南大周志华《机器学习》课程笔记

> Introduction：最近自学机器学习课程，注意到了南京大学周志华老师的课程。我是在学堂在线平台观看的，注意到b站上也有相应视频，但b站上并未获得授权，随时有消失的可能。
>
> 周志华老师的网络教学视频中，与其西瓜书相比确实少了一些内容。但幸运的是，缺失的内容实际上对于初学者来说并不会产生太大影响。目前这一笔记也遵循视频内容，相比西瓜书中也会有一些缺失，敬请谅解。可能以后如果有机会和时间，我会再阅读周志华老师的书籍将缺失内容补全。
>
> 一切内容敬请关注我的个人Page页面。
>
> 全系列笔记请见：click here
> **About Me:**点击进入我的Personal Page

第六章 神经网络

神经网络模型

什么是神经网络

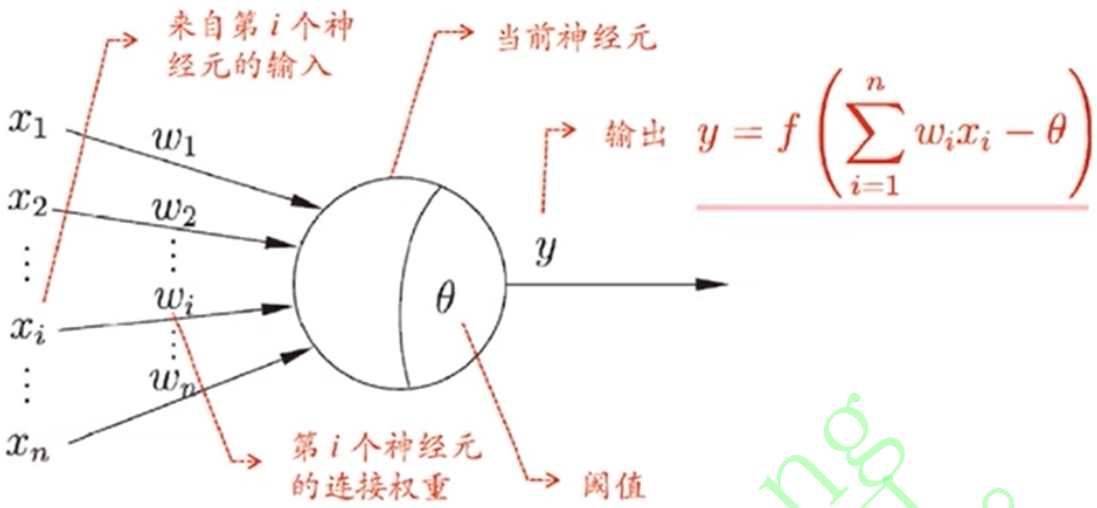
- "神经网络是由具有适应性的简单单元组成的广泛并行互连的网络，它的组织能够模拟生物神经系统对真实世界物体所作出的交互反应"

  [T. Kohonen, 1988，Neural Networks 创刊号]

- 神经网络是一个很大的学科领域，本课程仅讨论神经网络与机器学习的交集，即"神经网络学习"亦称"连接主义(connectionism)" 学习

"简单单元"——神经元模型

M-P 神经元模型 [McCulloch and Pitts, 1943]



神经网络学得的知识蕴含在连接权与阈值中

> 关于输出的理解：收到的来自其他神经元/细胞的信号，通过突触w进行放大，然后累加与阈值相比，如果大于阈值，就用函数f()进行处理，产生一个输出信号。

对于神经元要学习什么？与线性回归等一致，需要学习w和θ。线性模型是wx^Tx+b，b与这里的$-\theta$类似。可见，要学习的还是w和b，这与线性模型是一致的。实际上，对于每一个神经元，如果不考虑f()，那么其本身就是一个线性模型。

