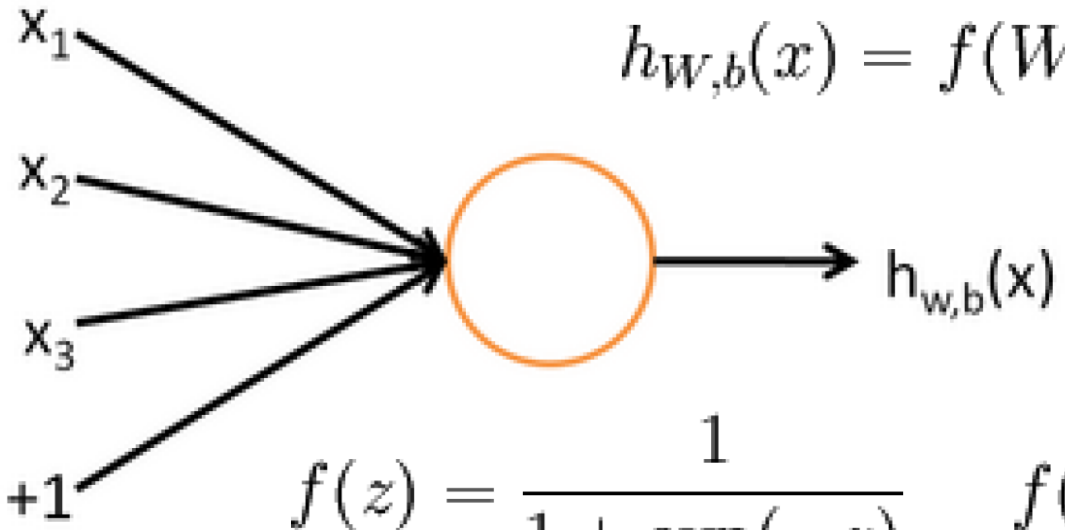
$$h_{W,b}(x) = f(W^T x) = f(\sum_{i=1}^{3} W_i x_i + b)$$

$$f(z) = \frac{1}{1 + \exp(-z)}$$

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$



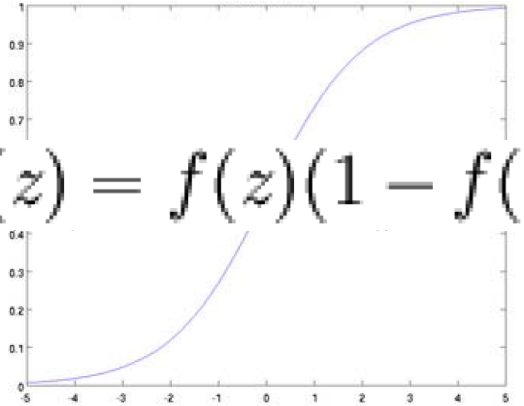




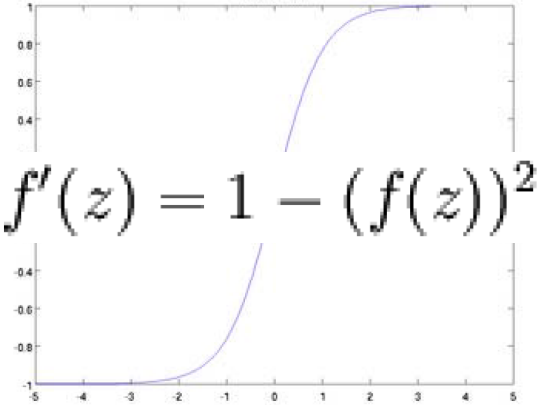
$$h_{W,b}(x) = f(W^T x) = f(\sum_{i=1}^{3} W_i x_i + b)$$

$$f(z) = \frac{1}{1 + \exp(-z)}$$

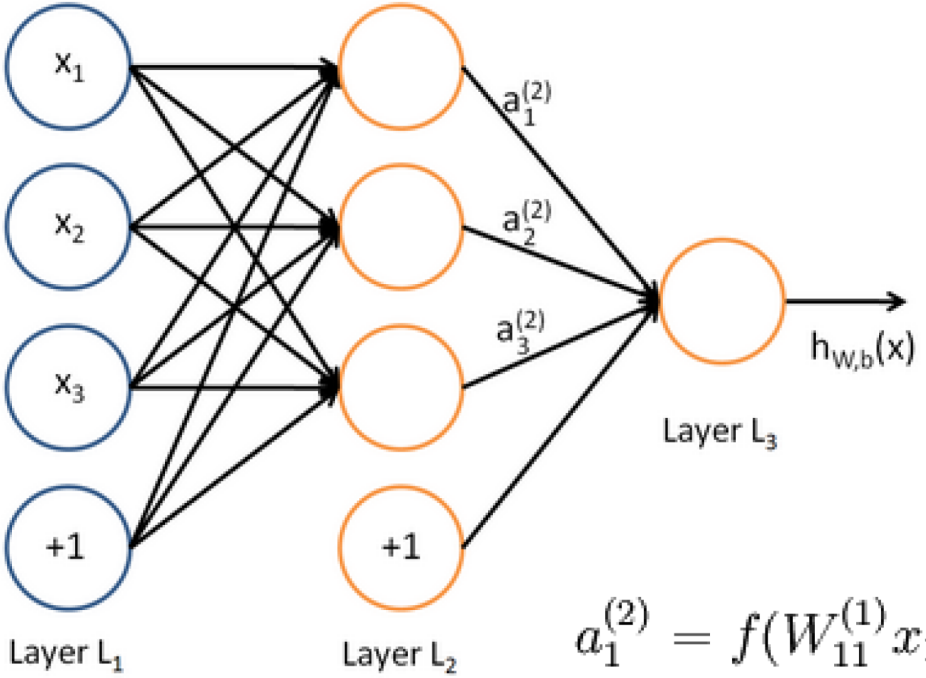
$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$



$$f'(z) = f(z)(1 - f(z))$$



$$f'(z) = 1 - (f(z))^2$$

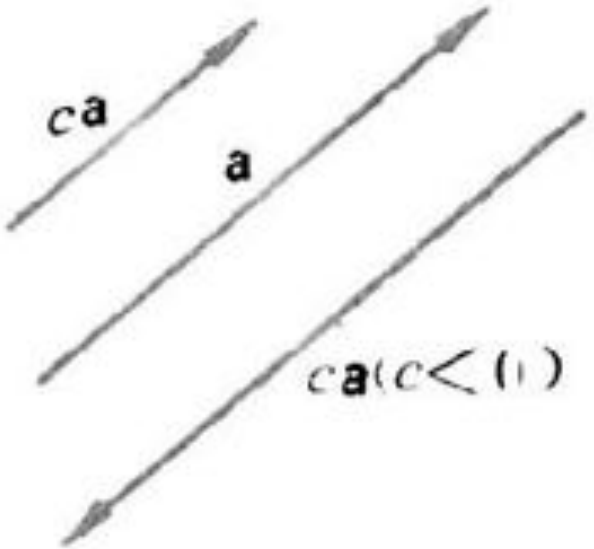


$$z^{(2)} = W^{(1)}x + b^{(1)}$$
$$a^{(2)} = f(z^{(2)})$$
$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$
$$h_{W,b}(x) = a^{(3)} = f(z^{(3)})$$

$$a_1^{(2)} = f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)})$$
$$a_2^{(2)} = f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)})$$
$$a_3^{(2)} = f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)})$$
$$h_{W,b}(x) = a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)})$$

关于向量

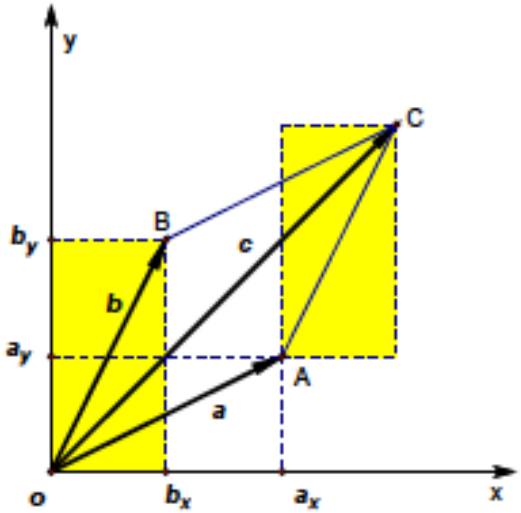
- 向量数乘的意义
  - 在原向量的直线上向量长度的伸长或缩短;

