

—武大本本科生课程



# 第13讲 神经网络

(Lecture 13 Neural Network)

武汉大学计算机学院机器学习课程组

# 第11章 神经网络

11.1 神经网络基本知识

11.2 感知器神经网络模型

11.3 BP神经网络模型

11.4 神经网络的发展-深度网络简介

11.5 神经网络Python程序举例

小结

# 11.1神经网络基本知识

---

神经网络(Neural Networks, NNs), 又称人工神经网络(Artificial Neural Networks, ANNs), 是模拟生物神经网络进行信息处理的一种数学模型。它以对大脑的生理研究成果为基础, 其目的在于模拟大脑的某些机理与机制, 实现一些特定的功能。

ANNs可以用硬件电路来实现(Neural Processing Unit, NPU), 也可以用计算机程序来模拟, 是人工智能研究的一种方法。

智能可以包含8个方面：

◆ 感知与认识客观事物、客观世界和自我的能力

-----感知是智能的基础——最基本的能力

◆ 通过学习取得经验与积累知识的能力

-----是人类在世界中能够不断发展的最基本能力

◆ 理解知识，运用知识和经验分析、解决问题的能力

这一能力可以看作是智能的高级形式。是人类对世界进行适当的改造，推动社会不断发展的基本能力。

◆ 联想、推理、判断、决策的能力

-----是智能的高级形式的又一方面

-----预测和认识

-----“主动”和“被动”之分(联想、推理、判断、决策的能力是“主动”的基础)

◆ 运用语言进行抽象、概括的能力

上述这5种能力，被认为是人类智能最为基本的能力。

作为5种能力综合表现形式的3种能力：

◆ 发现、发明、创造、创新的能力

◆ 实时、迅速、合理地应对复杂环境的能力

◆ 预测、洞察事物发展、变化的能力

使用计算机模拟人类的这些能力是人工智能所研究的问题，神经网络是人工智能研究的一种方法。

# 一.神经网络模型基本组成

## 1.生物神经元

科学研究发现：人脑大约有1000亿 ( $10^{11}$ )个神经元(每个神经元大约有 $10^4$ 个连接)，这些神经元通过1000万亿( $10^{15}$ )个连接构成一个大规模的神经网络系统。神经元是神经网络的基本信息处理单元。

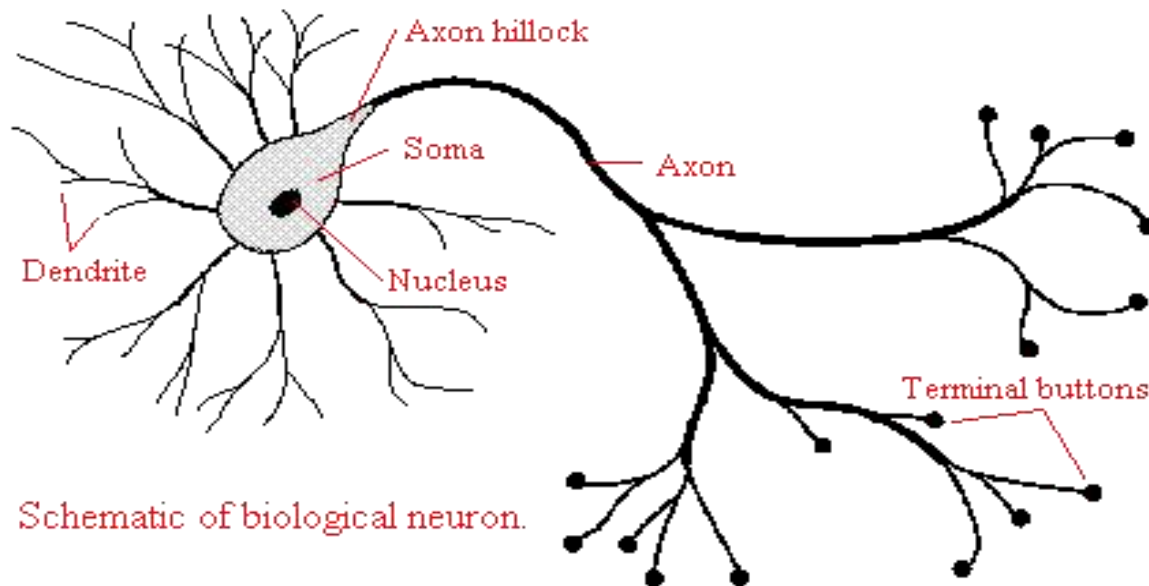
生物神经元  
的基本组成：

a.细胞体(soma)

b.树突(dendrite)

c.轴突(axon)

d.突触(synapse)



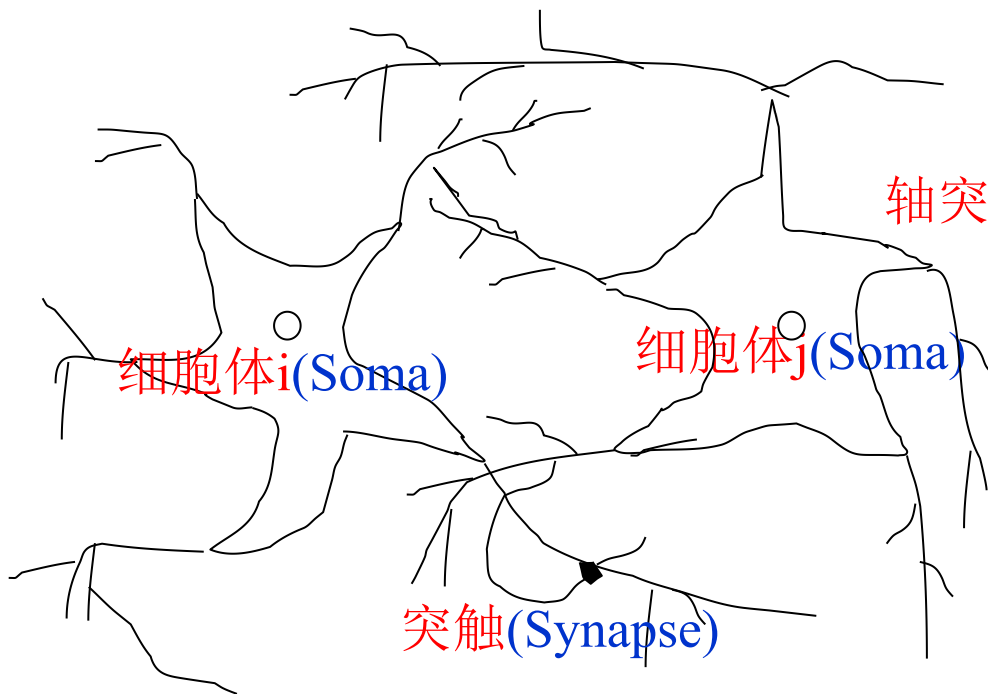
树突(Dendrite)

轴突(Axon)

细胞体i(Soma)

细胞体j(Soma)

突触(Synapse)



现代生理学研究表明：人类的大脑活动不是一个生物神经元所能完成的，也不是多个生物神经元功能的简单叠加，而是多个生物神经元构成的非线性动态处理系统。

在大脑神经系统中，每个神经元都通过突触与系统中的很多其他神经元相联系，突触的“连接强度”越大，接受的信号就越强，反之，突触的“连接强度”越小，接受的信号就越弱。突触的“连接强度”可以随着神经系统受到的训练而改变。例如，新记忆的形成就是通过改变突触的强度实现的，认识一位新朋友面孔的过程包含了各种突触的改变过程。

## 生物神经系统六大特征：

- (1)神经元及其连接；
- (2)神经元之间的连接强度决定了信号的传递强弱；
- (3)神经元之间的连接强度可以随训练而改变；
- (4)信号可以是起刺激(excite)作用的，也可以是起抑制(inhibit)作用的；
- (5)一个神经元接受信号的累积效果决定该神经元的状态；
- (6)每个神经元可以有一个阈值(threshold)。

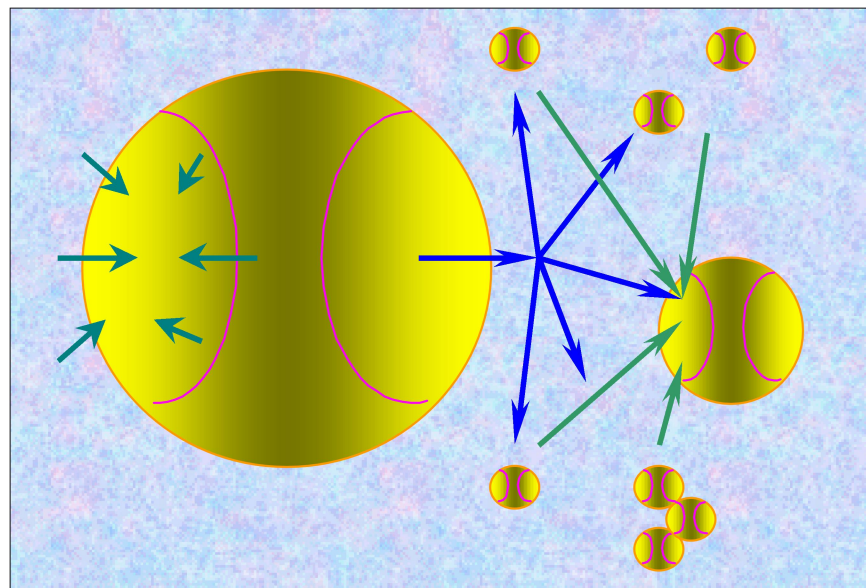


## 2.人工神经元模型

人工神经元是人工神经网络操作的基本信息处理单位。

人工神经元的基本结构：

- a.处理单元
- b.连接
- c.输入
- d.输出



人工神经网络没有生物神经网络那么复杂，但它们之间有两个关键的相似之处。首先，两个网络的构成都是可计算单元(处理单元)的高度互连；其次，处理单元之间的连接决定了网络的功能。

## (1)单输入神经元(Single-Input Neuron)

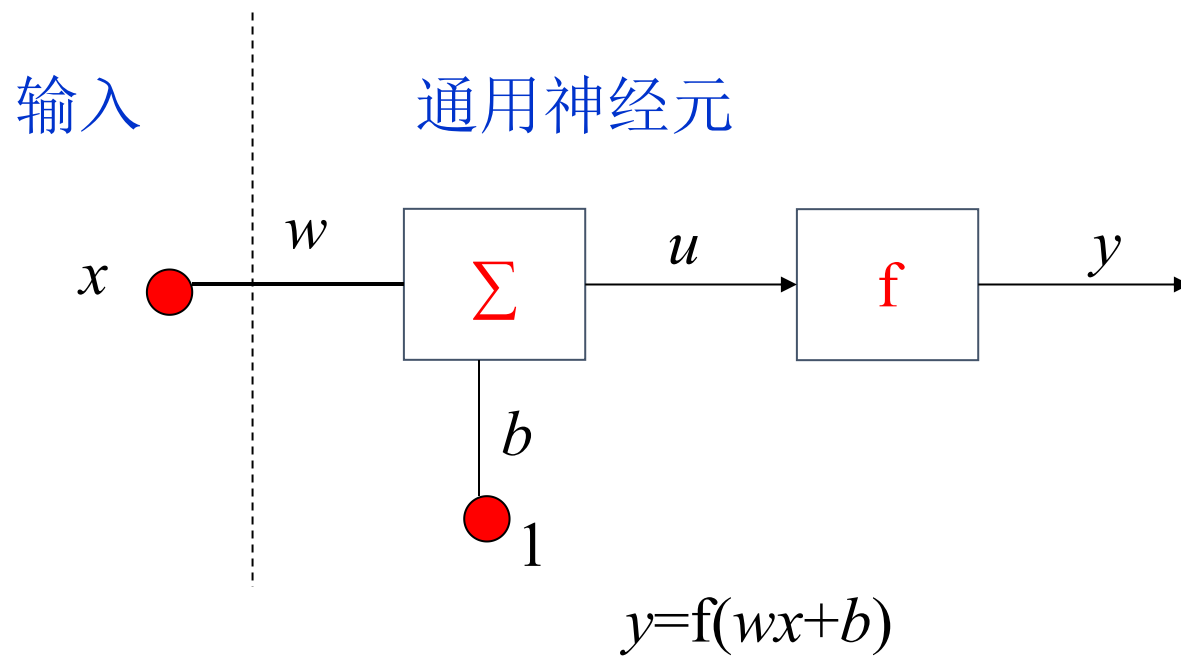


图 单输入神经元示意图

## 权值 偏置 净输入 传输函数

标量输入乘于标量权值 $w$ 得到 $wx$ ，再送到累加器，另一个输入1乘于偏置/阈值(bias/offset/threshold) $b$ ，再将其送到加法器。加法器输出 $u$ 通常被称为净输入(net input)，它被送到一个传输函数 $f$ (激活函数：用于限制神经元的输出幅度)，在 $f$ 中产生神经元的标量输出。

与前面的生物神经元对照，权值 $w$ 对应于突触的连接强度( $w$ 为正：激活activate， $w$ 为负：抑制inhibit)，细胞体对应于加法器和传输函数，神经元输出 $y$ 代表轴突的信号。

神经元输出按下式计算：

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

如，若 $w=3$ ,  $x=2$ ,  $b=-1.5$ , 则：

$$y=f(3\times 2-1.5)=f(4.5)$$

偏置值除了有常数输入值1外，它很像一个权值。但是如果不想使用偏置值，也可忽略它。

$w$ 和 $b$ 是神经元的可调整标量参数。设计者可以选择特定的传输函数，在一些学习规则中调整参数 $w$ 和 $b$ ，以满足特定的需要。

## (2)传输函数/激活函数(Transfer function/Activation function)

神经元中的传输函数可以是 $u$ 的线性函数或非线性函数。这里讨论常用的几个激活函数。

$$y = f(u) = f(wx + b)$$

a. 线性传输函数(线性单元)

$$y = f(u) = u$$

b. sign函数

$$y = f(u) = \operatorname{sgn}(u) = \begin{cases} 1, u > 0 \\ 0, u = 0 \\ -1, u < 0 \end{cases}$$

用该函数可以将输入分成-1和1两类。

c. 阶跃函数/阈值函数/阈值单元

$$y = f(u) = \frac{1}{2}[\text{sgn}(u) + 1] = \begin{cases} 1, & u > 0 \\ 0.5, & u = 0 \\ 0, & u < 0 \end{cases}$$

用该函数可以将输入分成0和1两类。

#### d.Sigmoid型传输函数(非线性单元)

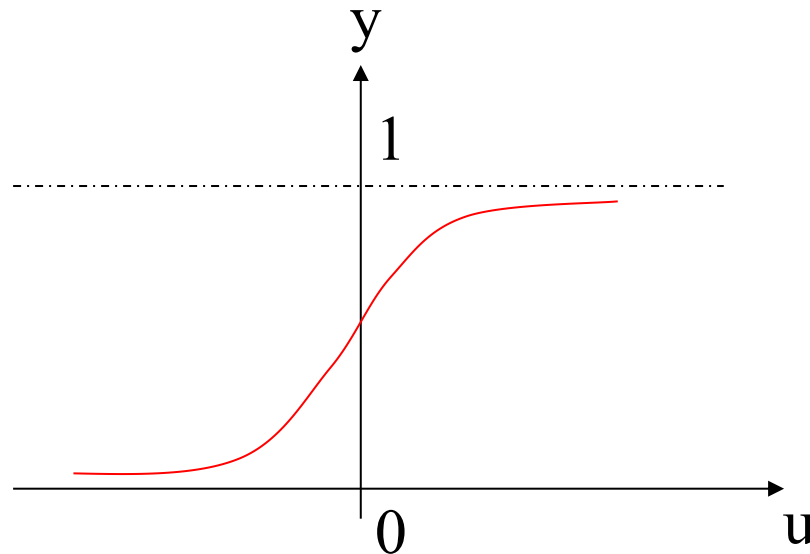
$$y = f(u) = \frac{1}{1 + \exp(-u)}$$

$$\{\exp(-u) = e^{-u}\}$$

该传输函数的输入在 $(-\infty, \infty)$ 区间取值，输出在0到1之间取值。

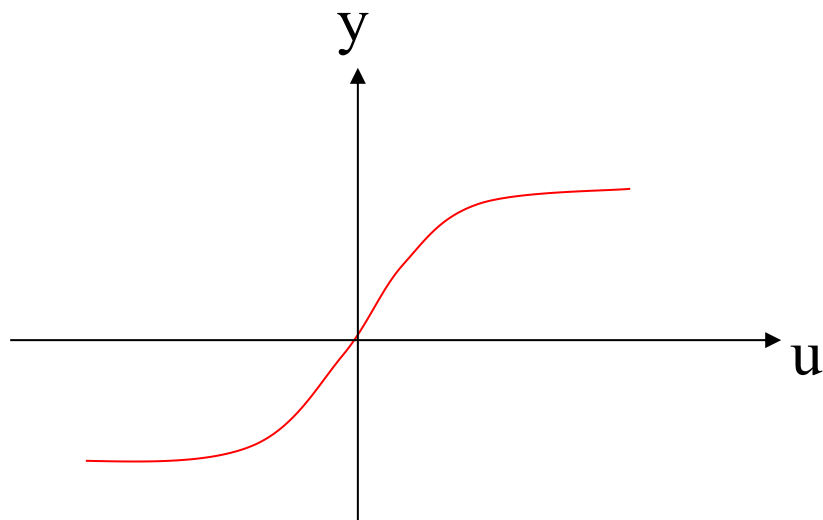
由于S型函数是可微的，能够体现数学计算上的优越性。在BP算法训练的多层网络中大多会采用该传输函数。

S型函数的图形：



e. 双曲正切S型函数(hyperbolic tangent sigmoid function)

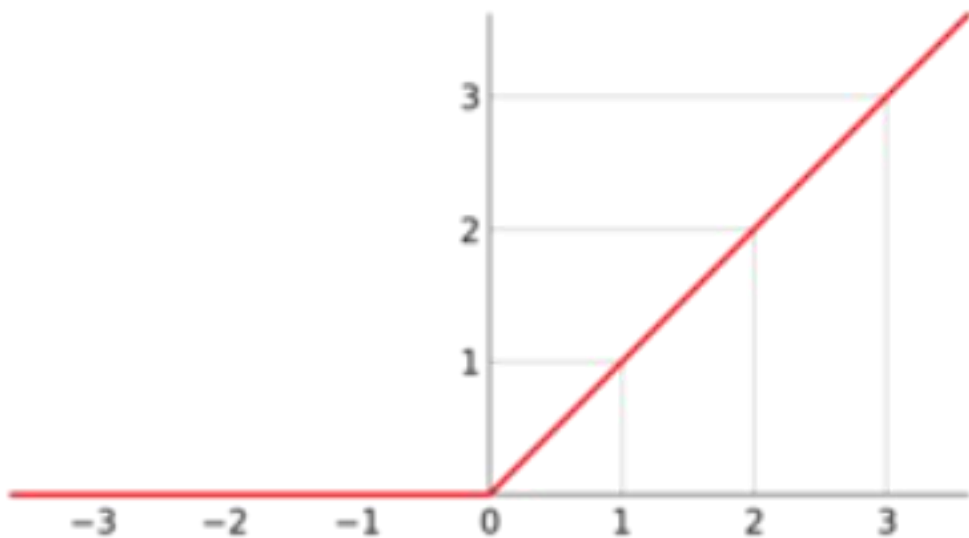
$$y = f(u) = \tanh(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}}$$