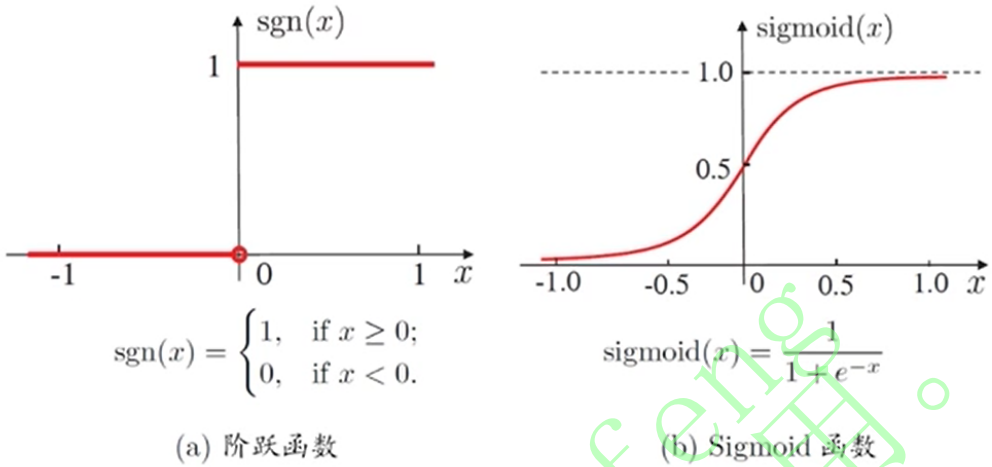
而现在，有了f()的存在，它就可以处理非线性的问题了，这个f()就被称为**激活函数**。

神经元的激活函数

激活函数分也称为响应函数、挤压函数(将输出挤压到0，1之间)

常用的Sigmoid函数为：$\frac{1}{1+e^{-x}}$，它有一个非常有趣的性质，即$f'(x)=f(x)\cdot(1-f(x))$。求导变成正类、负类的几率之乘积。

- 理想激活函数是阶跃函数, 0表示抑制神经元而1表示激活神经元
- 阶跃函数具有不连续、不光滑等不好的性质, 常用的是 Sigmoid 函数



$$\text{sgn}(x) = \begin{cases} 1, & \text{if } x \geq 0; \\ 0, & \text{if } x < 0. \end{cases} \qquad \text{sigmoid}(x) = \frac{1}{1+e^{-x}}$$

(a) 阶跃函数　　(b) Sigmoid 函数

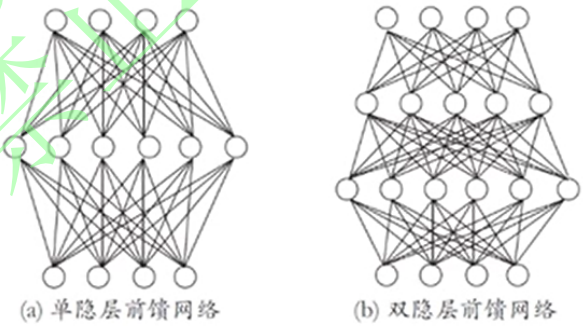
图 5.2　典型的神经元激活函数

多层前馈网络结构

多层网络：包含隐层的网络

前馈网络：神经元之间不存在同层连接也不存在跨层连接

隐层和输出层神经元亦称“功能单元”(Functional Unit)



(a) 单隐层前馈网络　　(b) 双隐层前馈网络

隐层也称为隐含层。

例如左侧的网络，我们可以称其为三层网络模型，但换个角度，只有隐含层和输出层才有激活函数处理，我们也可以称该网络为二层网络模型。为了避免歧义，便可以称呼其为单隐层网络模型，同理右侧网络为双隐层网络模型。

万有逼近能力

万有逼近性(Universicl Approximator)

多层前馈网络有强大的表示能力（“万有逼近性”）

仅需一个包含足够多神经元的隐层，多层前馈神经网络就能以任意精度逼近任意复杂度的连续函数　　[Hornik et al., 1989]

但是，如何设置隐层神经元数是未决问题(Open Problem). 实际常用“试错法”

