人工神经网络

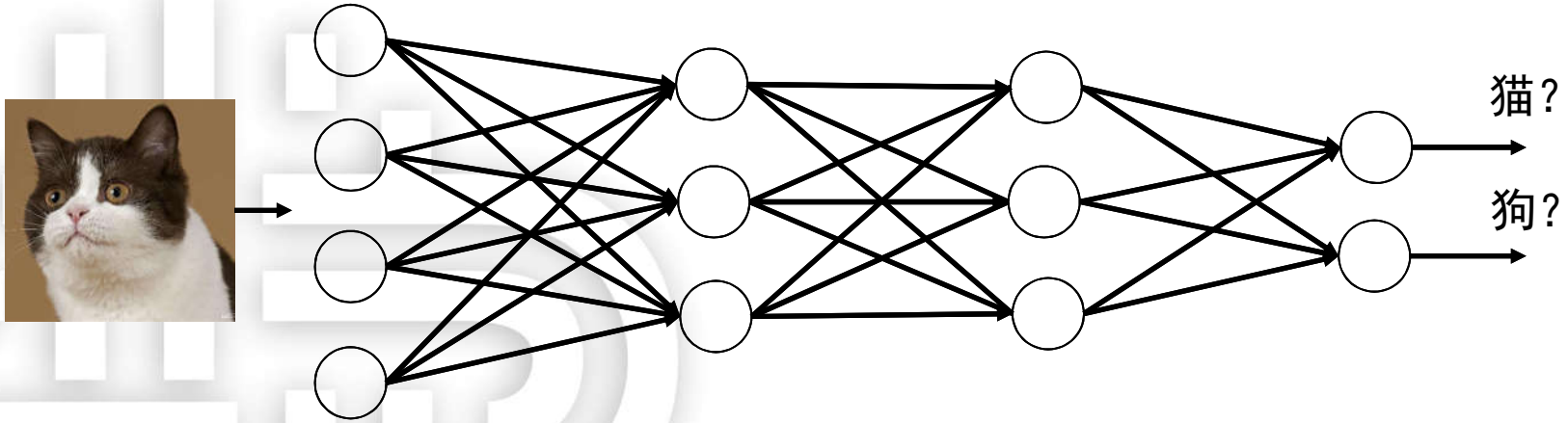
PDL, School of Computer, NUDT

主要内容

- 人工神经网络简介
- 人工神经网络的起源与发展
- 神经网络原理及功能结构
- Tensorflow神经网络训练方法

人工神经网络简介

- 人工神经网络（Artificial Neural Network，ANN）
- 一种信息处理数学模型



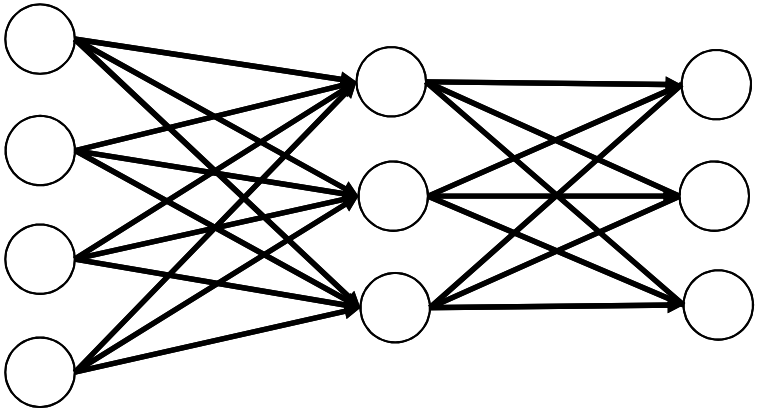
Artificial neural networks (**ANNs**) or **connectionist systems** are computing systems vaguely inspired by the biological neural networks that constitute animal brains.[1]

人工神经网络简介

- 人工神经网络
  - 生物神经网络假说：神经元等组成大型网络结构，产生意识，帮助思考和行动
  - 仿生：生物神经网络→形式化模型？
  - 复杂网络结构：大量的处理单元(神经元) 互相连接。单元怎么模拟？单元怎么连接？信息网络传播处理的机制？网络学习机制？

生物神经网络

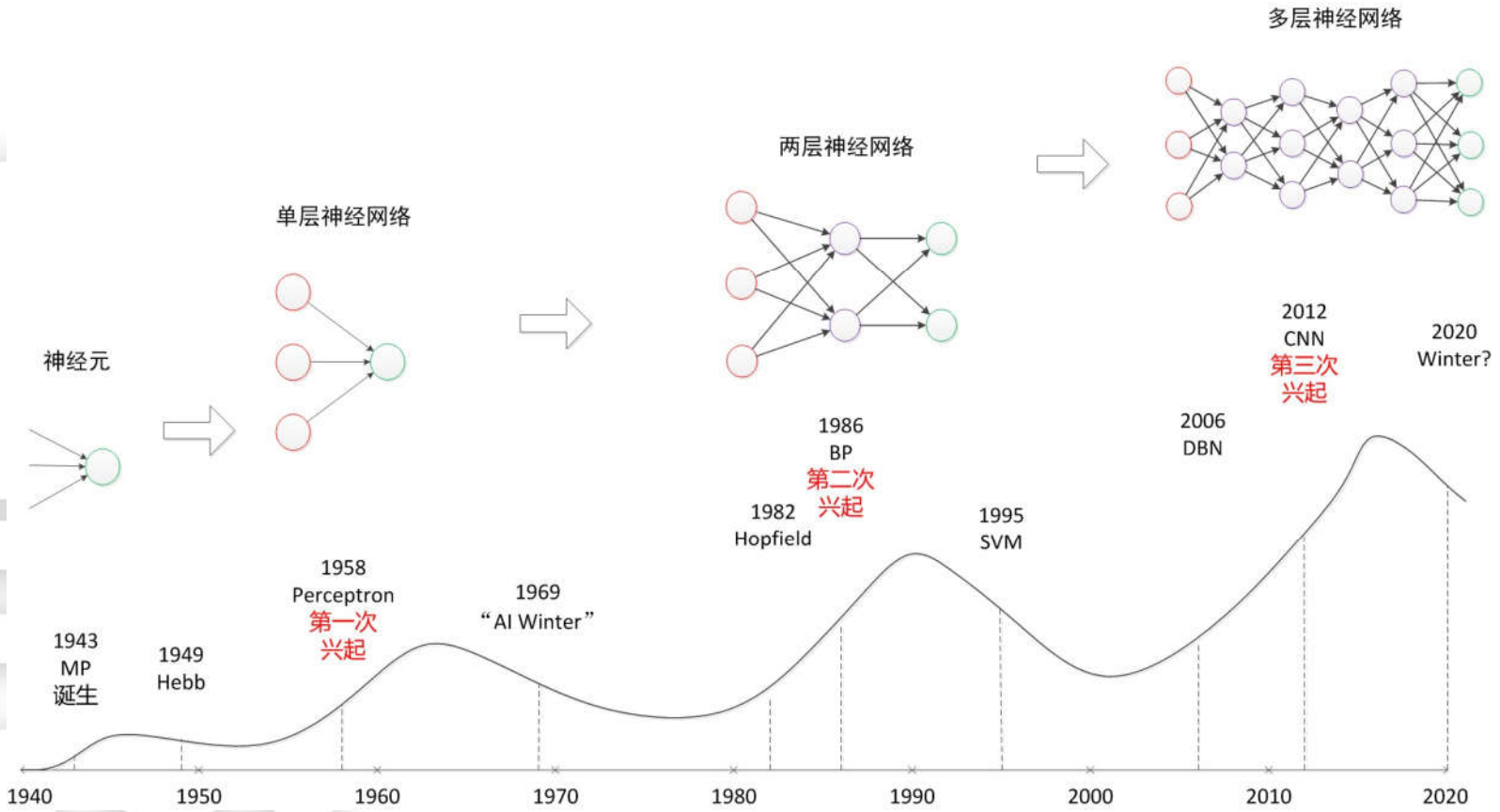
仿生



数学模型

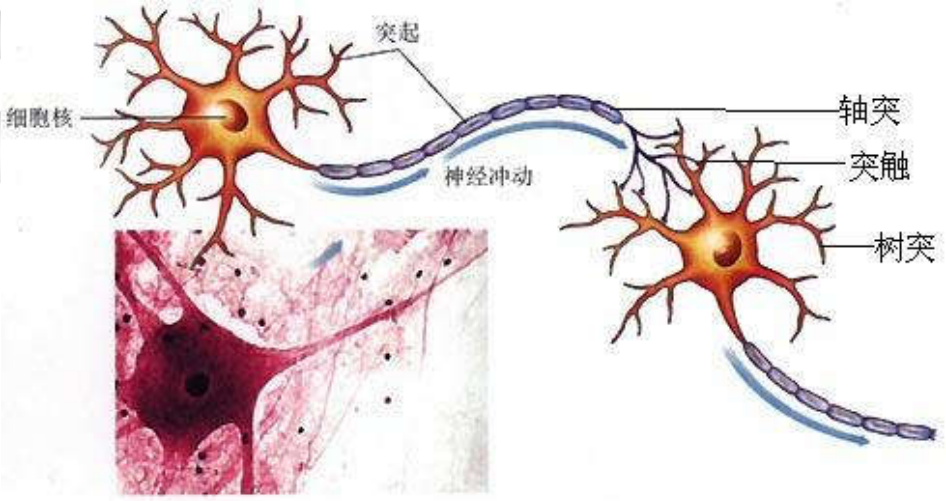
人工神经网络的起源与发展

- 发展脉络（三次高潮）



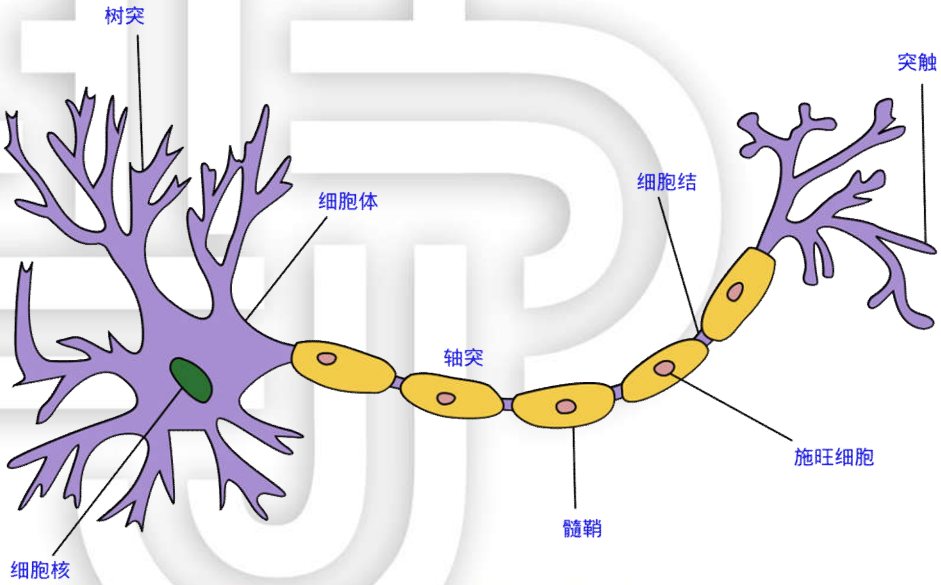
人工神经网络的起源与发展

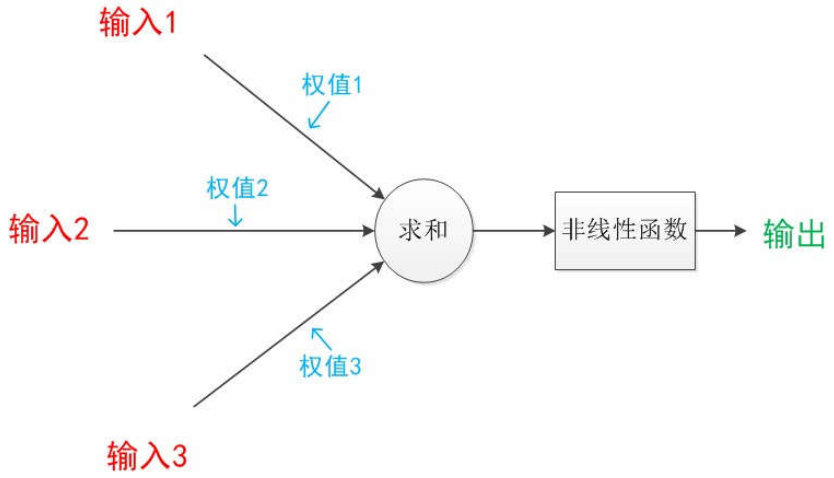
- 人工神经网络的基础结构：神经元
  - 大脑神经元：约10^12个，每个与10^2~10^4个其他神经元相连
  - 神经元的组成结构：1904年
  - 状态：兴奋（传递信息）和抑制
  - 突触：轴突末梢与其他树突的连接
  - 兴奋性突触和抑制性突触



人工神经网络的起源与发展

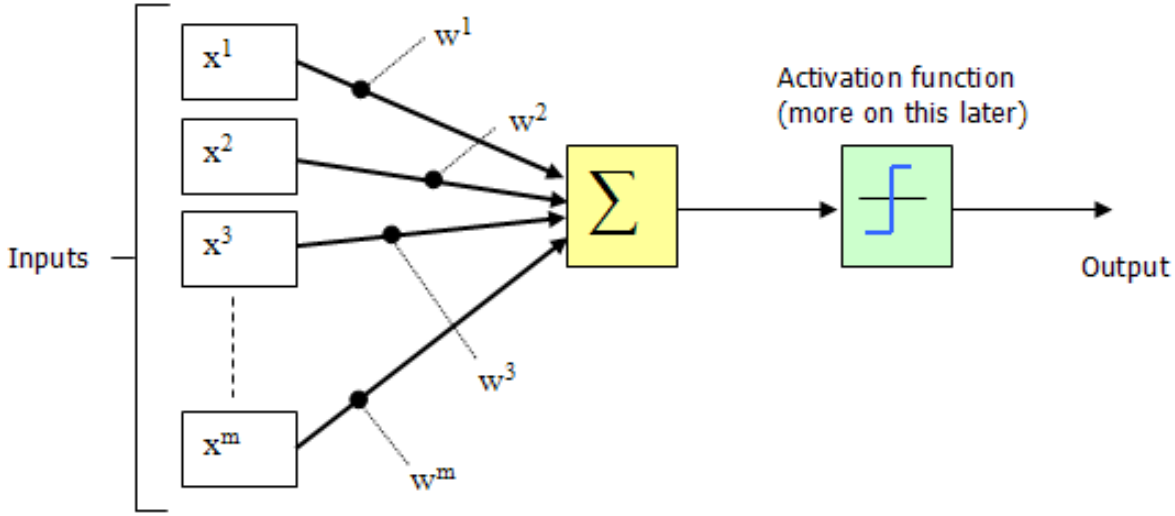
- 人工神经网络的基础结构：神经元
  - 1943年，心理学家McCulloch和数学家Pitts参考了生物神经元的结构，发表了抽象的神经元模型The McCulloch-Pitts Neuron
  - 输入：树突；输出：轴突；计算：胞体