## f.修正线性单元 (ReLU: Rectified Linear Unit, 整流线性函数)

$$y = f(u) = \max(0, u)$$



Sigmoid的优势在于全局可导，但是当在饱和区域非常平缓，接近于0，很容易造成梯度消失问题，减缓收敛速度。梯度消失在网络层数多的时候尤其明显，是加深网络结构的主要障碍之一。相反，ReLU的Gradient大多数情况下是常数，有助于解决深层网络的收敛问题。此外，在大规模网络训练时，Sigmoid函数相比ReLU的计算量更大，资源和时间消耗更为严重。

参考阅读：神经网络基础知识、常用激活函数及其Python图形绘制

<https://yuanxyx.blog.csdn.net/article/details/110325071>

## g.其他传输函数

饱和线性函数:

$$y = f(u) = \begin{cases} 0, & u < 0 \\ u, & 0 \leq u \leq 1 \\ 1, & u > 1 \end{cases}$$

(saturating linear function)

{MATLAB函数:  $y=\text{satlin}(u)$ }

对称饱和线性函数(symmetrical saturating linear function):

$$y = f(u) = \begin{cases} 0, & u < -1 \\ u, & -1 \leq u \leq 1 \\ 1, & u > 1 \end{cases}$$

{MATLAB函数:  $y=\text{satlins}(u)$ }

对称饱和线性函数也称分段线性函数, 该函数在 $[-1,1]$ 线性区间内的放大系数是一致的, 这种形式的传输函数可以看作是非线性放大器的近似。

### (3)多输入神经元 (Multi-input Neuron)-Perceptron

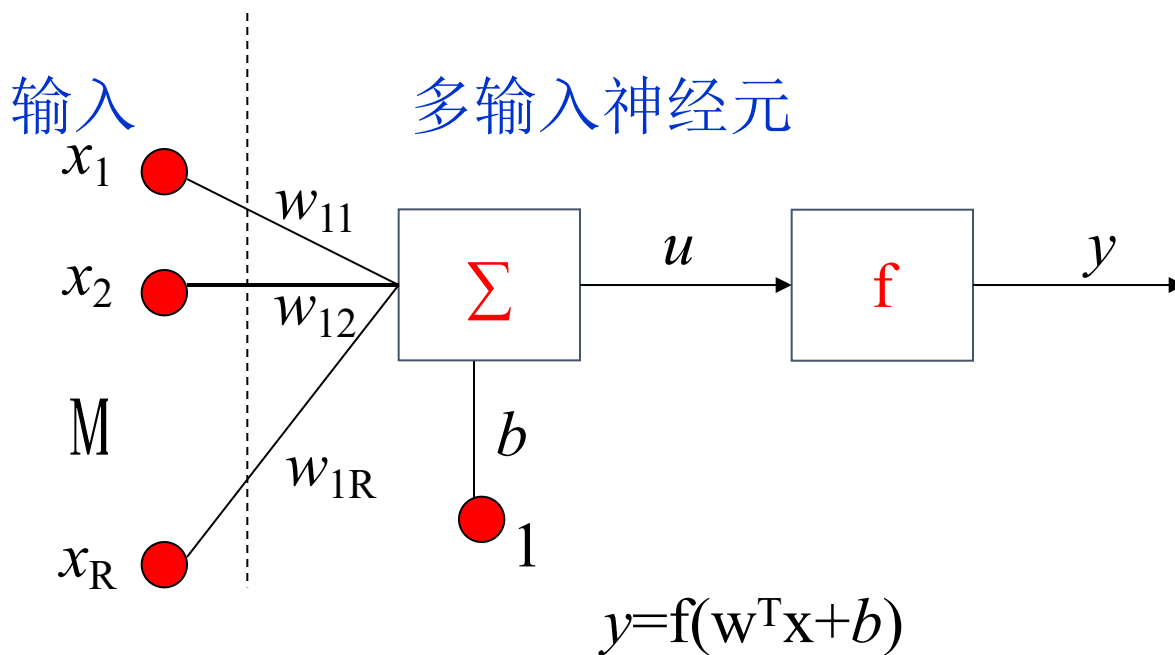


图 多输入神经元示意图

该神经元有一个偏置值 $b$ ，它与所有输入的加权和累加，从而形成净输入 $u$ ：

$$u = w_{11}x_1 + w_{12}x_2 + \dots + w_{1R}x_R + b$$

该表达式可写成矩阵形式：

$$u = \mathbf{w}^T \mathbf{x} + b$$

其中，

$$\mathbf{w} = (w_{11}, w_{12}, \dots, w_{1R})^T$$

$$\mathbf{x} = (x_1, x_2, \dots, x_R)^T$$

**权值下标：**采用人们习惯表示权值元素的下标。权值矩阵元素下标的第一个下标表示权值相应连接所指定的目标神经元编号，第二个下标表示权值相应连接的源神经元编号。

据此， $w_{12}$ 的含义是：该权值表示从第2个源神经元(此处即为第2个输入 $x_2$ )到第1个目标神经元的连接。

## (4)神经元的层

一般来说，有多个输入的单个神经元并不能满足实际应用的要求。在实际应用中需要多个并行的神经元。我们将这些可以并行操作的神经元组成的集合称为“层”。

下图是由 $S$ 个神经元组成的单层网络。 $R$ 个输入中的每一个均与每个神经元相连，权值矩阵 $W$ 有 $S$ 行。

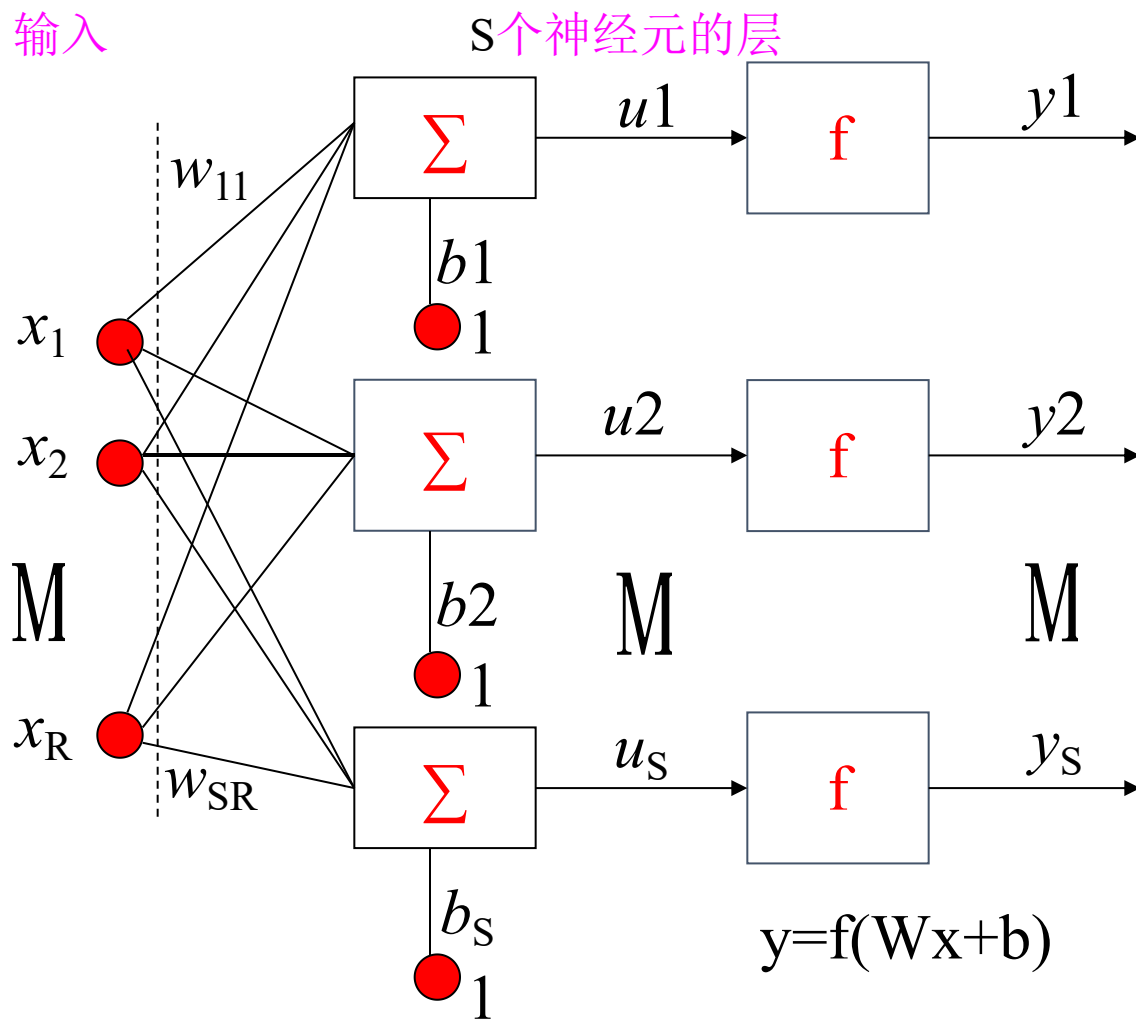


图  $S$ 个神经元组成的层

该层包括权值矩阵 $\mathbf{W}$ 、加法器 $\Sigma$ 、偏置(阈值)向量 $\mathbf{b}$ 、传输函数框和输出向量 $\mathbf{y}$ 。

输入向量 $\mathbf{x}$ 的每个元素均通过权值矩阵 $\mathbf{W}$ 和每个神经元相连。每个神经元有一个偏置值 $b_i$ 、一个加法器 $\Sigma$ 、一个传输函数 $f$ 和一个输出 $y_i$ 。将所有神经元的输出结合在一起，可以得到一个输出向量 $\mathbf{y}$ 。

通常，每层的输入个数并不等于该层中神经元的数目，即 $S \neq R$ 。

也许有人会问，同一层中所有神经元是否要有同样的传输函数？回答是否定的。可以把如上所述的两个并行操作网络组合在一起定义一种有不同传输函数的单个神经元(复合)层。两个网络都有同样的输入，而每个网络只产生一部分输出。

输入向量通过如下权值矩阵 $\mathbf{W}$  进入网络：

$$\mathbf{W} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1R} \\ w_{21} & w_{22} & \cdots & w_{2R} \\ \vdots & \vdots & \ddots & \vdots \\ w_{S1} & w_{S2} & \cdots & w_{SR} \end{pmatrix}$$

如前所述，矩阵 $\mathbf{W}$ 中元素的行下标代表该权值相应连接所指定的目标神经元，而列下标代表该权值相应连接的输入源神经元。那么， $w_{32}$ 的下标表示该元素是从第2个源神经元(此处即为第2个输入 $x_2$ )到第3个目标神经元的连接权值。

## 二. 神经网络结构

若将大量功能简单的神经元通过一定的拓扑结构组织起来，构成群体并行式处理的计算结构，这种结构就是人工神经网络。神经网络的特性和能力主要取决于网络拓扑结构 (network topology/architecture) 及学习方法 (learning method)。

根据神经元的不同连接方式，可将神经网络分为两大类：分层网络和相互连接型网络。

### 1. 分层网络

分层网络是将一个神经网络模型中的所有神经元按照功能分为若干层。一般有输入层、隐含层(中间层)和输出层，各层顺次连接。每一层的各神经元只能接受前一层神经元的输出, 作为自身的输入信号。



**输入层**：接收外部输入模式，并由各输入单元传送至相连的隐含层各单元。

**隐含层**：神经网络的内部处理单元层，神经网络所具有的模式变换能力，如模式分类、特征提取等，主要体现在隐含层单元的处理。根据模式变换功能的不同，隐含层可以有多层，也可以一层都没有。

**输出层**：若某层的输出是网络的输出，那么称该层为输出层。输出层产生神经网络的输出模式。

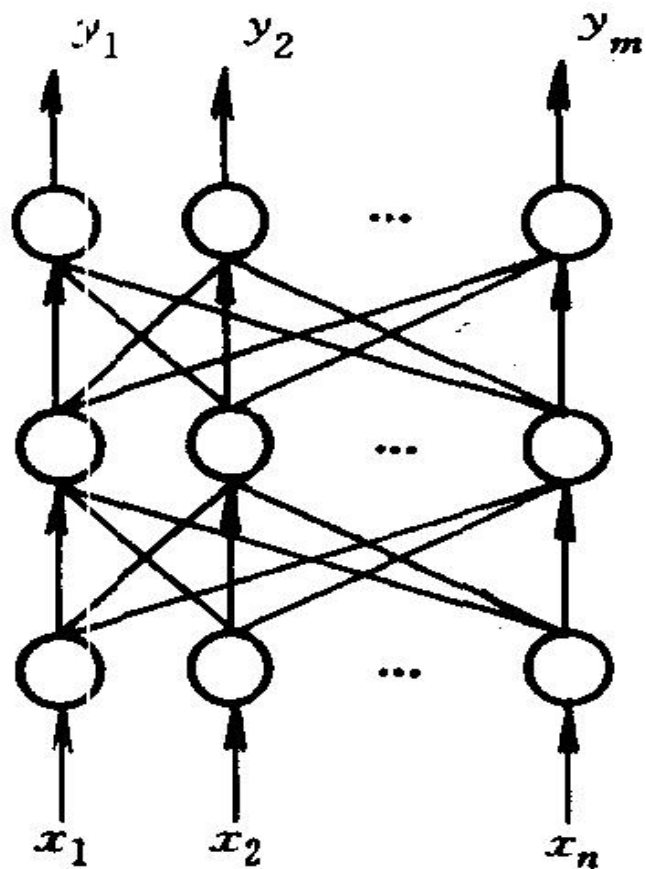
分层网络可细分为三种互连方式：

**(1)前向网络(前馈网络: Feedforward Neural Network)**

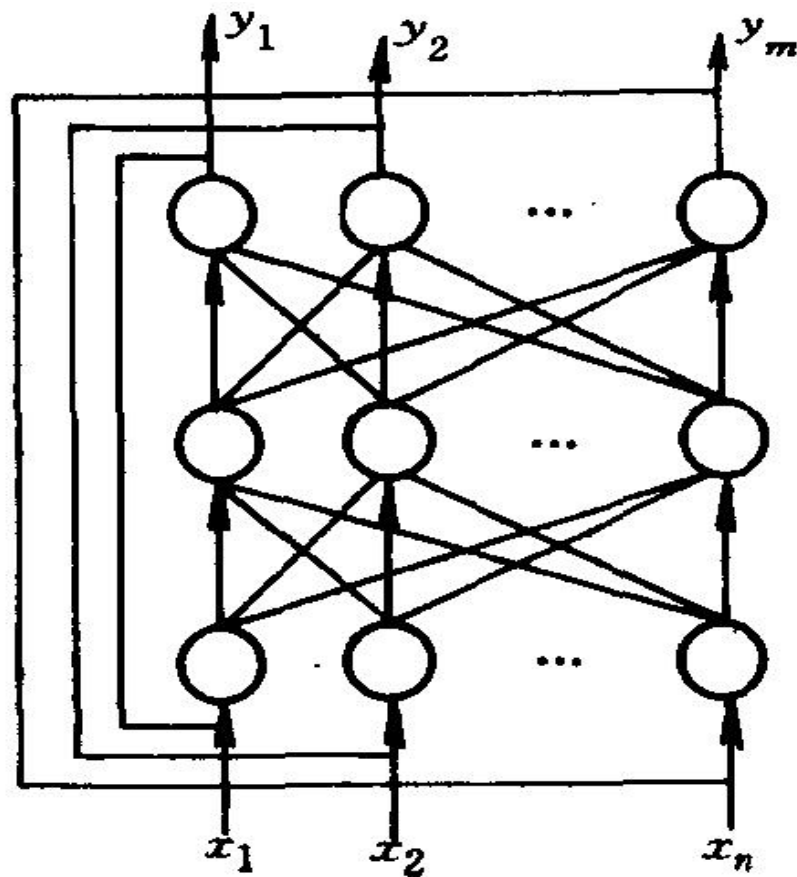
输入模式由输入层进入网络，经过中间各层的顺序模式变换，由输出层产生一个输出模式，完成一次网络状态的更新。见图a)所示。如**感知器**、**BP**(Back Propagation:误差反向传播，简称**反向传播**)神经网络、**RBF**(Radical Basis Function:**径向基函数**)网络采用此种连接方式。

**(2)具有反馈的前向网络**

反馈的结构形成封闭环路，从输出到输入具有反馈的单元也称为隐单元，其输出称为内部输出，而网络本身还是前向型的。见图b)所示。如**Fukushima**网络(**福岛**)采用此连接方式。



a) 前向网络

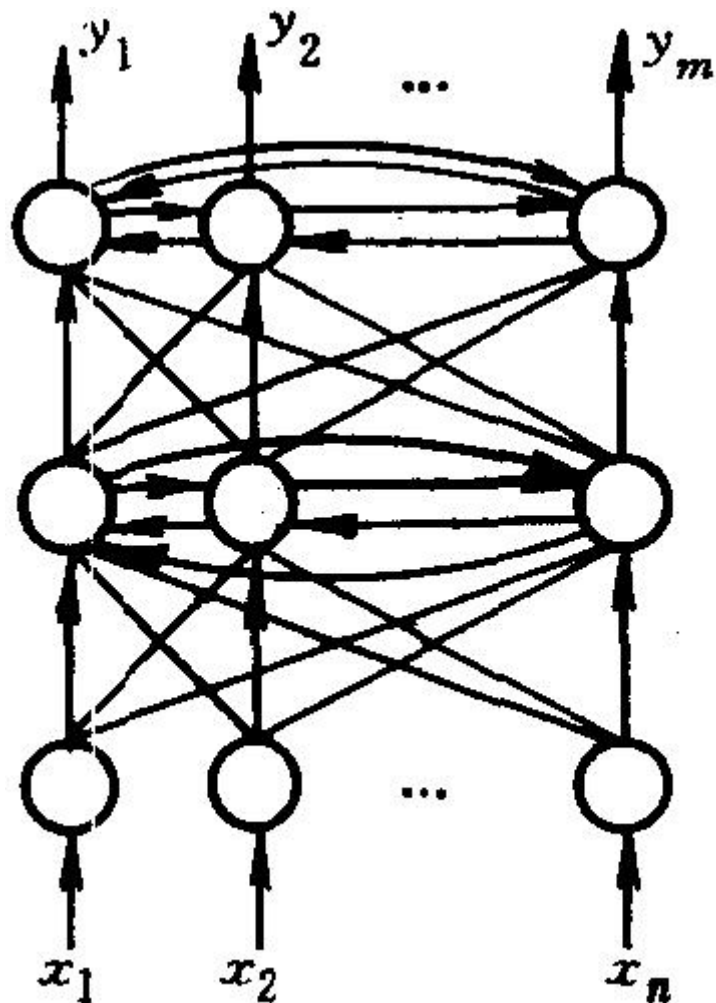


b) 具有反馈的前向网络

图 神经网络的连接方式

### (3)层内互连前向网络

同一层内单元的相互连接使它们彼此之间相互制约，限制同一层内能同时动作(激活)的神经元个数，而从外部看还是前向网络。见图c)所示。如很多自组织(竞争)神经网络采用此种连接。

