重庆邮电大学

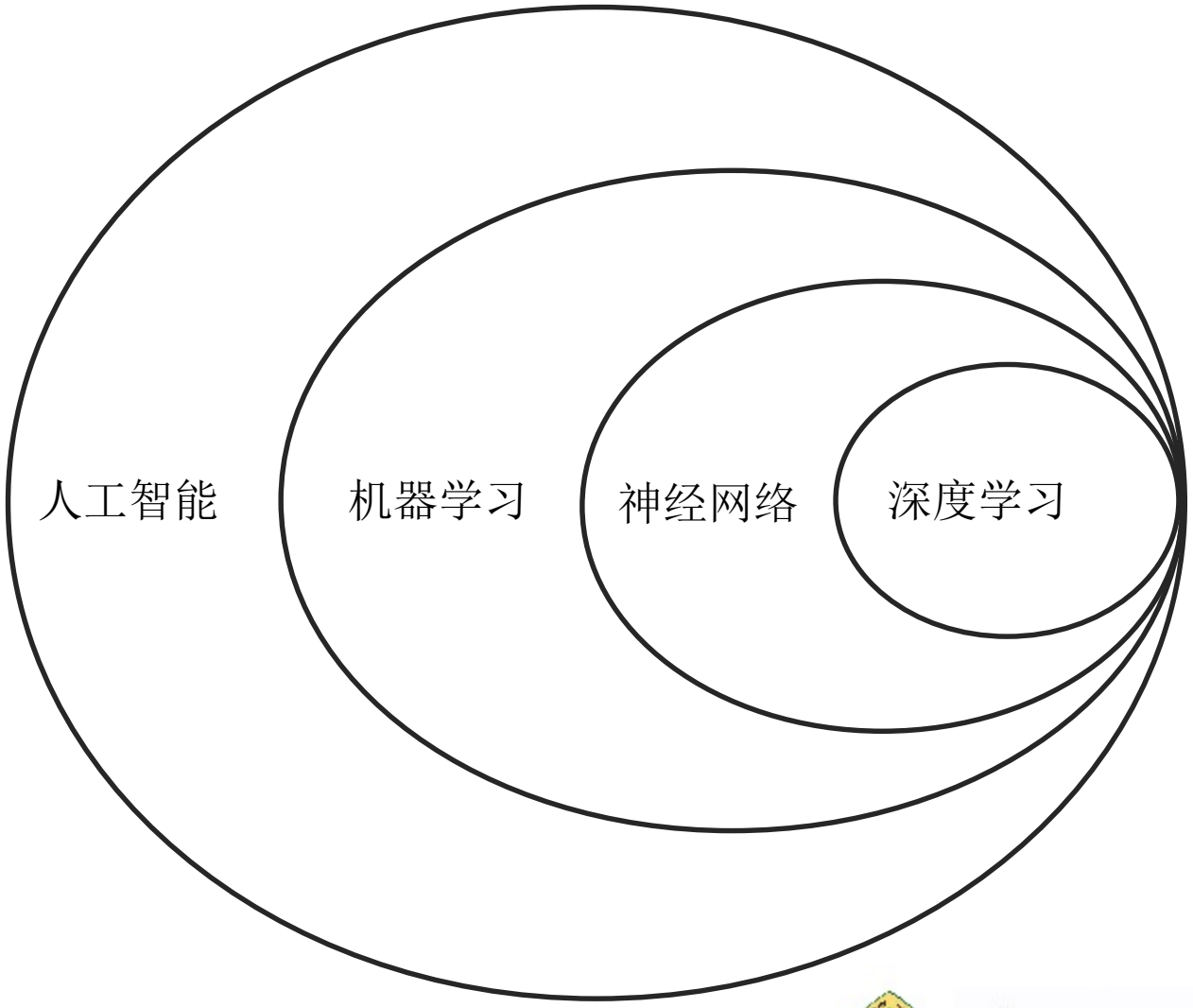
计算机科学与技术学院

人工智能原理

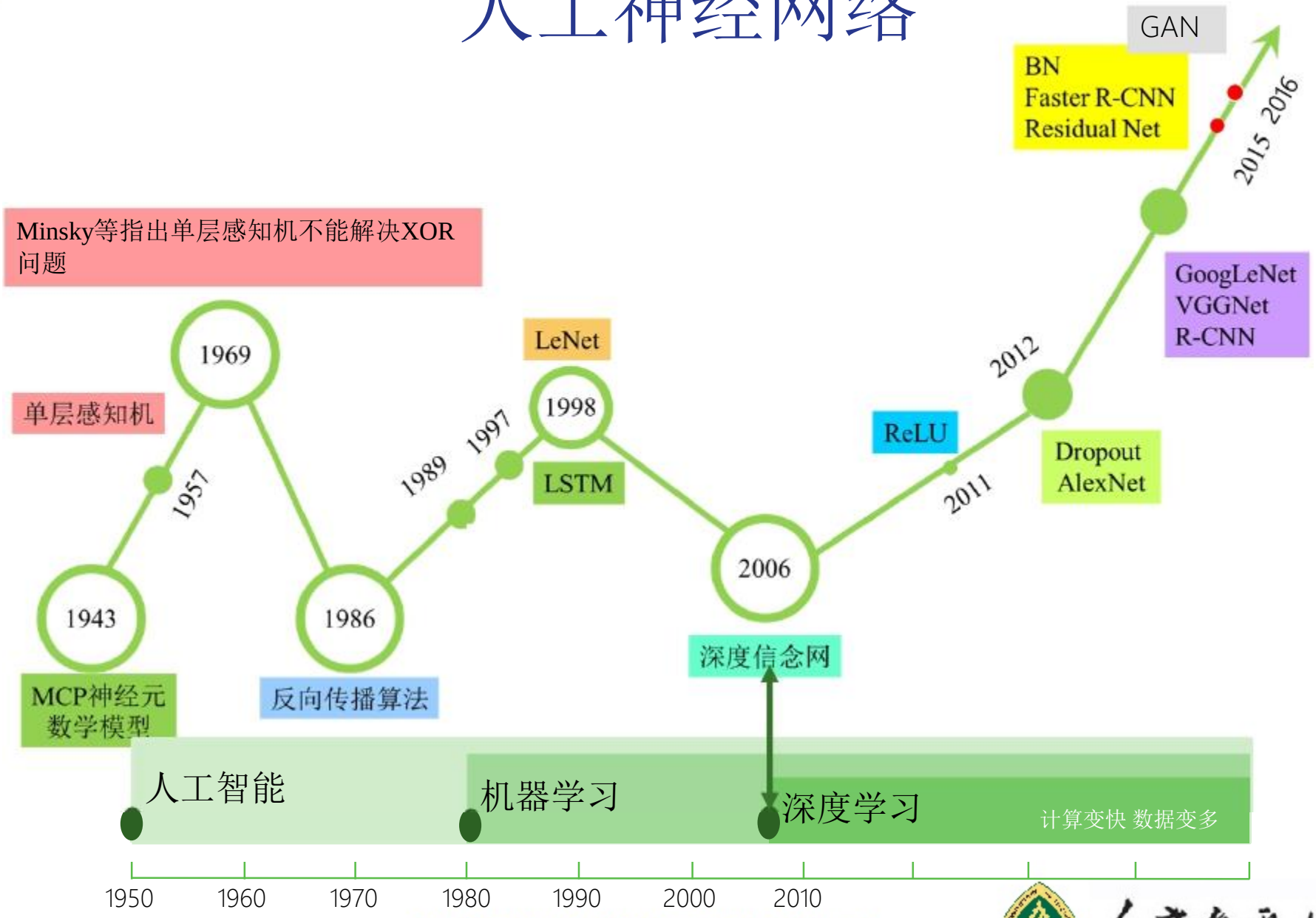
神经网络基础

包含关系

- 人工智能
- 机器学习
- 神经网络
- 深度学习

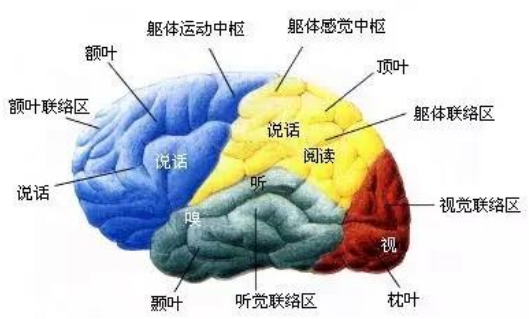


人工神经网络

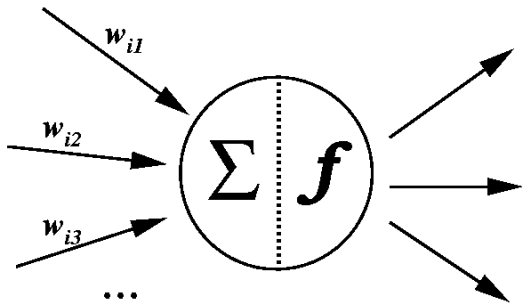

GAN
BN
Faster R-CNN
Residual Net
2015
2016
Minsky等指出单层感知机不能解决XOR问题
GoogLeNet
VGGNet
R-CNN
1969
LeNet
2012
单层感知机
1998
1997
1989
ReLU
Dropout
AlexNet
1957
LSTM
2011
2006
1943
1986
深度信念网
MCP神经元数学模型
反向传播算法
人工智能
机器学习
深度学习
计算变快 数据变多
1950
1960
1970
1980
1990
2000
2010