人工神经网络的起源与发展

- 神经元
  - 权重连接，权重：模拟兴奋或抑制连接
  - 偏置：激活阈值



$$y_i = f\left(\Sigma_i w_i x_i + b\right)$$

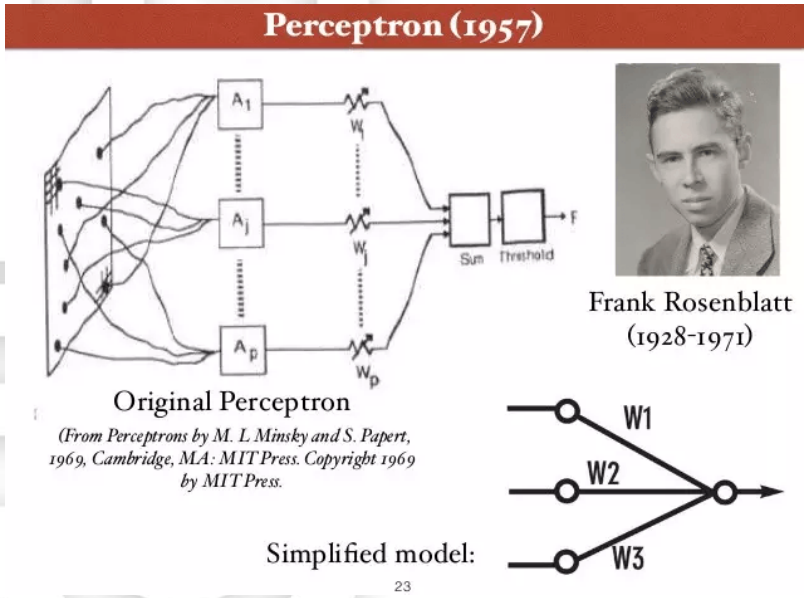
人工神经网络的起源与发展

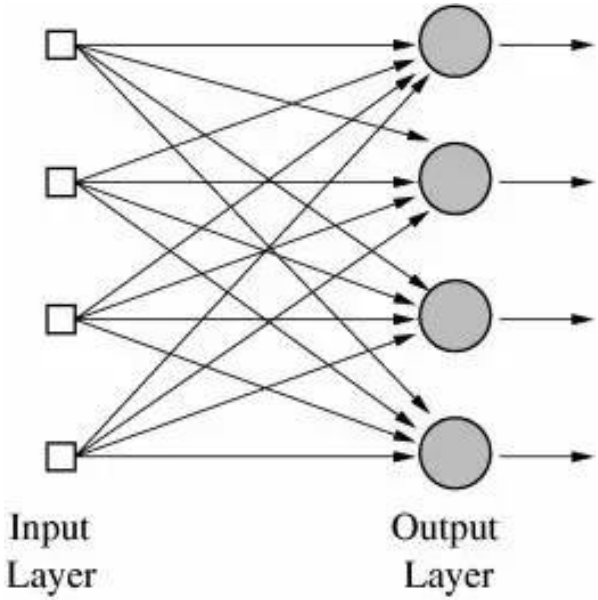
- MP神经元的缺陷
  - MP模型中，权重的值都是预先设置的，不能学习。
  - 1949年心理学家Hebb提出了Hebb学习规则——认为人脑神经细胞的突触（也就是连接）上的强度上可以变化的
  - 用调整权值的方法来让机器学习，奠定学习算法基础。
  - 无监督，强化常见模式的刺激
  - $delta\ Wj = k\ \times f\left(W_j^T X\right)X$

Donald Olding Hebb

人工神经网络的起源与发展

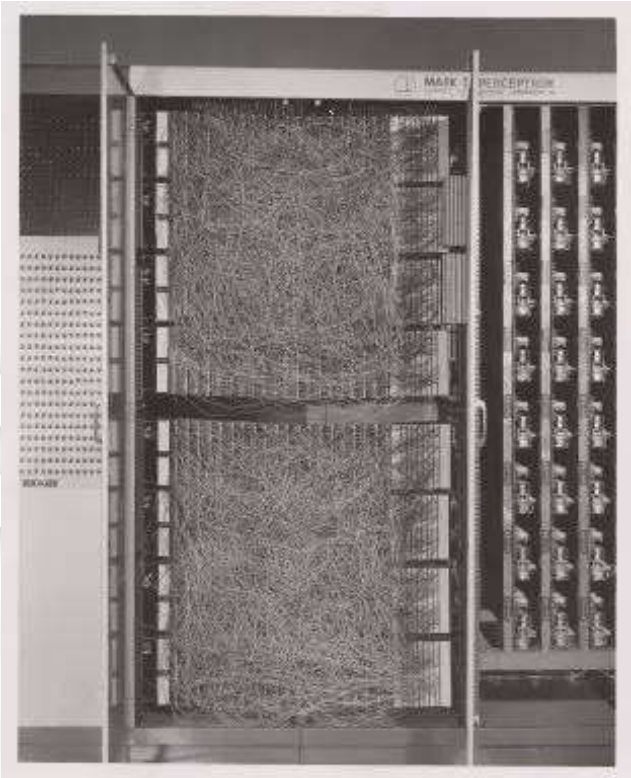
- 单层神经网络（感知器）
  - 历史：1958年，计算科学家Rosenblatt提出了由两层神经元组成的神经网络：“感知器”（Perceptron）
  - 感知机：MP神经元+一套学习算法
  - 多任务：single layer perceptron

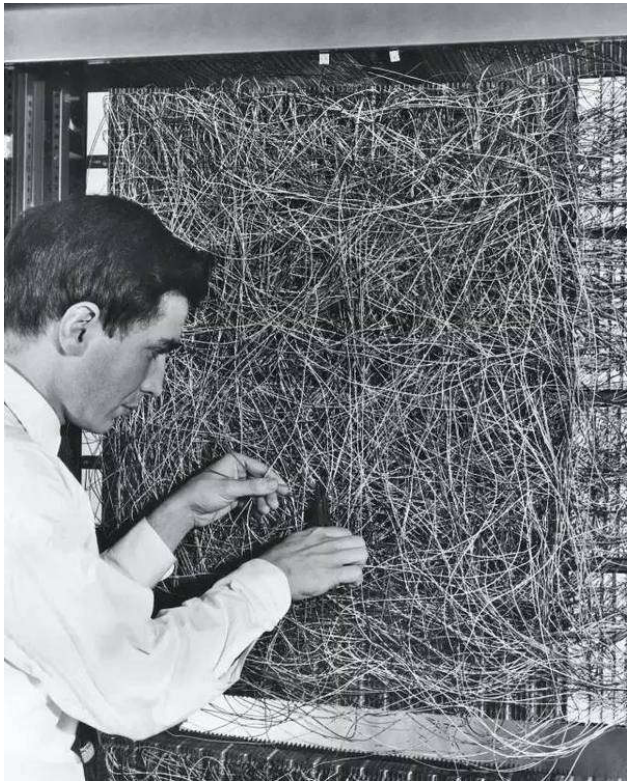




人工神经网络的起源与发展

- 纽约时报》当时报道说：“海军透露了一种电子计算机的雏形，它将能够走路、说话、看、写、自我复制并感知到自己的存在……据预测，不久以后，感知器将能够识别出人并叫出他们的名字，立即把演讲内容翻译成另一种语言并写下来。”

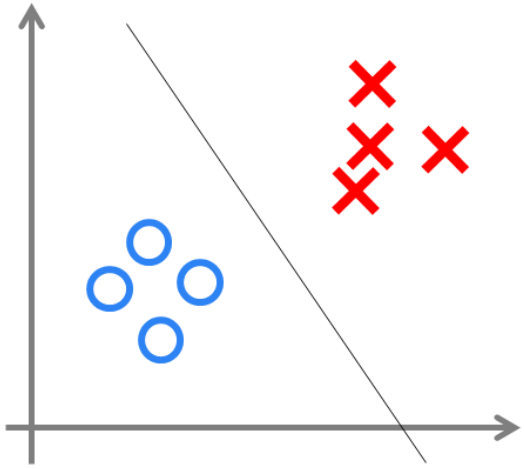




康奈尔航天实验室的Mark I 感知机 20x20 像素形状分类

人工神经网络的起源与发展

- 感知机可以解决的问题及其局限性
  - 本质解决二分类问题（怎样改成多分类？）
  - 线性可分问题（为什么?）



$$y = h\left(\sum_{i=1}^{n} w_i \cdot \boldsymbol{x_i} + b\right)$$

$$h(a) = \begin{cases} 1 & a > 0 \\ 0 & a <= 0 \end{cases}$$

人工神经网络的起源与发展

- 感知机的训练：知错能改（只管错的）

$$令x_0=1，\ w_0=b，\ 分割面\sum_{i=1}^{n+1}w_i\cdot\boldsymbol{x_i}=0$$

$$h_w(x)=f\left(\sum_{i=1}^{n+1}w_i\cdot\boldsymbol{x_i}\right)\ h(a)=\begin{cases}1 & a>0\\ 0 & a<=0\end{cases}$$

始终有：$(y^{(i)}-h_w(\vec{x}^{(i)}))w_T\vec{x}^{(i)}\leqslant 0$（分对：$=0$;分错：分错：$<0$）

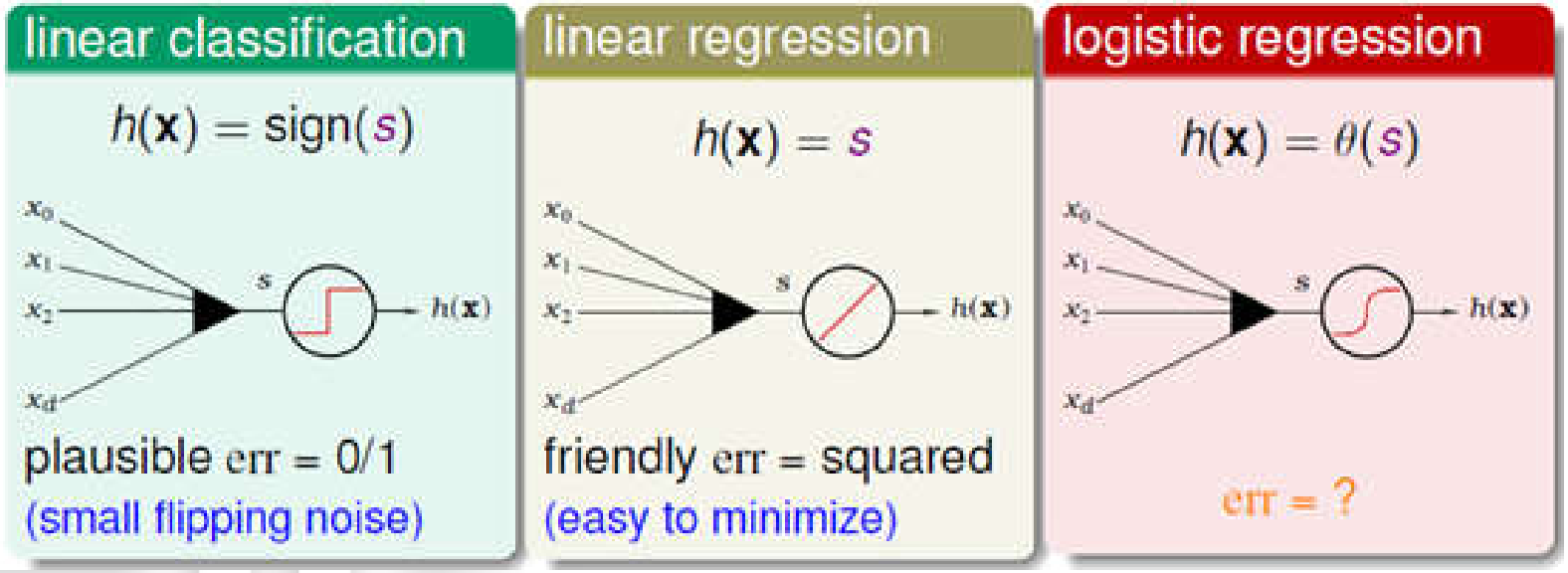
优化目标：错的离分割面距离总和最小

$$l(w)=-\sum_{i\in E}(y^{(i)}-h_w(x^{(i)}))w_Tx^{(i)}=-\sum_{i\in D}(y^{(i)}-h_w(x^{(i)}))w_Tx^{(i)}$$

$$w:=w-\alpha(\nabla(l(w)):=w+\alpha\sum_{i\in D}(y^{(i)}-h_w(x^{(i)}))x^{(i)}$$

人工神经网络的起源与发展

- 感知机、线性回归、逻辑回归
  - 感知机促进了逻辑回归、SVM的发展



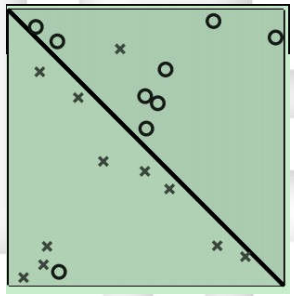
人工神经网络的起源与发展

- 感知机、线性回归分类对比
  - 感知机：令$x_0=1$，$w_0=b$，分割面$\sum_{i=1}^{n+1} w_i \cdot \boldsymbol{x_i}=0$
  - 错误数据到分割面的距离总和最小
  - 知错能改（只管错的）

$$h_w(x)=f\left(\sum_{i=1}^{n+1} w_i \cdot \boldsymbol{x_i}\right)$$

  - 全分对的时候？

$$l(w)=-\sum_{i\in E}\left(y^{(i)}-h_w(x^{(i)})\right)w^T x^{(i)}$$



$$\frac{w^T x^{(i)}}{|w|}=\cos<w,x^{(i)}>|x^{(i)}| \rightarrow x^{(i)}\text{到分割面法线的投影}$$

$x^{(i)}$到分割面的距离（$|w|$归一化后为$w^T x^{(i)}$）

  - 线性回归：

$$h_w(x)=\sum_{i=1}^{n+1} w_i \cdot \boldsymbol{x_i}$$

  - 假设分类问题，y=0或1，h(x)>0分为1反之分为0
  - 分对的尽量离分割面远，分错的尽量近

$$l(w)=\frac{1}{2}\sum_{i\in D}\left(y^{(i)}-h_w(x^{(i)})\right)^2$$

人工神经网络的起源与发展

- 感知机、逻辑回归分类对比：
  - 感知机：
  - 错误数据到分割面的距离总和最小
  - 知错能改（只管错的）

$$令x_0=1，\ w_0=b，\ 分割面\sum_{i=1}^{n+1}w_i\cdot x_i=0$$

$$h_w(x)=g(w^Tx)=\frac{1}{1+e^{-w^Tx}}\quad g(z)=\frac{1}{1+e^{-z}}$$

$$l(w)=-D(w)=-\sum_{i=1}^{m}(y^{(i)}logh_w(x^{(i)})+(1-y^{(i)})log(1-h_w(x^{(i)})))$$

  - 逻辑回归：
  - 和线性回归类似

$$Cost(h_w(x))=\begin{cases}-log(h_w(x)), & y=1\\ -log(1-h_w(x)), & y=0\end{cases}\quad（+1,-1时，取y=(y'+1)/2）$$

  - 取所有点到分割面距离最大为目标
  - 但距离用对数定义（分割面越近越敏感。为什么不用平方？损失非凸）

人工神经网络的起源与发展

- 感知机和SVM：SVM ≈ L2 regularized Perceptron

$$1.\ \min_{w,b} \frac{1}{n} \sum_{i=1}^{n} \max\left(0, -y_i(w \cdot x_i + b)\right)$$

$$2.\ \min_{w,b} \frac{1}{n} \sum_{i=1}^{n} \max\left(0, 1 - y_i(w \cdot x_i + b)\right)$$

1. 2. 问题等价，*svm*对应优化：$\min_{w,b} \frac{1}{n} \sum_{i=1}^{n} \max\left(0, 1 - y_i(w \cdot x_i + b)\right) + \lambda \|w\|_2^2$