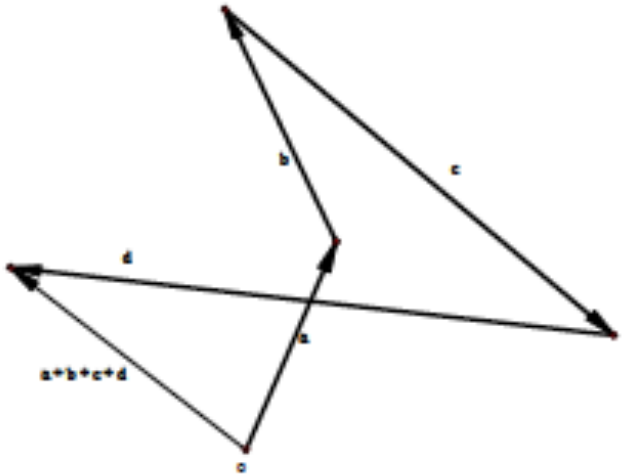


关于向量

向量加法的几何意义

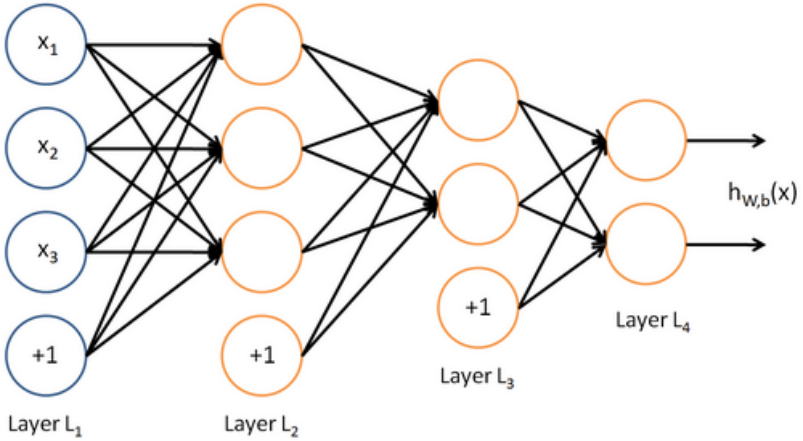
- 两向量相加的几何解释就是进行依照平行四边形法则对向量合并;
- 多向量加法的数学本质就是这些向量在坐标轴上的投影的合成的结果。





关于向量

- 向量的线性组合的几何意义
  - 向量的“线性组合”的几何意义就是对向量组内的向量长度进行缩放后依照平行四边形法则进行合并加；
  - “线性”表示的几何意义就是可以把一个向量依照平行四边形法则分解(或投影)为向量组上的和；



关于向量

向量的点积的几何意义

- 定义有两个，它们是一样的含义（用余玄定理可以证明）

$$\mathbf{a}\cdot\mathbf{b}=ab\cos\theta \quad \text{..... 1}$$

$$\mathbf{a}\cdot\mathbf{b}=a_xb_x+a_yb_y+a_zb_z \quad \text{..... 2}$$

- 向量内积的几何含义：一个向量在另一个向量上的投影的积，它反应了向量之间在方向上的“接近程度”
- 可知：两个向量内积为正，说明方向相近（夹角小于90度），内积为负，说明方向相反（夹角大于90度），内积为零，说明方向垂直。

关于向量

- 向量的叉积的几何意义
  - 定义有两个，它们是一样的含义（用余玄定理可以证明）

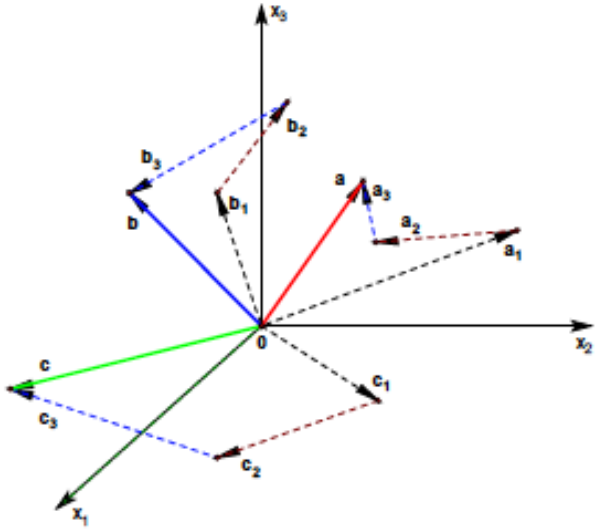
$$\mathbf{a}\times\mathbf{b}=(a_yb_z-a_zb_y,a_zb_x-a_xb_z,a_xb_y-a_yb_x) \quad \text{.....1}$$

$$\mathbf{a}\times\mathbf{b}=ab\sin\theta N \text{（其中N是垂直于 a 和 b 展成的平面的单位向量）.....2}$$

  - 向量叉积的物理意义：两个向量的叉积是把前一个向量旋转至后一个向量所需要的力矩。
  - 力矩在物理学里是指作用力使物体绕着转动轴或支点转动的趋向。力矩能够使物体改变其旋转运动。推挤或拖拉涉及到作用力 ，而扭转则涉及到力矩。

关于矩阵

- 矩阵加法的几何意义
  - 矩阵可以看作多个向量的集合
  - 矩阵加法可以看作相对应的行向量或列向量的加法。



关于矩阵

矩阵与向量的乘法

- 形式上的定义：

$$\mathbf{A}\cdot\mathbf{c}=\begin{bmatrix}a_1 & a_2 & a_3\\ b_1 & b_2 & b_3\end{bmatrix}\begin{pmatrix}c_1\\ c_2\\ c_3\end{pmatrix}=\begin{pmatrix}(a_1 \;\; a_2 \;\; a_3)\cdot\begin{pmatrix}c_1\\ c_2\\ c_3\end{pmatrix}\\ (b_1 \;\; b_2 \;\; b_3)\cdot\begin{pmatrix}c_1\\ c_2\\ c_3\end{pmatrix}\end{pmatrix}=\begin{pmatrix}a_1c_1+a_2c_2+a_3c_3\\ b_1c_1+b_2c_2+b_3c_3\end{pmatrix}$$

$$\mathbf{A}\cdot\mathbf{c}=\begin{bmatrix}a_1 & a_2 & a_3\\ b_1 & b_2 & b_3\end{bmatrix}\begin{pmatrix}c_1\\ c_2\\ c_3\end{pmatrix}=\begin{pmatrix}a_1\\ b_1\end{pmatrix}c_1+\begin{pmatrix}a_2\\ b_2\end{pmatrix}c_2+\begin{pmatrix}a_3\\ b_3\end{pmatrix}c_3=\begin{pmatrix}a_1c_1+a_2c_2+a_3c_3\\ b_1c_1+b_2c_2+b_3c_3\end{pmatrix}$$

关于矩阵

- 矩阵与向量的乘法

$$\mathbf{A}\cdot\mathbf{c}=\begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{bmatrix}\begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix}=\begin{pmatrix} a_1 \\ b_1 \end{pmatrix}c_1+\begin{pmatrix} a_2 \\ b_2 \end{pmatrix}c_2+\begin{pmatrix} a_3 \\ b_3 \end{pmatrix}c_3=\begin{pmatrix} a_1c_1+a_2c_2+a_3c_3 \\ b_1c_1+b_2c_2+b_3c_3 \end{pmatrix}$$

  - 看做向量对矩阵的操作：
  - 把矩阵A看做两个列向量，Ac的乘积可以理解为矩阵A的列向量的线性组合，组合系数是向量c的三个分量。
  - 几何解释就是：把矩阵 **A** 的列向量进行伸缩变换(比例变换,也可能改变方向)后首尾相连(即向量的和)得到了一个新向量,这个向量就是 $\mathbf{A}\cdot\mathbf{d}$

关于矩阵

矩阵与向量的乘法

$$\mathbf{A}\cdot\mathbf{c}=\begin{bmatrix}a_1 & a_2 & a_3\\ b_1 & b_2 & b_3\end{bmatrix}\begin{pmatrix}c_1\\ c_2\\ c_3\end{pmatrix}=\begin{pmatrix}(a_1 \;\; a_2 \;\; a_3)\cdot\begin{pmatrix}c_1\\ c_2\\ c_3\end{pmatrix}\\ (b_1 \;\; b_2 \;\; b_3)\cdot\begin{pmatrix}c_1\\ c_2\\ c_3\end{pmatrix}\end{pmatrix}=\begin{pmatrix}a_1c_1+a_2c_2+a_3c_3\\ b_1c_1+b_2c_2+b_3c_3\end{pmatrix}$$