生物神经网络

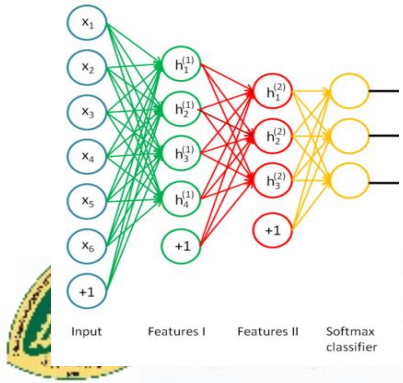
- 20世纪初期的神经生物学已经阐明，**人的智力功能定位在人的大脑皮层**，由大量的神经元和支持神经元的胶质细胞组成的，**神经元是基本的信息处理单元**。
- 1943年，McCulloch和Pitts提出了神经元的数理逻辑模型(MP模型)。**依据何在呢？**
- 由此，人们很自然想到了利用人工神经元网络来构造具有智能行为的机器系统，这就是智能理论与技术研究中的结构模拟：人工神经网络方法。



大脑皮层功能区示意图

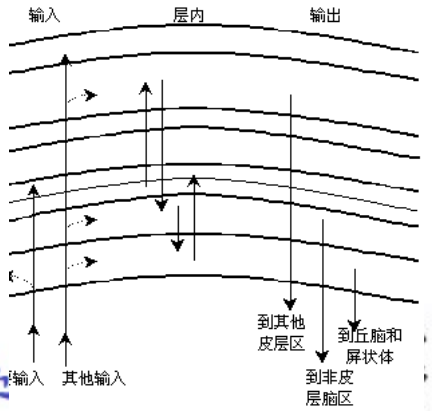


$y_i = f(net_i)$



大脑皮层

- 大脑半球表面覆盖着一层灰质，称为**大脑皮层，或大脑皮质，是由无数大小不等的神经细胞（称为神经元）和神经胶质细胞以及神经纤维构成**。
- 大脑皮层的神经元和神经纤维均分层排列，神经元之间形成复杂的联接，构成**生物神经网络**。
- 大脑皮层里神经网络作为**结构模拟**的对象。

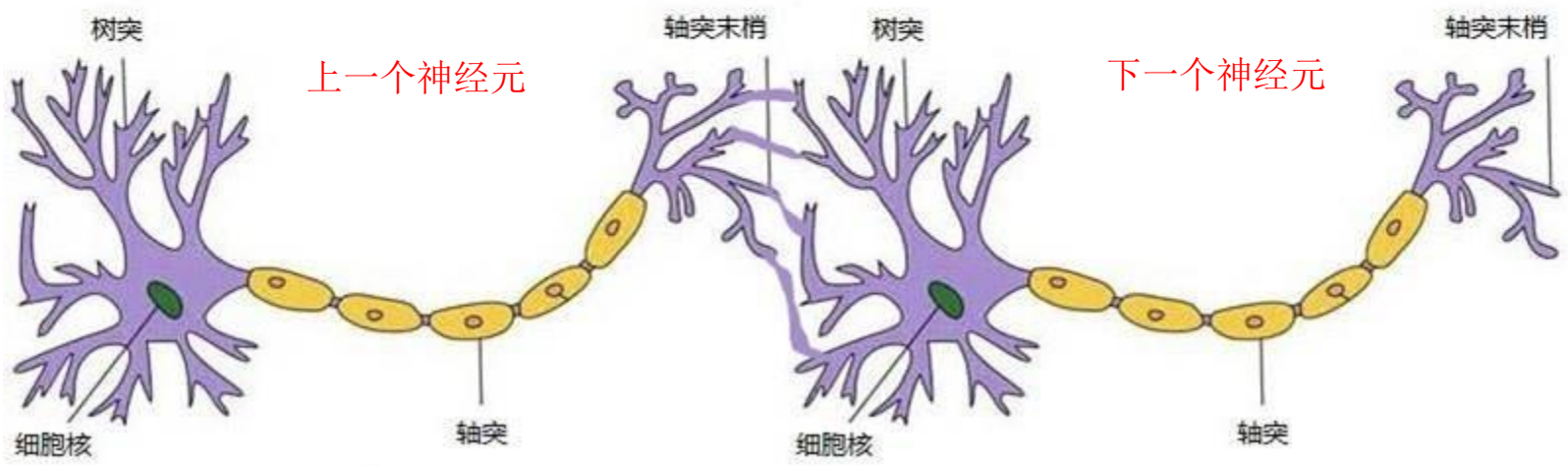


大脑皮层

- 大脑皮层大约包含有大约10^{12}个神经元，每个神经元又大约与10^{3}～10^{4}个其他神经元相连接（这些连接称为突触），形成极为错综复杂的而又灵活多样的神经网络。
- 每一个神经元虽然都十分简单，但如此大量的神经元之间如此复杂的连接却可以演化出极其丰富多彩变化万端的人类智能行为。

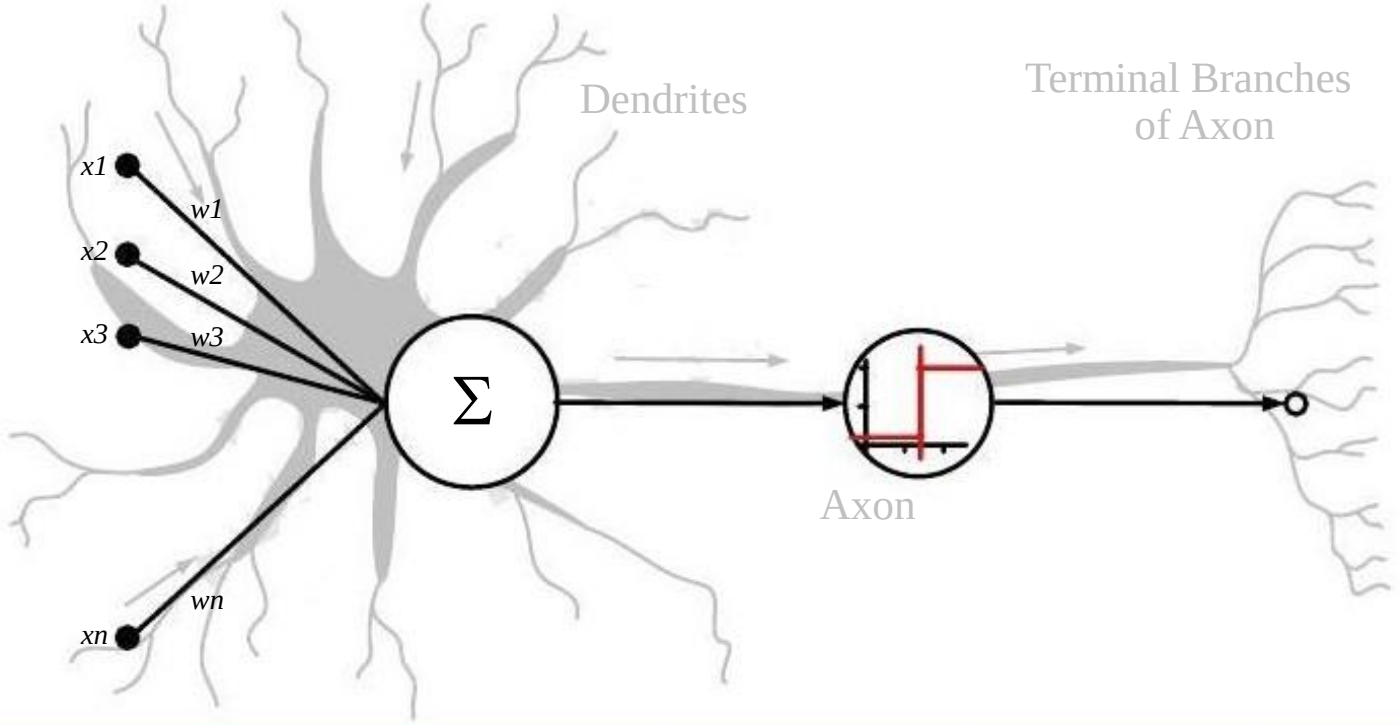
生物神经元

生物学领域，一个生物神经元有多个树突（接受传入信息）；有一条轴突，轴突尾端有许多轴突末梢（给其他多个神经元传递信息）。轴突末梢跟其它生物神经元的树突产生连接的位置叫做“突触”。