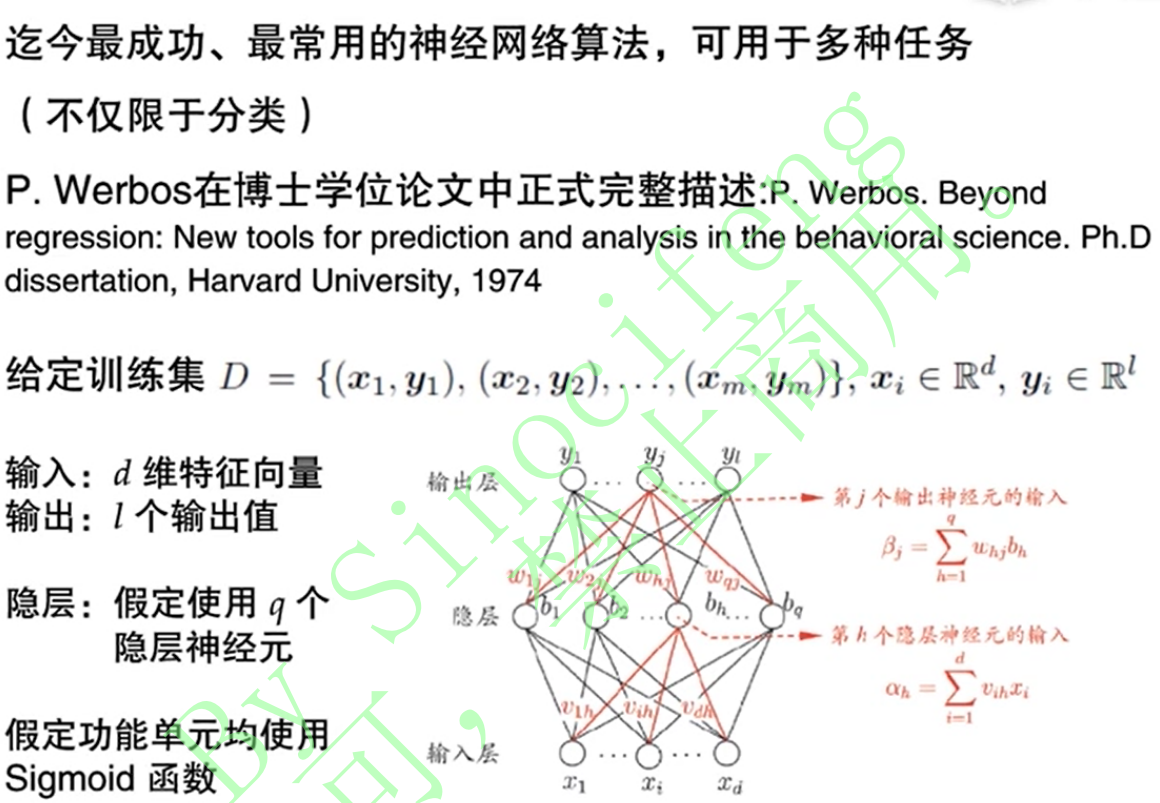
需要注意：有很多人将万有逼近性认为是神经网络所独有的一种优良属性，这一认知是错误的。

具有万有逼近性这一条件是我们可以将其作为机器学习中模型的一个前提，傅里叶变换、泰勒展开式等都具有万有逼近性。由于神经网络没有严格的数学证明过程，大家普遍怀疑其是否可以做到，因此提出其具有万有逼近性来证明其具有很好的逼近能力。但并不是说明因为神经网络具有万有逼近性就可以超越其他方法，因为最简单的傅里叶变换具有万有逼近性，而其如果作为一个模型是一个非常弱的模型。