- 看做矩阵对向量的操作：
- 把矩阵A看作两个行向量，矩阵与向量乘积比如 Ax 表现为矩阵 A 对一个向量 x 作用的结果。其作用的主要过程是对一个向量 进行旋转和缩放的综合过程(即线性变换的过程),一个向量就变换为另外一个向量。
- 一个 m 行 n 列的实矩阵$A_{m\times n}$就是一个$R^n \rightarrow R^m$ 上的线性变换，把一个n维空间的n维向量变换为一个m维空间的m维向量。

关于矩阵

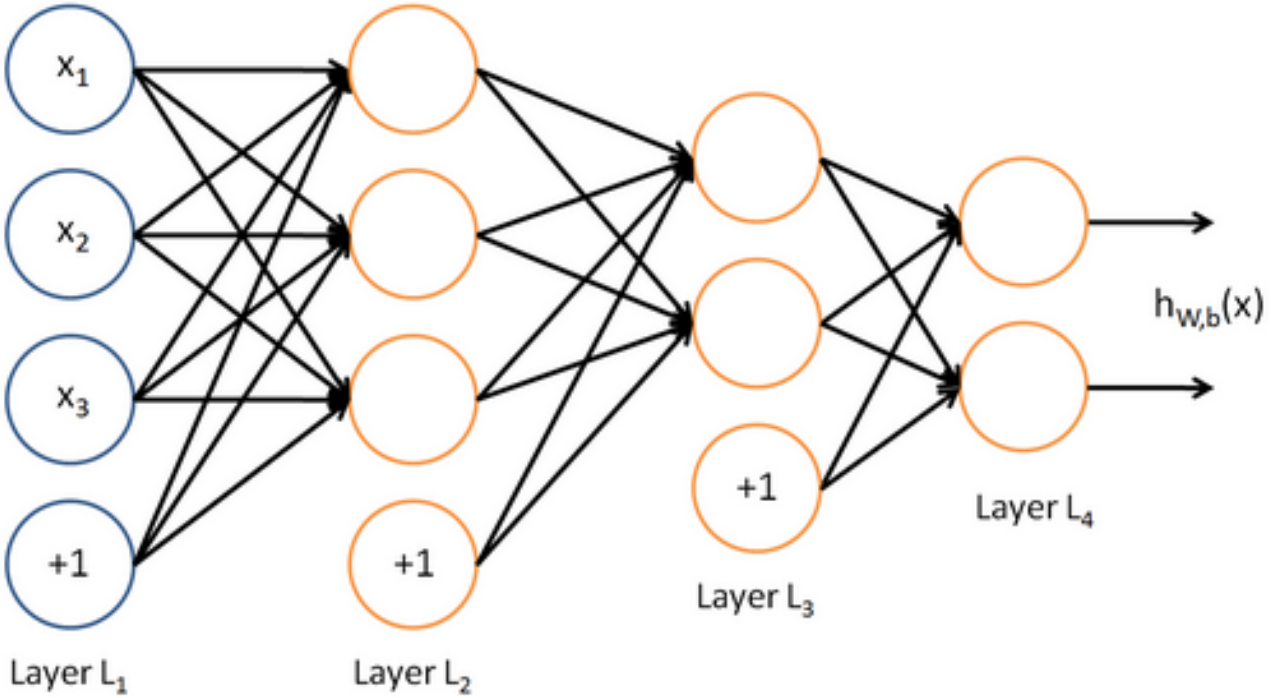
- 矩阵与矩阵之间的相乘
  - 矩阵与矩阵的乘法可以从矩阵与向量的乘法得到,因为一个矩阵与多个向量相乘,这多个向量就可以组成一个矩阵。
  - **AB = C** 的几何意义：

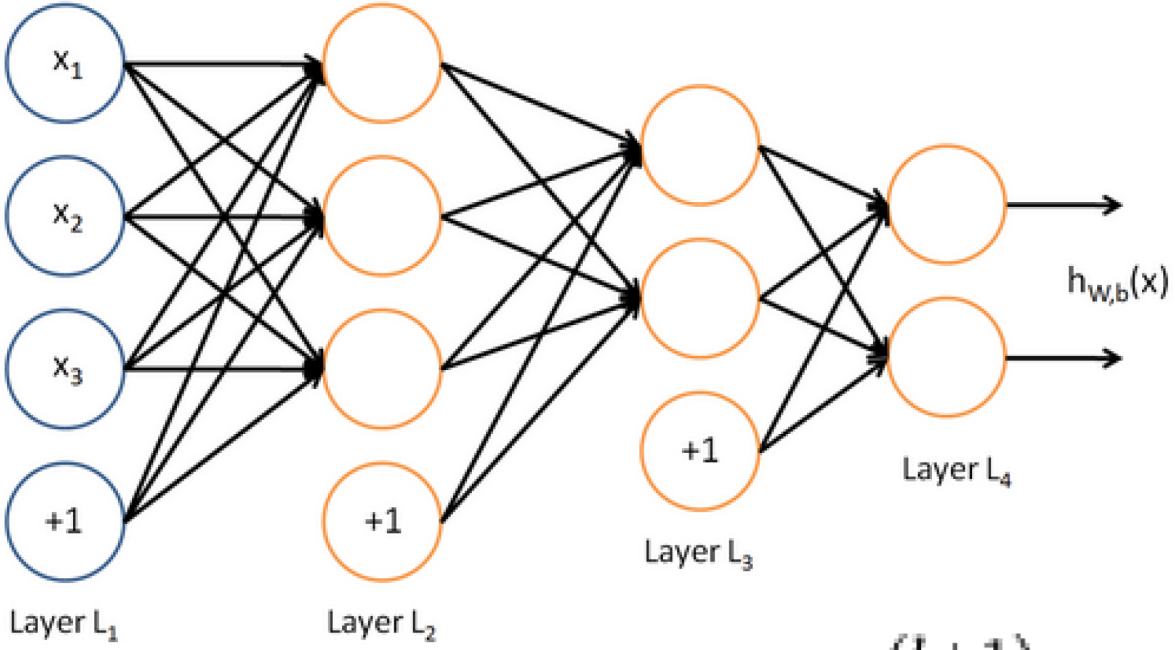
A的作用：把矩阵**B**的数个行向量或列向量构成的几何图形进行旋转、缩放、镜像等变换；

矩阵 **C**：构成的新的几何图形；

关于矩阵

- 再看神经网络





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

$$a^{(l+1)} = f(z^{(l+1)})$$

前向传播网络的表示能力

- 什么函数能够用前向传播网络表示？

1. 布尔函数
   - 任何布尔函数可以用具有两层神经元的神经网络表示；
     - 对于每个输入向量，创建不同的隐藏神经元；
     - 设置其的权值：当且仅当这个特定的向量输入到网络时，这个神经元被激活；
     - 因而，对于任意输入，仅有一个神经元被激活；
     - 输出层：仅由某个隐藏层被激活时输出；
   - 虽然：隐藏单元数量随着输入信号量的增长成指数级增长

前向传播网络的表示能力

- 什么函数能够用前向传播网络表示？

2. 连续函数

  - 任何有界的连续函数可以由一个两层的神经网络逼近（以任意小的误差）

【Cybenko 1989, Hornik et al. 1989】

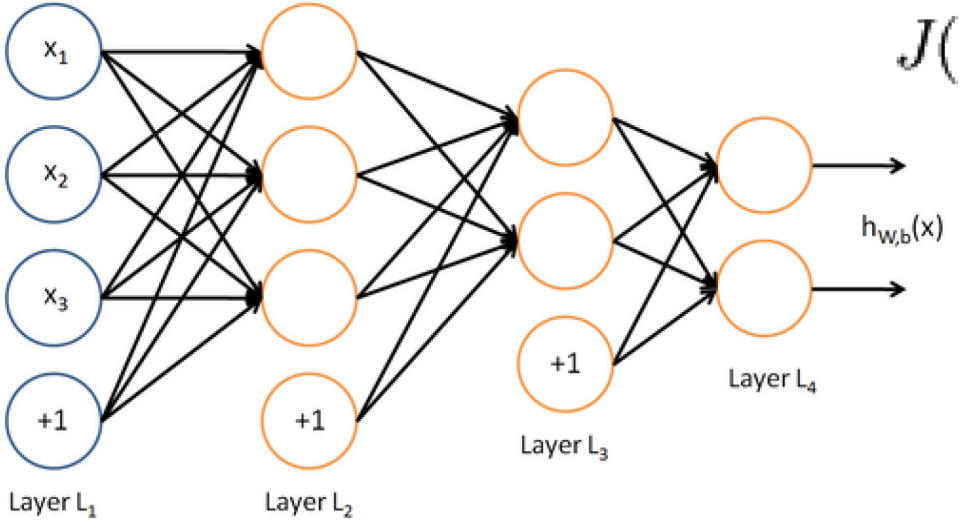
隐藏层使用Sigmod，输出层使用线性单元；

前向传播网络的表示能力

- 什么函数能够用前向传播网络表示？

3. 任意函数

  - 任何函数可以被一个有三层神经元的网络以任意精度逼近【Cybenko 1980】；
  - 隐藏层使用Sigmod，输出层使用线性单元，数量不定；
  - 两层的Sigmod神经元足以产生良好的局部逼近效果；



$$J(W,b;x,y) = \frac{1}{2}\left\|h_{W,b}(x) - y\right\|^2$$

$$J(W,b) = \left[\frac{1}{m}\sum_{i=1}^{m} J(W,b;x^{(i)},y^{(i)})\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2$$

$$= \left[\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\left\|h_{W,b}(x^{(i)}) - y^{(i)}\right\|^2\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2$$