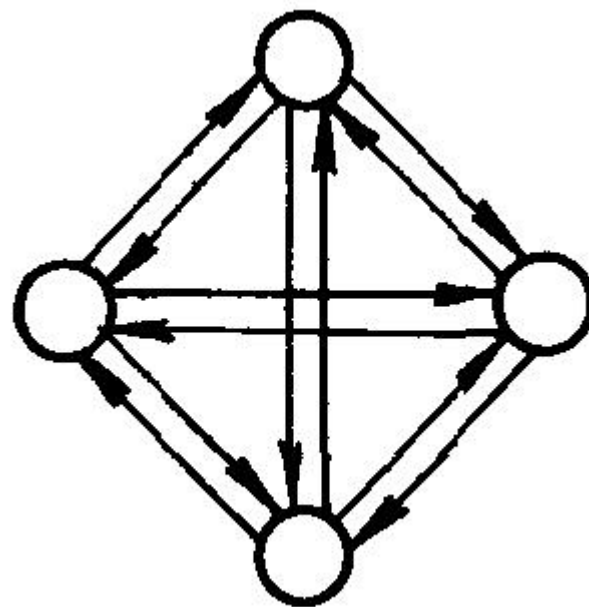


c)层内互连前向网络

图 神经网络的连接方式

## 2. 相互连接型网络

所谓相互连接是指网络中任意两个单元之间都是可达的，即存在连接路径。互连网络又细分为局部互连和全互连。全互连网络中每个神经元的输出都与其他神经元输入相连，而局部互连网络中，有些神经元之间没有连接关系。见图d)所示。如Hopfield网络和Boltzmann网络(又称Boltzmann machine)。



d)全互连网络

图 神经网络的连接方式

对于简单的前向网络，给定某一输入模式，网络能迅速产生一个相应的输出，并保持不变。但在相互连接的网络中，对于给定的某一输入模式，由某一网络参数出发，在一段时间内处于不断改变输出模式的动态变化中，网络最终可能产生某一稳定的输出模式，但也可能进入周期性振荡或混沌(chaos)状态。

### 三.神经网络基本学习算法

学习方法是人工神经网络研究中的核心问题。

神经网络的学习也称为训练，指的是神经网络在外部环境刺激下调整神经网络的参数，使神经网络以一种新的方式对外部环境作出反应的过程。

能够从环境中学习和在学习中提高自身性能是神经网络最有意义的性质，神经网络经过反复学习达到对环境的了解。

# 1.学习方式

分为有导师/教师学习、无导师/教师学习和再励学习。

## (1)有导师学习/有监督学习

(Supervised learning)

需组织一批正确的输入输出数据对。对每一个输入训练样本，都有一个期望得到的输出值(也称导师信号)，将它和实际输出值进行比较，根据两者之间的差值不断调整网络的连接权值，直到差值减小到允许范围之内。

## (2)无导师学习/无监督学习/自组织学习

(Unsupervised learning)

仅有一批输入数据。网络初始状态下，连接权值均设置为一小正数，网络按照预先设定的某种规则反复地自动调整网络连接权值，使网络最终具有模式分类等功能。这一自组织方式，使网络具有某种“记忆”能力。

## (3)增强学习/强化学习

(Reinforcement learning)

介于有导师学习和无导师学习之间，外部环境对系统输出结果只给出评价(奖和罚)，而不给出正确答案，学习系统通过强化那些受奖励的动作来改善自身的性能。此学习方式更适合于控制系统应用领域。



#### (4)半监督学习\* (Semi-supervised Learning, 了解)

是近几年来模式识别和机器学习领域的研究重点。它主要考虑如何利用少量的标注样本和大量的未标注样本进行训练和分类的问题。半监督学习对于减少标注代价，提高机器学习性能具有重大的实际意义。

基本假设:1)聚类假设:可直观地解释为如果一些模式紧凑地聚合在一起,它们就不可能属于两种类别;2)流形假设:认为高维数据总是落在低维流形上,据此假设,沿流形面上相近的点应该具有相似的类别标记,只要捕捉到数据所在的流形面就可以根据样本点之间的相似程度对未知样本的标记进行预测。

聚类假设反映的是模型的全局特征,而流形假设主要考虑是模型的局部平滑性,反映的是局部特征。

半监督学习的主要算法有五类:基于概率的算法;在现有的有监督算法基础上进行修改的方法,如半监督支持向量机方法等;直接依赖于聚类假设的方法;基于多试图的方法;基于图的方法。

## 2.基本学习算法

不同的学习算法对神经元的权值调整的表达式是不同的。没有一种独特的算法适用于设计所有的神经网络，选择或设计学习算法时还需考虑神经网络的结构及神经网络与外界环境相连接的形式。

权值的确定通常有两种方法：

①根据具体要求直接计算

如Hopfield网络作优化计算。

②通过学习得到

大多数人工神经网络都采用这种方法。

设 $w_{ij}$ 是神经元 $i$ 与神经元 $j$ 之间的连接权值， $\Delta w_{ij}$ 为连接权值 $w_{ij}$ 的修正值，即 $w_{ij}(n+1)=w_{ij}(n)+\Delta w_{ij}$ 。

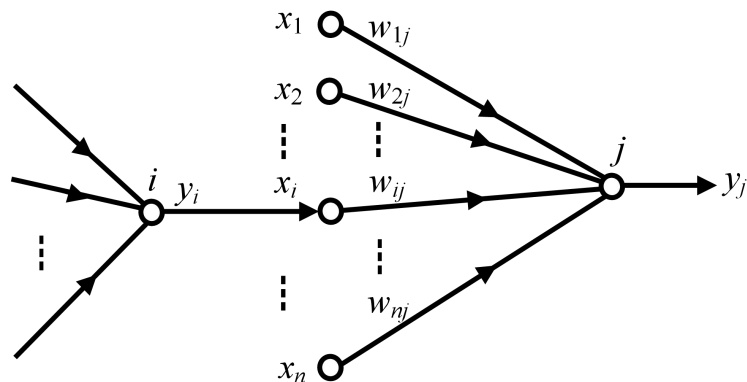
## (1)Hebb学习规则

它是由加拿大著名生理心理学家赫布Hebb根据生物学中的条件反射机理，于1949年提出的神经元连接强度变化规则，属于无导师学习。其基本思想是：如果有两个神经元同时兴奋(即同时被激活)，则它们之间的突触连接强度的增强与它们的激励的乘积成正比。假设 $t$ 时刻， $y_i(t)$ 为神经元 $i$ 的激活值(输出)， $y_j(t)$ 为神经元 $j$ 的激活值， $w_{ij}$ 表示神经元 $i$ 到神经元 $j$ 的连接权值，神经元 $j$ 的第 $i$ 个输入是另一个神经元 $i$ 的输出  $\{y_i(t)=x_i(t)\}$ (见图)，Hebb连接权值的学习规则 (learning rule) 可表示为：

$$\Delta w_{ij} = \eta y_j(t) y_i(t) = \eta y_j(t) x_i(t)$$

式中， $\eta$ 为学习速率参数。

Hebb学习规则是人工神经网络学习的基本规则，几乎所有神经网络的学习规则都可以看作Hebb学习规则的变种。



## (2) $\delta$ 学习规则

**$\delta$ 学习规则(误差校正学习规则)**是根据神经网络的输出误差对神经元的连接权值进行修正，属于有导师学习。

设 $d_j$ 为神经元 $j$ 的期望输出， $y_j$ 为神经元 $j$ 的实际输出， $d_j - y_j$ 为误差信号或学习信号，神经元 $i$ 到神经元 $j$ 的连接权为 $w_{ij}$ 。

在Hebb学习规则中引入教师信号，将Hebb学习规则公式中的 $y_j$ 换成神经元 $j$ 期望目标输出 $d_j$ 与神经元 $j$ 实际输出 $y_j$ 之差，即为有监督 **$\delta$ 学习规则**：

$$\Delta w_{ij}(k) = \eta [d_j(k) - y_j(k)] y_i(t) = \eta [d_j(k) - y_j(k)] x_i(t)$$

$$\delta = d_j(k) - y_j(k)$$

上式表明，两个神经元之间的连接强度的变化量与**教师信号** $d_j(k)$ 和网络实际输出 $y_j$ 之差成正比。

定义**均方误差函数**：

$$E = \frac{1}{2} (d_j - y_j)^2 = \frac{1}{2} \left( d_j - f \left( \sum_k w_{kj} x_k \right) \right)^2$$

从而：

$$\frac{\partial E}{\partial w_{ij}} = -\left(d_j - y_j\right) f' \left( \sum_k w_{kj} x_k \right) x_i$$

要使期望误差达到最小，要求取负梯度  $-\frac{\partial E}{\partial w_{ij}}$ ，得到基于梯度下降法的 $\delta$ 学习规则：

$$\begin{aligned} \Delta w_{ij} &= \eta \left( d_j - y_j \right) f' \left( \sum_k w_{kj} x_k \right) x_i \\ &= \eta \left( d_j - y_j \right) f' \left( neu_j \right) x_i \\ &= \eta \delta_j x_i \end{aligned}$$

其中， $\eta$ 为学习速率参数。一般 $\eta$ 选得很小； $\delta_j$ 为误差函数对神经元 $j$ 输入的偏导数。BP(反向传播)算法采用了 $\delta_j$ 规则。

### (3) 竞争学习算法(\*: 了解)

有导师的学习算法不能充分反映人脑神经系统的高级智能学习过程，人脑神经系统在学习过程中各个细胞始终存在竞争。竞争学习网络由一组性能基本相同，只是参数有所不同的神经元构成。对于一个输入模式内各子模式的作用，每个神经元通过相互竞争来做出不同的反映，每个神经元的激活范围遵循某种特定的限制。

竞争学习的基本思想：对竞争获胜的神经元权值进行修正，获胜神经元的输入状态为1时，相应的权值增加，状态为0时权值减小；学习过程中，权值越来越接近相应的输入状态。竞争学习算法属于无导师学习。

## 四.神经网络的特点及应用

### 1.神经网络的特点

#### (1)并行分布式处理

神经网络具有高度的并行结构和并行实现能力，具有高速寻找优化解的能力，能够发挥计算机的高速运算性能。

#### (2)非线性处理

人脑的思维是非线性的，故神经网络模拟人的思维也应是非线性的，这一特性有助于处理非线性问题。

#### (3)具有自学习功能

通过对过去的历史样本数据的学习，训练出一个具有归纳全部数据的特定神经网络，自学习功能对预测具有重要意义。

#### (4)神经网络的硬件实现

要使人工神经网络能更有效地解决大规模问题，可以采用超大规模集成电路(VLSI)实现，即把神经元和连接制作在一块芯片(多为CMOS)上构成ANNs。神经网络的VLSI设计方法近年来发展很快，硬件实现已成为ANNs的一个重要分支。

## 2.神经网络的应用领域

- ◆ 信号处理与分析
- ◆ 图像处理与分析
- ◆ 分类识别
- ◆ 自动控制中的智能控制
- ◆ 经济分析与决策
- ◆ 神经科学和生物学
- ◆ 信用分析
- ◆ 航空航天
- ◆ 医学诊断系统
- ◆ 预测分析
- ◆ .....



## 11.2 感知器神经网络模型

### 一. 单层感知器

感知器(Perceptron)是由美国计算机科学家罗森布拉特(Roseblatt)于1957年提出的有导师学习的神经网络模型。感知器是神经网络用来进行模式识别的一种最简单模型，是最早的神经网络。它属于前向神经网络模型，但是仅由一个神经元组成的**单层感知器**只能区分线性可分两类模式。

首先以最简单、只有一个神经元的单层感知器为例进行说明。

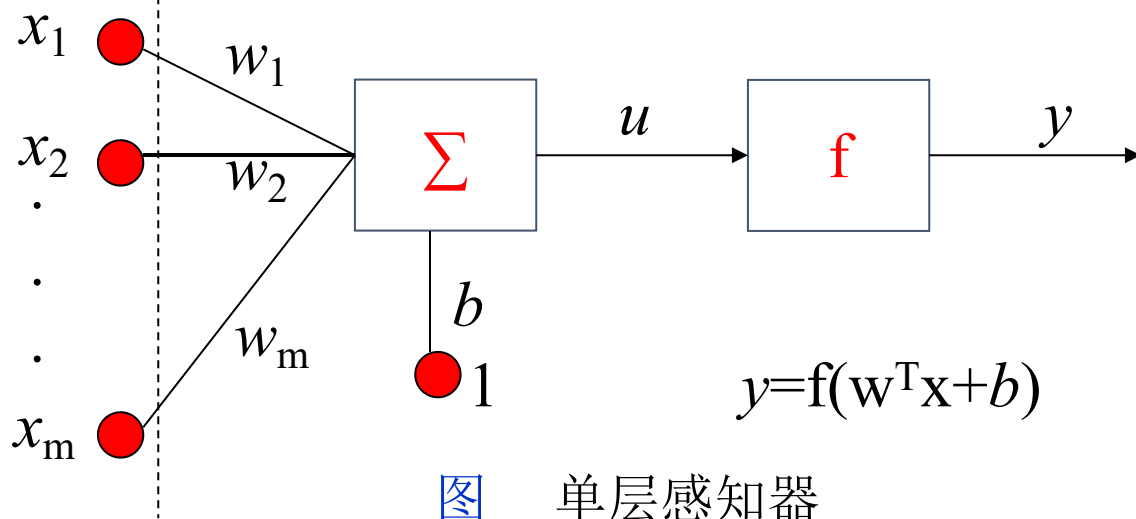


图 单层感知器

上图单层感知器中，通常 $f(u)$ 使用符号函数或阶跃函数(二值阈值元件)。单层感知器将外部输入模式分成两类：当感知器的输出为+1时，模式 $x$ 属于 $\omega_1$ 类，当感知器输出为-1时，模式 $x$ 属于 $\omega_2$ 类。

## 二.单层感知器的学习算法

单层感知器的学习算法是基于迭代的思想，通常采用 $\delta$ 学习规则的学习算法。考虑偏置值 $b$ 及输入常量1，可将输入向量和权值向量写成增广向量的形式：

$$\mathbf{x}(k) = (x_1(k), x_2(k), \text{L} \quad x_n(k), 1)^T$$

$$\mathbf{w}(k) = (w_1(k), w_2(k), \text{L} \quad w_n(k), b(k))^T$$

式中 $k$ 为迭代次数,若 $b(k)$ 用 $w_{n+1}(k)$ 表示,则二值阈值元件的输入可写为:

$$u = \sum_{i=1}^{n+1} w_i(k) x_i(k) = \mathbf{w}(k)^T \mathbf{x}(k)$$

单层感知器进行模式识别的决策面为：

$$\sum_{i=1}^n w_i x_i + b = 0$$

当它用于两类问题的模式分类时，相当于在高维样本空间中，用一个超平面将两类样本分开。

令上式等于0,即 $\mathbf{w}(k)^T \mathbf{x}(k) = 0$ ,可得到在 $n$ 维空间的单层感知器判决超平面。

一个神经元的二分类单层感知器学习算法描述(1)：

**Step 1:**设置变量和参数

$\mathbf{x}(k) = (x_1(k), x_2(k), \dots, x_n(k), 1)^T$  为输入向量,也可看成是训练样本

$\mathbf{w}(k) = (w_1(k), w_2(k), \dots, w_n(k), b(k))^T$  为权向量  
 $b(k)$ 为偏置值, $f(\cdot)$ 为传输函数, $y(k)$ 为网络实际输出, $d(k)$ 为期望输出, $k$ 为迭代次数, $e$ 为实际输出与期望输出的误差.

## Step 2: 初始化

置迭代次数 $k=0$ , 给权值向量 $\mathbf{w}(0)$ 的各个分量赋一个较小的随机非零值.

Step 3: 输入样本模式 $\mathbf{x}(k)$ 和它的期望输出 $d(k)$

Step 4: 计算实际输出

$$y(k) = f\left(\sum_{i=1}^{n+1} w_i(k)x_i(k)\right)$$

Step 5: 求出期望输出和实际输出的误差 $e$

$$e = d(k) - y(k)$$

根据误差 $e$ 判断当前输出是否满足条件. 若满足条件则算法结束; 否则 $\mathbf{w}(k+1) = \mathbf{w}(k) + \eta[d(k) - y(k)]\mathbf{x}(k)$ ,  $k \leftarrow k+1$ ,

式中, 
$$d(k) = \begin{cases} 1, & \mathbf{x}(k) \in \omega_1 \\ -1, & \mathbf{x}(k) \in \omega_2 \end{cases}$$

Step 6: 转到Step 3, 进入下一次计算过程。

在以上的学习算法中，Step 5需要判断是否满足算法结束条件，算法结束条件可以是误差小于设定的值  $\varepsilon$  或者是权值的变化已经很小。另外，在实现过程中还应设定最大的迭代次数，以防止算法不收敛时，学习算法进入死循环。

在单层感知器学习算法中，最关键的因素是引入了一个量化的期望输出，这样可以利用 $\delta$ 学习规则对权值向量逐步进行修正，最终达到问题所需要的精度。

对于线性可分的两类模式，可以证明单层感知器的学习算法是收敛的(证明此略)，即通过调整神经网络各连接权值可以得到合适的决策边界，从而正确区分两类模式；而对于线性不可分的两类模式，无法用一条直线区分两类模式，此时，单层感知器的学习算法不收敛，即单层感知器无法正确区分线性不可分的两类模式。

## 二分类单层感知器算法<sup>[1.P54]</sup>描述(2):

1. 将N个属于 $\omega_1$ 类和 $\omega_2$ 的训练样本写成增广向量并规范化(即将类 $\omega_2$ 的全部样本都乘于-1)。任取仅向量初始值 $\mathbf{w}(1)$ 开始迭代(括号中的1表示迭代次数 $k=1$ )。
2. 用全部训练样本进行一轮迭代。每输入一个样本 $\mathbf{x}$ 计算一次判别函数 $\mathbf{w} \cdot \mathbf{x}$ (两个向量的内积), 根据判别函数分类结果的正就只有修改权向量, 此时 $k=k+1$ 。假设进行第 $k$ 次迭代时, 输入样本为 $\mathbf{x}$ , 计算 $\mathbf{w}(k) \cdot \mathbf{x}$ 的值, 按下式修正权向量 $\mathbf{w}$ :

$$\mathbf{w}(k+1) = \begin{cases} \mathbf{w}(k), & \mathbf{w}^T(k)\mathbf{x} > 0 \text{ (分类正确)} \\ \mathbf{w}(k) + \rho \mathbf{x}(k), & \mathbf{w}^T(k)\mathbf{x} \leq 0 \text{ (分类错误)} \end{cases} \quad (\text{a})$$

( $\rho > 0$ )

3. 分析分类结果, 在这一轮的迭代过程中只要有一个样本的分类发生错误, 即出现了公式(a)的第2种情况, 则回到第2步进行第2轮迭代, 直到分类正确迭代过程结束。此时的权向量即为算法结果。

训练后的感知器输出:  $y = \text{sign}(\mathbf{w} \cdot \mathbf{x})$ 。

感知器算法就是一种赏罚(Reward-Punishment)过程: 当分类器发生分类错误时, 对分类器进行“罚”-修改权向量, 以使其向正确的方向转换; 分类正确时, 对其进行“赏”-表现为“不罚”, 即权向量不变。

M类问题多分类单层感知器算法描述<sup>[1.P227-228]</sup>: 不讲。

## 感知器固定增量算法<sup>[1.P60-62]</sup> - 梯度算法的应用 (\*: 选学)

准则函数:  $J(W, X) = \frac{1}{2} (|W^T X| - W^T X)$

该准则函数有唯一最小值“0”，且发生在  $W^T X > 0$  的时候。

求  $W(k)$  的递推公式:

设  $X = [x_1, x_2, \dots, x_n, 1]^T$  ,  $W = [w_1, w_2, \dots, w_n, w_{n+1}]^T$

1. 求  $J$  对  $W$  的梯度  $\nabla J = \frac{\partial J(W, X)}{\partial W} = ?$

方法: 函数对向量求导=函数对向量的分量求导, 即

$$\frac{\partial f}{\partial W} = \left[ \frac{\partial f}{\partial w_1}, \dots, \frac{\partial f}{\partial w_n}, \frac{\partial f}{\partial w_{n+1}} \right]^T$$

①首先求  $\mathbf{W}^T \mathbf{X}$  部分:

$$J(\mathbf{W}, \mathbf{X}) = \frac{1}{2} (\|\mathbf{W}^T \mathbf{X}\| - \mathbf{W}^T \mathbf{X})$$

$$\begin{aligned} \frac{\partial(\mathbf{W}^T \mathbf{X})}{\partial \mathbf{W}} &= \frac{\partial}{\partial \mathbf{W}} \left( \sum_{i=1}^n w_i x_i + w_{n+1} \right) \\ &= \left[ \frac{\partial}{\partial w_1} \left( \sum_{i=1}^n w_i x_i + w_{n+1} \right), \dots, \frac{\partial}{\partial w_k} \left( \sum_{i=1}^n w_i x_i + w_{n+1} \right), \dots, \frac{\partial}{\partial w_{n+1}} \left( \sum_{i=1}^n w_i x_i + w_{n+1} \right) \right]^T \\ &= [x_1, \dots, x_k, \dots, x_n, 1]^T = \mathbf{X} \end{aligned}$$

另: 矩阵论中有 (也可以直接按梯度定义演算)

$$\begin{aligned} \frac{d\mathbf{X}}{d\mathbf{X}^T} &= \frac{d\mathbf{X}^T}{d\mathbf{X}} = \mathbf{I}_{n \times n} \\ \therefore \frac{\partial(\mathbf{W}^T \mathbf{X})}{\partial \mathbf{W}} &= \mathbf{I}_{(n+1) \times (n+1)} \mathbf{X}_{(n+1) \times 1} = \mathbf{X}_{(n+1) \times 1} \end{aligned}$$



$$J(W, X) = \frac{1}{2} (|W^T X| - W^T X)$$

② 由①的结论  $\frac{\partial(W^T X)}{\partial W} = X$  有:

$$W^T X > 0 \text{ 时, } \frac{\partial(|W^T X|)}{\partial W} = \frac{\partial(W^T X)}{\partial W} = X$$

$$W^T X \leq 0 \text{ 时, } \frac{\partial(|W^T X|)}{\partial W} = \frac{\partial(-W^T X)}{\partial W} = -X$$

$$\therefore \frac{\partial(|W^T X|)}{\partial W} = [\text{sign}(W^T X)] \cdot X$$

$$\text{其中 } \text{sign}(W^T X) = \begin{cases} +1, & \text{若 } W^T X > 0 \\ -1, & \text{若 } W^T X \leq 0 \end{cases}$$

$$\therefore \nabla J = \frac{\partial J(W, X)}{\partial W} = \frac{1}{2} [X \text{sign}(W^T X) - X]$$

## 2. 求 $W(k+1)$

将  $\nabla J = \frac{\partial J(W, X)}{\partial W} = \frac{1}{2} [X \operatorname{sign}(W^T X) - X]$  代入梯度法公式

$$W(k+1) = W(k) - \rho \nabla J = W(k) - \rho \left[ \frac{\partial J(W, X)}{\partial W} \right]_{W=W(k)}$$

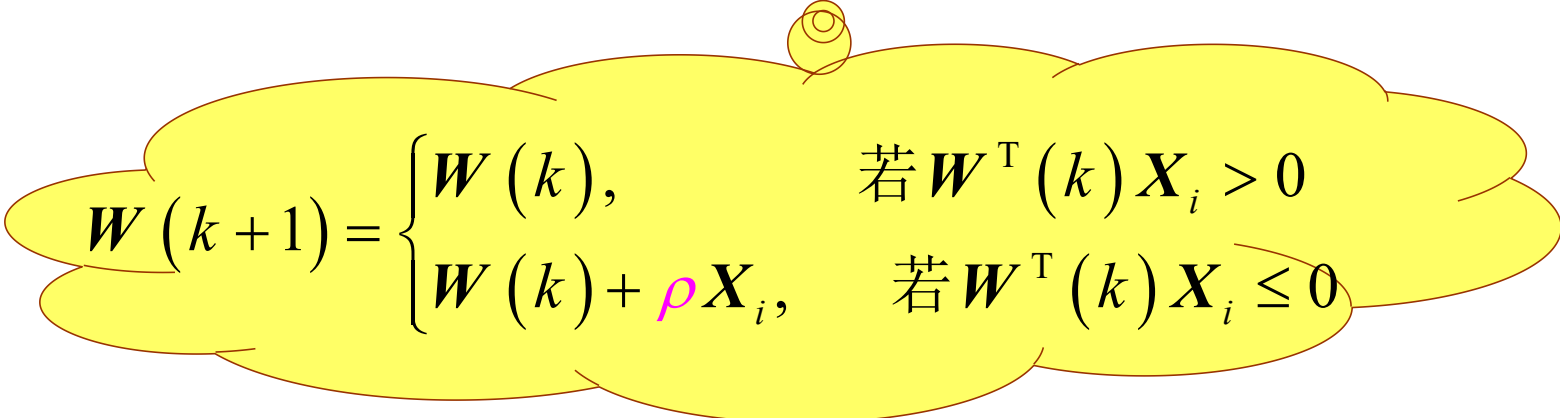
得:  $W(k+1) = W(k) - \rho \frac{1}{2} [X \operatorname{sign}(W^T(k) X) - X]$

$$= W(k) + \frac{\rho}{2} [X - X \operatorname{sign}(W^T(k) X)]$$

$$= W(k) + \begin{cases} 0, & \text{若 } W^T(k) X > 0 \\ \rho X, & \text{若 } W^T(k) X \leq 0 \end{cases} \quad (b)$$

$$\text{即: } \mathbf{W}(k+1) = \mathbf{W}(k) + \begin{cases} 0, & \text{若 } \mathbf{W}^T(k)\mathbf{X} > 0 \\ \rho \mathbf{X}, & \text{若 } \mathbf{W}^T(k)\mathbf{X} \leq 0 \end{cases}$$

即为**固定增量算法**(Fixed increment algorithm), 与神经网络中的感知器算法结论完全相同。



$$\mathbf{W}(k+1) = \begin{cases} \mathbf{W}(k), & \text{若 } \mathbf{W}^T(k)\mathbf{X}_i > 0 \\ \mathbf{W}(k) + \rho \mathbf{X}_i, & \text{若 } \mathbf{W}^T(k)\mathbf{X}_i \leq 0 \end{cases}$$

可以认为, 神经网络中的**感知器算法是梯度法的特例**。梯度法是将感知器算法中联立不等式求解 $\mathbf{w}$ 的问题, 转换为求目标函数极小值的问题, 将原来有多个解的情况, 变成求最优解的情况。

**收敛性**(Convergence): 如果经过算法的有限次迭代运算后, 求出了一个使训练集中所有样本都能正确分类的 $\mathbf{w}$ , 则称算法是收敛的。只要模型类别线性可分, **固定增量算法 (感知器算法)**就可以在有限的迭代步数里求出权向量 $\mathbf{w}$ 的解。

**举例:** Two class training examples are as follows:

$$\omega_1 : \mathbf{X}_1 = [0, 0]^T \quad \mathbf{X}_2 = [0, 1]^T$$

$$\omega_2 : \mathbf{X}_3 = [1, 0]^T \quad \mathbf{X}_4 = [1, 1]^T$$

Use the **Fixed Increment (Perceptron) Algorithm** to classify them and find out the weight vector and the **discriminant function**.

**Ans:** Extend: all example to extended vectors;

Normalize: samples in  $\omega_2$  is multiplied by (-1).

$$\mathbf{X}_1 = [0, 0, 1]^T \quad \mathbf{X}_2 = [0, 1, 1]^T \quad \mathbf{X}_3 = [-1, 0, -1]^T \quad \mathbf{X}_4 = [-1, -1, -1]^T$$

Take  $W(1)=\mathbf{0}=(0,0,0)^T$ ,  $\rho=1$ . Iterative process is:

1<sup>st</sup> run:

$$W^T(1)X_1 = [0,0,0] \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = 0 \leq 0, \quad \text{So } W(2) = W(1) + X_1 = [0,0,1]^T$$

$$W^T(2)X_2 = [0,0,1] \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} = 1 > 0, \quad \text{So } W(3) = W(2) = [0,0,1]^T$$

$$W^T(3)X_3 = [0,0,1] \begin{bmatrix} -1 \\ 0 \\ -1 \end{bmatrix} = -1 \leq 0, \quad \text{So } W(4) = W(3) + X_3 = [-1,0,0]^T$$

$$W^T(4)X_4 = [-1,0,0] \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} = 1 > 0, \quad \text{So } W(5) = W(4) = [-1,0,0]^T$$

There are two cases of  $W^T(k)X_i \leq 0$  (misclassify), go to second run.

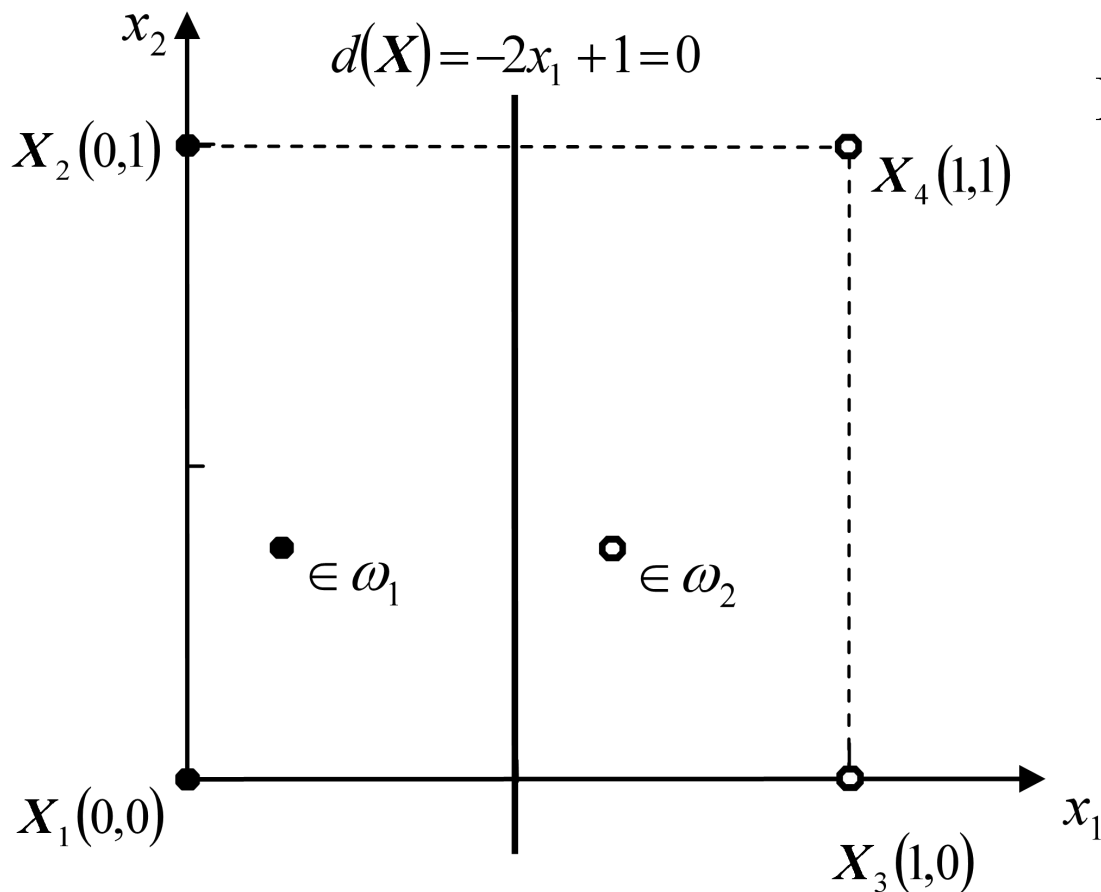
2<sup>nd</sup> run:  $W^T(5)X_1 = 0 \leq 0$ , So  $W(6) = W(5) + X_1 = [-1, 0, 1]^T$   
 $W^T(6)X_2 = 1 > 0$ , So  $W(7) = W(6) = [-1, 0, 1]^T$   
 $W^T(7)X_3 = 0 \leq 0$ , So  $W(8) = W(7) + X_3 = [-2, 0, 0]^T$   
 $W^T(8)X_4 = 2 > 0$ , So  $W(9) = W(8) = [-2, 0, 0]^T$

3<sup>rd</sup> run:  $W^T(9)X_1 = 0 \leq 0$ , So  $W(10) = W(9) + X_1 = [-2, 0, 1]^T$   
 $W^T(10)X_2 = 1 > 0$ , So  $W(11) = W(10)$   
 $W^T(11)X_3 = 1 > 0$ , So  $W(12) = W(11)$   
 $W^T(12)X_4 = 1 > 0$ , So  $W(13) = W(12)$

4<sup>th</sup> run:  $W^T(13)X_1 = 1 > 0$ , So  $W(14) = W(13)$   
 $W^T(14)X_2 = 1 > 0$ , So  $W(15) = W(14)$   
 $W^T(15)X_3 = 1 > 0$ , So  $W(16) = W(15)$   
 $W^T(16)X_4 = 1 > 0$ , So  $W(17) = W(16)$

All are classified correctly, So  $W = [-2, 0, 1]^T$

Discriminant function:  $d(X) = -2x_1 + 1$

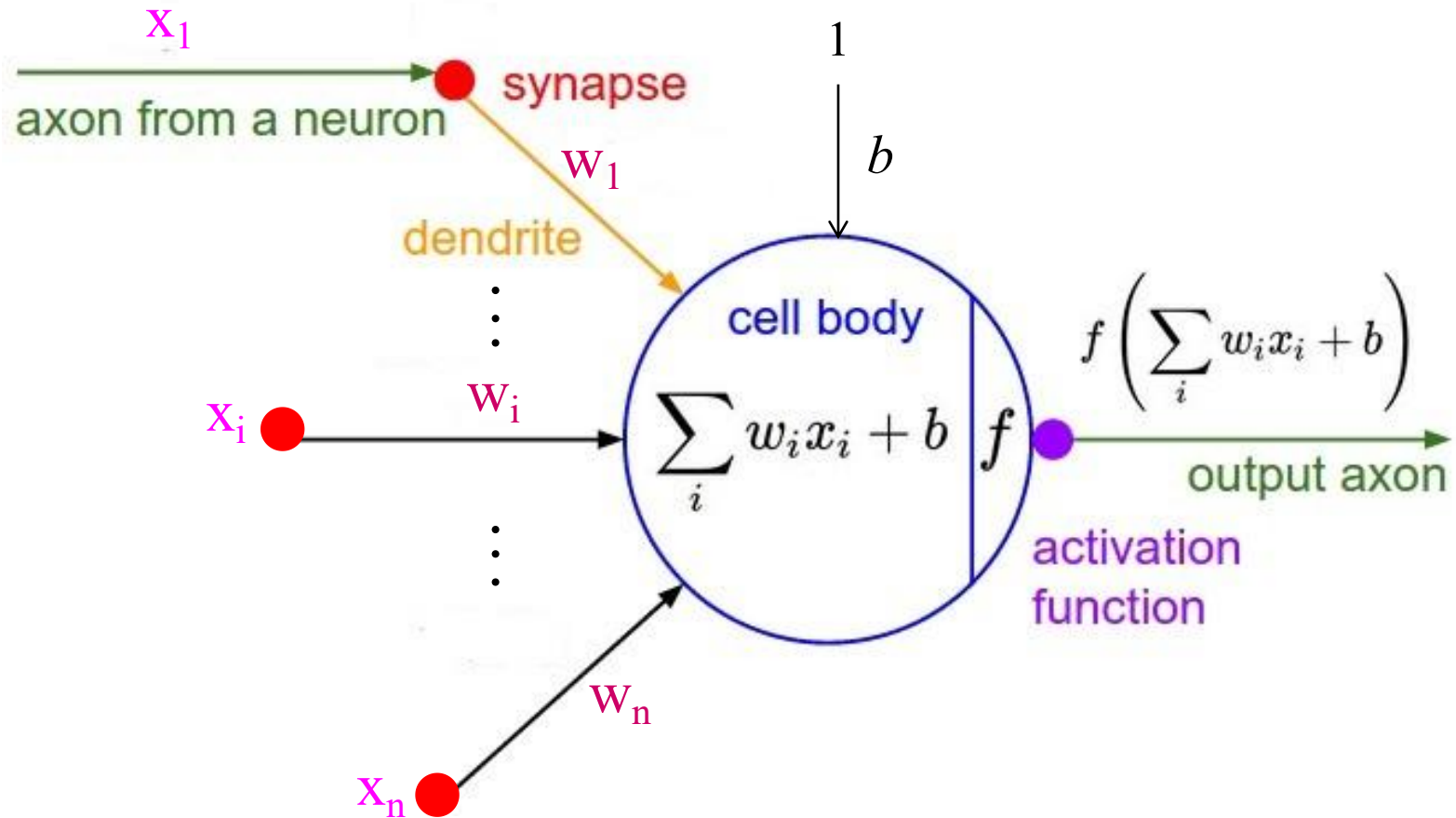


Discri.face  $d(X)=0$

if take  $\rho$  、  $W(1)$  into  
other values, result will not  
be the same.

Solution is **not unique**.