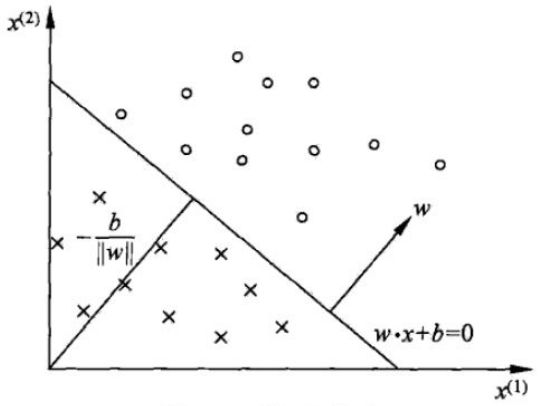


图 2.1　感知机模型

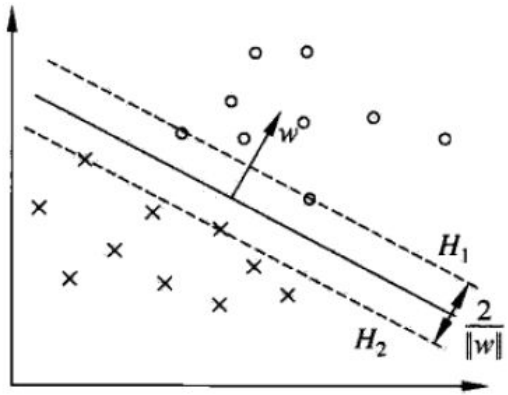


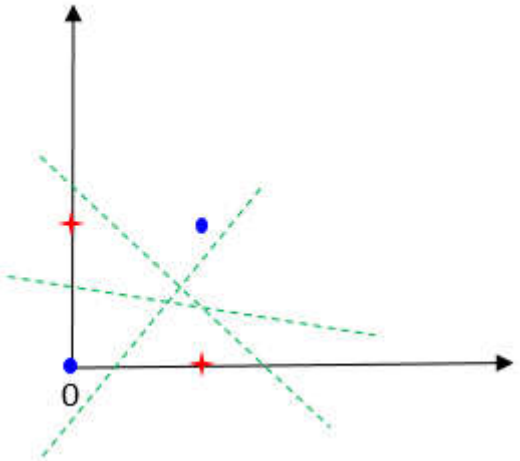
图 7.3　支持向量

- 两者也都可以用kernel trick, 加kernel相当于又加了一层

人工神经网络的基本原理

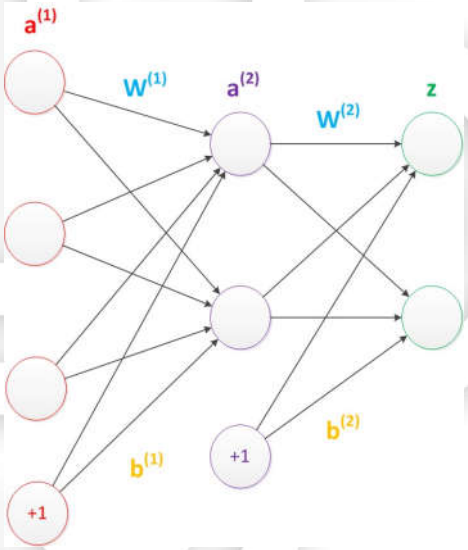
- 感知器的局限性：
  - 只能做简单的线性分类任务。
  - Minsky，1969，《Perceptron》：数学证明了感知器的弱点，尤其是感知器对XOR（异或）这样的简单分类任务都无法解决
  - 不光感知机，所有线性分类器都无法解决
  - Minsky：增加到两层，计算量则过大，而且没有有效的学习算法。

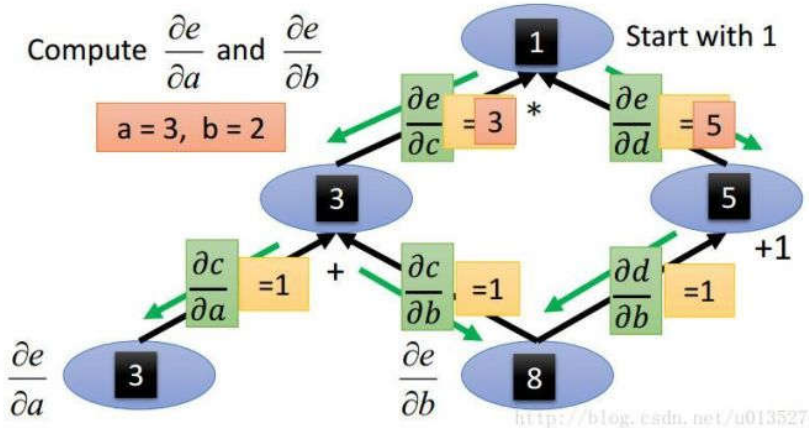
输入		输出
x_{i1}	x_{i2}	y_i
0	0	0
0	1	1
1	0	1
1	1	0



人工神经网络的基本原理

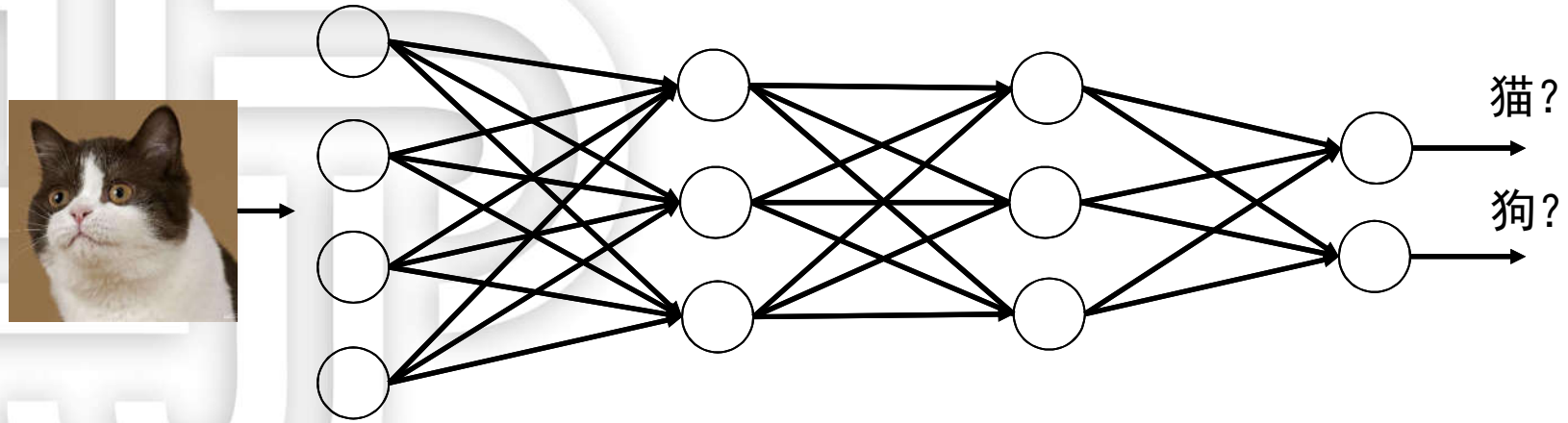
- 多层感知器： multi layer perceptron
  - Minsky:单层神经网络无法解决异或问题，两层可以，怎么优化？
  - 1986年，Rumelhar和Hinton：反向传播（Backpropagation，BP）解决复杂优化问题





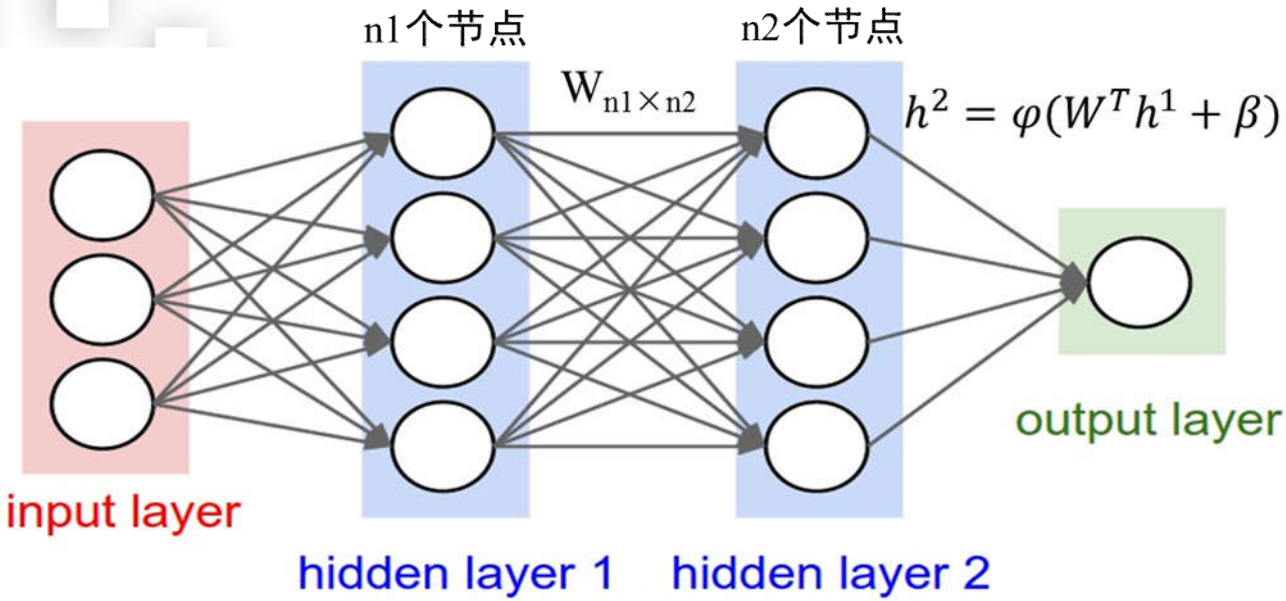
人工神经网络的基本原理

- 以多层感知机为例：
- 网络计算的意义？以层为单位，逐层映射，非线性映射
- 映射的意义？提取益于最终任务的特征（例如分类，提取类型的层次化显著特征）
- 怎样映射？结构、功能、计算过程



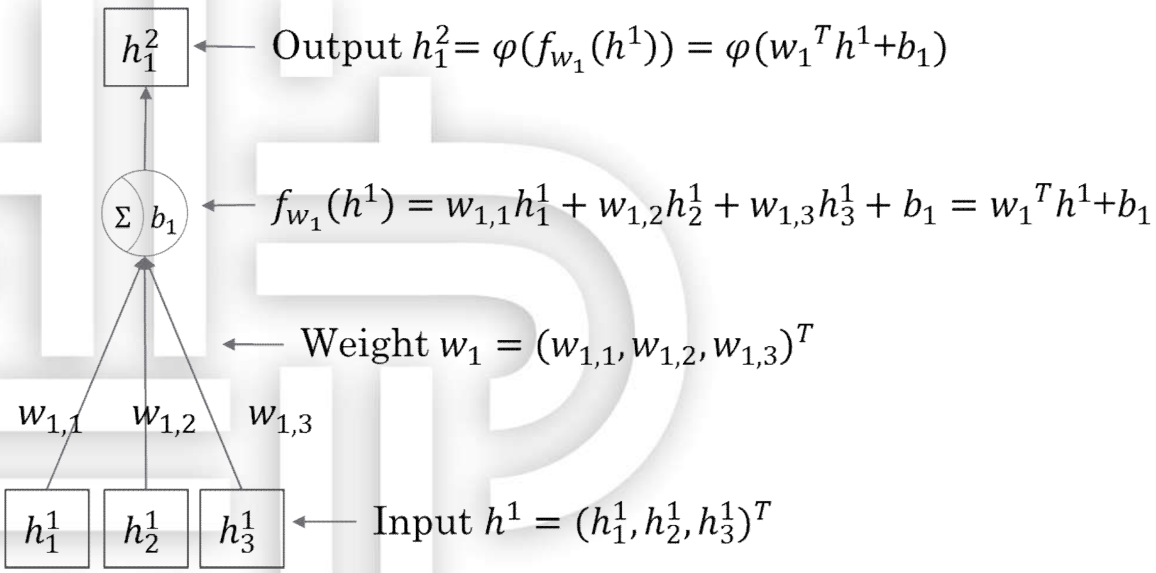
神经网络原理及功能结构

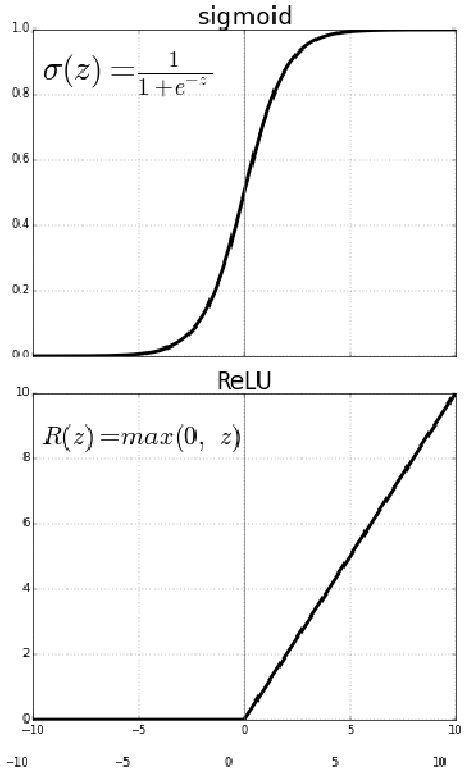
- 全连接神经网络
  - 多个隐含层，层级全连接（全连接？层级全连接？）
  - 每层功能：非线性映射
  - 每层结构：神经元层级全连接+非线性映射函数



神经网络原理及功能结构

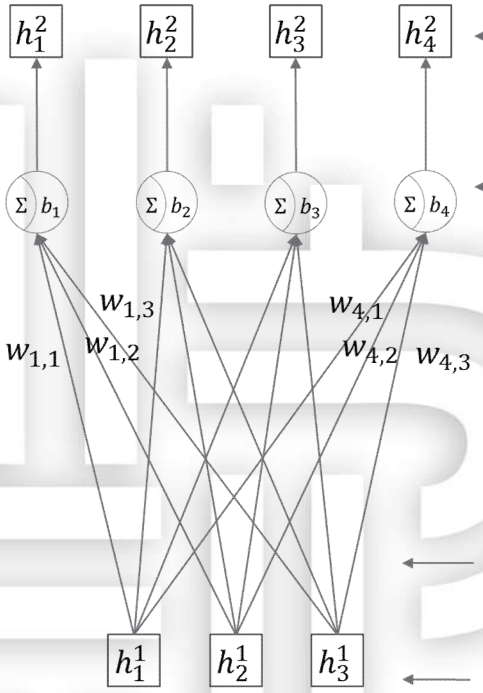
- 单个隐含层节点的结构及前向传播计算过程：





神经网络原理及功能结构

- 一个隐含层的结构及前向传播计算过程：



Output $h^2 = (h_1^2, h_2^2, h_3^2, h_4^2)^T = \varphi(f_W(h^1)) = \varphi(W^T h^1 + \beta)$

$$f_W(h^1) = \begin{pmatrix} w_{1,1}h_1^1 + w_{1,2}h_2^1 + w_{1,3}h_3^1 + b_1 \\ w_{2,1}h_1^1 + w_{2,2}h_2^1 + w_{2,3}h_3^1 + b_2 \\ w_{3,1}h_1^1 + w_{3,2}h_2^1 + w_{3,3}h_3^1 + b_3 \\ w_{4,1}h_1^1 + w_{4,2}h_2^1 + w_{4,3}h_3^1 + b_4 \end{pmatrix}$$