人工神经元

机器学习领域，人工神经元是一个包含输入，输出与计算功能的模型。不严格地说，其输入可类比为生物神经元的树突，其输出可类比为神经元的轴突，其计算可类比为细胞体。

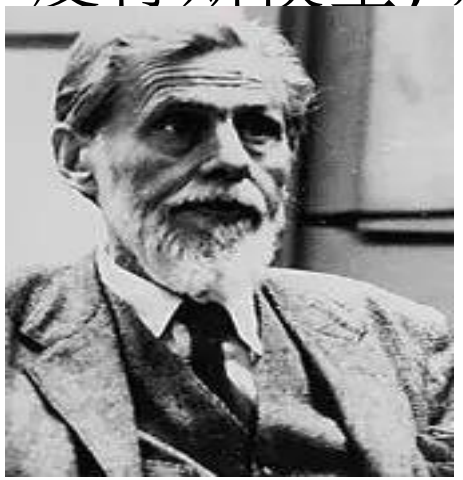


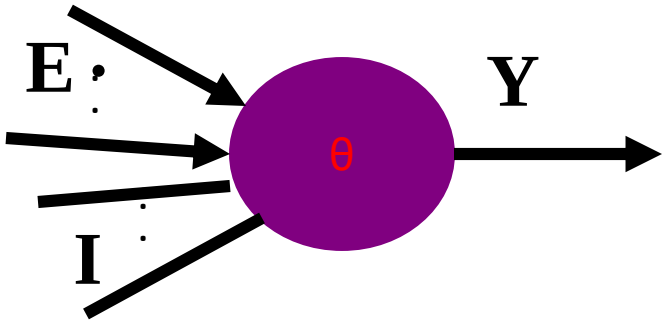
生物神经元：人工神经元=老鼠：米老鼠

人工神经元模型

早期M-P数学模型

- 1943年，心理学家McCulloch和数理逻辑学家Pitts提出了第一个系统的ANN研究——简化的形式神经元模型，称为McCulloch-Pitts模型(麦卡洛克-皮特斯模型)，简称为M-P模型。

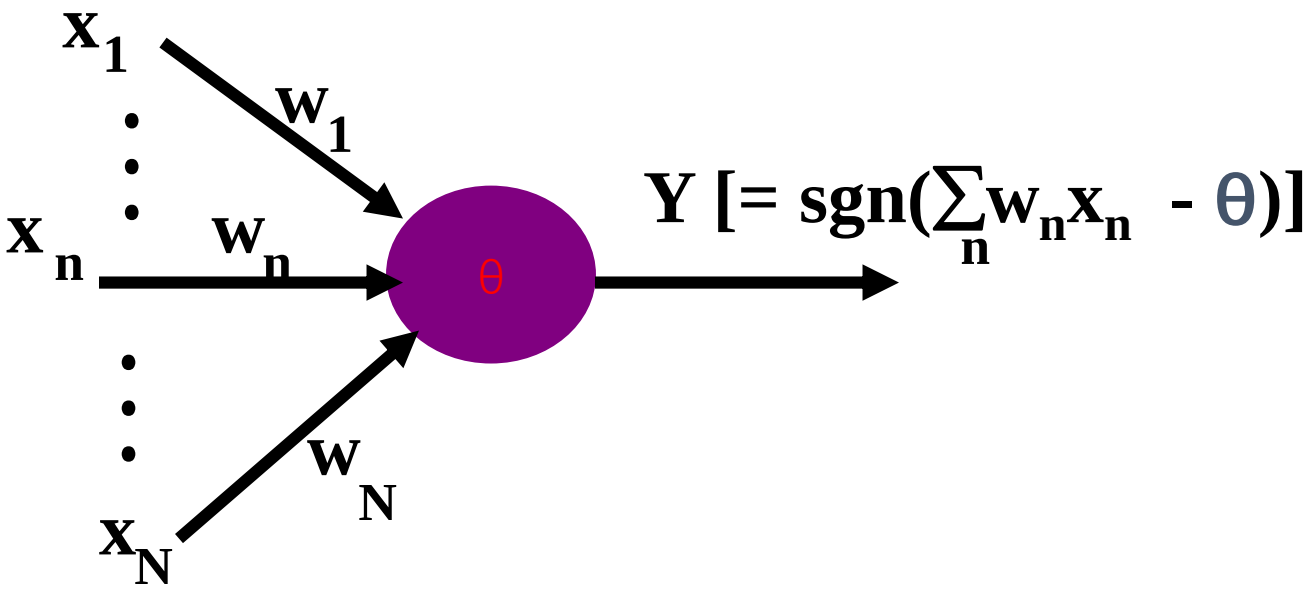




Input	Y
$\sum E \geq \theta, \sum I = 0$	1
$\sum E \geq \theta, \sum I > 0$	0
$\sum E < \theta, \sum I = 0$	0
$\sum E < \theta, \sum I > 0$	0

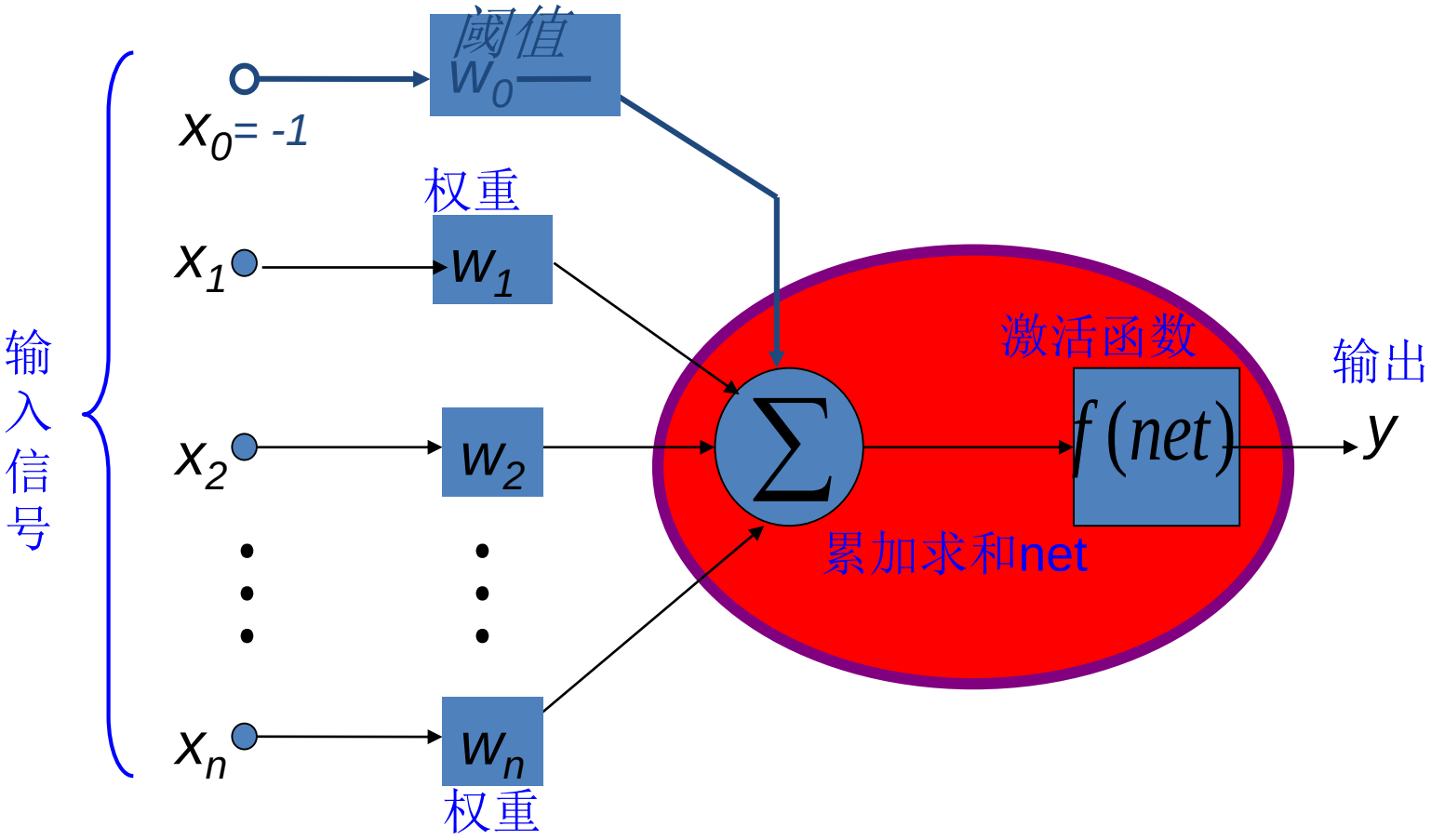
- M-P模型确实在一定程度上反映了生物神经元在结构和功能上的特征。
- M-P模型给抑制性输入赋予了“否决权”：
  - 只有当不存在抑制性输入而兴奋性输入的总和超过了阈值，神经元才兴奋。