# Summary of the Neuro and Single-Layer Perceptron:



**input** : dendrite

**output**: axon

$\Sigma + f$ (**processing unit**): cell body

**w**: synapse



## 11.3 BP神经网络模型

---

### 一.BP神经网络简介

#### 1. 多层感知器 (MLP)

单层感知器网络只能解决线性可分问题。在单层感知器网络的输入层和输出层之间加入一层或多层感知器单元(层中可以有多个感知器)作为隐含层，就构成了多层感知器 (Multilayer Perceptrons, MLP)。多层感知器可以解决线性不可分的输入向量的分类问题。这种由输入层、隐含层和输出层所构成的感知器神经网络就是多层前向神经网络，BP神经网络是其中的一种。

## 2. BP神经网络简介

BP神经网络是采用误差反向传播 (Back Propagation, BP) 算法的多层前向网络，其中，神经元的传输函数为S型函数，网络的输入和输出是一种非线性映射关系。

BP算法的学习过程由结果正向传播和误差反向传播组成。在正向传播过程中，输入模式从输入层经隐含层逐层处理后，传送至输出层。每一层神经元的状态只影响下一层神经元的状态。如果在输出层得不到期望输出，那么就转为反向传播，把误差信号沿原连接路径返回，并通过修改各层神经元的权值，使误差减小。

## 二. BP神经网络模型

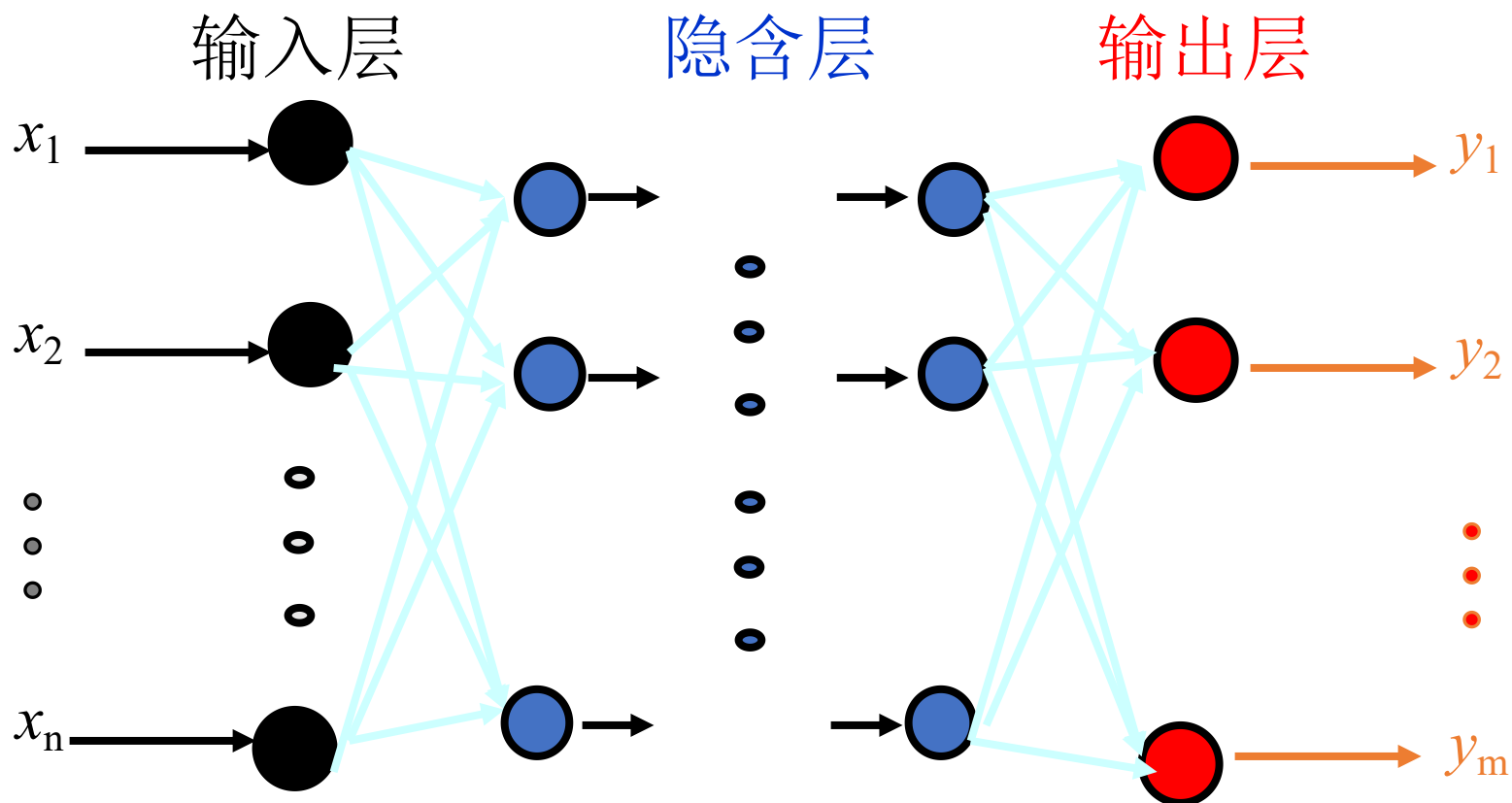


图 BP神经网络模型

BP网络为多层前向神经网络。

层与层之间多采用全互连方式，但同一层的节点之间不存在相互连接。

神经元的传输函数是可微的，早期的大多数设计者通常会采用S型(Sigmoid)函数。

神经元 $neu$ (neuron)的输入：

$$neu = x_1w_1 + x_2w_2 + \dots + x_nw_n + b$$

神经元 $neu$ 的输出：

$$y = f(neu) = \frac{1}{1 + e^{-neu}}$$

$$f'(neu) = -\frac{1}{(1 + e^{-neu})^2}(-e^{-neu})$$

$$= \frac{1 + e^{-neu} - 1}{(1 + e^{-neu})^2}$$

$$= \frac{1}{1 + e^{-neu}} - \frac{1}{(1 + e^{-neu})^2}$$

$$= y - y^2 = y(1 - y)$$

注意到:  $\lim_{neu \rightarrow +\infty} \frac{1}{(1 + e^{-neu})} = 1, \lim_{neu \rightarrow -\infty} \frac{1}{(1 + e^{-neu})} = 0$

根据S型传输函数可知,  $y$  的值域为 $(0,1)$ , 从而 $f'(neu)$  的值域为 $(0,0.25)$ , 见下图所示。

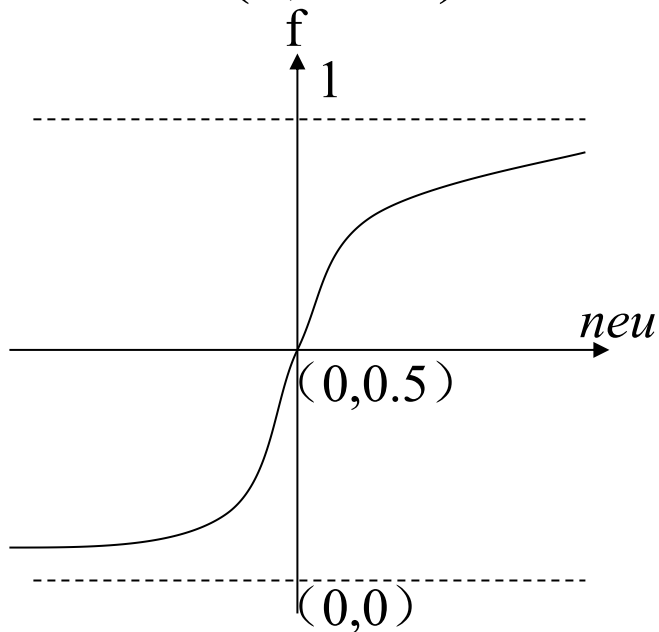


图 S型传输函数

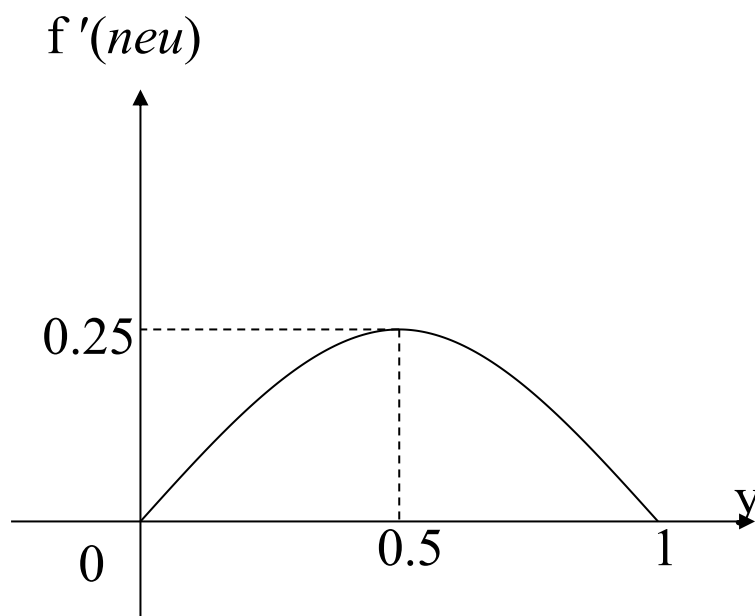


图  $f'(neu)$  的图像

由上图可知，对神经网络进行训练，应该将 $neu$ 的值尽量控制在收敛比较快的范围内。实际上，也可以用其他函数作为BP网络神经元的传输函数，只要该函数满足可导的条件即可。

在现代机器学习技术中，与Sigmoid、tanh等激活函数相比，ReLU函数是MLP(含BP网络)、深度神经网络隐含层更为广泛使用的激活函数。

### 三. BP网络训练过程

首先，上一讲已提到，ANNs的学习过程是根据模式样本集对神经元之间的连接权值进行调整的过程，BP网络也不例外。其次，BP网络执行是有导师学习。因此，其样本集是由形如：

(输入向量，理想输出向量)

的向量对构成的。

在开始训练前，所有的权都应用一些不同的小随机数进行初始化。“小随机数”用来确保网络不会因为权值过大而进入饱和状态，从而导致训练失败；“不同”用来保证网络可以正常学习。事实上，如果用相同的数去初始化权值矩阵，则网络无学习能力。



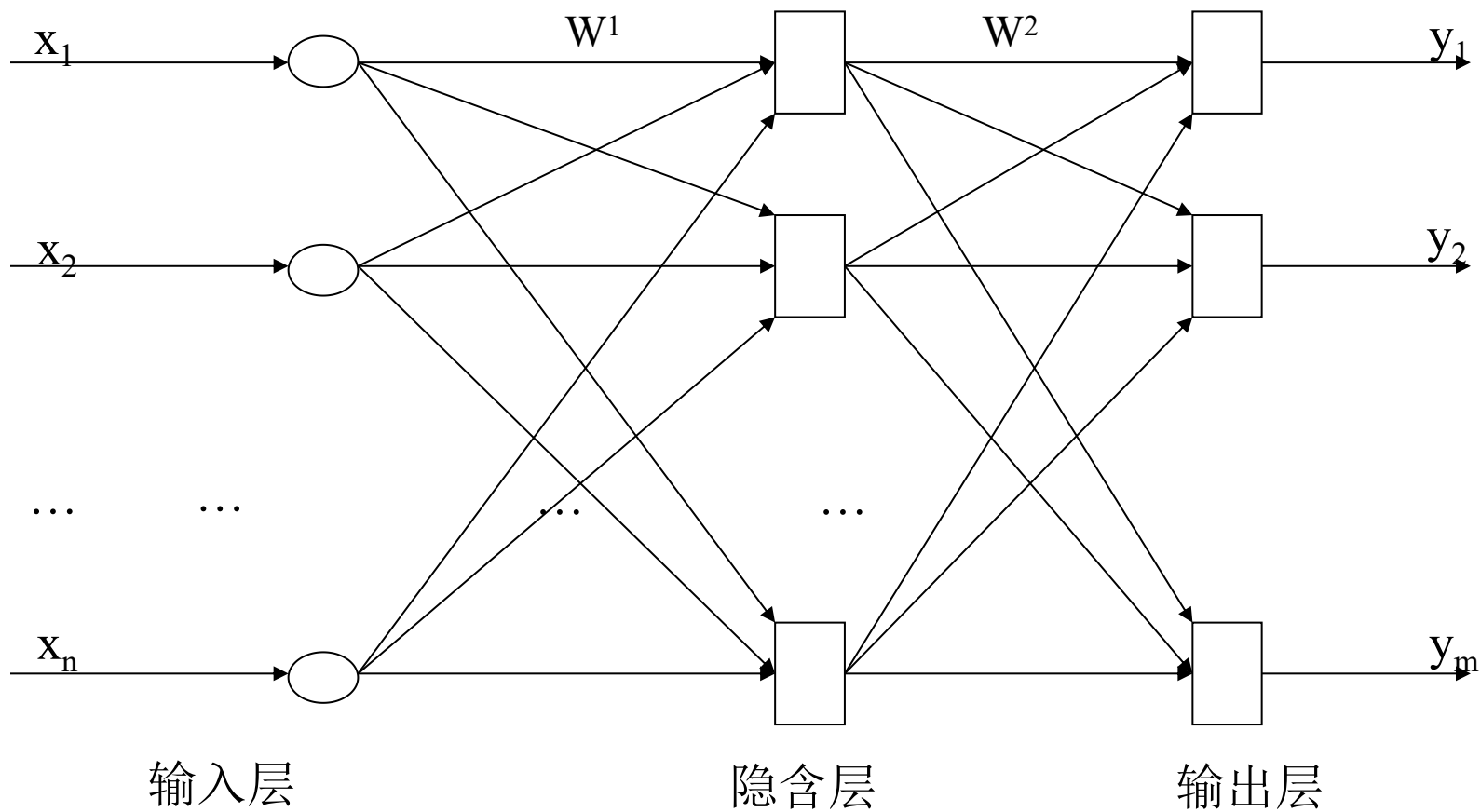


图 3(2层)层BP神经网络

BP基本算法主要包含4步，这4步被分成两个阶段：

## 1.前向(正向)传播输出阶段

Step 1: 给定输入样本  $\mathbf{X}_p = (x_1, x_2, \dots, x_n)^T$  和理想输出  $\mathbf{d}_p = (d_1, d_2, \dots, d_m)^T$ ；将 $\mathbf{X}_p$ 输入到网络。

Step 2: 计算相应的实际输出 $y_p$ ：

$$y_p = f_L(W^L \dots (f_2(W^2 f_1(W^1 X_p + b^1) + b^2) \dots) + b^L)$$

这里， $L$ 为网络层数， $W^i$ 为第 $i$ 层的权值矩阵 ( $i=1, 2, \dots, L$ )， $f_i$ 为第 $i$ 层神经元的传输函数。

## 2. 向后(误差反向)传播(修正权值)阶段

Step 1: 计算实际输出 $y_p$ 与相应的理想输出 $d_p$ 的差;

Step 2: 按极小化误差方式调整权值矩阵。

这两个步骤的工作一般要受到精度要求(期望误差 $\varepsilon$ )的控制, 这里取均方误差(损失)函数:

$$E_p = \frac{1}{2} \sum_{j=1}^m (d_j - y_j)^2 \quad \text{作为网络关于第} p \text{个样本的误差测度。}$$

将整个网络样本集的误差测度定义为:

$$E = \sum_p E_p$$

## 四. 误差反向传播原理分析

### 1. 输出层权值调整

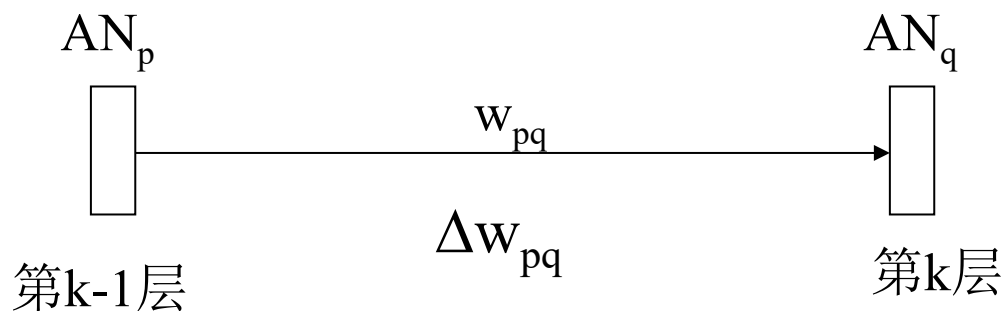


图  $AN_p$ 到 $AN_q$ 的连接

$$w_{pq} = w_{pq} + \Delta w_{pq}$$

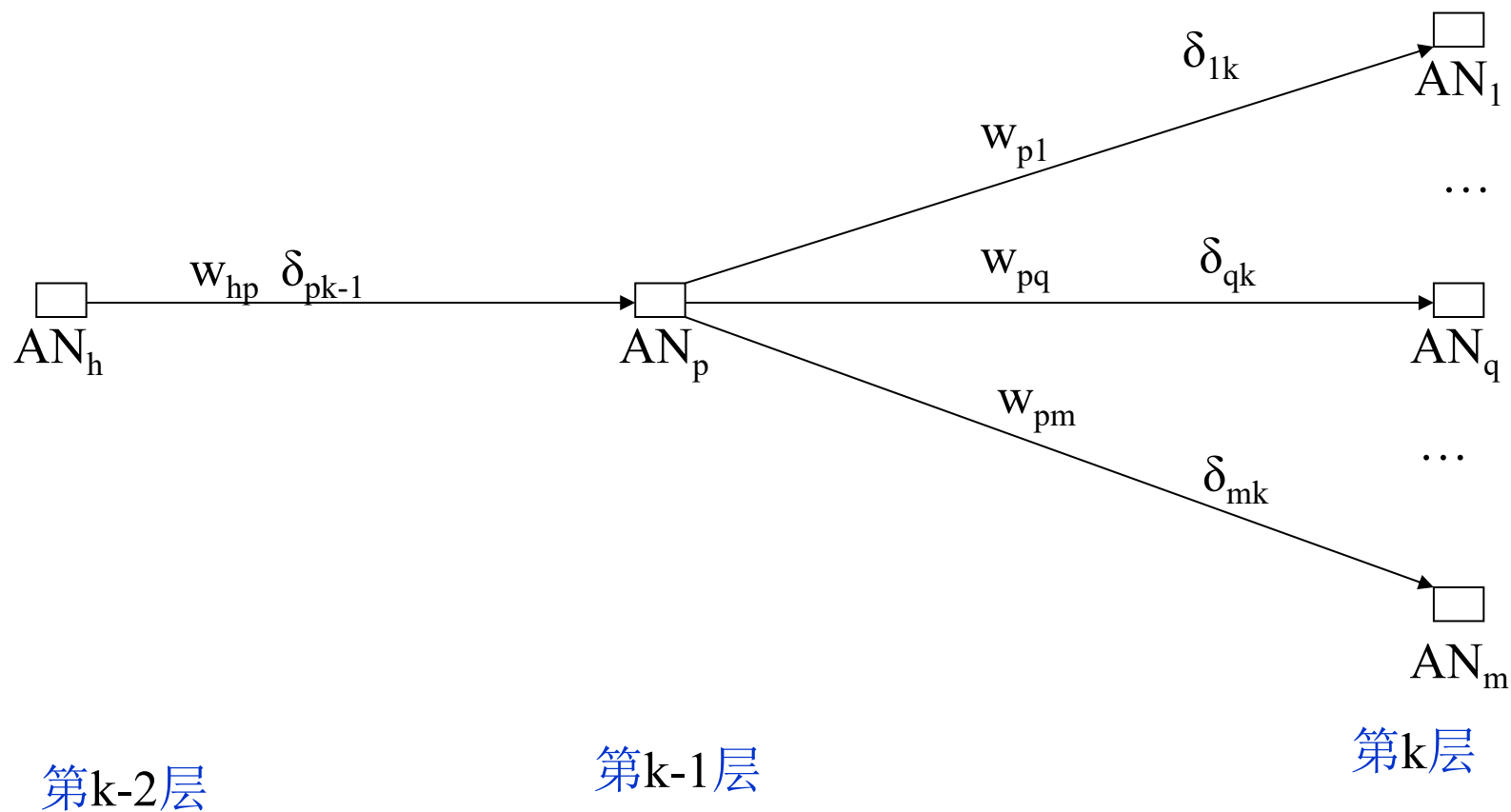
$$\Delta w_{pq} = \eta \delta_{qk} y_p$$

$$= \eta f'_n(\text{neu}_q)(d_q - y_q)y_p$$

$$= \eta y_q(1 - y_q)(d_q - y_q)y_p$$

基于梯度下降法的 $\delta$ 学习规则  
(详见前面的基本学习算法)

## 2. 隐含层权值调整



$\delta_{pk-1}$  的值和  $\delta_{1k}, \delta_{2k}, \dots, \delta_{mk}$  有关

不妨认为  $\delta_{pk-1}$

通过权值  $w_{p1}$  对  $\delta_{1k}$  做出贡献,

通过权值  $w_{p2}$  对  $\delta_{2k}$  做出贡献,

.....