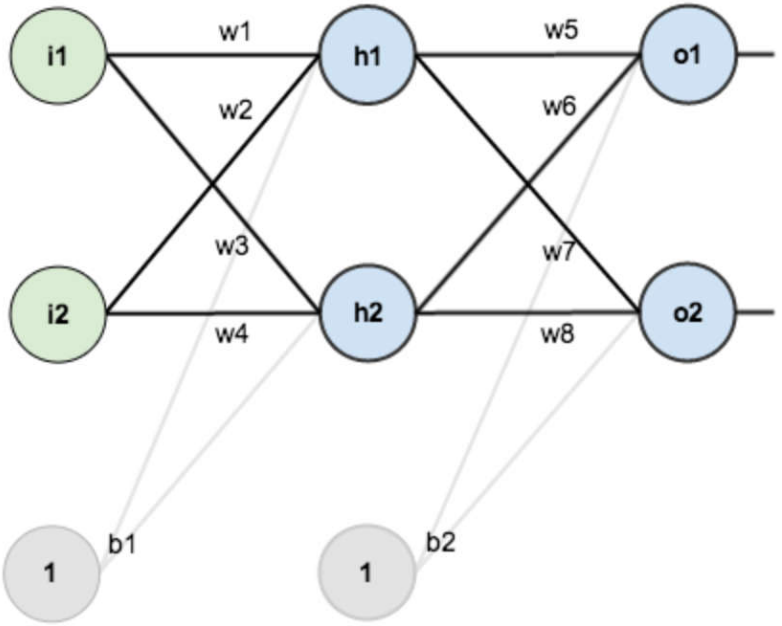
$$= \begin{pmatrix} w_{1,1}, w_{1,2}, w_{1,3} \\ w_{2,1}, w_{2,2}, w_{2,3} \\ w_{3,1}, w_{3,2}, w_{3,3} \\ w_{4,1}, w_{4,2}, w_{4,3} \end{pmatrix} \begin{pmatrix} h_1^1 \\ h_2^1 \\ h_3^1 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix} = W^T h^1 + \beta$$

Weight $W = (w_1, w_2, w_3, w_4) = \begin{pmatrix} w_{1,1}, w_{2,1}, w_{3,1}, w_{4,1} \\ w_{1,2}, w_{2,2}, w_{3,2}, w_{4,2} \\ w_{1,3}, w_{2,3}, w_{3,3}, w_{4,3} \end{pmatrix}_{3\times4}$

Input $h^1 = (h_1^1, h_2^1, h_3^1)^T$

神经网络原理及功能结构

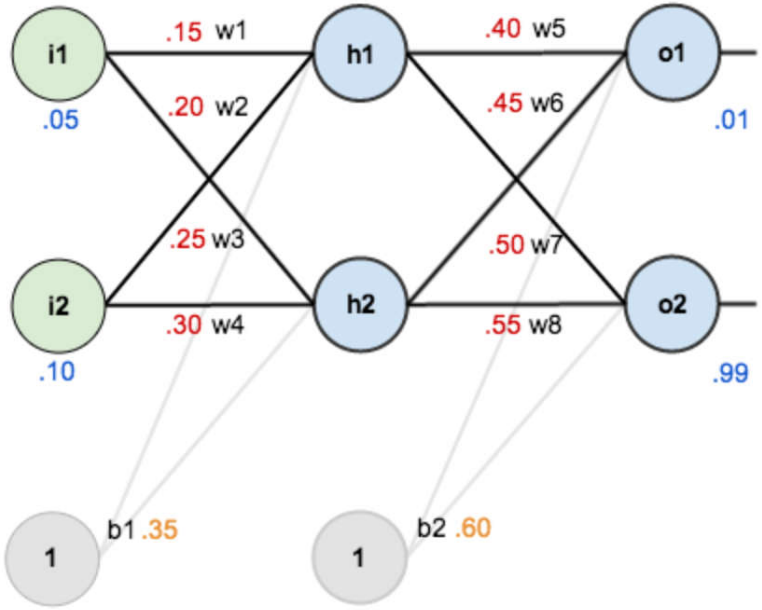
- 使用一个具体的例子来讲述前向计算过程
  - 第一层是输入层，包含两个神经元i1，i2，和偏置值b1；第二层是隐含层，包含两个神经元h1,h2和偏置值b2，第三层是输出o1,o2，每条线上标的wi是层与层之间连接的权重，激活函数我们默认为sigmoid函数。



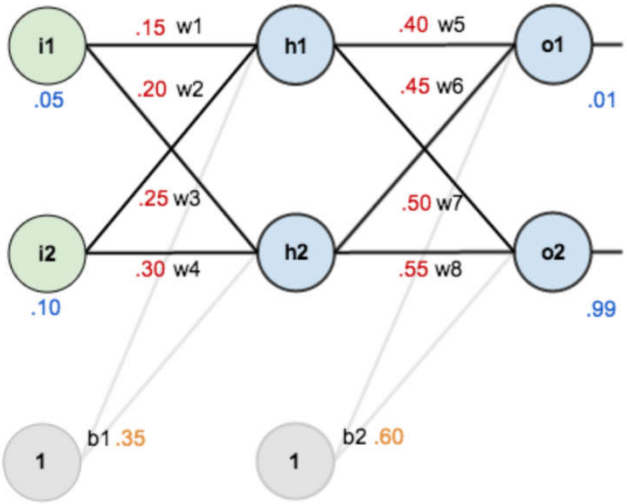
神经网络原理及功能结构

- 现在对他们赋上初值，如下图：
  - 其中，输入数据 i1=0.05，i2=0.10;
  - 输出数据 o1=0.01,o2=0.99;
  - 初始权重 w1=0.15,w2=0.20,w3=0.25,w4=0.30;
  - w5=0.40,w6=0.45,w7=0.50,w8=0.55

  - 目标：给出输入数据 i1,i2(0.05和0.10)，使输出尽可能与原始输出o1,o2(0.01和0.99)接近。



神经网络原理及功能结构



- **前向传播：**
- 1.输入层---->隐含层：
- 计算神经元h1的输入加权和：

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

- 神经元h1的输出o1:(此处用到激活函数为sigmoid函数)：

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$$

- 同理，可计算出神经元h2的输出o2：$out_{h2} = 0.596884378$

- 前向传播：
- 2.隐含层---->输出层：
- 计算输出层神经元o1和o2的值：

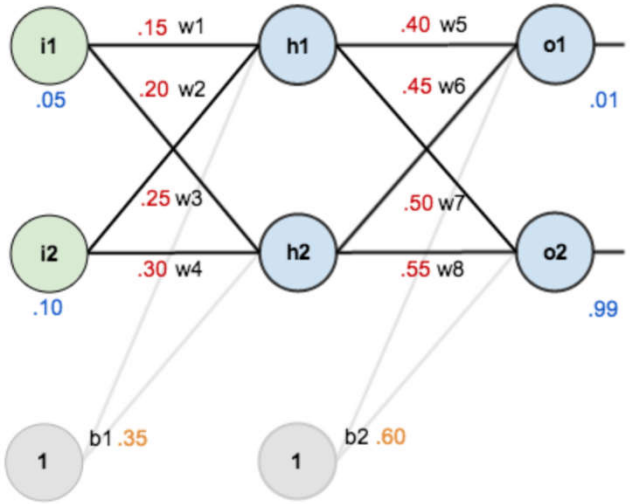
$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$$

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-1.105905967}} = 0.75136507$$

$$out_{o2} = 0.772928465$$

- 这样前向传播的过程就结束了，我们得到输出值为[0.75136079 , 0.772928465]，与实际值[0.01 , 0.99]相差还很远。

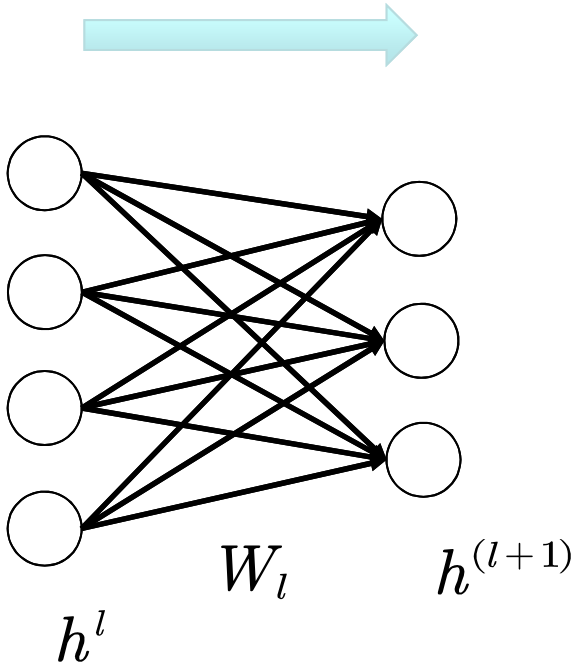


人工神经网络的基本原理

- 每层的前向计算可以简述为：

$$
\begin{aligned}
h^{(l+1)} &= \varphi\left(f_{W_l}(h^l)\right) \\
&= \varphi(W_l h^l + \beta) \\
&= \varphi(W_l' h^l)
\end{aligned}
$$

- 每层输入向量 h，和权值矩阵W，做矩阵乘法Wh，再逐元素非线性做非线性变换



人工神经网络的基本原理

- 矩阵乘法的意义？

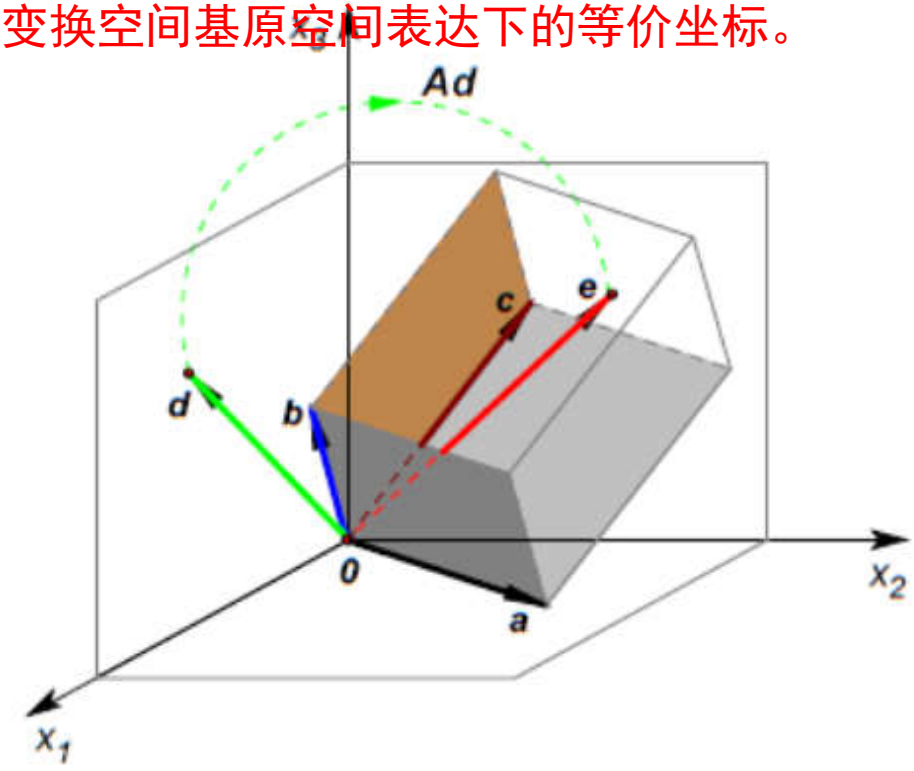
$$\mathbf{A}_1 \cdot \mathbf{d}_1 = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix} \begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix}$$

$$= d_1 \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} + d_2 \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} + d_3 \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix}$$

$$= \begin{pmatrix} e_1 \\ e_2 \\ e_3 \end{pmatrix}$$

3×3 矩阵 **A** 把一个三维向量 **d** 映射到一个三维向量 **e**。

待变换特征：d，指示了用变换空间基去映射该特征的方式
变换后的特征：e，新空间下的坐标
原空间的坐标d，变换后坐标e关系：d是e用变换空间基原空间表达下的等价坐标。



人工神经网络的基本原理

- 矩阵乘法的意义？

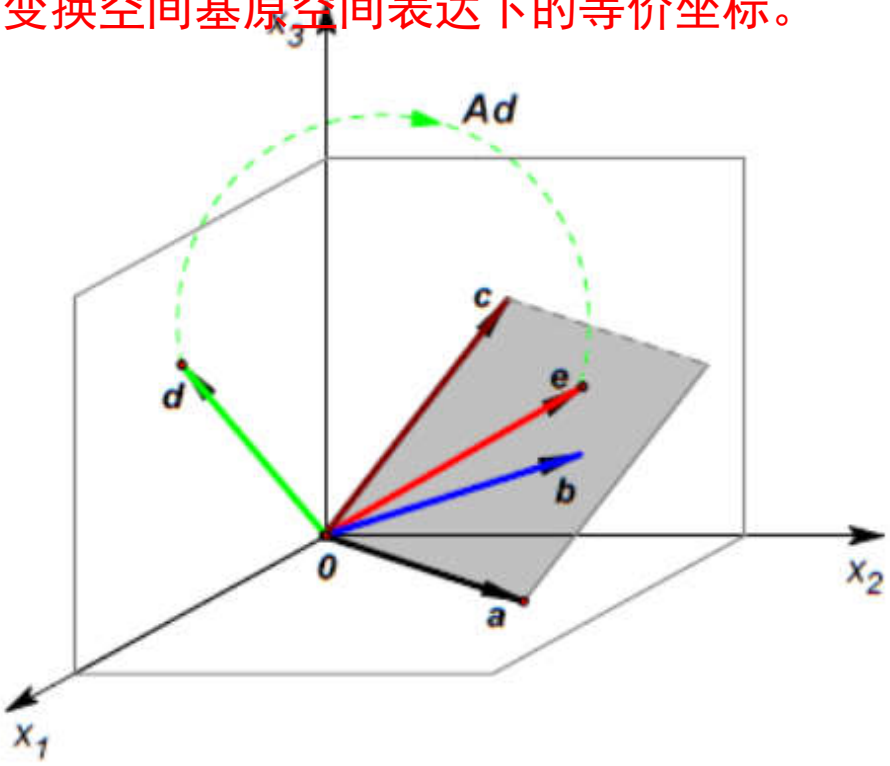
$$\mathbf{A}_2 \cdot \mathbf{d}_2 = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{bmatrix} \begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix}$$

$$= d_1 \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} + d_2 \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} + d_3 \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}$$

$$= \begin{pmatrix} e_1 \\ e_2 \end{pmatrix}$$

2×3 矩阵 **A** 把一个三维向量 **d** 映射到一个二维向量 **e**。

待变换特征：d，指示了用变换空间基去映射该特征的方式
变换后的特征：e，新空间下的坐标
原空间的坐标d，变换后坐标e关系：d是e用变换空间基原空间表达下的等价坐标。