阈值逻辑神经元模型



其中，X是输入向量，Y是输出结果，W是权值向量，θ是神经元的阈值，sgn表示符号函数 。

人工神经元统一模型



感知机模型

感知机模型是二分类的线性分类模型，它能对空间中线性可分的二分类样本点进行划分。

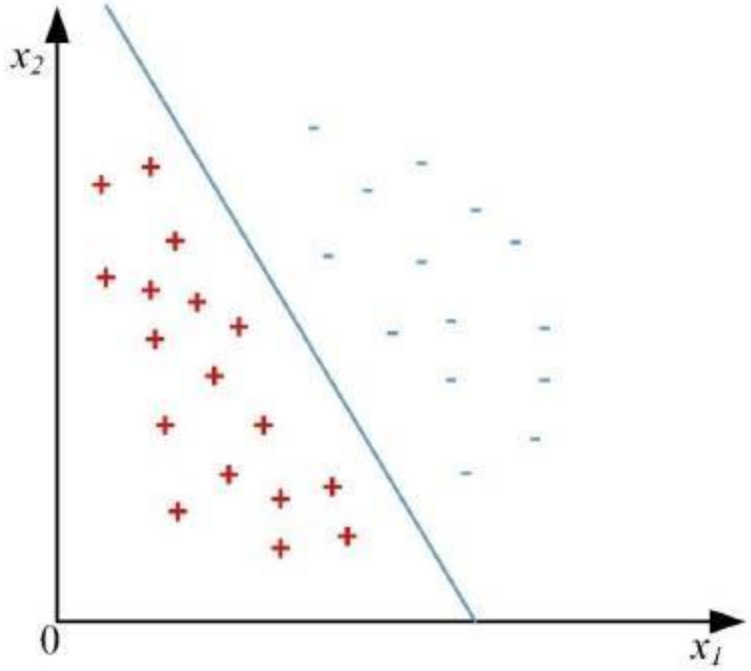
$$y = u\left(\sum_{i=1}^{n} w^{(i)}x^{(i)} + \theta\right) = u\left(\boldsymbol{W}\cdot\boldsymbol{x}^{\boldsymbol{T}} + \theta\right)$$

式中，$u(\cdot)$为单位阶跃函数，$\boldsymbol{W} = \left(w^{(1)}, w^{(2)}, \dots, w^{(n)}\right)$，$\boldsymbol{x} = \left(x^{(1)}, x^{(2)}, \dots, x^{(n)}\right)$。

$\boldsymbol{W}\cdot\boldsymbol{x}^{\boldsymbol{T}} + \theta = 0$表示空间中的一个超平面，该超平面将空间中的点划分为两类。对空间中的某点$\boldsymbol{x_i}$，如果被正确划分，易知$y_i\left(\boldsymbol{W}\cdot\boldsymbol{x}^{\boldsymbol{T}} + \theta\right) > 0$，。如果被错误划分，即为误分类点，则$y_i\left(\boldsymbol{W}\cdot\boldsymbol{x}^{\boldsymbol{T}} + \theta\right) \leq 0$。

感知机模型

感知机模型$H(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^T\boldsymbol{x} + b)$对应一个超平面$\boldsymbol{w}^T\boldsymbol{x} + b = 0$，模型参数是$(\boldsymbol{w},b)$。感知机的目标是找到一个$(\boldsymbol{w},b)$，将线性可分的数据集T中的所有的样本点正确地分为两类。



$$H(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^T\boldsymbol{x} + b)$$

$$sign(x) = \begin{cases} +1 & x \geq 0 \\ -1 & x < 0 \end{cases}$$

寻找损失函数，并将损失函数最小化

➢ 寻找损失函数

考虑一个训练数据集$D=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),..,(\boldsymbol{x}_m,y_m)\}$，其中，$\boldsymbol{x}_j \in R^n, y_j \in \{+1,-1\}$。如果存在某个超平面 $S:(\boldsymbol{w}^T\boldsymbol{x}+b=0)$，能将正负样本分到S两侧，则说明数据集可分，那么，如何求出这个超平面S的表达式？

策略：假设误分类的点为数据集M，使用误分类点到超平面的总距离来寻找损失函数（直观来看，总距离越小越好）

样本点$\boldsymbol{x}_j$到超平面S的距离：$d=\frac{1}{\|\boldsymbol{w}\|}|\boldsymbol{w}^T\boldsymbol{x}_j+b|$　　$\|\boldsymbol{w}\|$是$\boldsymbol{w}$的L^2范数

➢ 寻找损失函数

数据集中误分类点满足条件：$-y_j\left(\boldsymbol{w}^T\boldsymbol{x}_j+b\right)>0$

去掉点 $\boldsymbol{x}_j$到超平面S的距离的绝对值符号：

$$d=-\frac{1}{\|\boldsymbol{w}\|}y_j\left(\boldsymbol{w}^T\boldsymbol{x}_j+b\right)$$

所有误分类点到超平面S的总距离为

$$d=-\frac{1}{\|\boldsymbol{w}\|}\sum_{\boldsymbol{x}_j\in M}y_j(\boldsymbol{w}^T\boldsymbol{x}_j+b)$$

由此寻找到感知机的损失函数 $L(\boldsymbol{w},b)=-\sum_{\boldsymbol{x}_j\in M}y_j(\boldsymbol{w}^T\boldsymbol{x}_j+b)$

损失函数极小化的最优化问题可使用：随机梯度下降法

$$L(\boldsymbol{w},b) = -\sum_{\boldsymbol{x}_j \in M} y_j(\boldsymbol{w}^T\boldsymbol{x}_j + b)$$

$$\nabla_{\boldsymbol{w}} L(\boldsymbol{w},b) = -\sum_{\boldsymbol{x}_j \in M} y_j\boldsymbol{x}_j \Rightarrow \boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha y_j\boldsymbol{x}_j$$

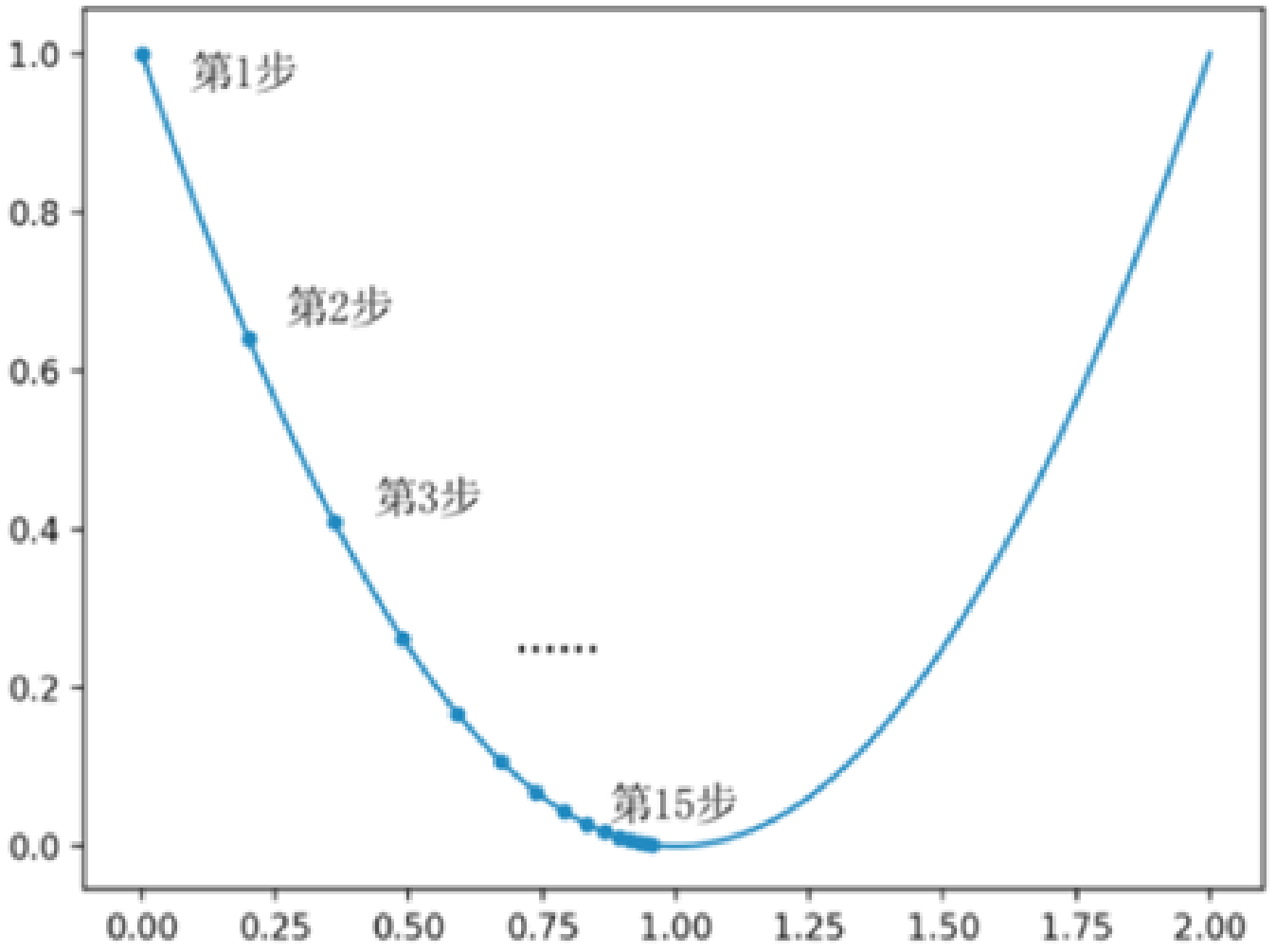
$$\nabla_b L(\boldsymbol{w},b) = -\sum_{\boldsymbol{x}_j \in M} y_j \Rightarrow b \leftarrow b + \alpha y_j$$