通过权值  $w_{pm}$  对  $\delta_{mk}$  做出贡献。

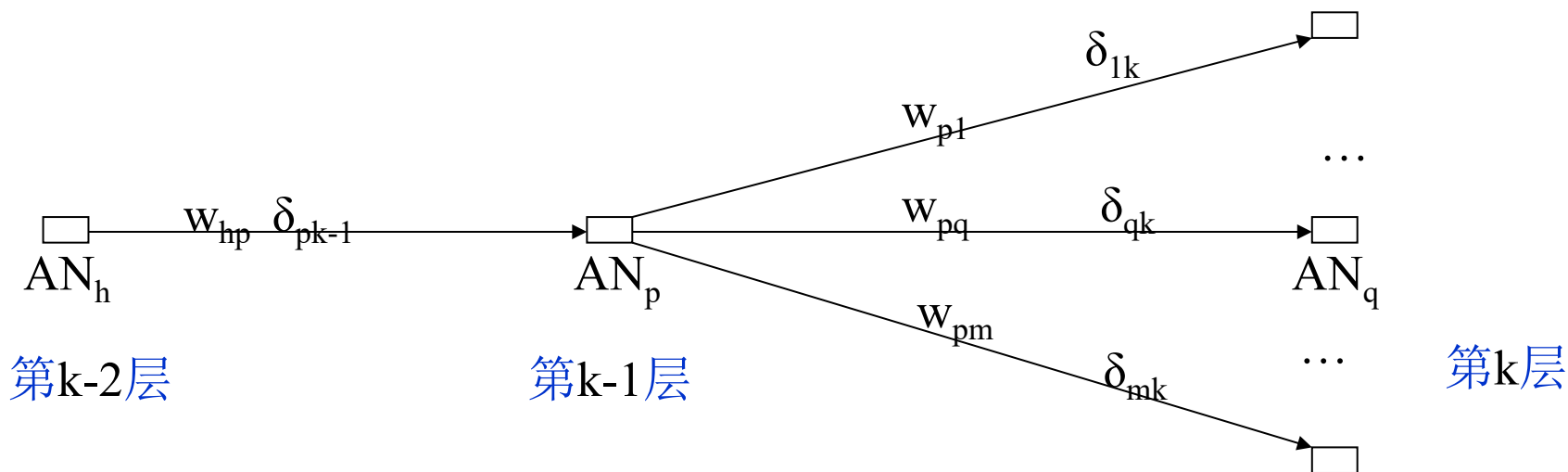
$$\delta_{pk-1} = f_{k-1}'(\text{neu}_p) (w_{p1}\delta_{1k} + w_{p2}\delta_{2k} + \dots + w_{pm}\delta_{mk})$$

$$w_{hp} = w_{hp} + \Delta w_{hp}$$

$$\Delta w_{hp} = \eta \delta_{pk-1} y_{hk-2}$$

$$= \eta f'_{k-1}(\text{neu}_p) (w_{p1} \delta_{1k} + w_{p2} \delta_{2k} + \dots + w_{pm} \delta_{mk}) y_{hk-2}$$

$$= \eta y_{pk-1} (1 - y_{pk-1}) (w_{p1} \delta_{1k} + w_{p2} \delta_{2k} + \dots + w_{pm} \delta_{mk}) y_{hk-2}$$



## 五. 基本的BP算法思想

模式样本集： $S=\{(x_1,d_1),(x_2,d_2),\dots,(x_N,d_N)\}$

基本的BP算法思想：

(1)逐一地根据样本集中的样本 $(x_k,d_k)$ 计算出实际输出 $y_k$ 和误差测度 $E_p$ ，对 $W^1$ ， $W^2$ ， $\dots$ ， $W^L$ 各做一次调整，重复这个循环，直到 $\sum E_p < \varepsilon$ 。

(2)用输出层的误差调整输出层权值矩阵，并用此误差估计输出层的直接前导层的误差，再用输出层前导层误差估计更前一层的误差。如此获得所有其他各层的误差估计，并用这些估计实现对权值矩阵的修改。形成将输出端表现出的误差沿着与输入信号相反的方向逐级向输入端传递的过程。



基本的BP神经网络学习算法使用基于梯度下降法的 $\delta$ 学习规则(MATLAB使用traingd作为梯度下降学习算法标识符)，学习过程是通过调整权值和阈值，使输出期望值和神经网络的实际输出值的均方误差趋于最小实现的，但是它只用到均方误差函数对权值和阈值的一阶导数(梯度)信息，使得算法收敛速度较慢，易陷入局部极小等缺陷。

需特别说明的是，在现代机器学习中常使用交叉熵损失函数用于MLP (BP) 分类问题，常使用均方误差损失函数用于MLP (BP) 分类问题。

**损失函数的进一步说明：**损失函数(Loss function)又称成本函数、代价函数、目标函数或优化函数；损失函数不仅以衡量模型预测值与真实值偏离的程度，而且通过求解损失函数的最小值可实现求解模型参数、优化模型参数和评价模型参数学习效果的目的。

下面是常用的两种损失函数：

(1) **均方误差损失函数：**某样本 $x_i$ 预测值 $y_i$ 与其真实值 $d_i$ 差值的平方定义为**单样本损失** $L(y_i, d_i) = (y_i - d_i)^2$ ，评估 $m$ 个样本的损失用全部损失的均方误差损失表示为 $J(w, b) = 1/m * \text{Sum}[L(y_i, d_i)]$

(2) **交叉熵损失函数：**

以逻辑回归为例，**单样本交叉熵损失函数：**

$$L(y_i, d_i) = -[d_i \log(y_i) + (1 - d_i) \log(1 - y_i)]$$

**M个样本的交叉熵损失函数：**

$$J(w, b) = -1/m * \text{Sum}[d_i \log(y_i) + (1 - d_i) \log(1 - y_i)]$$

在以上两种损失函数中，**均方误差损失函数**常用于**回归问题**，**交叉熵损失函数**一般用于**分类问题**。各种流行的机器学习框架预定义了多种损失函数，实践中可灵活选择，也可根据问题需要自定义损失函数。

## 六. BP神经网络分类器设计

在设计BP神经网络分类器时，需从网络的层数、每层包含的神经元数、初始权值以及学习速率等几方面考虑。

### 1.网络层数

已经证明：三层BP神经网络可以实现多维单位立方体 $R^n$ 到 $R^m$ 的映射，即能够逼近任意有理函数。这实际上给出了设计BP神经网络的基本原则。增加层数可以更进一步地降低误差、提高精度，但同时也使网络复杂化，从而增加网络权值的训练时间。而误差精度的提高实际上也可以通过增加隐含层中的神经元数目来获得，其训练结果也比增加层数更容易观察和调整。因此，一般情况下，应优先考虑增加隐含层的神经元数目。

## 2.隐含层的神经元数目

网络训练精度的提高，可以通过采用一个隐含层增加神经元数目获得，这在结构实现上要比增加更多的隐含层简单得多。

隐含层神经元数目的确定是目前没有解决的一个难题，一般根据经验和实验确定。

## 3.初始权值的选取

由于系统是非线性的，初始权值的选取对于学习是否达到局部极小、是否能够收敛以及训练时间的长短有很大的关系。

初始权值过大、太小都会影响学习速度，初始仅值一般可取较小的随机数。

## 4.学习速率

过大的学习速率可能导致系统不稳定，过低的学习速率导致较长的训练时间，可能收敛很慢。在一般情况下，倾向于选取较小的学习速率以保证系统的稳定性。

学习速率  $\eta$  一般在0.01~0.8之间选取。

## 七. 隐含层神经元数目的讨论(\*: 选学)

BP网络隐含层神经元数目对网络性能影响很大，因此需要适当地选取。

1988年Cybenko指出，当各神经元均采用S型函数时，一个隐含层就足以解决任意分类问题；两个隐含层则足以实现输入的任意函数；在线性可分情况下，不需要隐含层(即采用单层感知器即可)。

设 $n$ 、 $m$ 、 $h$ 分别表示输入节点数(输入向量维数)、输出神经元数目和隐含神经元数， $N$ 为训练样本数。

## 1.单隐含层

(1)  $h = 2n + 1$ ;

(2) 对于医疗诊断神经网络系统，通常 $n$ 较大，

$$h_{\max} = m(n + 1) \text{ 或 } h_{\max} = 3m ;$$

(3) 对于 $n > m$ 小型神经网络系统,  $h = \sqrt{n \cdot m}$  ;

(4) 在图像识别中，当输入维数较多时，  $h_{\min} = 0.02 \times n$  ;

(5) 当用于统计过程控制时，  $h = 4n$  ;

(6) 当用于数据压缩时，  $h = \log(2n)$  ;

(7)  $h = \sqrt{n + m} + a, a \in [1, 10]$ 。

## 2.双(多)隐含层

- (1)用于图像识别时，第二隐含层神经元数目 $h=2m$ ；
- (2)当为高维输入向量时，第一隐含层对第二隐含层的比例为3:1。

## 3.隐含神经元数目与训练样本数的关系

- (1) $h = \log_2 N$ ；
- (2)训练次数最小的隐含层最佳神经元数目 $h \approx N-1$ 。

以上是从一些文献中收集的确定隐含层神经元数目的经验性结论，供大家设计神经网络模型时参考。需要特别强调的是，由于应用的目的和条件不同，所得结果彼此有所不同。根据实际经验，在实际工程应用问题时，首先是选择已有的经验公式大致确定隐含层神经元数目，然后在实验中反复对隐含层神经元数目进行调整，以使神经网络达到良好的工作性能。

## 11.4 神经网络的发展-深度网络简介

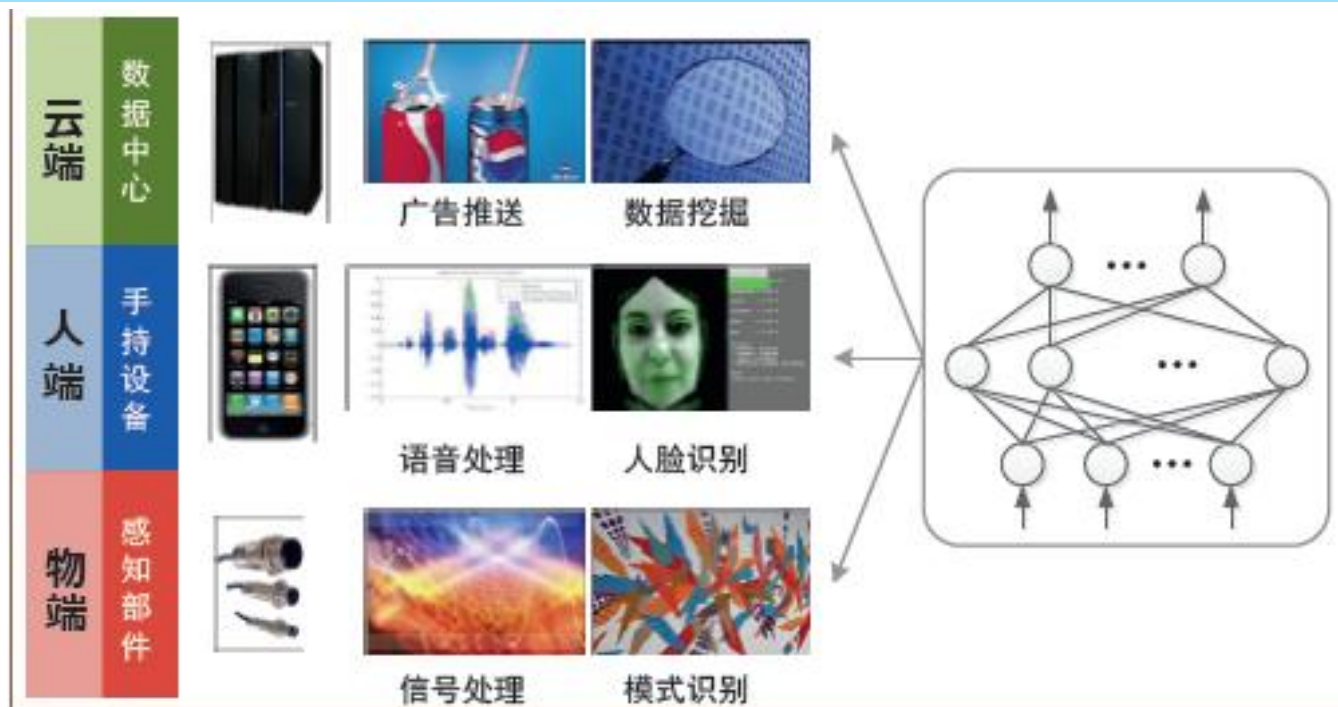


图 人工神经网络的应用覆盖了云端、人端和物端

学术界已经提出了许多不同的人工神经网络结构模型和算法。这些方法在对很多图像、声音和文字的理解和分析上表现出了很好的性能。无论是云端的大数据处理(如广告推荐、数据挖掘和视频图像自动标记), 还是个人移动终端任务(如语音识别和自动翻译), 都能看到人工神经网络的身影。



以深度信念网络(Deep Belief Network, **DBN**)、卷积神经网络(Convolutional Neural Network, **CNN**)、循环神经网络(Recurrent Neural Network, **RNN**)为代表的“深度学习”(Deep Learning, 一种多层的神经网络)是当前学术界和工业界流行的名词, 典型的深度学习模型见所示。如, Google、微软、科大讯飞、百度利用深度神经网络在语音识别和图像处理方面做了许多有价值的工作, 很多工作已经进入产品化阶段。2011 年, Google和斯坦福大学合作构建了一个有10 亿个突触(Synapse)的深度神经网络来进行猫脸识别。他们用16000 个处理器核花了好几天时间训练此神经网络, 使得其对猫脸识别的准确率比前人最好的结果提高了70%。

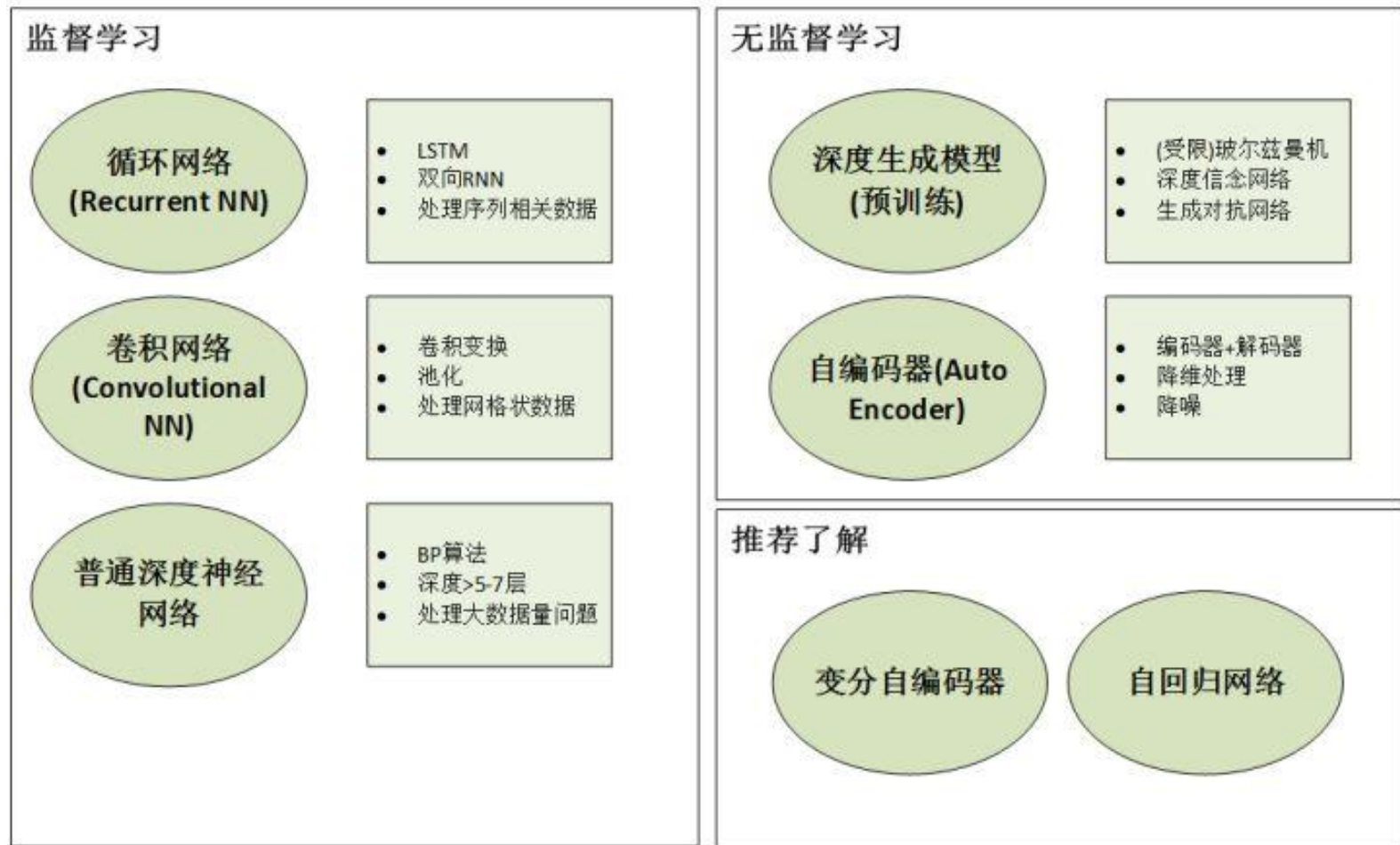
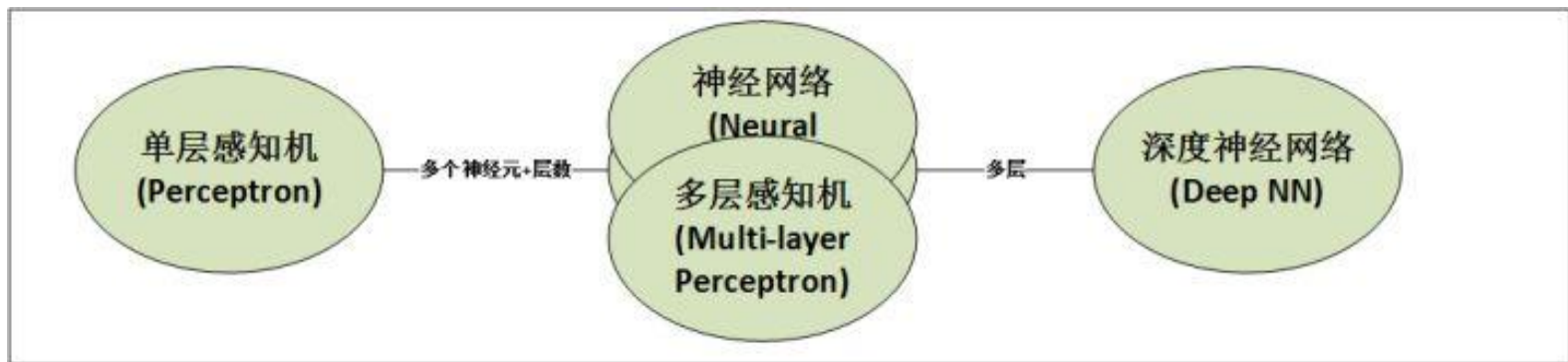