人工神经网络的基本原理

- 矩阵乘法的意义？

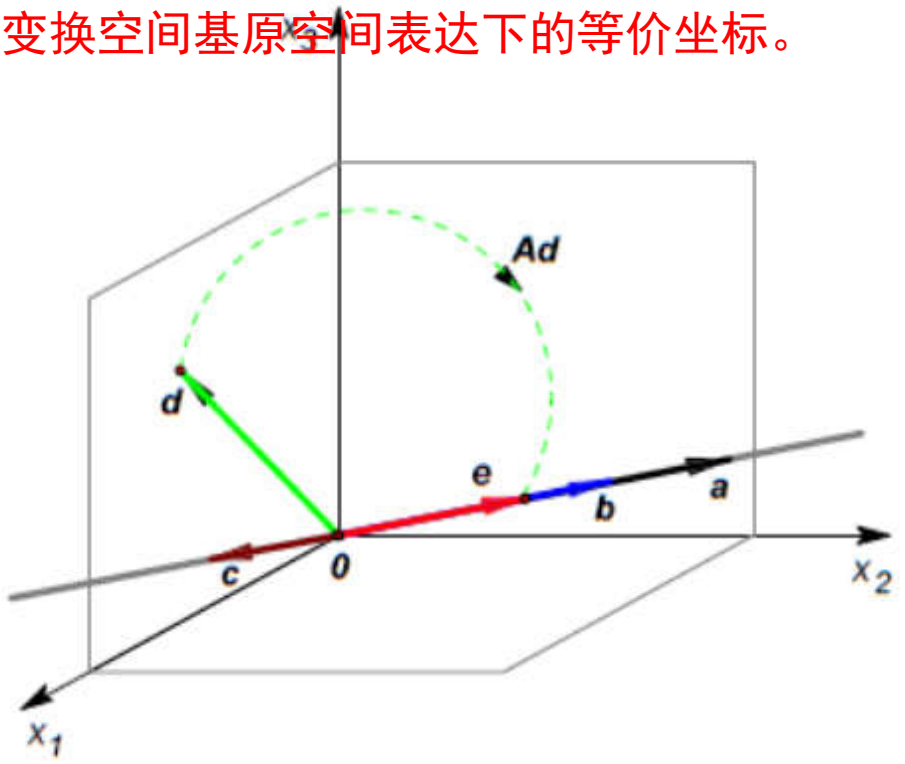
$$\mathbf{A}_3 \cdot \mathbf{d}_3 = \begin{bmatrix} a_1 & b_1 & c_1 \end{bmatrix} \begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix}$$

$$= d_1 a_1 + d_2 b_1 + d_3 c_1$$

$$= e$$

1×3 矩阵 **A** 把一个三维向量 **d** 映射到一个一维向量 **e**（或实数 e）。

待变换特征：d，指示了用变换空间基去映射该特征的方式
变换后的特征：e，新空间下的坐标
原空间的坐标d，变换后坐标e关系：d是e用变换空间基原空间表达下的等价坐标。

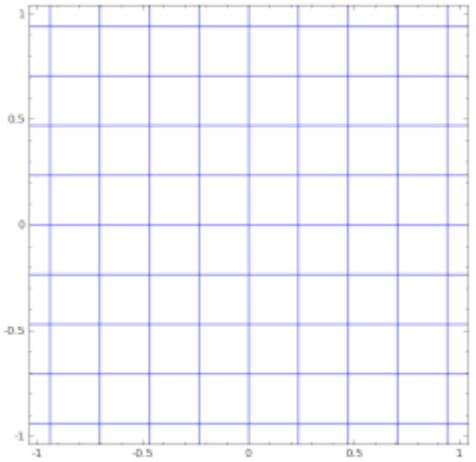


人工神经网络的基本原理

- 矩阵乘法的意义？
- 变换、映射
- 把特征从一个空间的表达变换为另一种空间的表达
- 待变换特征：d，指示了用变换空间基去映射该特征的方式
- 变换后的特征：e，新空间下的坐标
- 原空间的坐标d，变换后坐标e关系：d是e用变换空间基原空间表达下的等价坐标。
- 线性空间：一组基可以定义
- 矩阵乘法特点：只能进行线性映射
- 线性空间特点：平面、直线

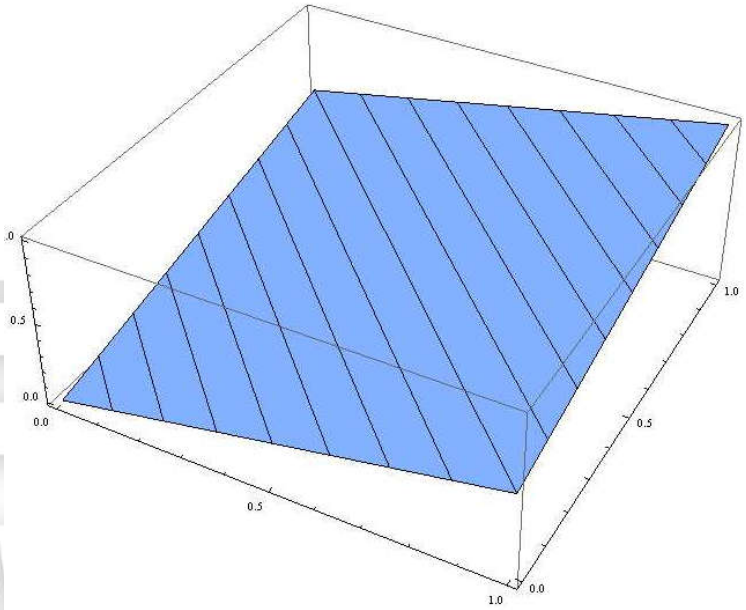
人工神经网络的基本原理

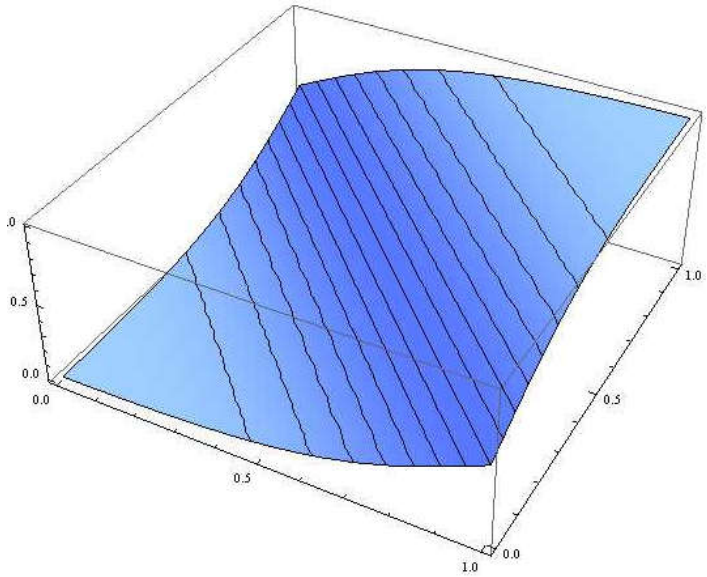
- 线性网络超平面分割
  - 神经网络由一层一层构建，每层都在做什么？
  - 数学理解：$\vec{y} = a(W \cdot \vec{x} + b)$，其中$\vec{x}$是输入向量，$\vec{y}$是输出向量，b是偏移量，W是权重矩阵，a()是激活函数。通过如下5种对输入空间（输入向量的集合）的操作，完成 **输入空间 ——> 输出空间** 的变换 (矩阵的行空间到列空间)。
  - Wx
    - 1. 升维/降维 （矩阵乘输入输出维度）
    - 2. 放大/缩小
    - 3. 旋转
    - 4. 平移
  - a(.)
    - 5. “弯曲”



神经网络中的主要功能结构

- 例：非线性激活函数带来的映射空间弯曲
  - $z = 0.5x + 0.5y + 0$ 只是矩阵乘法，变换依然是平面
  - $z = \mathrm{sig}moid(0.5x + 0.5y + 0)$ 非线性变换后，空间弯曲





神经网络中的主要功能结构

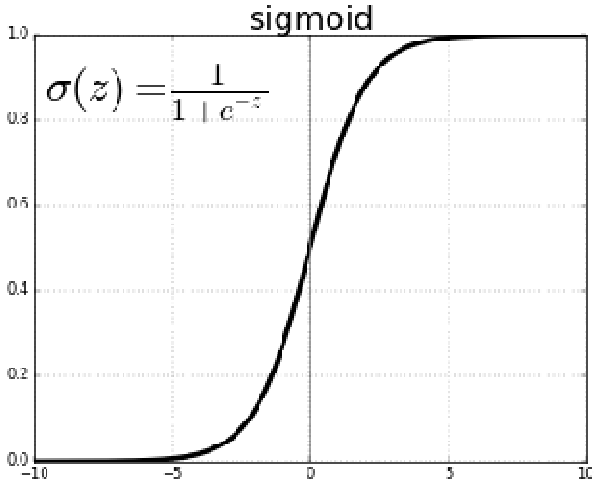
- 非线性映射(激活函数)
  - Sigmoid（其中x为激活函数输入）:

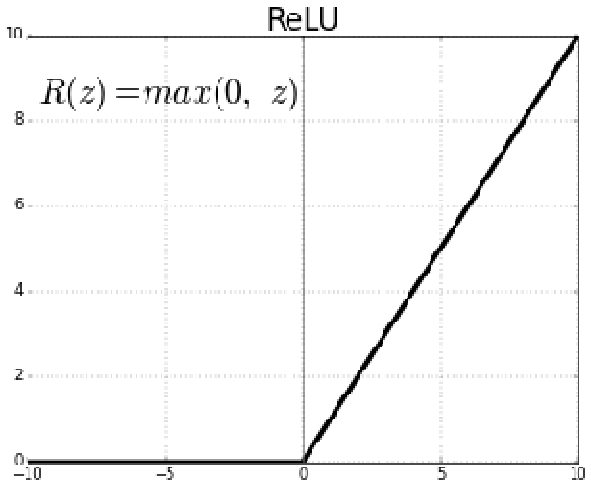
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

    - 避免丢失信息
    - 梯度具有连续性
  - ReLU:

$$R(x) = \max(0, x)$$

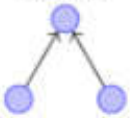
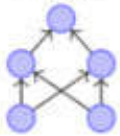
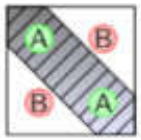
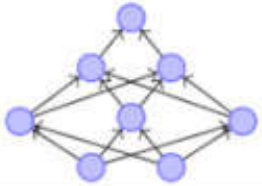
    - 收敛速度更快
    - 有助于防止过拟合
  - Tensorflow函数定义：
    - （1）tf.nn.relu(features, name='None'), 其中features为输入特征
    - （2）tf.nn.sigmoid(features, name='None'), 其中features为输入特征





人工神经网络的基本原理

- 为什么要逐层非线性映射？
- 大多数问题是非线性的
- 逐层非线性可以组合更加复杂的非线性空间结构

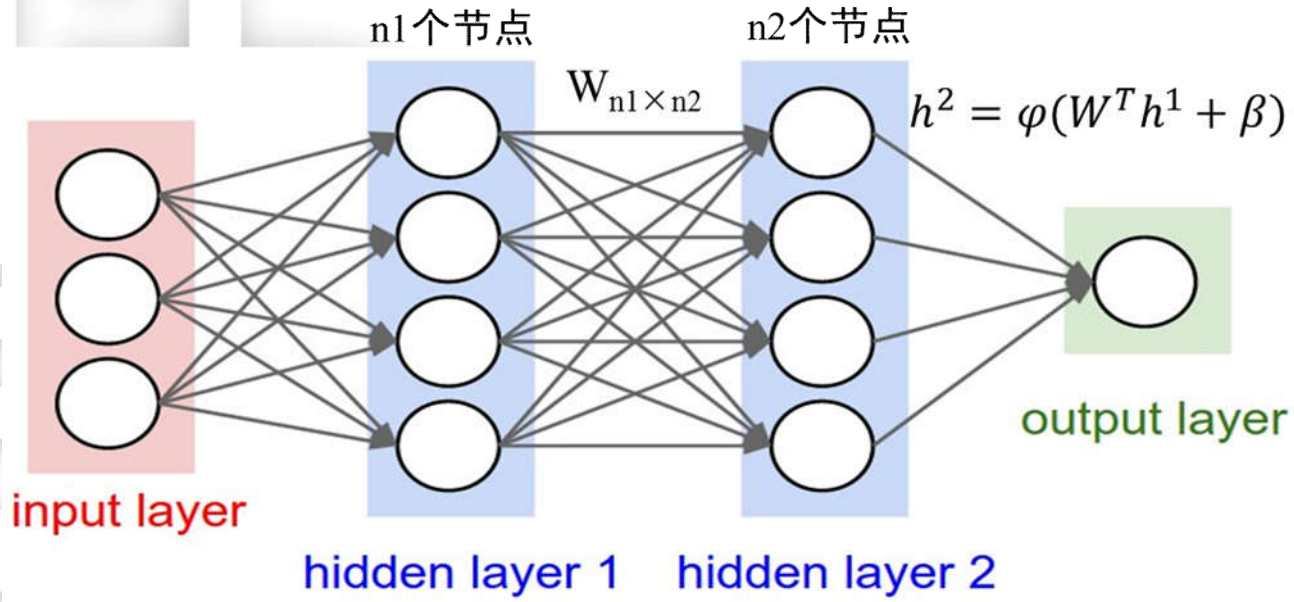
结构	决策区域类型	区域形状	异或问题
无隐层 	由一超平面分成两个		
单隐层 	开凸区域或闭凸区域		
双隐层 	任意形状（其复杂度由单元数目确定）		

人工神经网络的基本原理

- 为什么要逐层非线性映射？

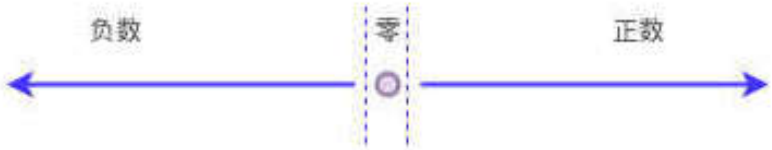
$$h^l = f_{W_l}(h^l) = f_{W_l}\left(f_{W_{l-1}}(h^{(l-1)})\right) = \ldots = W_l' \ldots W_1' W_0' h^0 = W h^0$$

- 即是做很多层，也只等价于一个线性映射

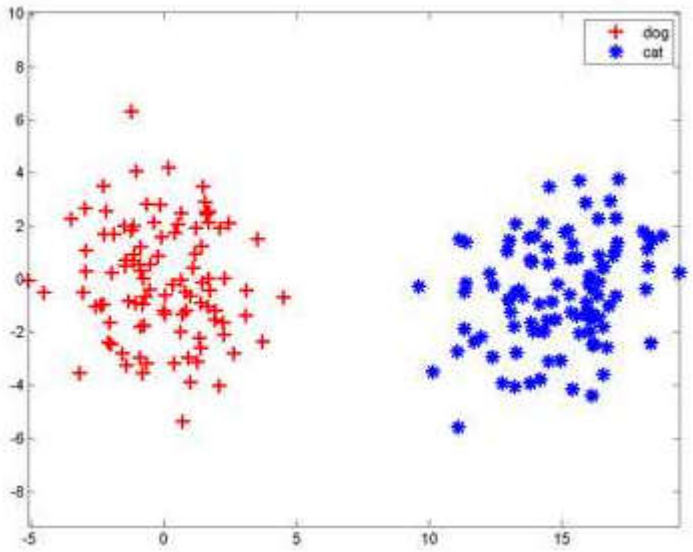


人工神经网络的基本原理

- 线性网络超平面分割
  - 每一层的这种行为是如何完成识别任务的？
  - 一维情景：

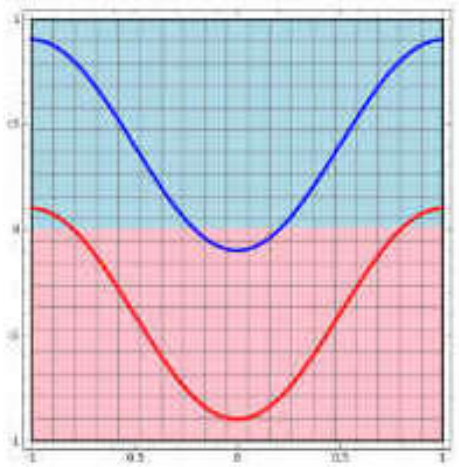


  - 二维情景：



人工神经网络的基本原理

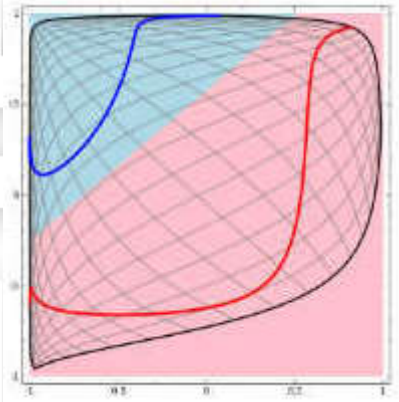
- 线性网络超平面分割
  - 二维情景：平面的四个象限也是线性可分。但下图的红蓝两条线就无法找到一超平面去分割。