随机选取误分类点(x_j,y_j)对$\boldsymbol{w},b$以α为步长进行更新，通过迭代可以使得损失函数 $L(\boldsymbol{w},b)$不断减小，直到为0

$$L(\boldsymbol{w},b) \to 0$$

梯度下降算法

梯度下降算法的思路可概括为：假定目标函数可微，梯度下降算法从空间中任一给定初始点开始进行指定轮数的搜索。在每一轮搜索中都计算目标函数在当前点的梯度，并沿着与梯度相反的方向按照一定步长移动到下一可行点。


1.0
0.8
0.6
0.4
0.2
0.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
第1步
第2步
第3步
.....
第15步

梯度下降算法

$\boldsymbol{w} = \mathbf{0}$

For $t = 1,2,\ldots,N$:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla F(\boldsymbol{w})$$

Return $\boldsymbol{w}$

例： 上图的目标函数 $F(w) = (w-1)^2$。当 $w = 1$ 时 F 达到其最小值。梯度下降算法验证了这一事实。在第1行定义梯度下降的搜索轮数为20，学习速率 $\eta = 0.1$。从第3行开始20轮循环，$\nabla F(w) = 2(w-1)$ 。

运行下图中的算法得到 $w = 0.996$ 。它十分接近最优解。

```
1  N, eta = 20, 0.1
2  w = 0
3  for t in range(N):
4      w = w - eta * 2 * (w - 1)
5  print(w)
```

随机梯度下降算法

随机梯度下降法是梯度下降法的改进算法。随机梯度下降算法适用于训练数据规模较大的场合。

随机梯度下降算法的每次迭代可以从所有训练数据中取一个采样来估计目标函数梯度。因而它能够大幅度地降低算法的时间复杂度。

随机梯度下降法不需要计算大量的数据，所以速度较快，但得到的并不是真正的梯度，可能会造成不收敛的问题。

随机梯度下降算法

$\boldsymbol{w} = \mathbf{0},\ \boldsymbol{w}_{sum} = \mathbf{0}$

For $t = 1,2,\ldots,N$:

Sample $\left(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\right) \sim S$

$$\eta_t = \frac{\eta_0}{\eta_1 + t}$$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta_t \nabla l\left(h_w\left(\boldsymbol{x}^{(i)}\right), \boldsymbol{y}^{(i)}\right)$$

$$\boldsymbol{w}_{sum} \leftarrow \boldsymbol{w}_{sum} + \boldsymbol{w}$$

Return $\overline{\boldsymbol{w}} = \boldsymbol{w}_{sum}/N$

感知机模型

设感知机的样本集$\mathbf{S}=\{\boldsymbol{s}_1,\boldsymbol{s}_2,\ldots,\boldsymbol{s}_m\}$包含个$m$样本，每个样本$\boldsymbol{s}_i=(\boldsymbol{x}_i,y_i)$包括一个实例$\boldsymbol{x}_i$和一个标签$y_i$，$y_i\in\{1,\ 0\}$。

步数	操作
1	设定步长η，随机选取$\boldsymbol{W}$和θ初值
2	从样本集S中选取一个样本$s_i=(\boldsymbol{x}_i,y_i)$，计算 $\hat{y}_i=u(\boldsymbol{W}\cdot\boldsymbol{x}_i^{\boldsymbol{T}}+\theta)$，调整系数：$\boldsymbol{W}\leftarrow\boldsymbol{W}+\eta(y_i-\hat{y}_i)\boldsymbol{x}_i$，$\theta\leftarrow\theta+\eta(y_i-\hat{y}_i)_i$
3	重复第2步，直到没有误分类点

小练习

假设有一个简单的训练数据集，正实例点是$x_1 = (3,3)^T$， $x_2 = (4,3)^T$，负实例点$x_3 = (1,1)^T$，试利用最基础的感知机学习算法求模型$f(x) = sign(w \cdot x + b)$。其中，$w = (w^{(1)}, w^{(2)})^T, x = (x^{(1)}, x^{(2)})^T$。 w和b初值都取0。

假设当x>0，sign(x)=1;当x=0，sign(x)=0;当x<0， sign(x)=-1。

若构建损失函数$L(w,b) = -\sum_{x_i \in M} y_i(w \cdot x + b)$。设步长为1。更新过程将如何?

思考题解答

根据损失函数$L(w,b)=-\sum_{x_i\in M} y_i(w\cdot x+b)$可得

$$\frac{dL(w,b)}{dw}=-\sum_{x_i\in M} y_i x,$$

$$\frac{dL(w,b)}{db}=-\sum_{x_i\in M} y_i。$$

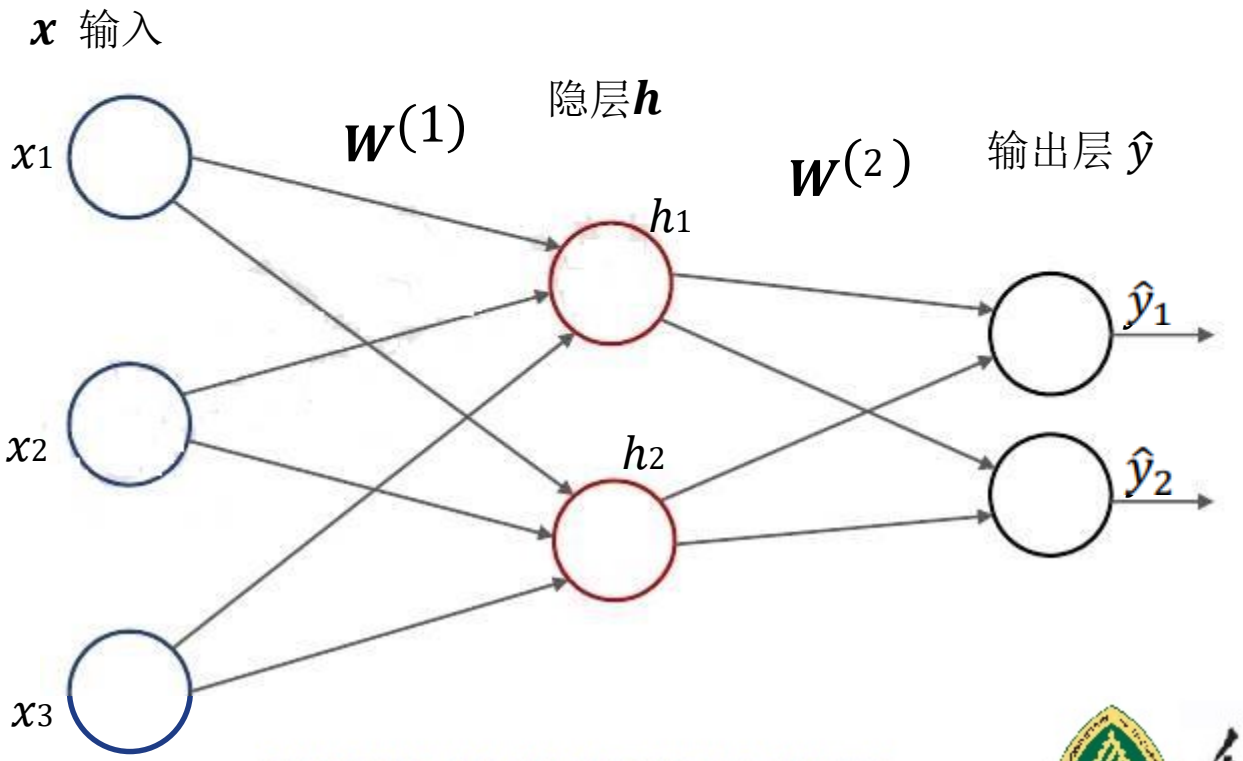
即梯度向量可以表示为（$-\sum_{x_i\in M} y_i x$，$-\sum_{x_i\in M} y_i$）。若每次选取一个错误点进行参数更新。则第一步$x_1=(3,3)^T$被分类错误，对应$y_1=1$，更新$w_1=w_0-1*(-y_1x_1)=(3,3)^T$, $b_1=b_0-1*(-y_1)=1$。迭代过程如下：