范数

向量的范数

- 向量的范数，可以粗略理解为，对向量“长度”的度量。

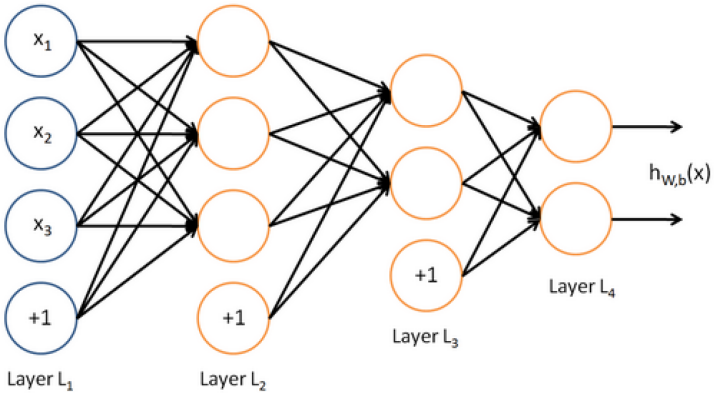
对向量x

x的l_1范数：

$$\|x\|_1 = \sum_{i=1}^{n} |x_i|$$

x的l_2范数：

$$\|x\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}. \qquad \|x\|_2^2 = x^T x.$$

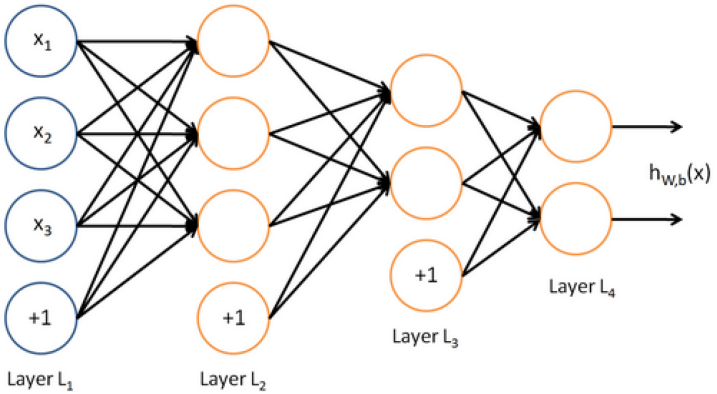


$$J(W,b;x,y) = \frac{1}{2}\left\|h_{W,b}(x) - y\right\|^2$$

$$J(W,b) = \left[\frac{1}{m}\sum_{i=1}^{m} J(W,b;x^{(i)},y^{(i)})\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2$$

$$= \left[\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\left\|h_{W,b}(x^{(i)}) - y^{(i)}\right\|^2\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2$$

- 关于规则化项（权重衰减项）
  - 目的是减小权重的幅度，防止过度拟合；
  - 通常不使用偏置项作为规则化项；
  - 把偏置项包含在权重衰减项中只会对最终的神经网络产生很小的影响；



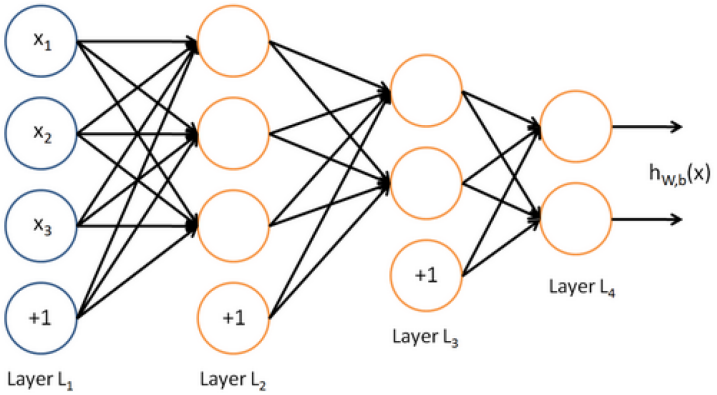
$$J(W,b;x,y) = \frac{1}{2}\left\|h_{W,b}(x) - y\right\|^2$$

$$J(W,b) = \left[\frac{1}{m}\sum_{i=1}^{m} J(W,b;x^{(i)},y^{(i)})\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2$$

$$= \left[\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\left\|h_{W,b}(x^{(i)}) - y^{(i)}\right\|^2\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2$$

关于权重初始化

- 随机初始化的目的是使对称失效
  - 如果全部置为0或全部相同的值，那么所有隐藏层单元最终会得到相同的值；
- 通常将参数初始化为一个很小的、接近零的随机值
  - 例如：使用正态分布生成的随机值



$$J(W,b;x,y) = \frac{1}{2}\left\|h_{W,b}(x) - y\right\|^2$$

$$J(W,b) = \left[\frac{1}{m}\sum_{i=1}^{m} J(W,b;x^{(i)},y^{(i)})\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2$$

$$= \left[\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\left\|h_{W,b}(x^{(i)}) - y^{(i)}\right\|^2\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2$$

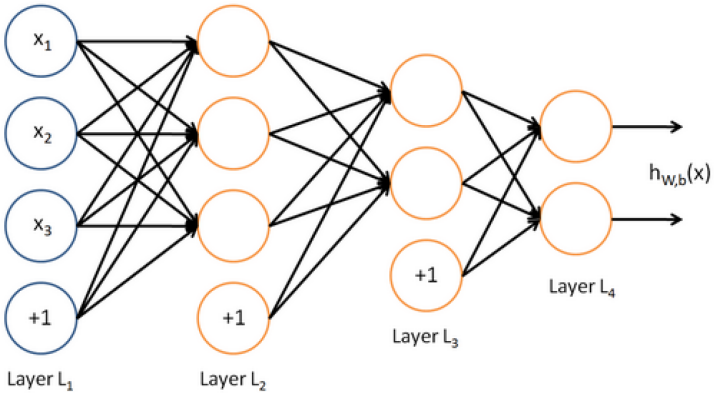
$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha\frac{\partial}{\partial W_{ij}^{(l)}}J(W,b)$$

$$b_i^{(l)} = b_i^{(l)} - \alpha\frac{\partial}{\partial b_i^{(l)}}J(W,b)$$

对矩阵求导

$$A \in \mathbb{R}^{m \times n} \qquad \nabla_A f(A) \in \mathbb{R}^{m \times n} = \begin{bmatrix} \frac{\partial f(A)}{\partial A_{11}} & \frac{\partial f(A)}{\partial A_{12}} & \cdots & \frac{\partial f(A)}{\partial A_{1n}} \\ \frac{\partial f(A)}{\partial A_{21}} & \frac{\partial f(A)}{\partial A_{22}} & \cdots & \frac{\partial f(A)}{\partial A_{2n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f(A)}{\partial A_{m1}} & \frac{\partial f(A)}{\partial A_{m2}} & \cdots & \frac{\partial f(A)}{\partial A_{mn}} \end{bmatrix}$$

$$x \in \mathbb{R}^n \qquad \nabla_x f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix}$$



$$J(W,b;x,y) = \frac{1}{2}\left\|h_{W,b}(x) - y\right\|^2$$

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha\frac{\partial}{\partial W_{ij}^{(l)}}J(W,b)$$

$$b_i^{(l)} = b_i^{(l)} - \alpha\frac{\partial}{\partial b_i^{(l)}}J(W,b)$$

$$J(W,b) = \left[\frac{1}{m}\sum_{i=1}^{m}J(W,b;x^{(i)},y^{(i)})\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2$$

$$= \left[\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\left\|h_{W,b}(x^{(i)}) - y^{(i)}\right\|^2\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2$$

$$\frac{\partial}{\partial W_{ij}^{(l)}}J(W,b) = \left[\frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial W_{ij}^{(l)}}J(W,b;x^{(i)},y^{(i)})\right] + \lambda W_{ij}^{(l)}$$

$$\frac{\partial}{\partial b_i^{(l)}}J(W,b) = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial b_i^{(l)}}J(W,b;x^{(i)},y^{(i)})$$