  - 神经网络的解决方法依旧是转换到另外一个空间下，用的是所说的**5种空间变换操作**。

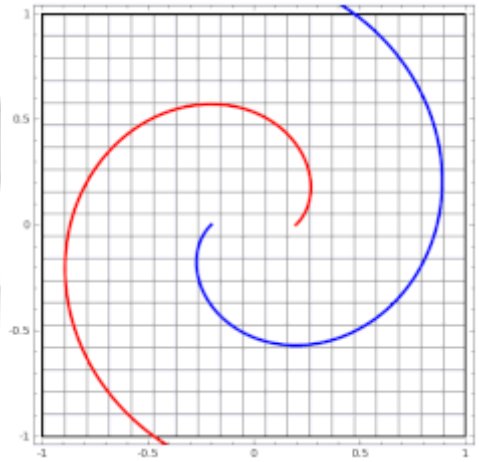
人工神经网络的基本原理

- 线性网络超平面分割
  - 比如下图就是经过放大、平移、旋转、扭曲原二维空间后，在三维空间下就可以成功找到一个超平面分割红蓝两线 (同SVM的思路一样)。



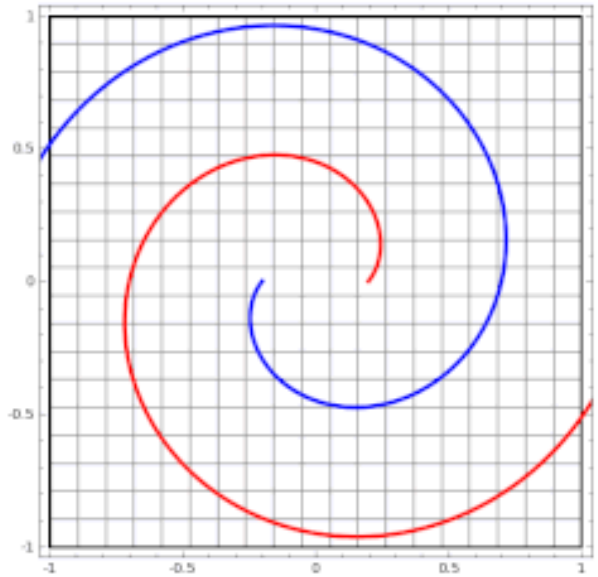
人工神经网络的基本原理

- 线性网络超平面分割
  - 上面是一层神经网络可以做到的，如果把y当做新的输入再次用这5种操作进行第二遍空间变换的话，网络也就变为了二层。最终输出是$\vec{y} = a_2\left(W_2 a_1(W_1 \cdot \vec{x} + b_1)\right) + b_2)$
  - 设想网络拥有很多层时，对原始输入空间的“扭曲力”会大幅增加，如下图，最终我们可以轻松找到一个超平面分割空间。



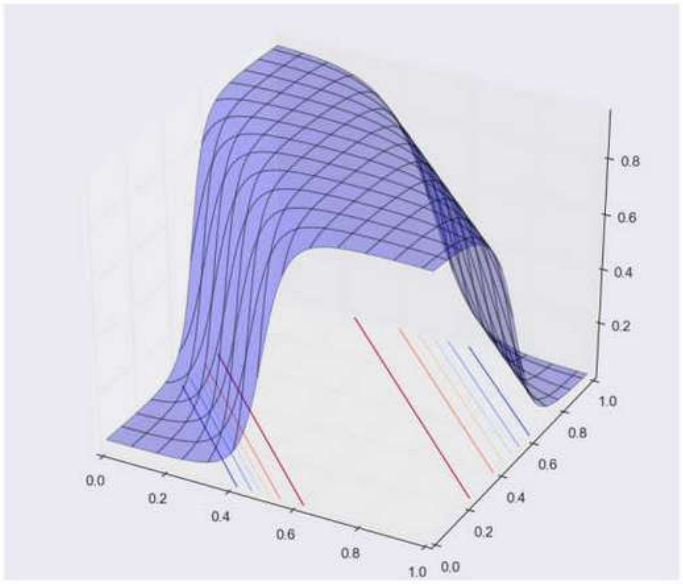
人工神经网络的基本原理

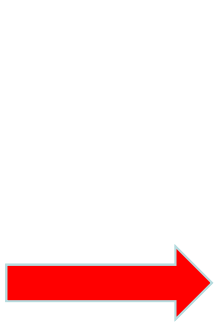
- 线性网络超平面分割
  - 当然也有如下图失败的时候，关键在于“如何扭曲空间”。所谓监督学习就是给予神经网络网络大量的训练例子，让网络从训练例子中学会如何变换空间。每一层的权重W就**控制着如何变换空间**，我们最终需要的也就是训练好的神经网络的所有层的权重矩阵。

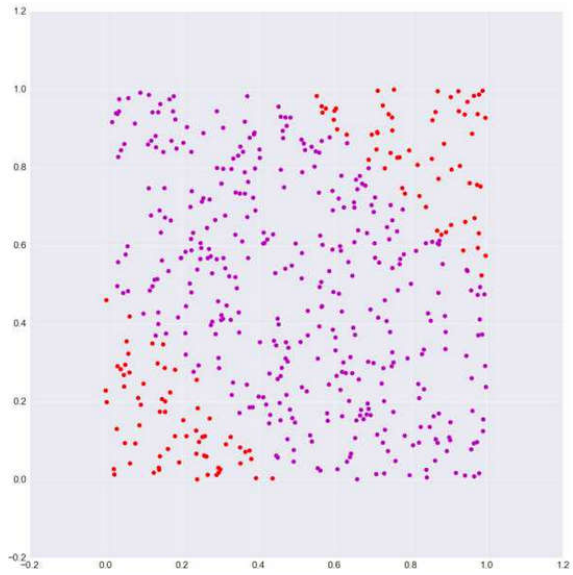


神经网络中的主要功能结构

- 举例：多层感知机解决XOR（异或）问题
  - 对于XOR这个线性不可分问题，无论何种平面都无法把两类点分开
  - 是否可以找到某种非线性映射将其分开？

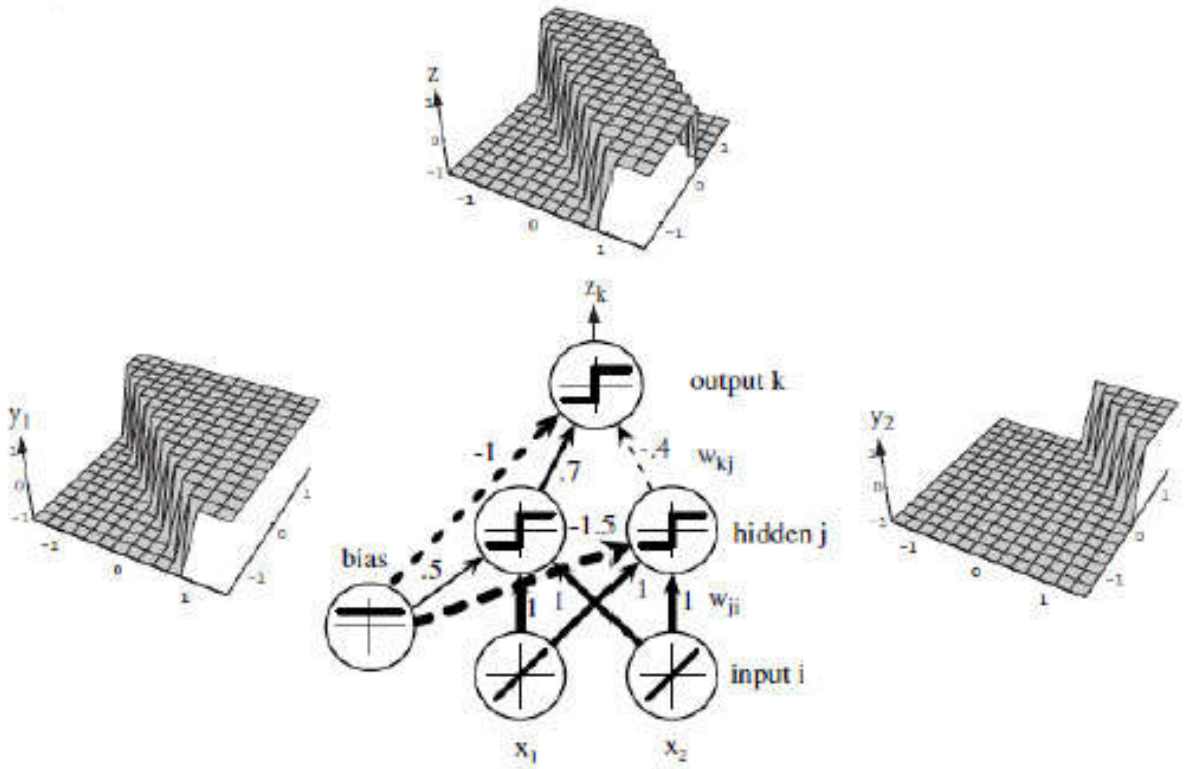






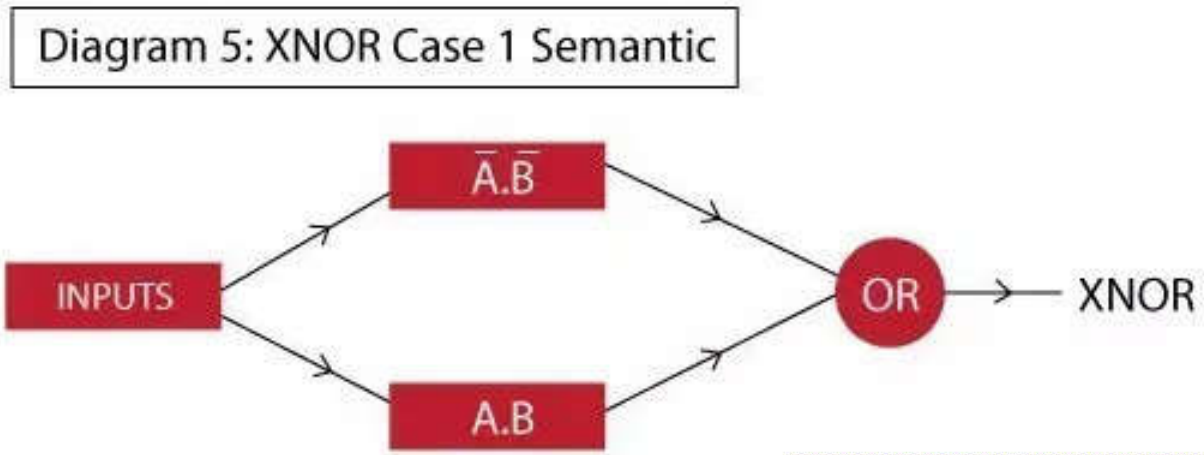
神经网络中的主要功能结构

- 举例：多层感知机解决XOR（异或）问题
  - 这种复杂的非线性结构，可以通过多层非线性组合完成



神经网络中的主要功能结构

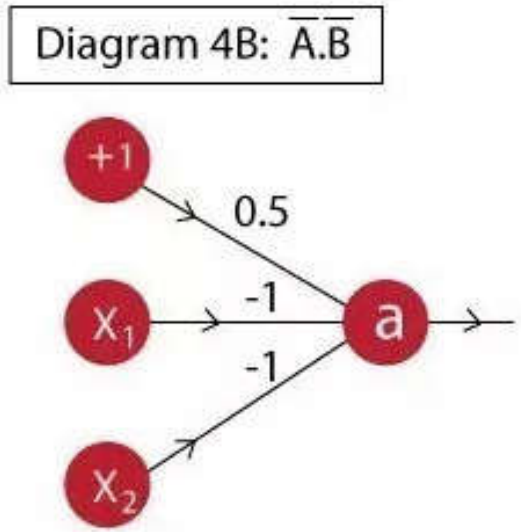
- 举例：多层感知机解决XOR（异或）问题
  - 这种复杂的非线性结构，可以通过多层非线性组合完成

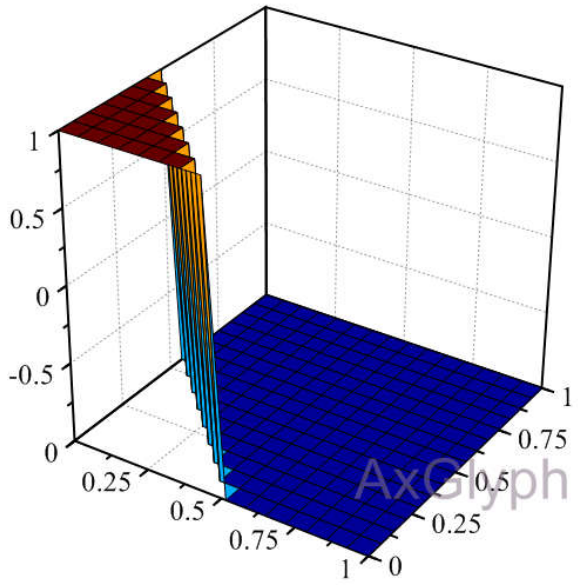


X1	X2	X1 XNOR X2
0	0	1
0	1	0
1	0	0
1	1	1

神经网络中的主要功能结构

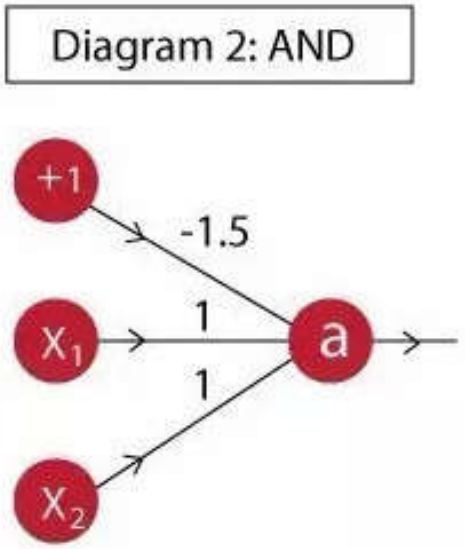
- 举例：多层感知机解决XOR（异或）问题
  - 这种复杂的非线性结构，可以通过多层非线性组合完成

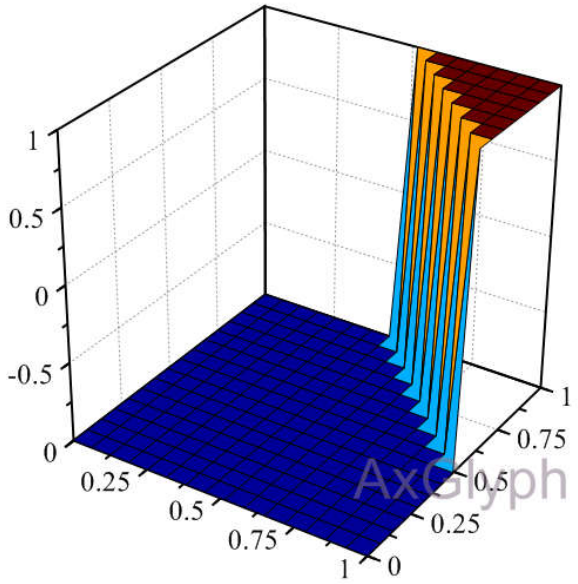




神经网络中的主要功能结构

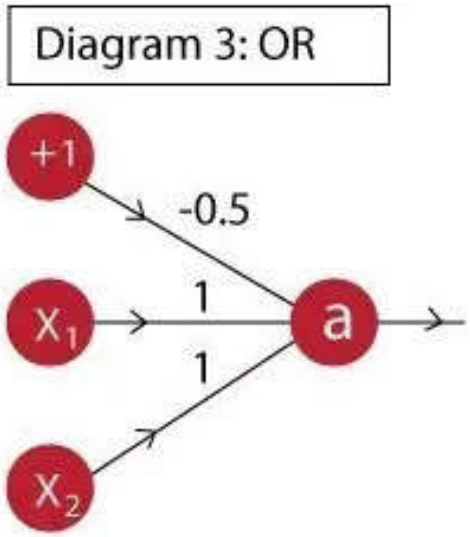
- 举例：多层感知机解决XOR（异或）问题
  - 这种复杂的非线性结构，可以通过多层非线性组合完成