思考题解答

迭代次数	误分类点	y_i	w	b	w·x+b
0			0	0	0
1	$x_1 = (3,3)^T$	1	$(3,3)^T$	1	$3x^{(1)} + 3x^{(2)} + 1$
2	$x_3 = (1,1)^T$	-1	$(2,2)^T$	0	$2x^{(1)} + 2x^{(2)}$
3	$x_3 = (1,1)^T$	-1	$(1,1)^T$	-1	$x^{(1)} + x^{(2)} - 1$
4	$x_3 = (1,1)^T$	-1	$(0,0)^T$	-2	-2
5	$x_1 = (3,3)^T$	1	$(3,3)^T$	-1	$3x^{(1)} + 3x^{(2)} - 1$
6	$x_3 = (1,1)^T$	-1	$(2,2)^T$	-2	$2x^{(1)} + 2x^{(2)} - 2$
7	$x_3 = (1,1)^T$	-1	$(1,1)^T$	-3	$x^{(1)} + x^{(2)} - 3$
8	0		$(1,1)^T$	-3	$x^{(1)} + x^{(2)} - 3$

两层神经网络-多层感知机

- 将大量的神经元模型进行组合，用不同的方法进行连接并作用在不同的激活函数上，就构成了人工神经网络模型。
- 多层感知机一般指全连接的两层神经网络模型



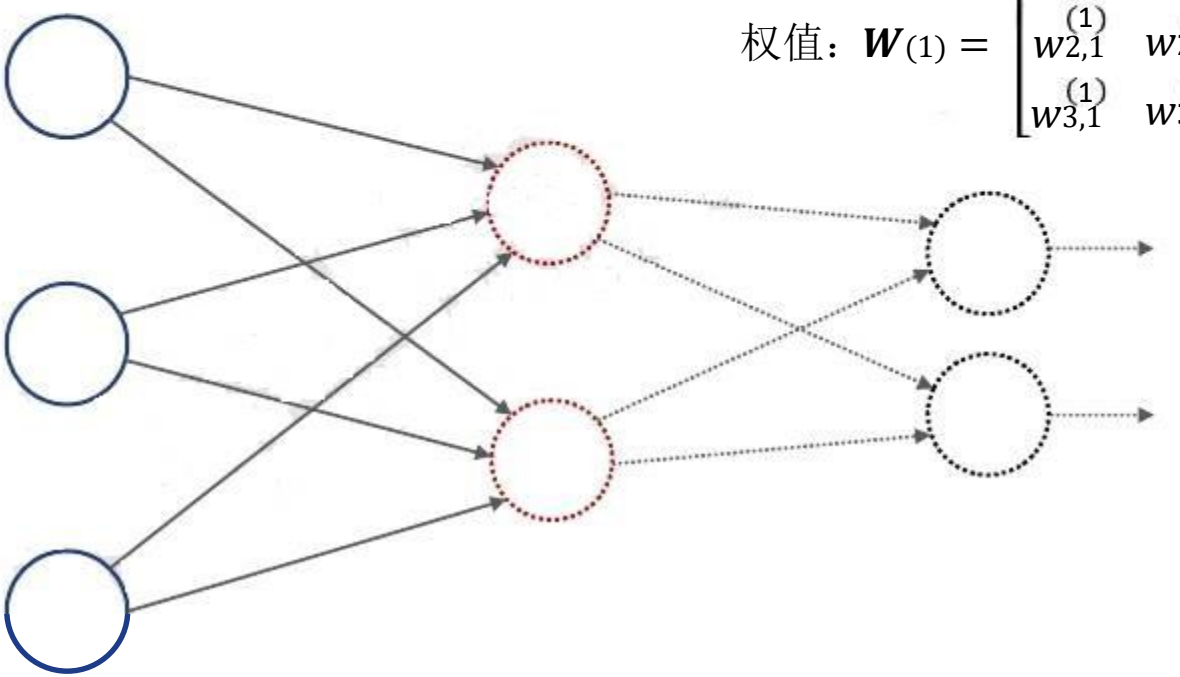
https://www.bilibili.com/video/BV1f64y1T7sw?from=search&seid=4861413253999417876

输入

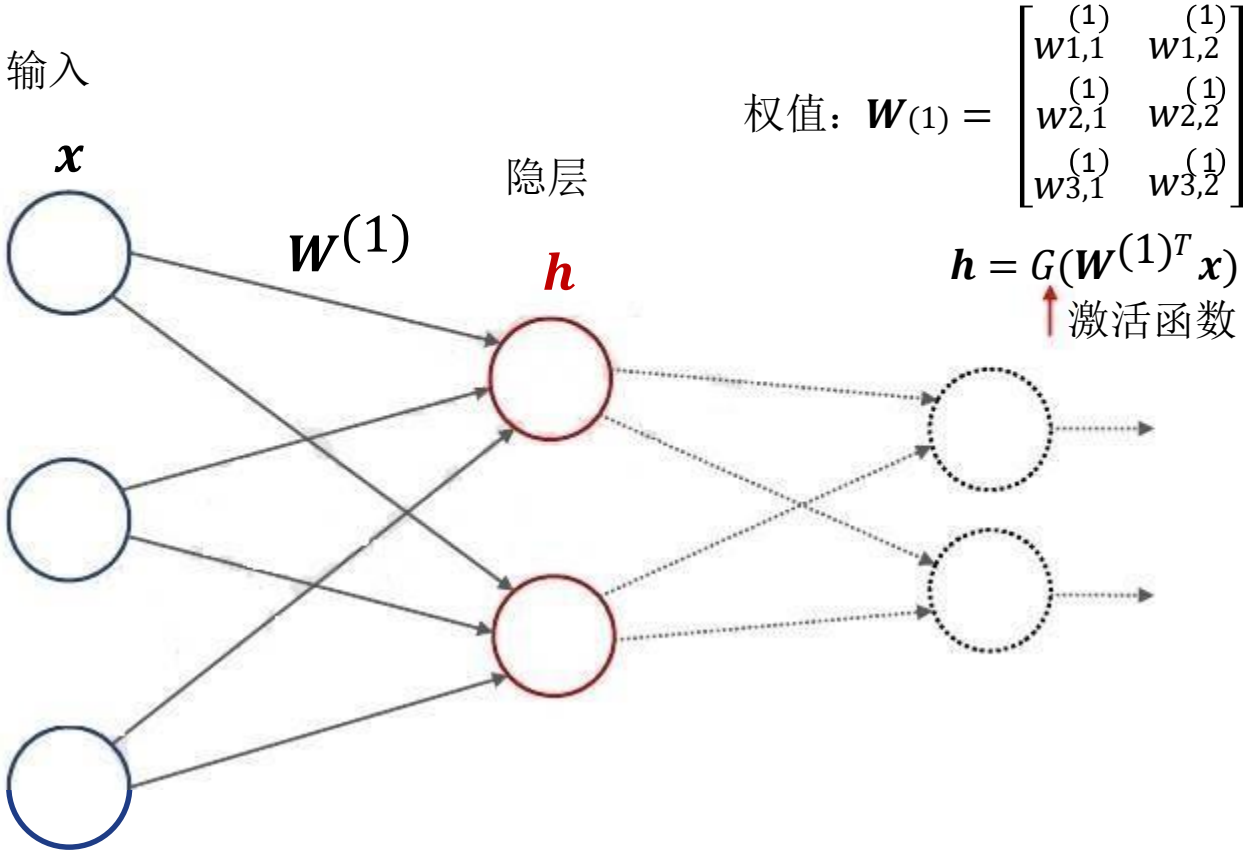
$\boldsymbol{x}$ $\boldsymbol{W}^{(1)}$

输入样本：$\boldsymbol{x} = [x_1; x_2; x_3]$

权值：$\boldsymbol{W}_{(1)} = \begin{bmatrix} w_{1,1}^{(1)} & w_{1,2}^{(1)} \\ w_{2,1}^{(1)} & w_{2,2}^{(1)} \\ w_{3,1}^{(1)} & w_{3,2}^{(1)} \end{bmatrix}$