图 典型的深度学习模型

# 11.5 神经网络Python程序举例

---

## 一. 感知器Python程序举例

例：鸢尾花数据集共有150个样本(0-149行)、4个特征(0,1,2,3列)，数据集分成3类；加载鸢尾花数据集、取花瓣特征(即第2、3列特征: petal length, petal width)；采用感知器算法对鸢尾花进行分类(70%用于训练集、30%用于测试集)，试编写感知器Python程序。

程序清单：

#Filename: perceptron\_iris.ipynb

#读取数据

from sklearn import datasets

import numpy as np

iris = datasets.load\_iris()

X = iris.data[:,[2,3]] #鸢尾花数据集有4个特征(0,1,2,3列)，

取鸢尾花数据集中的第2、3列特征(petal length, petal width)

y = iris.target #鸢尾花模式样本标签(150个模式样本)

# 训练数据和测试数据分为7:3

```
from sklearn.model_selection import train_test_split
```

```
x_train,x_test,y_train,y_test
```

```
=train_test_split(X,y,test_size=0.3,random_state=0)
```

# 标准化数据

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

#Fit only to the training data

```
sc.fit(x_train)
```

#Now apply the transformations to the data

```
x_train_std = sc.transform(x_train)
```

```
x_test_std = sc.transform(x_test)
```

```
# 导入sklearn的Perceptron, 用fit训练perceptron模型
from sklearn.linear_model import Perceptron
ppn=Perceptron(max_iter=40,eta0=0.01,random_state=0)
#train perceptron model
ppn.fit(x_train_std,y_train)
y_pred = ppn.predict(x_test_std)
miss_classified = (y_pred != y_test).sum()
print("错误分类数: ",miss_classified)
from sklearn.metrics import accuracy_score
print('准确率: %.2f' % accuracy_score(y_test, y_pred))
```

#画超平面(可选的可视化部分)

```
from matplotlib.colors import ListedColormap
```

```
import matplotlib.pyplot as plt
```

```
def plot_decision_regions(X, y, classifier, test_idx=None, resolution=0.02):
```

```
    # setup marker generator and color map
```

```
    markers = ('s', 'x', 'o', '^', 'v')
```

```
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
```

```
    cmap = ListedColormap(colors[:len(np.unique(y))])
```

```
    # plot the decision surface
```

```
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
```

```
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
```

```
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),  
                           np.arange(x2_min, x2_max, resolution))
```

```
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
```

```
    Z = Z.reshape(xx1.shape)
```

```
    plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
```

```
    plt.xlim(xx1.min(), xx1.max())
```

```
    plt.ylim(xx2.min(), xx2.max())
```

```
    for idx, cl in enumerate(np.unique(y)):
```

```
        plt.scatter(x=X[y == cl, 0],
```

```
                    y=X[y == cl, 1],
```

```
                    alpha=0.8,
```

```
                    c=colors[idx],
```

```
                    marker=markers[idx],
```

```
                    label=cl,
```

```
                    edgecolor='black')
```

#高亮测试样本点

if test\_idx:

# plot all samples

X\_test, y\_test = X[test\_idx, :], y[test\_idx]

plt.scatter(X\_test[:, 0],  
 X\_test[:, 1],  
 c="",  
 edgecolor='black',  
 alpha=1.0,  
 linewidth=1,  
 marker='o',  
 s=100,  
 label='test set')

#htack vstack: 水平叠加及垂直叠加

```
X_combined_std = np.vstack((x_train_std, x_test_std))
```

```
y_combined = np.hstack((y_train, y_test))
```

```
plot_decision_regions(X=X_combined_std, y=y_combined,  
                      classifier=ppn, test_idx=range(105,150))
```

```
plt.xlabel('petal length [standardized]')
```

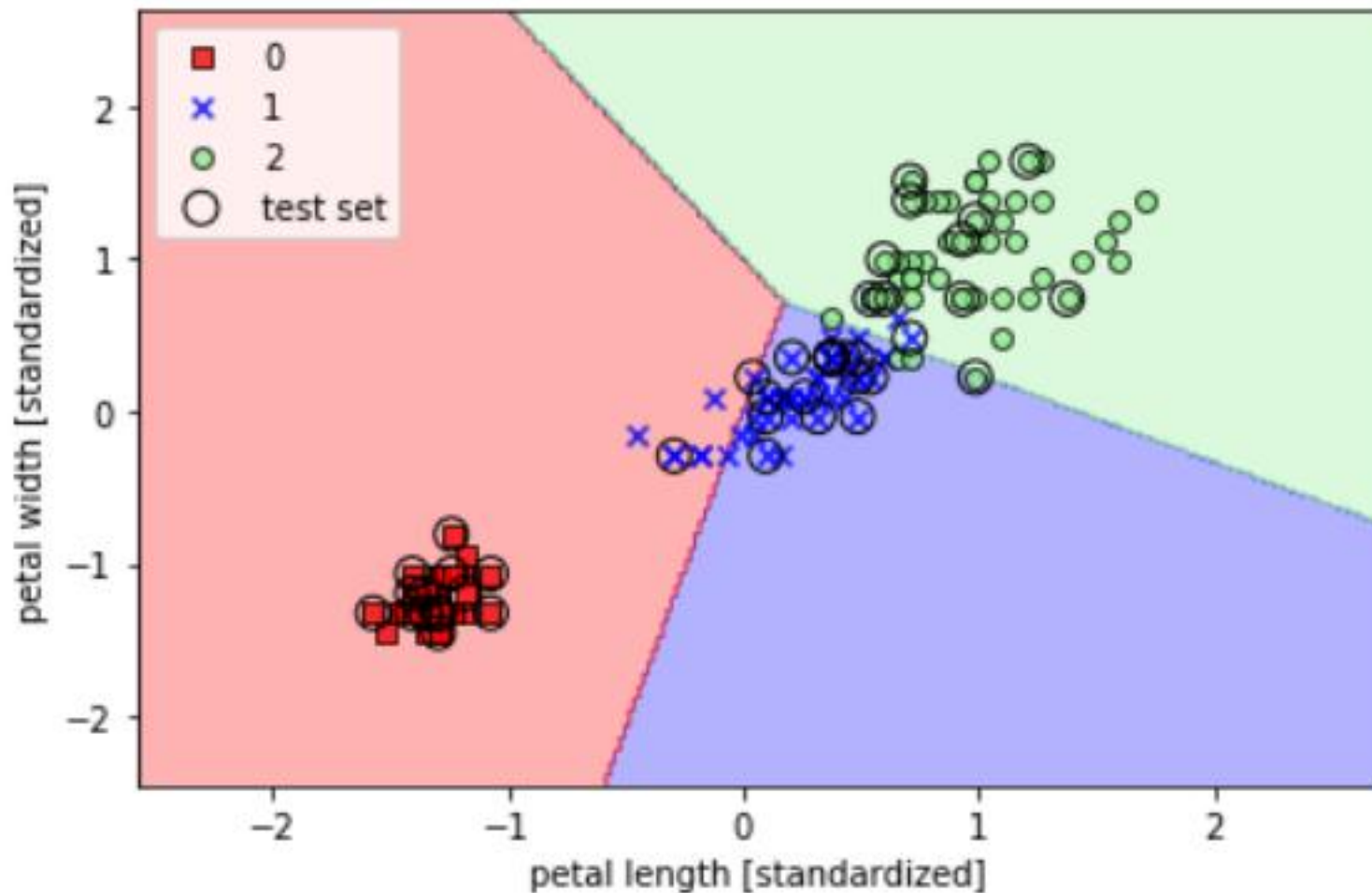
```
plt.ylabel('petal width [standardized]')
```

```
plt.legend(loc='upper left')
```

```
plt.tight_layout()
```

```
plt.show()
```





说明：从本图可以看出，打圈的测试集中有4个样本点分类错误，2个1类样本错分到0类，1个1类样本错分到2类，1个2类样本错分到1类。

由于感知机只能对线性数据进行正确分类，因而针对非线性的鸢尾花数据集，采用感知器分类必然会发生一定数量的样本错分情况；对于非线性数据，我们可采用逻辑回归等分类方法。下面是采用逻辑回归(logistic regression )对鸢尾花花瓣特征数据进行3分类的核心代码：

```
#scikit-learn实现逻辑回归
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(C = 100.0, random_state = 1)
lr.fit(x_train_std,y_train)
plot_decision_regions(X=X_combined_std,
                      y = y_combined,
                      classifier = lr,
                      test_idx = range(105,150) )

plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.tight_layout()
#plt.savefig('images/04_17.png', dpi=300)
plt.show()
```

## 二. 多层感知器Python程序举例

例：鸢尾花数据集共有150个样本(0-149行)、4个特征(0,1,2,3列)，数据集分成3类；加载鸢尾花数据集、取花瓣特征(即第2、3列特征: petal length, petal width)；采用感知器算法对鸢尾花进行分类(70%用于训练集、30%用于测试集)，试编制多层神经网络Python程序。

程序清单：

```
#Filename: mlp_iris.ipynb
```

```
#读取数据
```

```
from sklearn import datasets
```

```
import numpy as np
```

```
iris = datasets.load_iris()
```

```
X = iris.data[:,[2,3]] #鸢尾花数据集有4个特征(0,1,2,3列)，取  
鸢尾花数据集中的第2、3列特征(petal length, petal width)
```

```
y = iris.target      #鸢尾花模式样本标签(150个模式样本)
```

#训练数据和测试数据分为7:3

#from sklearn.cross\_validation import train\_test\_split

from sklearn.model\_selection import train\_test\_split

x\_train,x\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.3,random\_state=0)

# 标准化数据

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

#Fit only to the training data

sc.fit(x\_train)

#Now apply the transformations to the data

x\_train\_std = sc.transform(x\_train)

x\_test\_std = sc.transform(x\_test)

#导入sklearn的多层感知器分类器模型(MLP), 用fit训练MLP模型

from sklearn.neural\_network import MLPClassifier

#solver/optimizer: lbfgs-Quasi Newton method(拟牛顿法)

mlp=MLPClassifier(solver='lbfgs',activation='tanh',hidden\_layer\_sizes=(100,100),max\_iter=10000)

#train MLP model

mlp.fit(x\_train\_)

miss\_classified = (y\_std,y\_train)

y\_pred=mlp.predict(x\_test\_stdpred != y\_test).sum()

print("错误分类数: ",miss\_classified)

from sklearn.metrics import accuracy\_score

print('准确率: %.2f % accuracy\_score(y\_test, y\_pred))

## Scikit-learn **MLPClassifier**分类器常见参数说明:

(1) **hidden\_layer\_sizes**: 如hidden\_layer\_sizes=(100, 100), 表示有两层隐含层, 第一层隐含层有100个神经元, 第二层有100个神经元

(2) **activation**: 激活函数, {'identity', 'logistic', 'tanh', 'relu'}, 默认relu

- identity:  $f(x)=x$

- logistic: 就是sigmoid,  $f(x)=1/(1+\exp(-x))$

- tanh:  $f(x)=\tanh(x)$

- relu:  $f(x)=\max(0, x)$

(3) **solver**: {'lbfgs', 'sgd', 'adam'}, 默认adam, 用来优化权重

- lbfgs: quasi-Newton法优化器

- **sgd**: 随机梯度下降法优化器

解释: 传统的梯度下降方法是每次更新w都需要遍历所有data, 当数据量太大或者一次无法获取全部数据时, 此法并不可行; 一种解决该问题基本思路是只**通过**一个**随机选取的数据**( $x_i, y_i$ )来获取“**梯度**”, 以此对**w进行更新**, 这种优化方法称为**随机梯度下降法**(SGD, Stochastic Gradient Descent)。

- adam: Kingma, Diederik等于2015年提出的一种改进随机梯度优化器, 是sgd优化器的扩展

注意: 默认solver adam在相对较大的数据集上效果比较好 (几千个样本或者更多), 对小数据集来说, lbfgs收敛更快、效果更好。

(4) **alpha**: float, 可选的正则化项参数, 是一个L2惩罚项, 用于控制正则化的程度, 默认值为0.0001

.....

#以下是可选的可视化部分

#画超平面

```
from matplotlib.colors import ListedColormap
```

```
import matplotlib.pyplot as plt
```

```
def plot_decision_regions(X, y, classifier, test_idx=None, resolution=0.02):
```

```
    # setup marker generator and color map
```

```
    markers = ('s', 'x', 'o', '^', 'v')
```

```
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
```

```
    cmap = ListedColormap(colors[:len(np.unique(y))])
```

```
    # plot the decision surface
```

```
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
```

```
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
```

```
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),  
                           np.arange(x2_min, x2_max, resolution))
```

```
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
```

```
    Z = Z.reshape(xx1.shape)
```

```
    plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
```

```
    plt.xlim(xx1.min(), xx1.max())
```

```
    plt.ylim(xx2.min(), xx2.max())
```

```
    for idx, cl in enumerate(np.unique(y)):
```

```
        plt.scatter(x=X[y == cl, 0],
```

```
                    y=X[y == cl, 1],
```

```
                    alpha=0.8,
```

```
                    c=colors[idx],
```

```
                    marker=markers[idx],
```

```
                    label=cl,
```

```
                    edgecolor='black')
```

#高亮测试样本点

```
if test_idx:
```

```
    # plot all samples
```

```
    X_test, y_test = X[test_idx, :], y[test_idx]
```

```
    plt.scatter(X_test[:, 0],  
                X_test[:, 1],  
                c="",  
                edgecolor='black',  
                alpha=1.0,  
                linewidth=1,  
                marker='o',  
                s=100,  
                label='test set')
```

#htack vstack: 水平叠加及垂直叠加

```
X_combined_std = np.vstack((x_train_std, x_test_std))
```

```
y_combined = np.hstack((y_train, y_test))
```

```
plot_decision_regions(X=X_combined_std, y=y_combined,  
                      classifier=mlp, test_idx=range(105,150))
```

```
plt.xlabel('petal length [standardized]')
```

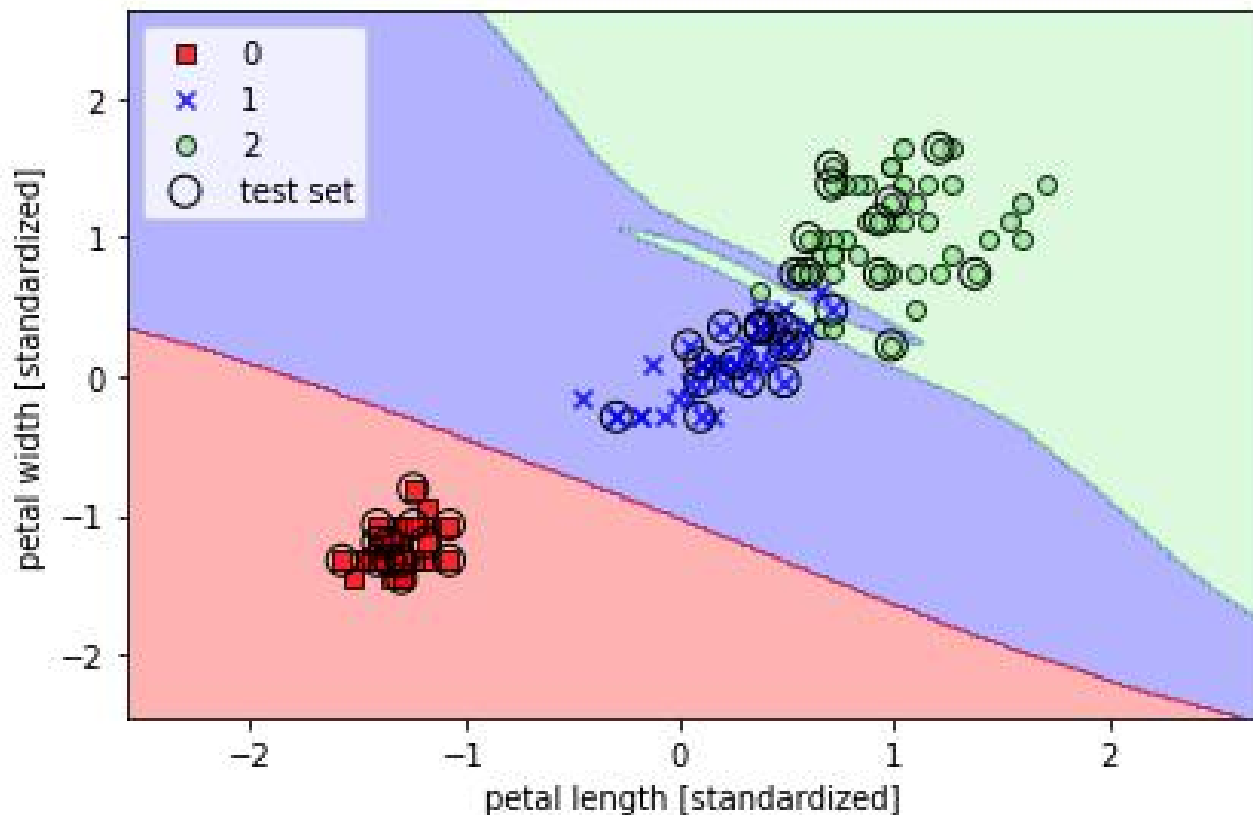
```
plt.ylabel('petal width [standardized]')
```

```
plt.legend(loc='upper left')
```

```
plt.tight_layout()
```

```
plt.show()
```



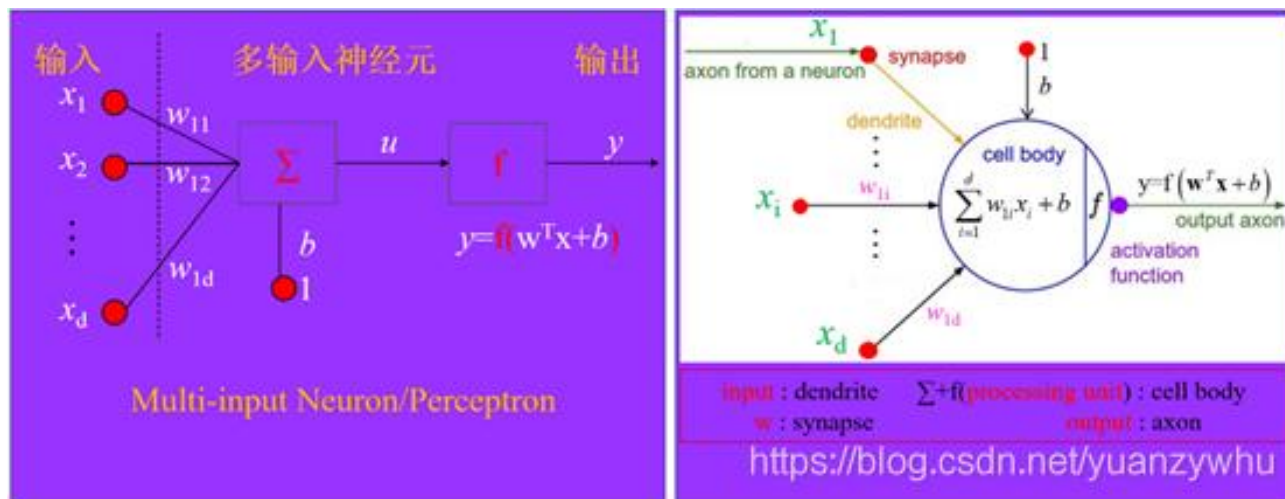


说明：从本图可看出，打圈的测试集中无样本点错误分类(最理想的情况)。表明MLP比前面的单层线性感知器Python程序的分类准确率要高出许多。

常见的几种方法调整MLP神经网络模型复杂度：(1)调整NN每层隐含层结点数；(2)调节NN隐含层的层数；(3)调节activation的激活函数类型；(4)调整alpha值控制NN模型正则化的程度 (模型复杂度控制)。[实验略]

# 本章小结:

## 1. 感知器 (Perceptron)



感知器结构及感知器算法。

## 2. BP神经网络

人工神经网络可分为前向网络、具有反馈的前向网络、层内互连前向网络和全互连网络四种类型。**多层感知器** (Multilayer Perceptrons, **MLP**)属于前向网络，可用来解决线性不可分的输入向量的分类问题，BP神经网络是一种MLP。**反向传播**(BP)算法是一种迭代算法，每次迭代包含两个阶段：第1阶段是向前传播，第2阶段是(误差)反向传播(一种**基本BP算法**：链式求导法则+**梯度下降法**优化网络参数)。

## 3. 神经网络中的常用激活函数

Sigmoid、ReLU、Softmax等。

## 4. 神经网络Python编程

感知器Python编程、MLP Python编程及应用。

## 本章主要参考文献:

- [01]齐敏等. 模式识别导论, 清华版, 2009
- [02]李弼程等. 模式识别原理与应用, 西电版, 2008
- [03]孙即祥. 现代模式识别(第二版), 高教版, 2008
- [04]神经网络基础知识、常用激活函数及其Python图形绘制  
(<https://yuanyx.blog.csdn.net/article/details/110335071>)
- [05]**Softmax**激活函数计算解析及**MLP**多分类器应用  
(<https://yuanyx.blog.csdn.net/article/details/113738679>)

End of this lecture.