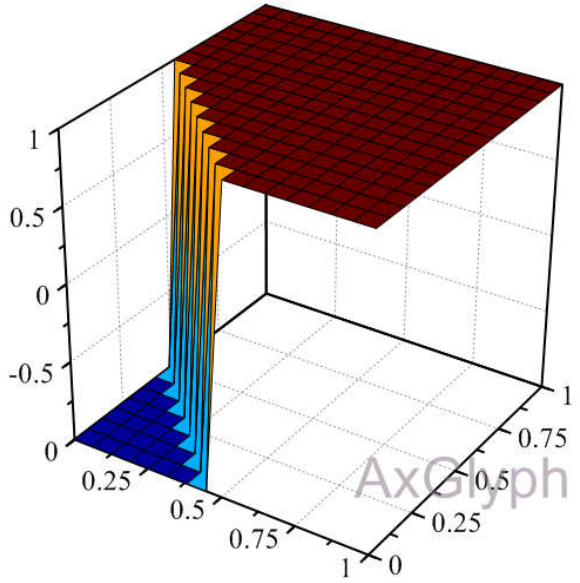




神经网络中的主要功能结构

- 举例：多层感知机解决XOR（异或）问题
  - 这种复杂的非线性结构，可以通过多层非线性组合完成

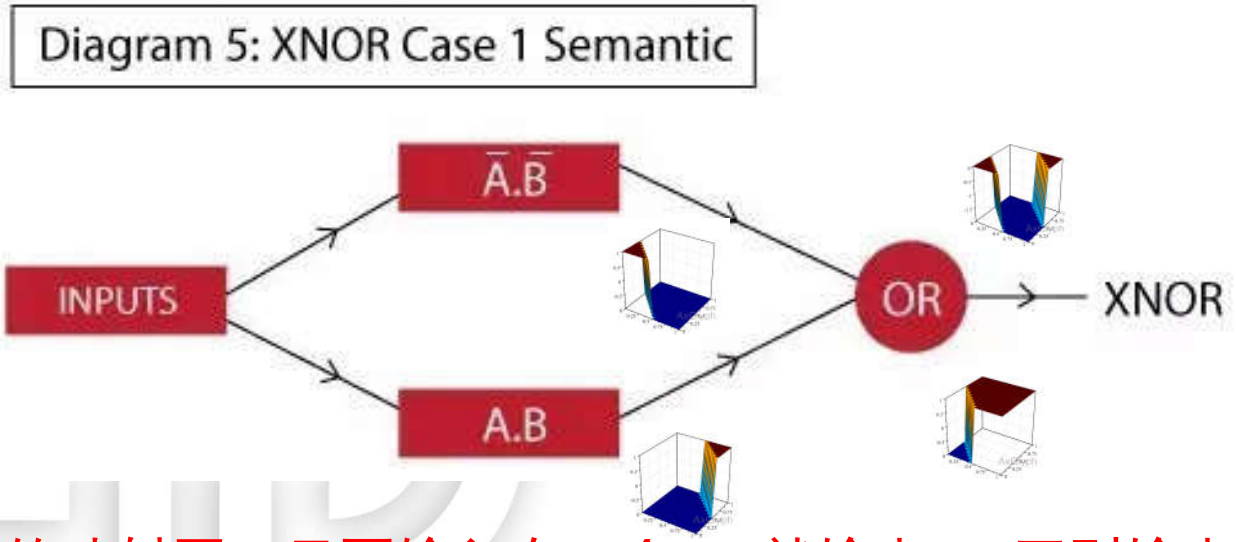




  - Or的映射图：只要输入有一个1，就输出1；否则输出0。那么就看前面输出的那些部分输出是1。

神经网络中的主要功能结构

- 举例：多层感知机解决XOR（异或）问题
  - 这种复杂的非线性结构，可以通过多层非线性组合完成



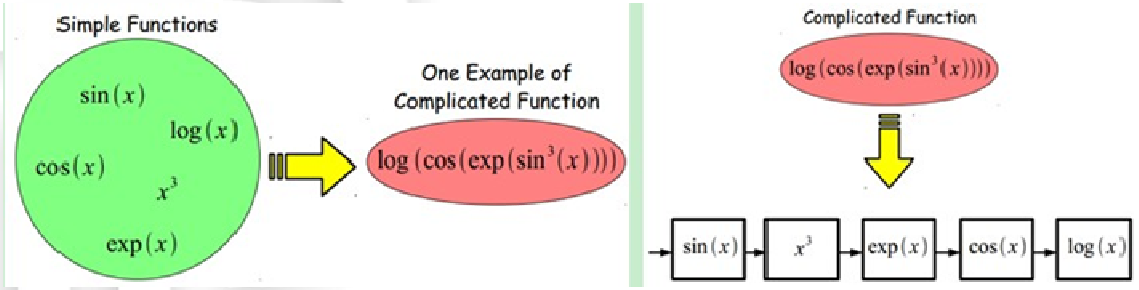
  - Or的映射图：只要输入有一个1，就输出1；否则输出0。那么就看前面输出的哪些部分输出是1。也就是之前两个图输出是1的部分都是1，其他部分是0。所以结果是两个图的一个“并”

神经网络中的主要功能结构

- XOR感知机不能一层解决，是不是设计的问题？
- 例如：用比较复杂的非线性映射函数

$$sign(x) = \begin{cases} 1, x > 0.5 \\ 0, 0.5 \geqslant x \geqslant -0.5 \\ -1, x < -0.5 \end{cases}$$

- 那里不好？



$$F(x) = \log(\cos(\exp(\sin^3(x)))) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n + O(x^{n+1})$$

- 我们希望用简单的，去组合复杂的，而不是一把设计复杂的

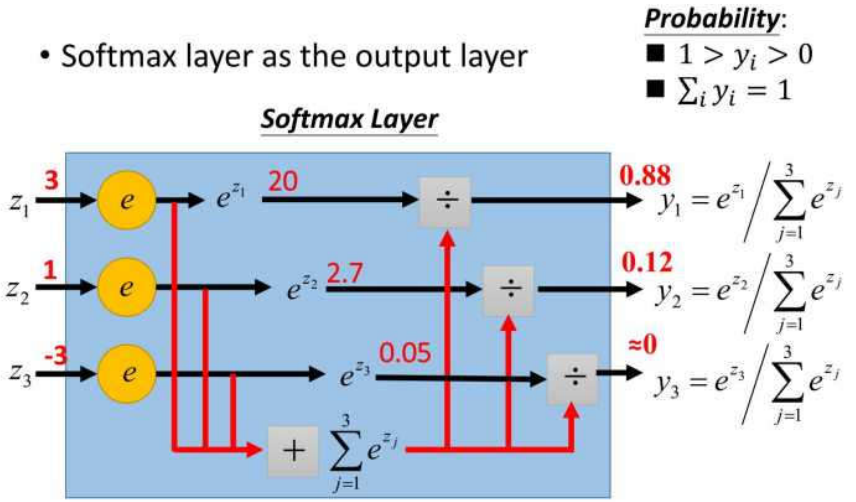
Softmax与交叉熵

- 多分类问题：怎样衡量样本预测为某类的可能性？或从另一角度，模型将样本分为某类的概率？
- 例：逻辑回归，实际上用sigmod 映射（0,1）区间，表达类型概率
- 多分类情况：常采用softmax表达样本属于某类概率

$$y_i = \frac{e^{a_i}}{\sum_{k=1}^{C} e^{a_k}} \quad \forall i \in 1...C \quad \sum_{i=1}^{C} y_i = 1$$

网络输出结点数为C，代表C个类别，

其中a_i为第i个网络输出结点输出值



Softmax与交叉熵

- 怎样对类型预测概率进行优化？
- 回想：回归问题，目标是让预测值与真实值尽量接近
- 例：线性回归采用平方差作为优化目标

$$h_w(x) = \sum_{i=1}^{n+1} w_i \cdot \boldsymbol{x_i}$$

$$l(w) = -\frac{1}{2}\sum_{i \in D}\left(y^{(i)} - h_w\left(x^{(i)}\right)\right)^2$$

- 概率的优化目标？或者说，我们期望的类型预测概率应该有什么特点？

Softmax与交叉熵

- 1. 最大似然估计：Maximum likelihood estimation
- 最大似然思想：既然都看见了，说明概率挺大的哦

- 术语：当从模型总体随机抽取n组样本观测值后，最合理的参数估计量应该使得从模型中抽取该n组样本观测值的概率最大

- 对于输入的X，其对应的类标签为t($t=(t_1,t_2,\ldots,t_C)$) t_i为第i类标签(t_i=0 or 1)，对于单个样本我们的目标是优化参数w使得p(t|x)最大。

Softmax与交叉熵

- 求一个样本的样本的概率，其类型的出现事件互斥，t(t=(t_1,t_2,…,t_C)) t_i为第i类标签(t_i=0 or 1)

$$p(t|x) = \prod_{i=1}^{C} P\left(t_i|x\right)^{t_i} = \prod_{i=1}^{C} y_i^{t_i}$$

- n个样本假设独立，求其联合概率

$$p(T|X) = \prod_{j=1}^{n} P\left(t|x_j\right)$$

- 对其取对数，单个样本问题转换为最小化对数似然函数：

$$-log\, p\left(t|x\right) = -log \prod_{i=1}^{C} y_i^{t_i} = -\sum_{i=i}^{C} t_i log\left(y_i\right)$$

- n个样本：

$$L = -\sum_{k=1}^{n}\sum_{i=1}^{C} t_{ki} log\left(y_{ki}\right)$$

Softmax与交叉熵

- 分布距离度量：数据真实分布p与估计分布q的差距
- KL散度（Kullback-Leibler Divergence）：又称为相对熵

$$
\begin{aligned}
D_{KL}(p||q) &= \sum_{x \in X} p(x) log \frac{p(x)}{q(x)} \\
&= \sum_{x \in X} p(x) log\, p(x) - \sum_{x \in X} p(x) log\, q(x) \\
&= -H(p) - \sum_{x \in X} p(x) log\, q(x)
\end{aligned}
$$

- 给定样本集上的交叉熵：

$$
CE(p,q) = -\sum_{x \in X} p(x) log\, q(x) = H(p) + D_{KL}(p||q)
$$

- 交叉熵和相对熵相差了H(p)，而当p已知的时候，H(p)是个常数，故交叉熵反映了估计分布q与真实分布p的相似程度

Softmax与交叉熵

- 多分类中，类标签可看作为分布：
- One-hot 编码方式：
- 例：5个类别的分类中，[0, 1, 0, 0, 0]表示该样本属于第二个类，其概率值为1
- 将样本的标签视作真实分布p，该分布中，ti=1当样本属于类型i。Softmax 预测概率为:

$$y = (y_1, y_2, \ldots, y_c) \qquad \sum_{i=1}^{C} y_i = 1$$

- 例：[0.1, 0.8, 0.05, 0.05, 0]
- 则对此样本，其交叉熵为：

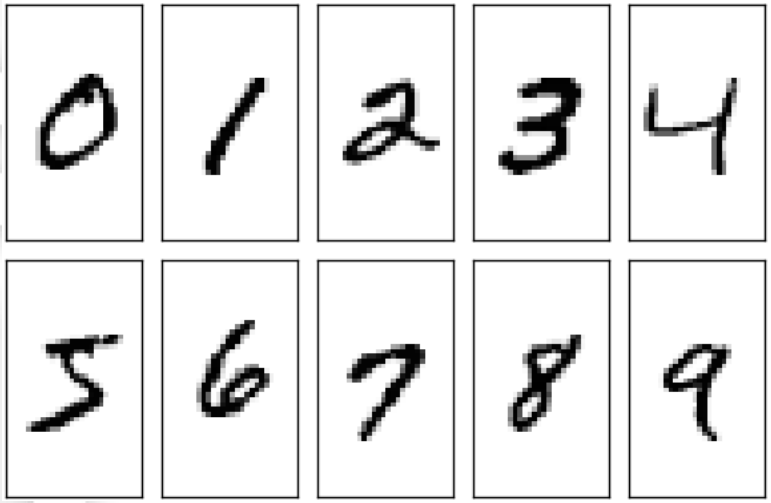
$$l_{CE} = -\sum_{i=1}^{C} t_i log(y_i)$$

- 最终对所有样本：我们的交叉熵优化目标为：

$$L = -\sum_{k=1}^{n}\sum_{i=1}^{C} t_{ki} log(y_{ki})$$

Tensorflow神经网络训练方法

- Yann Lecun, 1998: 手写字符识别
- 数据集：MNIST



Tensorflow神经网络训练方法

- Tensorflow中神经网络训练和测试通用框架：
  - 四个大阶段：
  - 1. 数据导入、格式化及预处理
  - 2. 定义所需的神经网络（构建计算图）
  - 3. 会话中循环执行训练网络
  - 4. 评估网络性能

Tensorflow神经网络训练方法

- 定义网络，构建计算图方法：
  - 针对网络中三种主要结构分别定义：
  - 1. 定义网络输入：占位符定义输入变量
  - 2. 逐层定义网络各个功能层：
    - 确定每层的网络参数变量
    - 定义功能层相关操作
  - 3. 定义网络输出：设计和预测类型匹配的输出函数