输入：$\boldsymbol{x} = [x_1; x_2; x_3]$



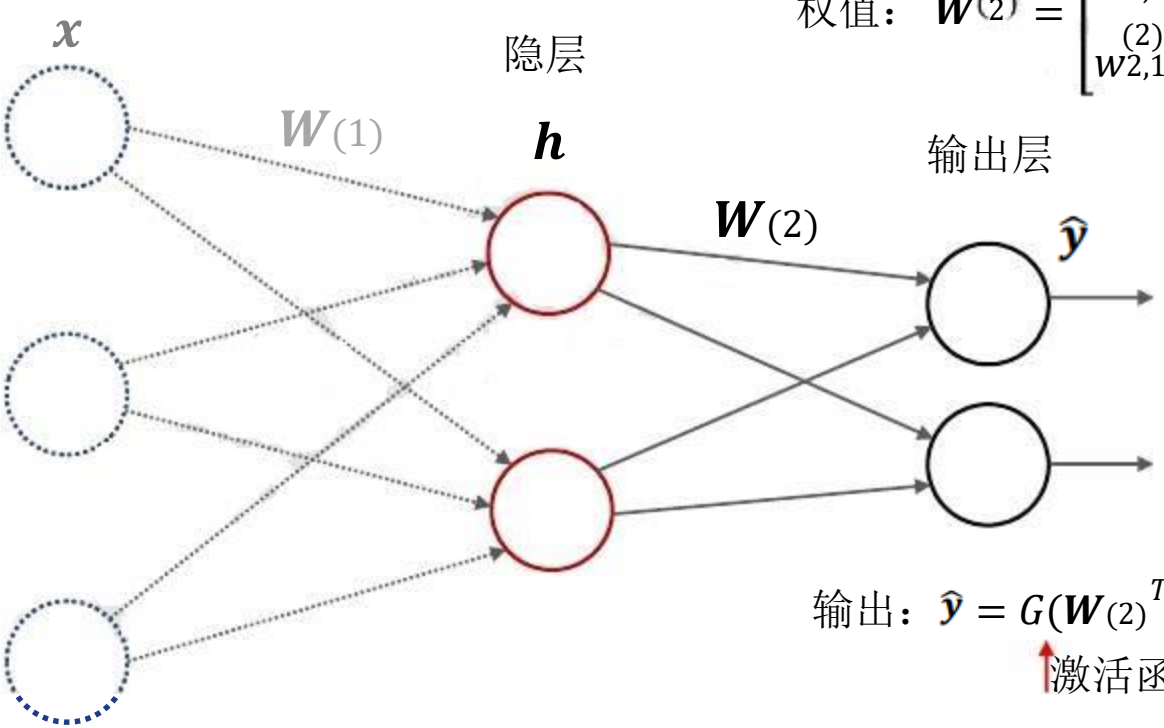
权值：$\boldsymbol{W}_{(1)} = \begin{bmatrix} w_{1,1}^{(1)} & w_{1,2}^{(1)} \\ w_{2,1}^{(1)} & w_{2,2}^{(1)} \\ w_{3,1}^{(1)} & w_{3,2}^{(1)} \end{bmatrix}$

$\boldsymbol{h} = G(\boldsymbol{W}^{(1)^T}\boldsymbol{x})$

↑激活函数

输入： $\boldsymbol{h} = G(\boldsymbol{W}_{(1)}{}^{T}\boldsymbol{x})$

权值： $\boldsymbol{W}^{(2)} = \begin{bmatrix} w_{1,1}^{(2)} & w_{1,2}^{(2)} \\ w_{2,1}^{(2)} & w_{2,2}^{(2)} \end{bmatrix}$



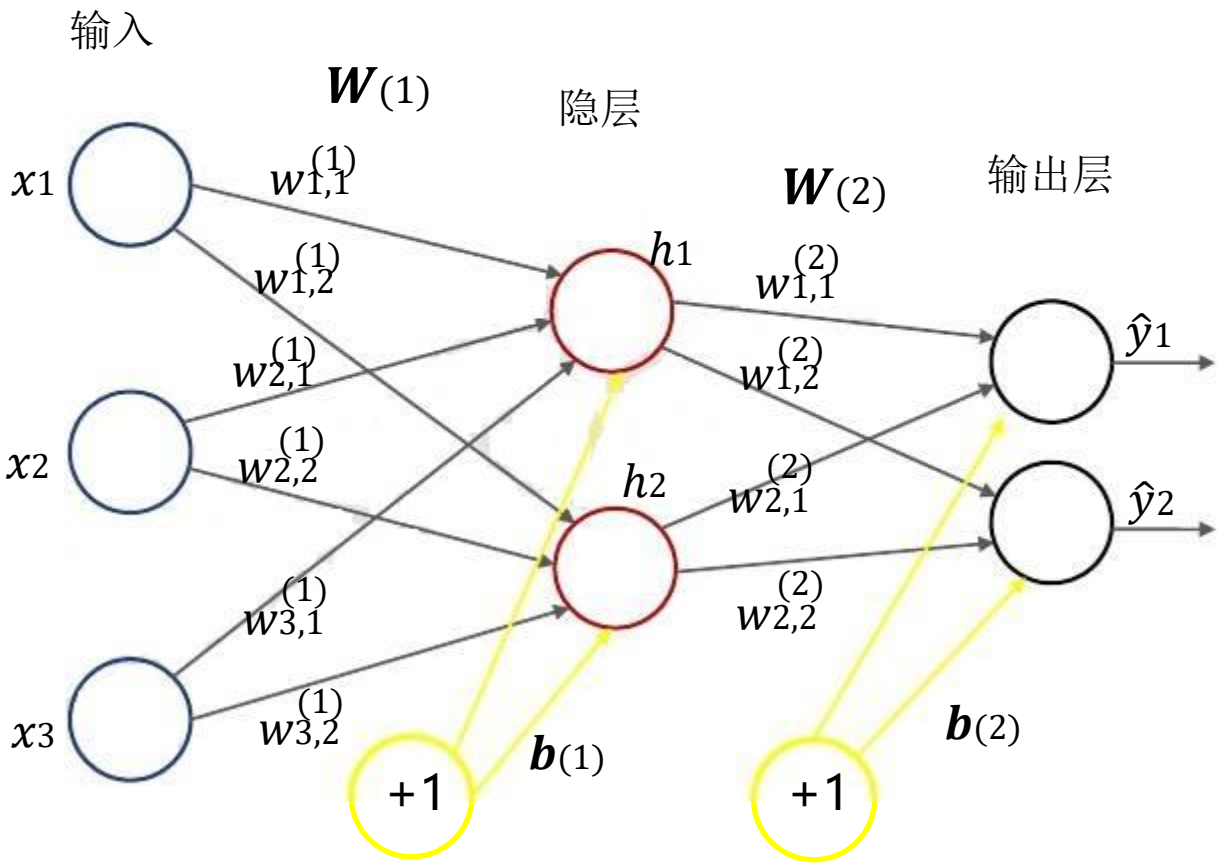
输出： $\hat{\boldsymbol{y}} = G(\boldsymbol{W}_{(2)}{}^{T}\boldsymbol{h})$

激活函数

偏置节点

$$\boldsymbol{h} = G(\boldsymbol{W}^{(1)T}\boldsymbol{x} + \boldsymbol{b}^{(1)})$$

$$\boldsymbol{y} = G(\boldsymbol{W}^{(2)T}\boldsymbol{h} + \boldsymbol{b}^{(2)})$$



在神经网络中，除了输出层以外，都会有一个偏置单元b，与后一层的所有节点相连接。

$\boldsymbol{W}$称之为权重， $\boldsymbol{b}$为偏置， $(\boldsymbol{W},\boldsymbol{b})$ 合称为神经网络的参数

浅层神经网络特点

需要数据量小、训练快，其局限性在于对复杂函数的表示能力有限，针对复杂分类问题其泛化能力受到制约

Why Not Go Deeper? Kurt Hornik证明了理论上两层神经网络足以拟合任意函数，过去也没有足够的数据和计算能力

多层神经网络（深度学习）

2006年，Hinton在Science发表了论文（Reducing the dimensionality of data with neural networks. Science, Vol. 313. no. 5786），给多层神经网络相关的学习方法赋予了一个新名词--“深度学习”。他和LeCun以及Bengio三人被称为深度学习三位开创者