1. 进行前馈传导计算，利用前向传导公式，得到 $L_2, L_3, \ldots$ 直到输出层 L_{n_l} 的激活值。
2. 对于第 n_l 层（输出层）的每个输出单元 i，我们根据以下公式计算残差：

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \left\| y - h_{W,b}(x) \right\|^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$

$$\begin{aligned}
\delta_i^{(n_l)} &= \frac{\partial}{\partial z_i^{n_l}} J(W,b;x,y) = \frac{\partial}{\partial z_i^{n_l}} \frac{1}{2} \left\| y - h_{W,b}(x) \right\|^2 \\
&= \frac{\partial}{\partial z_i^{n_l}} \frac{1}{2} \sum_{j=1}^{S_{n_l}} (y_j - a_j^{(n_l)})^2 = \frac{\partial}{\partial z_i^{n_l}} \frac{1}{2} \sum_{j=1}^{S_{n_l}} (y_j - f(z_j^{(n_l)}))^2 \\
&= -(y_i - f(z_i^{(n_l)})) \cdot f'(z_i^{(n_l)}) = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})
\end{aligned}$$

$$\begin{aligned}
z^{(2)} &= W^{(1)} x + b^{(1)} \\
a^{(2)} &= f(z^{(2)}) \\
z^{(3)} &= W^{(2)} a^{(2)} + b^{(2)} \\
h_{W,b}(x) &= a^{(3)} = f(z^{(3)})
\end{aligned}$$

3. 对 $l = n_l - 1, n_l - 2, n_l - 3, \ldots, 2$ 的各个层，

第 l 层的第 i 个节点的残差计算方法如下：

$$\delta_i^{(n_l-1)} = \frac{\partial}{\partial z_i^{n_l-1}} J(W,b;x,y) = \frac{\partial}{\partial z_i^{n_l-1}} \frac{1}{2} \left\| y - h_{W,b}(x) \right\|^2 = \frac{\partial}{\partial z_i^{n_l-1}} \frac{1}{2} \sum_{j=1}^{S_{n_l}} (y_j - a_j^{(n_l)})^2$$

$$= \frac{1}{2} \sum_{j=1}^{S_{n_l}} \frac{\partial}{\partial z_i^{n_l-1}} (y_j - a_j^{(n_l)})^2 = \frac{1}{2} \sum_{j=1}^{S_{n_l}} \frac{\partial}{\partial z_i^{n_l-1}} (y_j - f(z_j^{(n_l)}))^2$$

$$= \sum_{j=1}^{S_{n_l}} -(y_j - f(z_j^{(n_l)})) \cdot \frac{\partial}{\partial z_i^{(n_l-1)}} f(z_j^{(n_l)}) = \sum_{j=1}^{S_{n_l}} -(y_j - f(z_j^{(n_l)})) \cdot f'(z_j^{(n_l)}) \cdot \frac{\partial z_j^{(n_l)}}{\partial z_i^{(n_l-1)}}$$

$$= \sum_{j=1}^{S_{n_l}} \delta_j^{(n_l)} \cdot \frac{\partial z_j^{(n_l)}}{\partial z_i^{n_l-1}} = \sum_{j=1}^{S_{n_l}} \left(\delta_j^{(n_l)} \cdot \frac{\partial}{\partial z_i^{n_l-1}} \sum_{k=1}^{S_{n_l-1}} f(z_k^{n_l-1}) \cdot W_{jk}^{n_l-1} \right)$$

$$= \sum_{j=1}^{S_{n_l}} \delta_j^{(n_l)} \cdot W_{ji}^{n_l-1} \cdot f'(z_i^{n_l-1}) = \left(\sum_{j=1}^{S_{n_l}} W_{ji}^{n_l-1} \delta_j^{(n_l)} \right) f'(z_i^{n_l-1})$$

4. 计算我们需要的偏导数，计算方法如下：

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x,y) = a_j^{(l)} \delta_i^{(l+1)}$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W,b;x,y) = \delta_i^{(l+1)}.$$

$$\frac{\partial J(W,b;x,y)}{\partial W_{ij}^{(l)}} = \frac{\partial J(W,b;x,y)}{\partial z_i^{(l+1)}} * \frac{\partial z_i^{(l+1)}}{\partial w_{ij}^{(l)}},$$

$$z_i^{(l+1)} = \sum_{k=1}^{s_l} W_{ik}^{(l)} a_k^{(l)} + b_i^{(l)}$$

$$\frac{\partial z_i^{(l+1)}}{\partial w_{ij}^{(l)}} = a_j^{(l)},$$

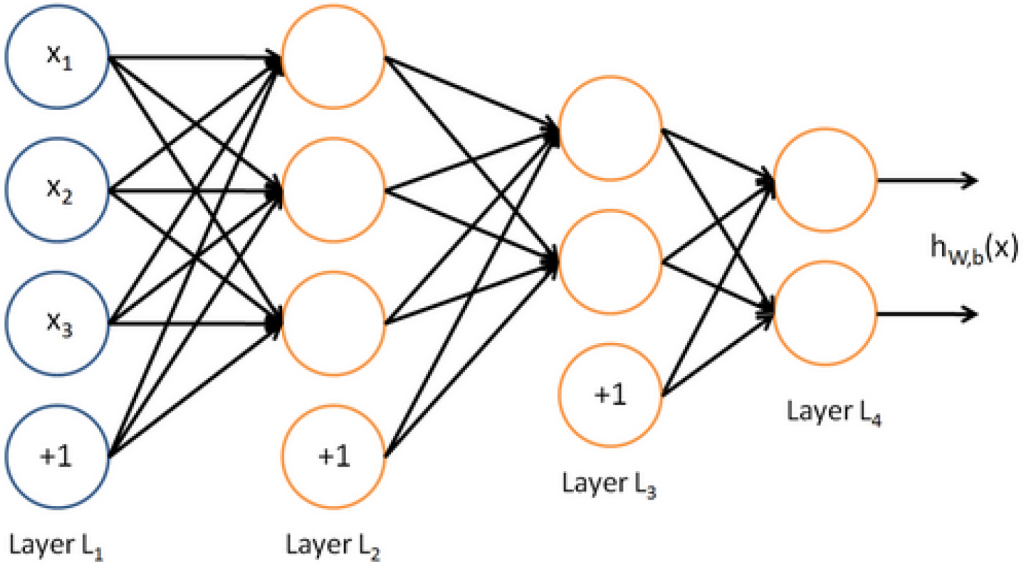
$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{n_l}} J(W,b;x,y)$$

$$\frac{\partial J(W,b;x,y)}{\partial W_{ij}^{(l)}} = a_j^{(l)} \delta_i^{(l+1)}$$

梯度下降算法总结

Step 1.

- 由前向后，依次计算出各层各个神经元的激活值：



$$f(z) = \frac{1}{1+\exp(-z)}$$

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

$$a^{(l+1)} = f(z^{(l+1)})$$

梯度下降算法总结

- Step 2.
  - 计算输出层（n_l层）各个神经元的$\delta_i^{(n_l)}$

$$\delta_i^{(n_l)} = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$

梯度下降算法总结

- Step 3.
  - 由后向前，依次计算出（l层）各个神经元的$\delta_i^{(l)}$

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)})$$

梯度下降算法总结

- Step 4.

  - 计算各层各个$\frac{\partial J(W,b;x,y)}{\partial W_{ij}^{(l)}}$和$\frac{\partial J(W,b;x,y)}{\partial \mathrm{b}_{i}^{(l)}}$的值：

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x,y) = a_j^{(l)} \delta_i^{(l+1)}$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W,b;x,y) = \delta_i^{(l+1)}.$$

梯度下降算法总结

- Step 5.