Thanks !

## 课外作业题11 (神经网络):

1. 画出人工神经元(单层感知器)示意图，并写出数学表达式。
2. 人工神经网络分为哪4种类型？
3. Hebb学习规则和 $\delta$ 学习规则的内容分别是什么？
4. 简述基本的BP算法思想。
5. BP网络神经元的激活函数(传输函数)有什么特殊要求？
6. 以ORL人脸数据集(每张人脸图像大小为 $112 \times 92$ )为例，说明采用PCA对人脸做特征提取并用BP神经网络进行人脸识别的技术方案。
7. 已知两类训练样本为

$$\omega_1: [0,0,0]^T, [1,0,0]^T, [1,0,1]^T, [1,1,0]^T$$

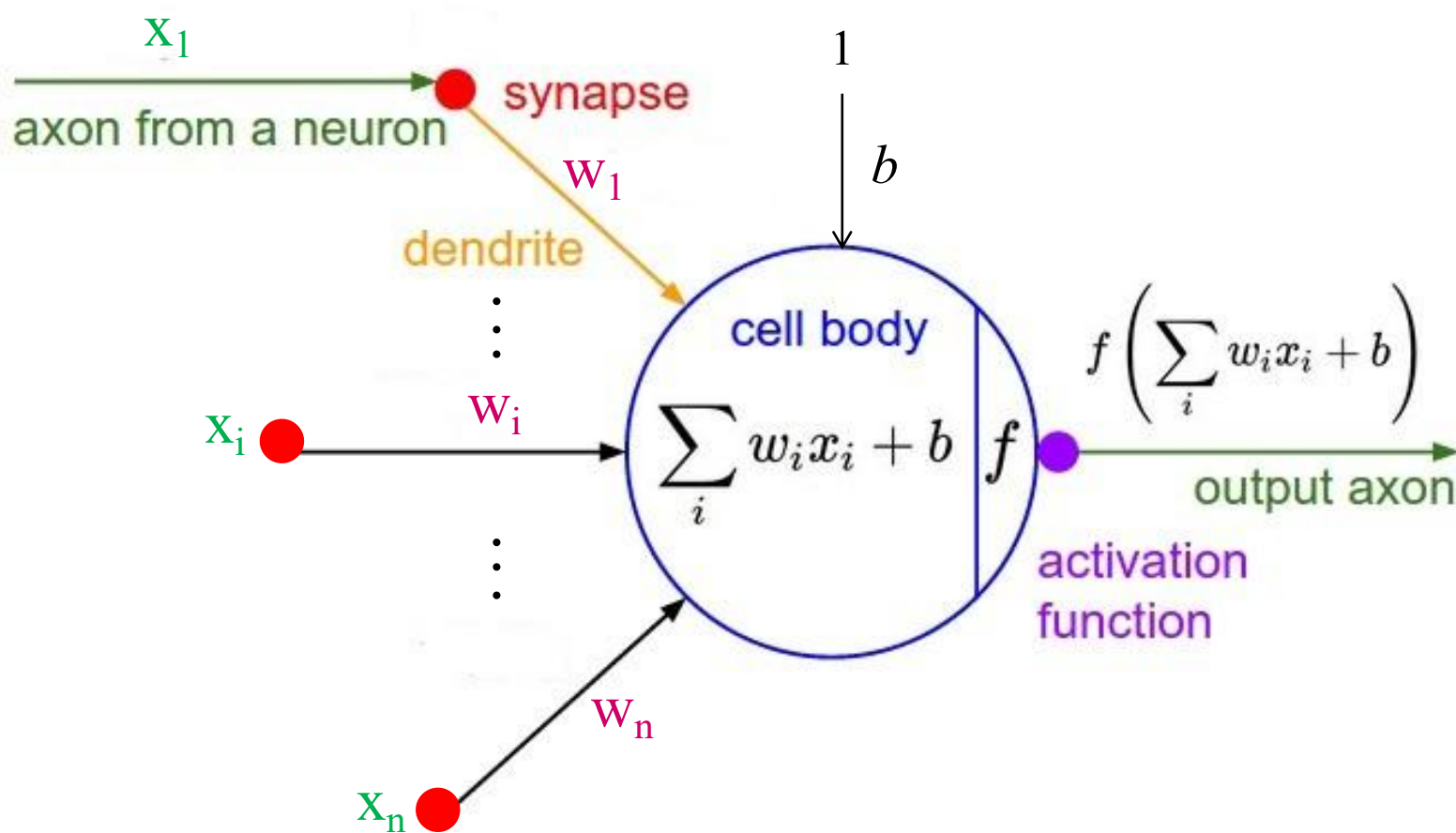
$$\omega_2: [0,0,1]^T, [0,1,1]^T, [0,1,0]^T, [1,1,1]^T$$

设  $W(1) = [-1, -2, -2, 0]^T$ ，用感知器算法求解判别函数，并绘出判别界面。

**课外单元编程作业4：**使用**ORL**人脸数据集 (网上下载或用课堂提供的**orl-faces.rar**文件)，编写采用**PCA**对人脸做特征提取并用**BP**神经网络进行人脸识别的**Python**程序并撰写实验报告。

## 课外作业11参考解答:

1.答



2.答: 前向网络, 具有反馈的前向网络, 层内互连前向网络, 全互连网络。

3.答：见本课件(此略)。

4.答：见本课件(此略)。

5.答：由于BP常用于解决非线性模式识别问题，故传输(激活)函数大多使用非线性函数，如S型函数、ReLU函数等。在现代机器学习技术中，ReLU函数是MLP(含BP网络)、深度神经网络使用更为广泛的激活函数。

6.答(要点)：(1)采用PCA对112x92图像(按列排列成10304维向量)做特征提取(如特征提取后的向量为10~30维特征向量，即特征脸Eigenface)，(4)设计BP神经网络(设计隐含层数及各层神经元数目等)，10~30维特征向量作为BP网络的输入，……。



7. 解：感知器算法为：

$$\mathbf{W}(k+1) = \begin{cases} \mathbf{W}(k), & \text{若 } \mathbf{W}^T(k)\mathbf{X}_i > 0 \\ \mathbf{W}(k) + c\mathbf{X}_i, & \text{若 } \mathbf{W}^T(k)\mathbf{X}_i \leq 0 \end{cases}, \quad c \text{ 为正的校正增量。}$$

首先，将所有样本写成增广向量的形式并编号，属于  $\omega_2$  的样本乘以(-1)：

$$\mathbf{X}_1 = [0, 0, 0, 1]^T, \quad \mathbf{X}_2 = [1, 0, 0, 1]^T, \quad \mathbf{X}_3 = [1, 0, 1, 1]^T, \quad \mathbf{X}_4 = [1, 1, 0, 1]^T$$

$$\mathbf{X}_5 = [0, 0, -1, -1]^T, \quad \mathbf{X}_6 = [0, -1, -1, -1]^T, \quad \mathbf{X}_7 = [0, -1, 0, -1]^T, \quad \mathbf{X}_8 = [-1, -1, -1, -1]^T$$

取  $c=1$  开始迭代：

第一轮：

$$\mathbf{W}^T(1)\mathbf{X}_1 = [-1 \quad -2 \quad -2 \quad 0] \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = 0 \leq 0, \quad \text{故 } \mathbf{W}(2) = \mathbf{W}(1) + \mathbf{X}_1 = [-1, -2, -2, 1]^T$$

$$\mathbf{W}^T(2)\mathbf{X}_2 = [-1 \quad -2 \quad -2 \quad 1] \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = 0 \leq 0, \quad \text{故 } \mathbf{W}(3) = \mathbf{W}(2) + \mathbf{X}_2 = [0, -2, -2, 2]^T$$

$$\mathbf{W}^T(3)\mathbf{X}_3 = [0 \quad -2 \quad -2 \quad 2] \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} = 0 \leq 0, \quad \text{故 } \mathbf{W}(4) = \mathbf{W}(3) + \mathbf{X}_3 = [1, -2, -1, 3]^T$$

$$\boldsymbol{W}^{\mathrm{T}}(4)\boldsymbol{X}_4 = [1 \quad -2 \quad -1 \quad 3] \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} = 2 > 0, \text{ 故 } \boldsymbol{W}(5) = \boldsymbol{W}(4)$$

$$\boldsymbol{W}^{\mathrm{T}}(5)\boldsymbol{X}_5 = [1 \quad -2 \quad -1 \quad 3] \begin{bmatrix} 0 \\ 0 \\ -1 \\ -1 \end{bmatrix} = -2 \leq 0, \text{ 故 } \boldsymbol{W}(6) = \boldsymbol{W}(5) + \boldsymbol{X}_5 = [1, -2, -2, 2]^{\mathrm{T}}$$

$$\boldsymbol{W}^{\mathrm{T}}(6)\boldsymbol{X}_6 = [1 \quad -2 \quad -2 \quad 2] \begin{bmatrix} 0 \\ -1 \\ -1 \\ -1 \end{bmatrix} = 2 > 0, \text{ 故 } \boldsymbol{W}(7) = \boldsymbol{W}(6)$$

$$\boldsymbol{W}^{\mathrm{T}}(7)\boldsymbol{X}_7 = [1 \quad -2 \quad -2 \quad 2] \begin{bmatrix} 0 \\ -1 \\ 0 \\ -1 \end{bmatrix} = 0 \leq 0, \text{ 故 } \boldsymbol{W}(8) = \boldsymbol{W}(7) + \boldsymbol{X}_7 = [1, -3, -2, 1]^{\mathrm{T}}$$

$$\boldsymbol{W}^{\mathrm{T}}(8)\boldsymbol{X}_8 = [1 \quad -3 \quad -2 \quad 1] \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \end{bmatrix} = 3 > 0, \text{ 故 } \boldsymbol{W}(9) = \boldsymbol{W}(8)$$

第二轮:

$$\boldsymbol{W}^{\mathrm{T}}(9)\boldsymbol{X}_1 = 1 > 0, \text{ 故 } \boldsymbol{W}(10) = \boldsymbol{W}(9)$$

$$\boldsymbol{W}^{\mathrm{T}}(10)\boldsymbol{X}_2 = 2 > 0, \text{ 故 } \boldsymbol{W}(11) = \boldsymbol{W}(10)$$

$$\boldsymbol{W}^{\mathrm{T}}(11)\boldsymbol{X}_3 = 0 \leq 0, \text{ 故 } \boldsymbol{W}(12) = \boldsymbol{W}(11) + \boldsymbol{X}_3 = [2, -3, -1, 2]^{\mathrm{T}}$$

$$\boldsymbol{W}^{\mathrm{T}}(12)\boldsymbol{X}_4 = 1 > 0, \text{ 故 } \boldsymbol{W}(13) = \boldsymbol{W}(12)$$

$$\boldsymbol{W}^{\mathrm{T}}(13)\boldsymbol{X}_5 = -1 \leq 0, \text{ 故 } \boldsymbol{W}(14) = \boldsymbol{W}(13) + \boldsymbol{X}_5 = [2, -3, -2, 1]^{\mathrm{T}}$$

$$\boldsymbol{W}^{\mathrm{T}}(14)\boldsymbol{X}_6 = 4 > 0, \text{ 故 } \boldsymbol{W}(15) = \boldsymbol{W}(14)$$

$$\boldsymbol{W}^{\mathrm{T}}(15)\boldsymbol{X}_7 = 2 > 0, \text{ 故 } \boldsymbol{W}(16) = \boldsymbol{W}(15)$$

$$\boldsymbol{W}^{\mathrm{T}}(16)\boldsymbol{X}_8 = 2 > 0, \text{ 故 } \boldsymbol{W}(17) = \boldsymbol{W}(16)$$

第三轮:

$$\boldsymbol{W}^{\mathrm{T}}(17)\boldsymbol{X}_1 = 1 > 0, \text{ 故 } \boldsymbol{W}(18) = \boldsymbol{W}(17)$$

$$\boldsymbol{W}^{\mathrm{T}}(18)\boldsymbol{X}_2 = 3 > 0, \text{ 故 } \boldsymbol{W}(19) = \boldsymbol{W}(18)$$

$$\boldsymbol{W}^{\mathrm{T}}(19)\boldsymbol{X}_3 = 1 > 0, \text{ 故 } \boldsymbol{W}(20) = \boldsymbol{W}(19)$$

$$\boldsymbol{W}^{\mathrm{T}}(20)\boldsymbol{X}_4 = 0 \leq 0, \text{ 故 } \boldsymbol{W}(21) = \boldsymbol{W}(20) + \boldsymbol{X}_4 = [3, -2, -2, 2]^{\mathrm{T}}$$

$$\boldsymbol{W}^{\mathrm{T}}(21)\boldsymbol{X}_5 = 0 \leq 0, \text{ 故 } \boldsymbol{W}(22) = \boldsymbol{W}(21) + \boldsymbol{X}_5 = [3, -2, -3, 1]^{\mathrm{T}}$$

$$\boldsymbol{W}^{\mathrm{T}}(22)\boldsymbol{X}_6 = 4 > 0, \text{ 故 } \boldsymbol{W}(23) = \boldsymbol{W}(22)$$

$$\boldsymbol{W}^{\mathrm{T}}(23)\boldsymbol{X}_7 = 1 > 0, \text{ 故 } \boldsymbol{W}(24) = \boldsymbol{W}(23)$$

$$\boldsymbol{W}^{\mathrm{T}}(24)\boldsymbol{X}_8 = 1 > 0, \text{ 故 } \boldsymbol{W}(25) = \boldsymbol{W}(24)$$

第四轮：

$$\boldsymbol{W}^T(25)\boldsymbol{X}_1 = 1 > 0, \text{ 故 } \boldsymbol{W}(26) = \boldsymbol{W}(25)$$

$$\boldsymbol{W}^T(26)\boldsymbol{X}_2 = 4 > 0, \text{ 故 } \boldsymbol{W}(27) = \boldsymbol{W}(26)$$

$$\boldsymbol{W}^T(27)\boldsymbol{X}_3 = 1 > 0, \text{ 故 } \boldsymbol{W}(28) = \boldsymbol{W}(27)$$

$$\boldsymbol{W}^T(28)\boldsymbol{X}_4 = 2 > 0, \text{ 故 } \boldsymbol{W}(29) = \boldsymbol{W}(28)$$

$$\boldsymbol{W}^T(29)\boldsymbol{X}_5 = 2 > 0, \text{ 故 } \boldsymbol{W}(30) = \boldsymbol{W}(29)$$

$$\boldsymbol{W}^T(30)\boldsymbol{X}_6 = 4 > 0, \text{ 故 } \boldsymbol{W}(31) = \boldsymbol{W}(30)$$

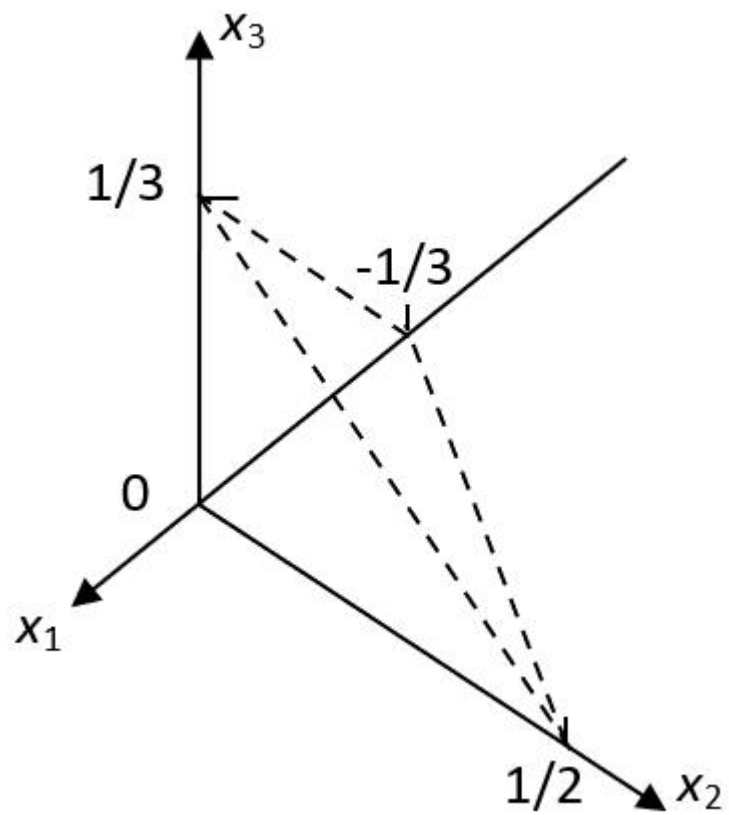
$$\boldsymbol{W}^T(31)\boldsymbol{X}_7 = 1 > 0, \text{ 故 } \boldsymbol{W}(32) = \boldsymbol{W}(31)$$

$$\boldsymbol{W}^T(32)\boldsymbol{X}_8 = 1 > 0, \text{ 故 } \boldsymbol{W}(33) = \boldsymbol{W}(32)$$

该轮迭代分类结果全部正确，故解向量  $\boldsymbol{W} = [3, -2, -3, 1]^T$ ，对应的判别函数为：

$$d(\boldsymbol{X}) = 3x_1 - 2x_2 - 3x_3 + 1$$

判别界面  $d(\boldsymbol{X}) = 0$  如下图所示，图中虚线为判别界面与坐标面  $x_1Ox_2$ ， $x_1Ox_3$ ， $x_2Ox_3$  的交线。



第7题解图：判别界面