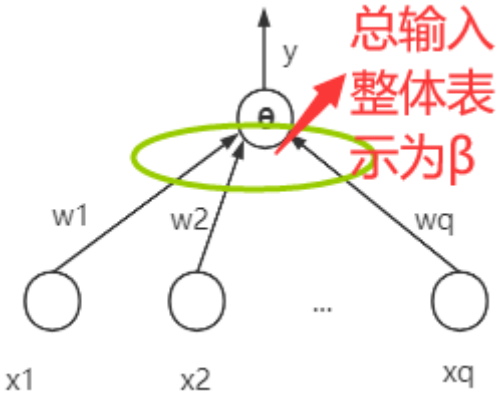
正是由于神经网络本身数学基础薄弱，在机器学习使用时不放心，所以需要有这么一个性质来告诉我们：解一定在其中，而我们需要做的是寻找这个解。

BP算法推导

BP(BackPropagation: 误差逆传播算法)



注：考虑输出层的神经元，β表示所有输入加和称为净输入，从而便于后面进行描述。



对于一个训练样例(x_k, y_k)，假定网络的实际输出为$\hat{y}_k = \{y_1^k, y_2^k, \ldots, y_l^k\}$。

可以得到，对于输出层神经元y_j，有：$\hat{y}_j^k = f(\beta - \theta)$，其中$\theta$为阈值。

那么网络在训练样例(x_k, y_k)上的均方误差为：$E_k = \frac{1}{2}\sum_{j=1}^{l}(\hat{y}_j^k - y_j^k)^2$

> 注意：在这个学习过程中，我们需要学习确定的参数有$(d + l + 1)q + l$个
>
> - 输入层d个神经元，隐层q个神经元，则输入层与隐层之间就包含dq个未知参数

> - 隐层q个神经元，输出层l 个神经元，则隐层与输出层之间就包含l q个未知参数
> - 隐层和输出层神经元分别含有q个和l 个阈值
>
> 将上面所有参数个数相加，即得需要学习确定的参数有$(d+l+1)q+l$个

BP算法是一个迭代学习算法，在迭代的每一轮中采用**广义感知学习规则** $v<-v+\triangle v$

为什么要每一次迭代过程中对量进行改变？一定是由于误差的存在，而存在误差要如何修改呢？常见的想法是求导，如果导数为0，且函数为凸，则极值点就是误差最小处。然而神经网络中，无法保证必然是凸的，也无法保证求导后必然可以等于0。但是，可以选择梯度下降的方法，这样获得的结果总归是好的。