  - 计算各层各个$\frac{\partial J(W,b)}{\partial W_{ij}^{(l)}}$和$\frac{\partial J(W,b)}{\partial \mathrm{b}_{i}^{(l)}}$的值：

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b) = \left[\frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x^{(i)},y^{(i)}) \right] + \lambda W_{ij}^{(l)}$$

$$\frac{\partial}{\partial b_{i}^{(l)}} J(W,b) = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial b_{i}^{(l)}} J(W,b;x^{(i)},y^{(i)})$$

梯度下降算法总结

- Step 6.
  - 对各层的各个$W_{ij}^{(l)}$ 和 $b_{ij}^{(l)}$的值进行更新：

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b)$$

$$b_{i}^{(l)} = b_{i}^{(l)} - \alpha \frac{\partial}{\partial b_{i}^{(l)}} J(W,b)$$

梯度下降算法总结

- Step 7.
  - 直到$J(W,b)$的值“足够小”　：

$$J(W,b) = \left[\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\left\|h_{W,b}(x^{(i)}) - y^{(i)}\right\|^2\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2$$

矢量化编程

- 提高算法速度的一种有效方法。
- 矢量化编程的思想就是尽量使用这些被高度优化的数值运算操作来实现我们的学习算法。

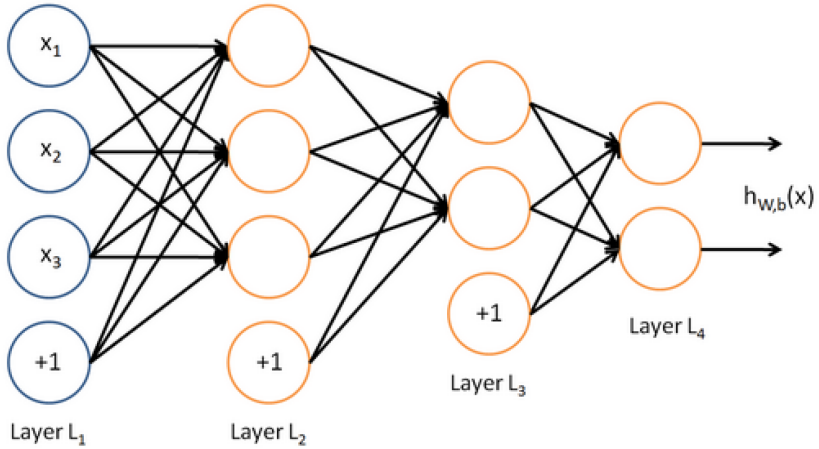
假设$x \in \Re^{n+1}$和$\theta \in \Re^{n+1}$为向量，需要计算$z = \theta^T x$

```
z = 0;
for i=1:(n+1),
  z = z + theta(i) * x(i);
end;
```

```
z = theta' * x;
```

代码中尽可能避免显式的for循环

神经网的矢量编程

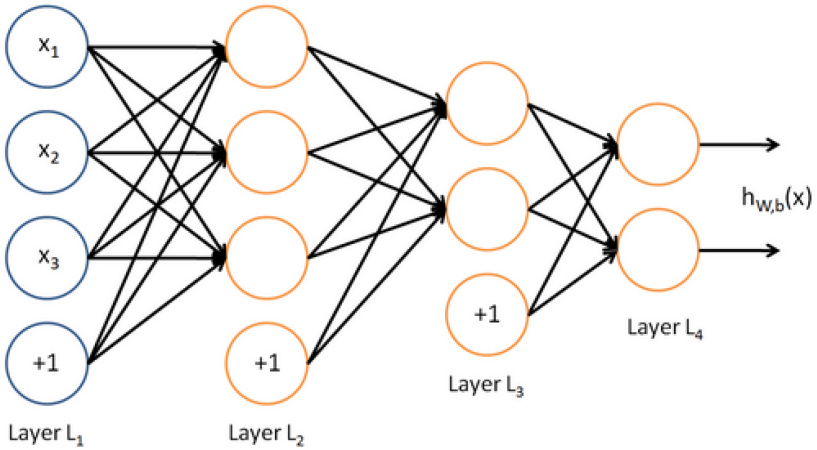


$$z^{(2)} = W^{(1)}x + b^{(1)}$$
$$a^{(2)} = f(z^{(2)})$$
$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$
$$h_{W,b}(x) = a^{(3)} = f(z^{(3)})$$

- 每个样本对应一个列向量，把这些列向量组成一个矩阵；
  - $a^{(2)}$就成了一个$s^{(2)}$ * m的矩阵。
  - 第i列表示：当第i个训练样本x(:i)输入到网络中时，该输入对隐神经元网络第二层的激励结果。

```
% 非向量化实现
for i=1:m,
  z2 = W1 * x(:,i) + b1;
  a2 = f(z2);
  z3 = W2 * a2 + b2;
  h(:,i) = f(z3);
end;
```

神经网的矢量编程

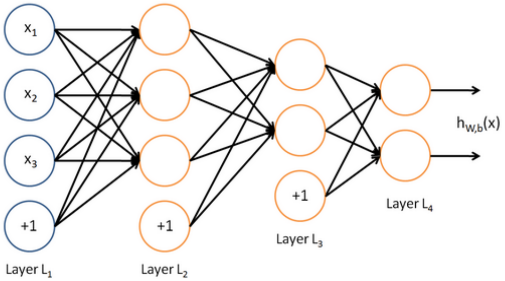


$$z^{(2)} = W^{(1)}x + b^{(1)}$$
$$a^{(2)} = f(z^{(2)})$$
$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$
$$h_{W,b}(x) = a^{(3)} = f(z^{(3)})$$

```
% 正向传播的向量化实现
z2 = W1 * x + repmat(b1,1,m);
a2 = f(z2);
z3 = W2 * a2 + repmat(b2,1,m);
h = f(z3)
```

```
% 非向量化实现
for i=1:m,
  z2 = W1 * x(:,i) + b1;
  a2 = f(z2);
  z3 = W2 * a2 + b2;
  h(:,i) = f(z3);
end;
```

神经网的矢量编程



$$\delta_i^{(n_l)} = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$

$$\delta^{(n_l)} = -(y - a^{(n_l)}) \bullet f'(z^{(n_l)})$$

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)}\right) f'(z_i^{(l)})$$

$$\delta^{(l)} = \left((W^{(l)})^T \delta^{(l+1)}\right) \bullet f'(z^{(l)})$$

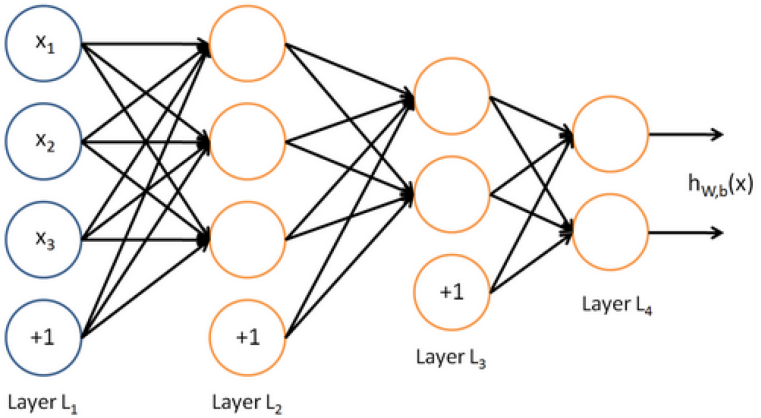
$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x,y) = a_j^{(l)} \delta_i^{(l+1)}$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W,b;x,y) = \delta_i^{(l+1)}.$$

$$\nabla_{W^{(l)}} J(W,b;x,y) = \delta^{(l+1)} (a^{(l)})^T$$

$$\nabla_{b^{(l)}} J(W,b;x,y) = \delta^{(l+1)}.$$

神经网



1. 进行前馈传导计算，利用前向传导公式，得到 $L_2, L_3, \ldots$ 直到输出层 L_{n_l} 的激活值。
2. 对输出层（第 n_l 层），计算：

$$\delta^{(n_l)} = -(y - a^{(n_l)}) \bullet f'(z^{(n_l)})$$

3. 对于 $l = n_l - 1, n_l - 2, n_l - 3, \ldots, 2$ 的各层，计算：

$$\delta^{(l)} = \left((W^{(l)})^T \delta^{(l+1)}\right) \bullet f'(z^{(l)})$$

4. 计算最终需要的偏导数值：

$$\nabla_{W^{(l)}} J(W, b; x, y) = \delta^{(l+1)} (a^{(l)})^T,$$

$$\nabla_{b^{(l)}} J(W, b; x, y) = \delta^{(l+1)}.$$

```
gradW1 = zeros(size(W1));
gradW2 = zeros(size(W2));
for i=1:m,
  delta3 = -(y(:,i) - h(:,i)) .* fprime(z3(:,i));
  delta2 = W2'*delta3(:,i) .* fprime(z2(:,i));

  gradW2 = gradW2 + delta3*a2(:,i)';
  gradW1 = gradW1 + delta2*a1(:,i)';
end;
```

初步梯度检验方法

设目标函数为$J(\theta)$其中θ为参数，则

- 可以通过如下方法验证计算$\frac{\partial J(\theta)}{\partial \theta}$的函数$g(\theta)$的正确性：

$$\frac{d}{d\theta}J(\theta) = \lim_{\epsilon \to 0} \frac{J(\theta+\epsilon) - J(\theta-\epsilon)}{2\epsilon}$$

$$g(\theta) \approx \frac{J(\theta + \text{EPSILON}) - J(\theta - \text{EPSILON})}{2 \times \text{EPSILON}}$$