Tensorflow神经网络训练方法

- 训练方法：
  - 1. 定义loss函数
  - 2. 选择反向传播算法
  - 3. 启动模型，初始化变量
  - 4. 开始循环训练

Tensorflow神经网络训练方法

- 评估网络方法：
  - 运行会话，得到预测标签与真实标签对比得到准确率

Tensorflow神经网络训练方法

- 实现一个单层的Softmax Regression 手写字符识别：
- 第一步：数据导入、格式化及预处理

首先导入用于获取数据的Python代码文件

```
import tensorflow.examples.tutorials.mnist.input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

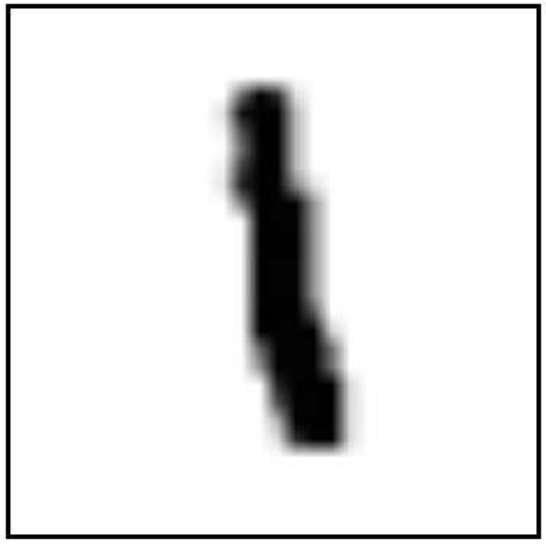
下载下来的数据集被分成两部分：60000行的训练数据集（mnist.train）和10000行的测试数据集（mnist.test）。这样的切分很重要，在机器学习模型设计时必须有一个单独的测试数据集不用于训练而是用来评估这个模型的性能，从而更加容易把设计的模型推广到其他数据集上（泛化）。

正如前面提到的一样，每一个MNIST数据单元有两部分组成：一张包含手写数字的图片和一个对应的标签。我们把这些图片设为“xs”，把这些标签设为“ys”。训练数据集和测试数据集都包含xs和ys，比如训练数据集的图片是 mnist.train.images，训练数据集的标签是 mnist.train.labels。

Tensorflow神经网络训练方法

- **第一步：数据导入、格式化及预处理**

每一张图片包含28X28个像素点。我们可以用一个数字数组来表示这张图片：



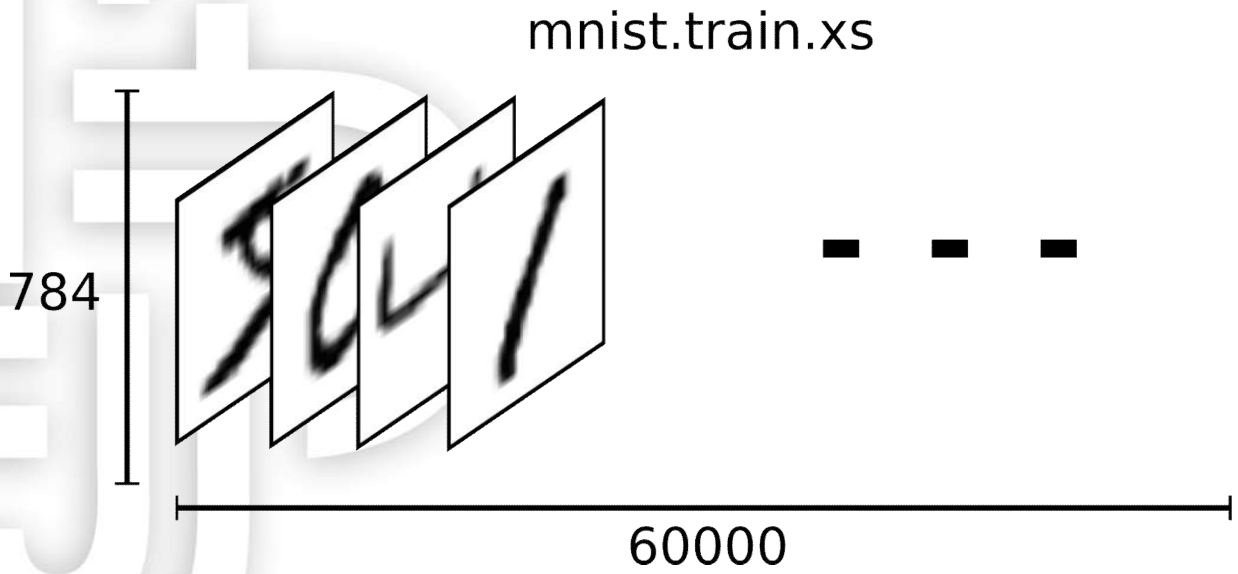
≈

$$\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .6 & .8 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .5 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .7 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & .9 & 1 & .1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & .3 & 1 & .1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

Tensorflow神经网络训练方法

- **第一步：数据导入、格式化及预处理**

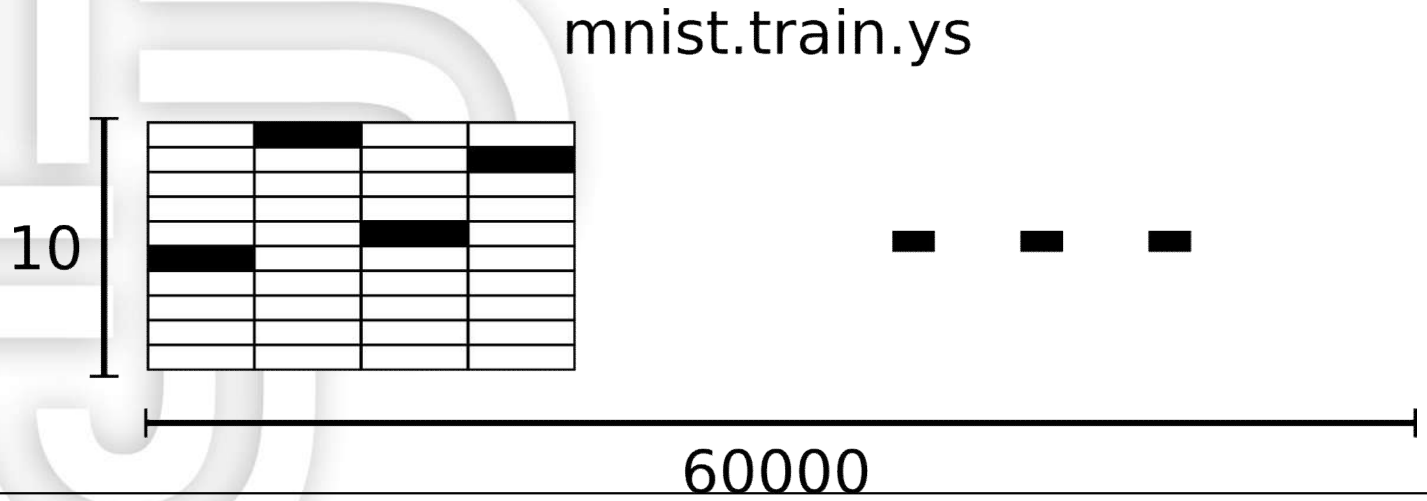
因此，在MNIST训练数据集中，mnist.train.images 是一个形状为 [60000, 784] 的张量，第一个维度数字用来索引图片，第二个维度数字用来索引每张图片中的像素点。在此张量里的每一个元素，都表示某张图片里的某个像素的强度值，值介于0和1之间。



Tensorflow神经网络训练方法

- 第一步：数据导入、格式化及预处理

相对应的MNIST数据集的标签是介于0到9的数字，用来描述给定图片里表示的数字。为了用于这个课程，我们使标签数据是“one-hot vectors”。一个one-hot向量除了某一位的数字是1以外其余各维度数字都是0。所以在此课程中，数字n将表示成一个只有在第n维度（从0开始）数字为1的10维向量。比如，标签0将表示成([1,0,0,0,0,0,0,0,0,0])。因此，mnist.train.labels 是一个 [60000, 10] 的数字矩阵。

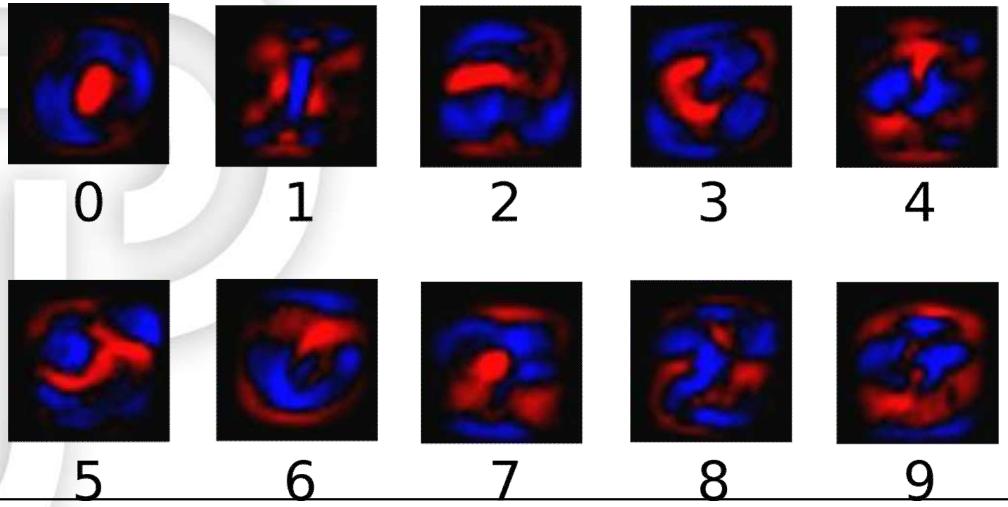


Tensorflow神经网络训练方法

- 第二步：定义所需的神经网络（构建计算图）

为了得到一张给定图片属于某个特定数字类的证据（evidence），我们对图片像素值进行加权求和。如果这个像素具有很强的证据说明这张图片不属于该类，那么相应的权值为负数，相反如果这个像素拥有有利的证据支持这张图片属于这个类，那么权值是正数。

下面的图片显示了一个模型学习到的图片上每个像素对于特定数字类的权值。红色代表负数权值，蓝色代表正数权值。



Tensorflow神经网络训练方法

- 第二步：定义所需的神经网络（构建计算图）

我们也需要加入一个额外的偏置量（bias），因为输入往往会带有一些无关的干扰量。因此对于给定的输入图片 x 它代表的是数字 i 的证据可以表示为

$$\text{evidence}_i = \sum_j W_{i,j} x_j + b_i$$

其中W_i代表权重，b_i代表数字 i 类的偏置量，j 代表给定图片 x 的像素索引用于像素求和。

- 第二步：定义所需的神经网络（构建计算图）
  - 然后用softmax函数可以把这些证据转换成概率 y：

$$y = \text{softmax}(\text{evidence})$$

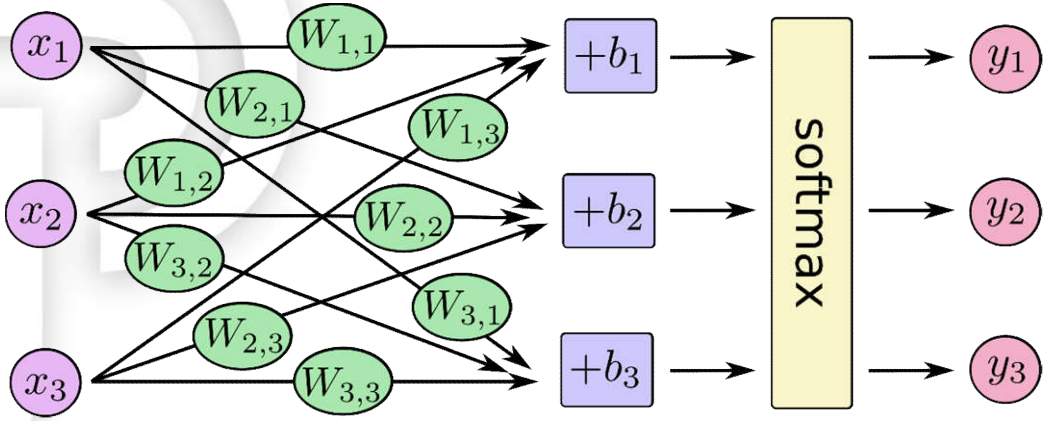
  - 这里的softmax可以看成是一个激励（activation）函数或者链接（link）函数，把我们定义的线性函数的输出转换成我们想要的格式，也就是关于10个数字类的概率分布。因此，给定一张图片，它对于每一个数字的吻合度可以被softmax函数转换成为一个概率值。softmax函数可以定义为：

$$\text{softmax}(x) = \text{normalize}(\exp(x))$$

- 第二步：定义所需的神经网络（构建计算图）
  - 展开等式右边的子式，可以得到：

$$\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

  - 对于softmax回归模型可以用下面的图解释，对于输入的xs加权求和，再分别加上一个偏置量，最后再输入到softmax函数中：



- 第二步：定义所需的神经网络（构建计算图）
  - 如果把它写成一个等式，我们可以得到：

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{matrix} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3 \end{matrix} \right)$$

  - 我们也可以用向量表示这个计算过程：用矩阵乘法和向量相加。这有助于提高计算效率。

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

  - 更进一步，可以写成更加紧凑的方式：

$$y = \text{softmax}(Wx + b)$$

Tensorflow神经网络训练方法

- 第二步：定义所需的神经网络（构建计算图）
  - Tensorflow不单独地运行单一的复杂计算，而是让我们可以先用图描述一系列可交互的计算操作，然后全部一起在Python之外运行。（这样类似的运行方式，可以在不少的机器学习库中看到。）
  - 使用TensorFlow之前，首先导入它：

```
import tensorflow as tf
```

  - 我们通过操作符号变量来描述这些可交互的操作单元，可以用下面的方式创建一个（（1）构建输入层）：

```
x = tf.placeholder(tf.float32, [None, 784])
```

Tensorflow神经网络训练方法

- 第二步：定义所需的神经网络（构建计算图）
  - 我们的模型也需要权重值和偏置量（（2）构建中间网络），当然我们可以把它们当做是另外的输入（使用占位符），但TensorFlow有一个更好的方法来表示它们：Variable 。 一个Variable代表一个可修改的张量，存在在TensorFlow的用于描述交互性操作的图中。它们可以用于计算输入值，也可以在计算中被修改。对于各种机器学习应用，一般都会有模型参数，可以用Variable表示。

```
W = tf.Variable(tf.zeros([784,10]))
b = tf.Variable(tf.zeros([10]))
```

Tensorflow神经网络训练方法

- 第二步：定义所需的神经网络（构建计算图）
  - 现在，我们可以定义最后一层输出（（3）构建输出层），只需要一行代码确定了我们完整的模型

```
y = tf.nn.softmax(tf.matmul(x,W) + b)
```

我们用tf.matmul(X，W)表示x乘以W，对应之前等式里面的Wx，这里x是一个2维张量拥有多个输入。然后再加上b，把和输入到tf.nn.softmax函数里面。

Tensorflow神经网络训练方法

- 第三步：会话中循环执行训练网络
  - （1）选择Loss函数，然后尽量最小化这个指标这里我们选择一个非常常见的，非常漂亮的成本函数“交叉熵”（cross-entropy）。
  - 它的定义如下：

$$H_{y'}(y) = -\sum_i y'_i \log(y_i)$$

  - y 是我们预测的概率分布, y' 是实际的分布（我们输入的one-hot vector)。
  - 为了计算交叉熵，我们首先需要添加一个新的占位符用于输入正确值：

```
y_ = tf.placeholder("float", [None,10])
```

Tensorflow神经网络训练方法

- 第三步：会话中循环执行训练网络
  - 然后我们可以用 $-\sum y' \log(y)$ 计算交叉熵：

```
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y),
reduction_indices=[1]))
```

Tensorflow神经网络训练方法

- 第三步：会话中循环执行训练网络
  - 用TensorFlow来训练它是非常容易的。因为TensorFlow拥有一张描述你各个计算单元的图，它可以自动地使用反向传播算法(backpropagation algorithm)来有效地确定你的变量是如何影响你想要最小化的那个loss值。然后，TensorFlow会用你选择的优化算法来不断地修改变量以降低loss。
  - （2）我们选择梯度下降算法优化网络

```
train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)
```

Tensorflow神经网络训练方法

- 第三步：会话中循环执行训练网络
  - 现在，我们已经设置好了我们的模型。在运行计算之前，我们需要添加一个操作来初始化我们创建的变量：

```
init = tf.initialize_all_variables()
```

现在我们可以在一个Session里面（3）启动我们的模型，并且初始化变量：

```
sess = tf.Session()
sess.run(init)
```

Tensorflow神经网络训练方法

- 第三步：会话中循环执行训练网络
  - 然后（4）开始训练模型，这里我们让模型循环训练1000次：

```
for i in range(1000):
  batch_xs, batch_ys = mnist.train.next_batch(100)
  sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
```

Tensorflow神经网络训练方法

- 第四步：评估网络性能
  - 首先让我们找出那些预测正确的标签。：

```
correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
```

  - 这行代码会给我们一组布尔值。为了确定正确预测项的比例，我们可以把布尔值转换成浮点数，然后取平均值。例如，[True, False, True, True] 会变成 [1,0,1,1]，取平均值后得到 0.75.

```
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
```

  - 最后，我们计算所学习到的模型在测试数据集上面的正确率。

```
print sess.run(accuracy, feed_dict={x: mnist.test.images, y_:
mnist.test.labels})
```