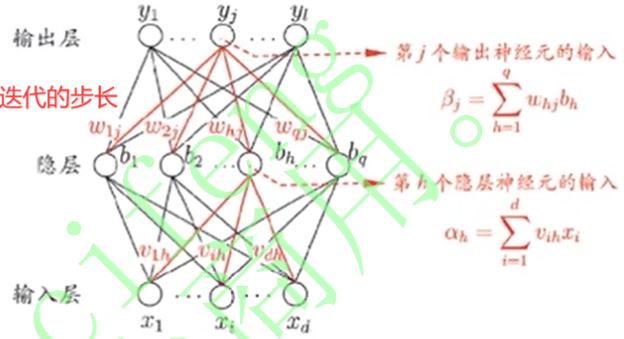
BP 算法基于梯度下降策略，以目标的负梯度方向对参数进行调整

以 w_{hj} 为例

决定每次迭代的步长

对误差 E_k，给定学习率 η，有：

$$\Delta w_{hj}=-\eta\frac{\partial E_k}{\partial w_{hj}}$$



注意到 w_{hj} 先影响到 β_j，再影响到 $\hat{y}_j^k$，然后才影响到 E_k，有：

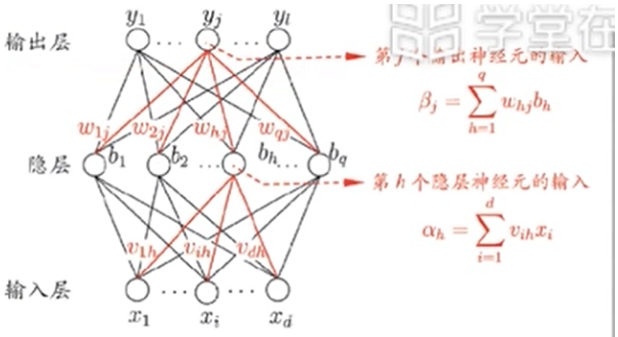
$$\frac{\partial E_k}{\partial w_{hj}}=\frac{\partial E_k}{\partial \hat{y}_j^k}\cdot\frac{\partial \hat{y}_j^k}{\partial \beta_j}\cdot\frac{\partial \beta_j}{\partial w_{hj}}$$

“链式法则”

$$\frac{\partial E_k}{\partial w_{hj}}=\frac{\partial E_k}{\partial \hat{y}_j^k}\cdot\frac{\partial \hat{y}_j^k}{\partial \beta_j}\cdot\frac{\partial \beta_j}{\partial w_{hj}}$$

$=(\hat{y}_j^k-y_j^k)$

$=b_h$



注意到 $\hat{y}_j^k=f(\beta_j-\theta_j)$

$f'(\beta_j-\theta_j)$

$\hat{y}_j^k(1-\hat{y}_j^k)$

对 $\text{sigmoid}(x)=\frac{1}{1+e^{-x}}$，有

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

令 $g_j=-\frac{\partial E_k}{\partial \hat{y}_j^k}\cdot\frac{\partial \hat{y}_j^k}{\partial \beta_j}$

$=\hat{y}_j^k(1-\hat{y}_j^k)(y_j^k-\hat{y}_j^k)$

于是，$\Delta w_{hj}=-\eta\frac{\partial E_k}{\partial w_{hj}}=\eta g_j b_h$

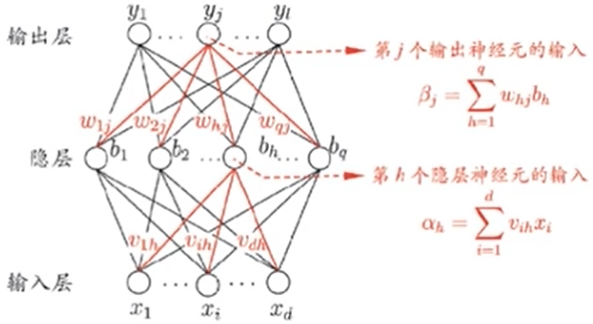
类似于上述对w_{hj}的推导，可以得到其他参数的推导结果如下

类似地，有：

$$\Delta\theta_j = -\eta g_j$$

$$\Delta v_{ih} = \eta e_h x_i$$

$$\Delta\gamma_h = -\eta e_h$$



其中：

$$e_h = -\frac{\partial E_k}{\partial b_h} \cdot \frac{\partial b_h}{\partial \alpha_h}$$

$$= -\sum_{j=1}^{l} \frac{\partial E_k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial b_h} f'(\alpha_h - \gamma_h) \quad = \sum_{j=1}^{l} w_{hj} g_j f'(\alpha_h - \gamma_h)$$

$$= b_h(1-b_h) \sum_{j=1}^{l} w_{hj} g_j$$

学习率 $\eta \in (0,1)$ 不能太大、不能太小

学习率如果太大，会产生**震荡**现象，但震荡是不用担心的，我们可以进行处理。

- 方案一：设置迭代次数，例如迭代1000轮后停止
- 方案二：设置动态学习率，开始用较大的学习率，到某个部分减半，再走某些部分再次减半，以此类推。

关于神经网络执行过程

在输入层获得x，在输出层获得y，这就是一个训练样例。

- 输入x，对权重、阈值等所有参数均随机初始化，然后隐层输入和输出均可算出，再可以得到输出层的输入和输出$\hat{y}$。
- 得到$\hat{y}$后，可以根据其和y，x等算出误差，再根据上面推得的各个步长△，从而调整各个参数
- 针对下一条数据进行迭代学习，重复上面的过程