- 取EPSILON为10^{-4}则上式结果大致相同（ 10^{-4} 范围内基本相同）

初步梯度检验方法

- 可以通过如下方法验证计算$\frac{\partial J(\theta)}{\partial \theta}$的函数$g(\theta)$的正确性：

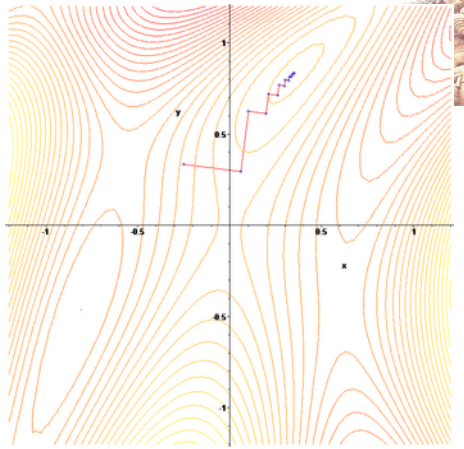
$$\theta^{(i+)} = \theta + \text{EPSILON} \times \vec{e}_i \qquad \theta^{(i-)} = \theta - \text{EPSILON} \times \vec{e}_i$$

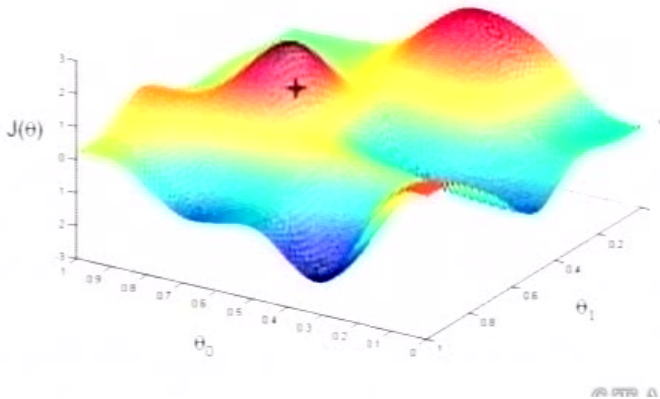
$$g_i(\theta) \approx \frac{J(\theta^{(i+)}) - J(\theta^{(i-)})}{2 \times \text{EPSILON}} \qquad \vec{e}_i = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

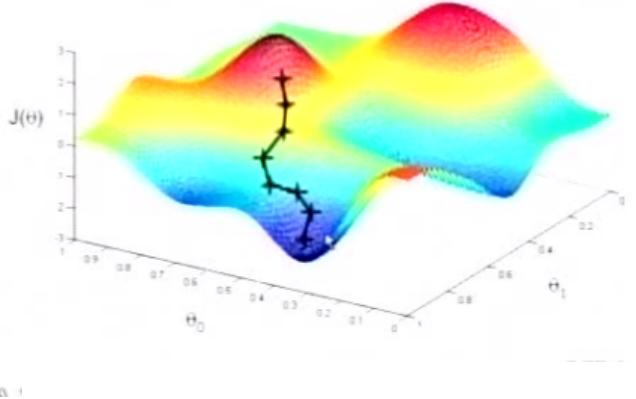
- 取EPSILON为10^{-4}则上式结果大致相同（ 10^{-4} 范围内基本相同）

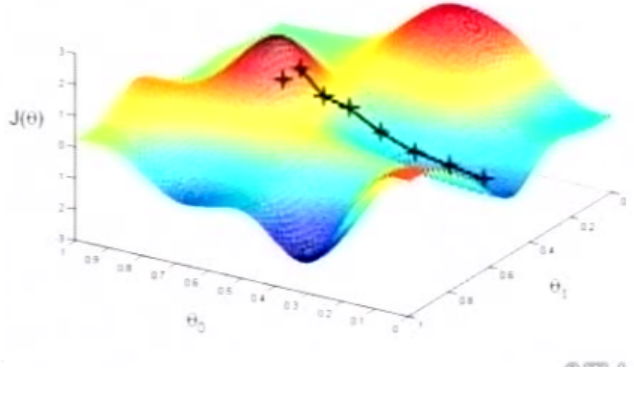
梯度下降法的缺点

- 局部最优解；
- 靠近极小值时速度减慢；
- 直线搜索可能会产生一些问题；
- 可能会‘之字型’地下降；







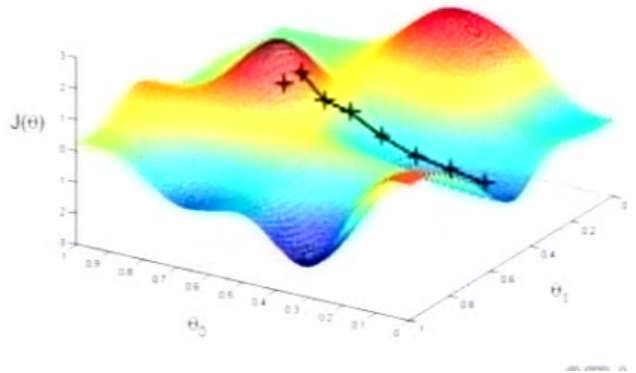


应对梯度下降缺点的办法

1. 调整权重调整的幅度

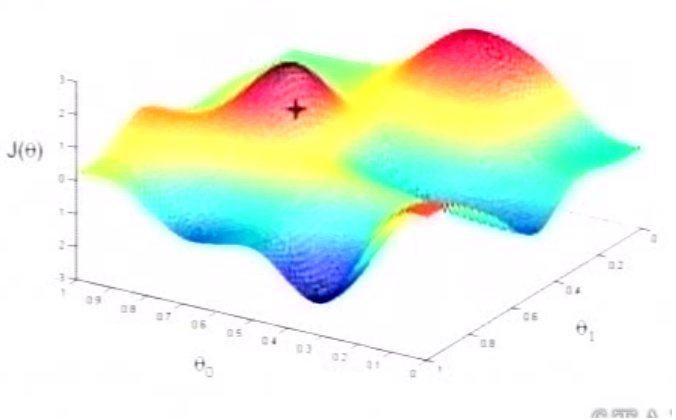
$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b)$$

$$b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W,b)$$



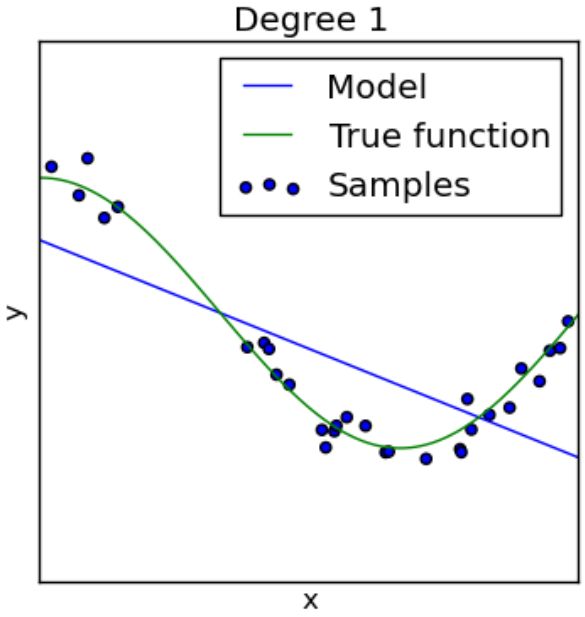
2. 使用不同的权重初始化值

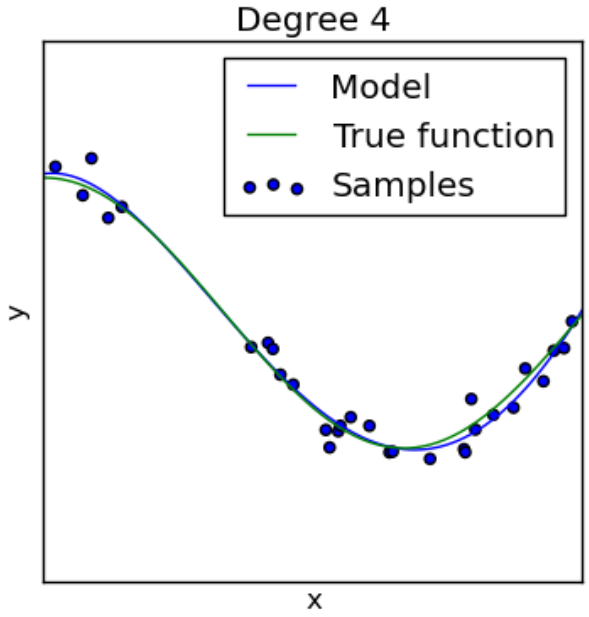
- 初始化值不同，结果可能不同
- 对权重调整过程进行干预

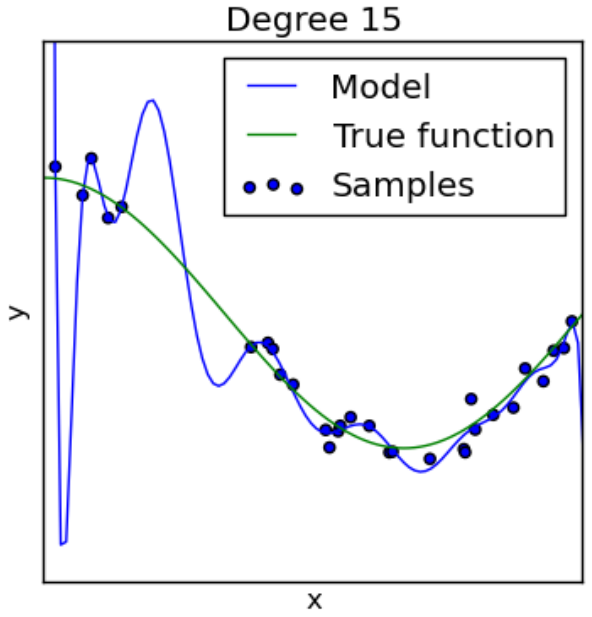


3. 使用随机梯度下降方法

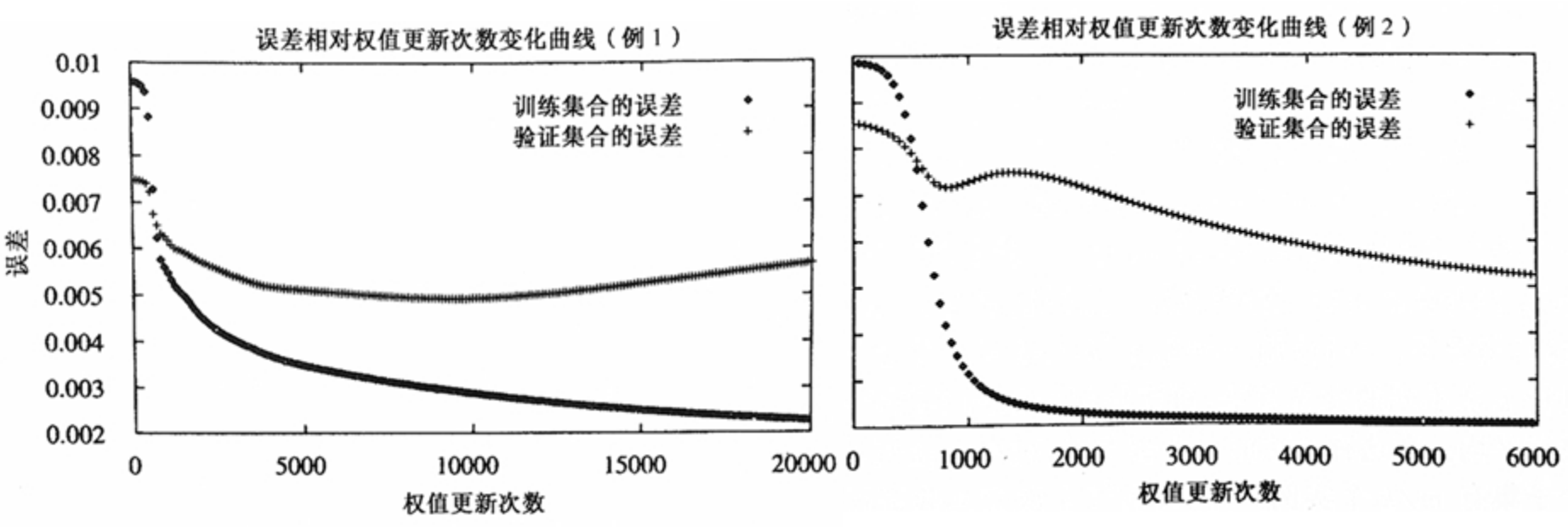
Underfitting vs. Overfitting



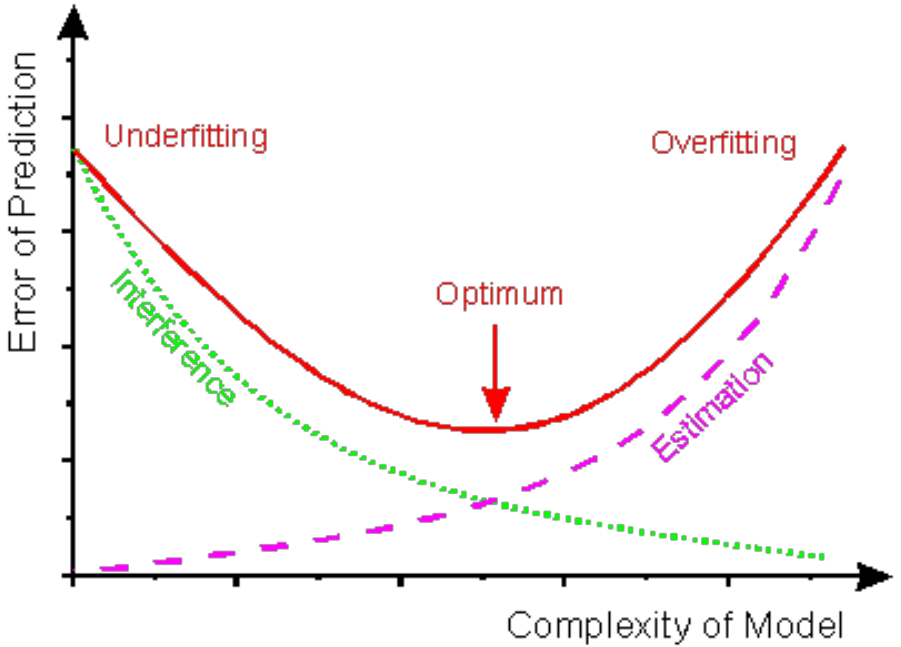


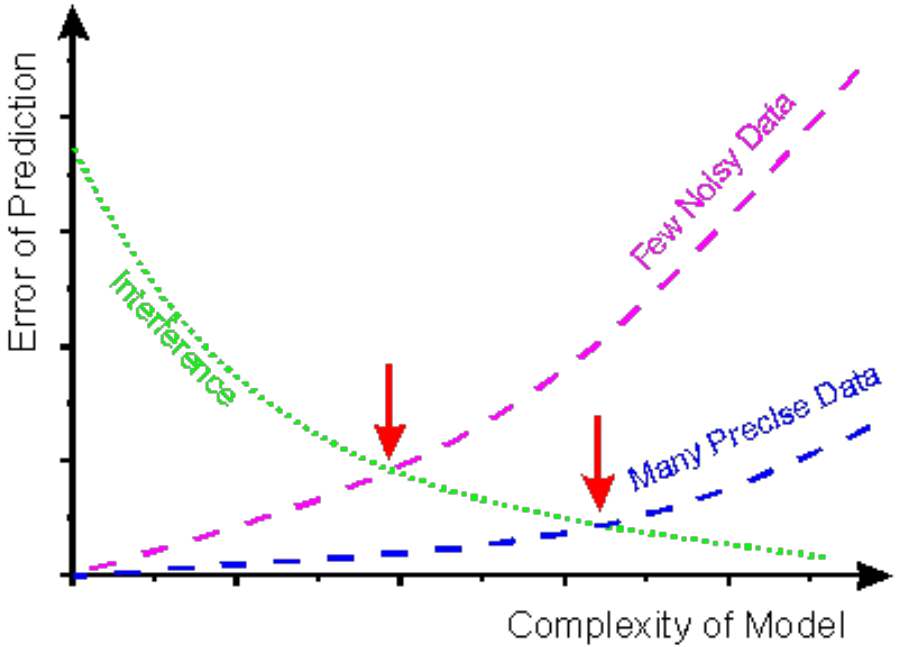


应对过度拟合的方法


误差相对权值更新次数变化曲线（例1）
训练集合的误差
验证集合的误差
误差
0.01
0.009
0.008
0.007
0.006
0.005
0.004
0.003
0.002
0
5000
10000
15000
20000
权值更新次数
误差相对权值更新次数变化曲线（例2）
训练集合的误差
验证集合的误差
0
1000
2000
3000
4000
5000
6000
权值更新次数

Model Complexity





Thanks.