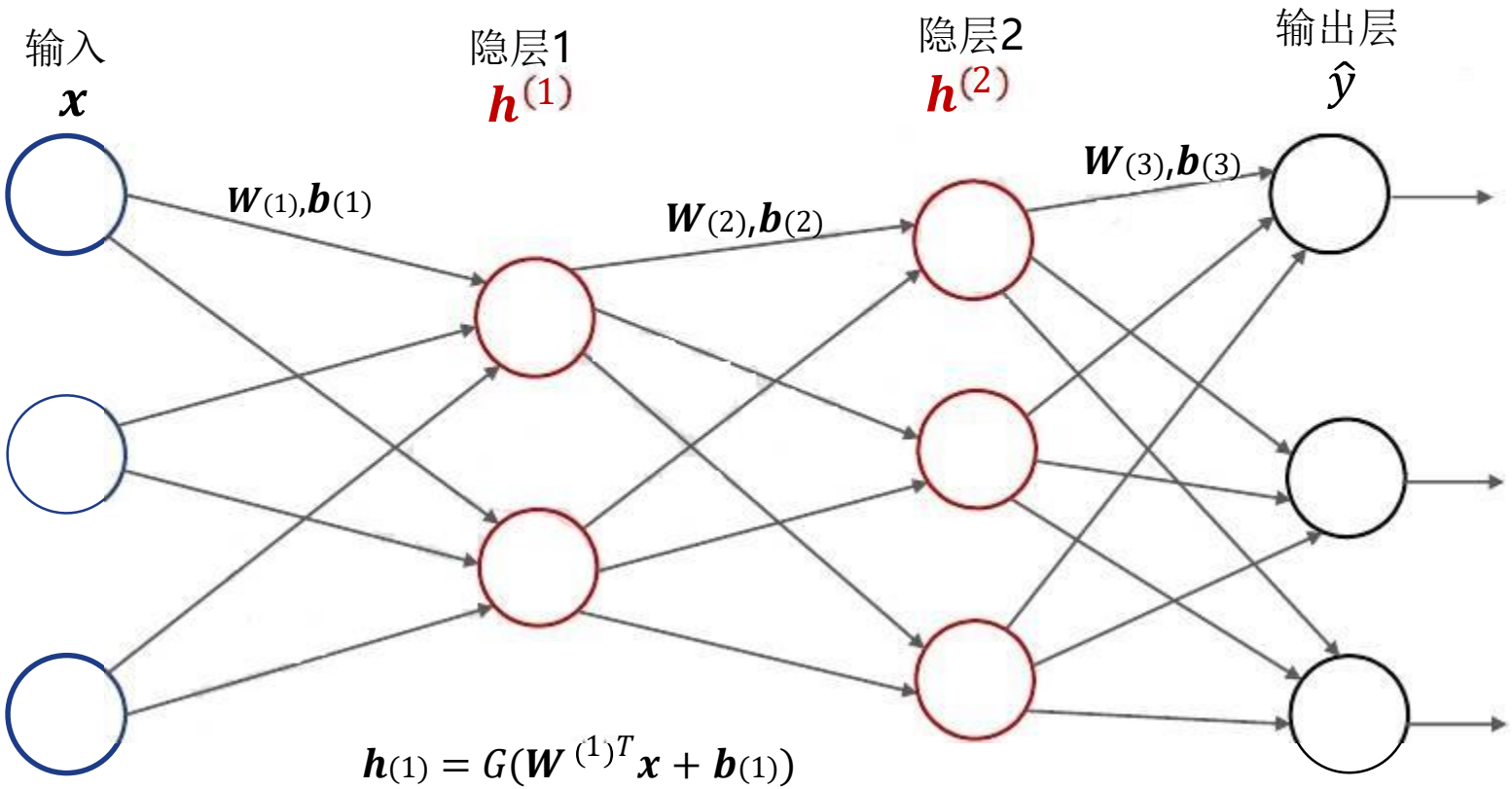
Geoffery Hinton

深度神经网络的成功：ABC

深度神经网络不断发展不仅依赖于自身的结构优势，也依赖于如下一些外在因素

- Algorithm：算法日新月异，优化算法层出不穷（学习算法->BP 算法-> Pre-training，Dropout等方法）
- Big data：数据量不断增大（10-> 10 k ->100M）
- Computing：处理器计算能力的不算提升（晶体管->CPU ->集群/GPU ->智能处理器）

多层神经网络（深度学习）-层数不断增加

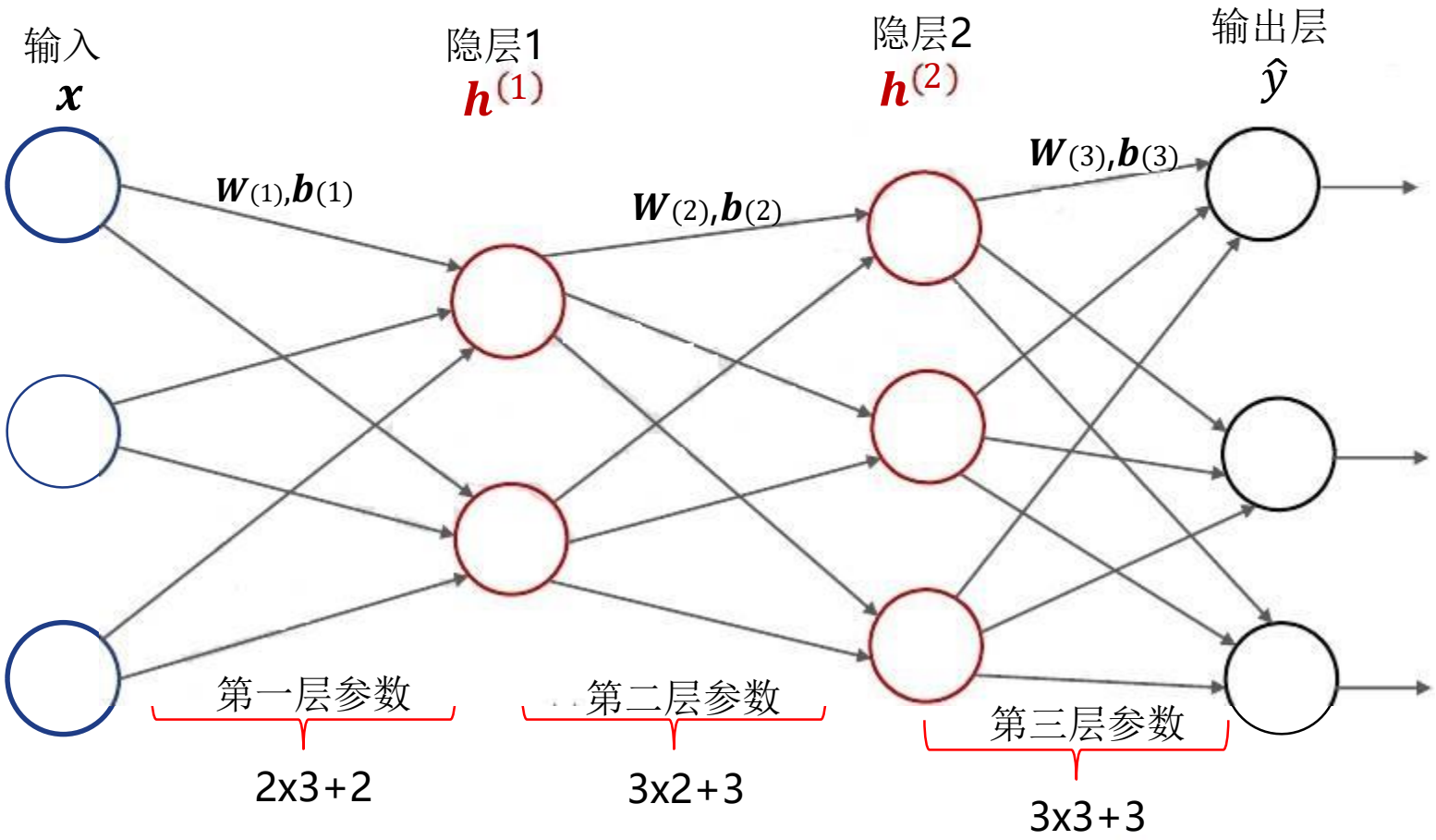


推导公式

$$\boldsymbol{h}(1) = G(\boldsymbol{W}^{(1)T}\boldsymbol{x} + \boldsymbol{b}(1))$$

$$\boldsymbol{h}(2) = G(\boldsymbol{W}^{(2)T}\boldsymbol{h}^{(1)} + \boldsymbol{b}(2))$$

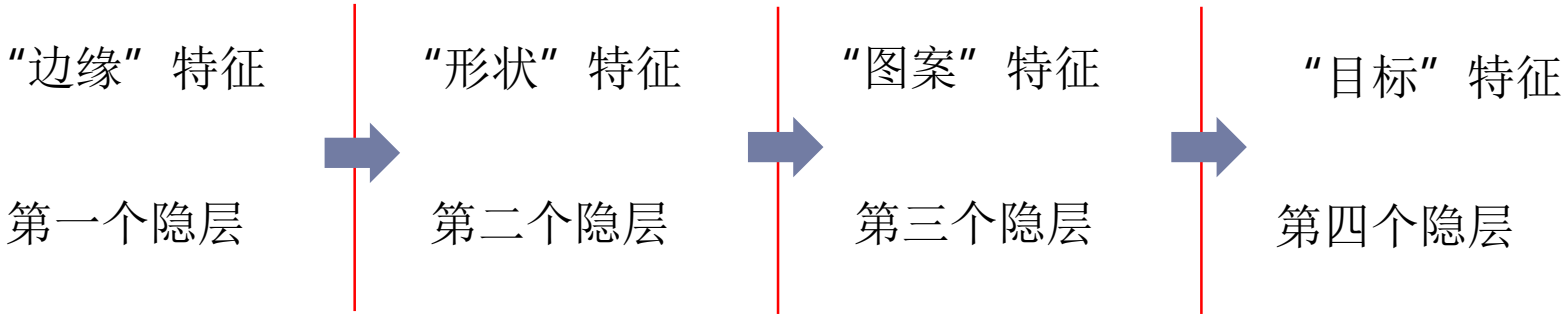
$$\hat{y} = G(\boldsymbol{W}^{(3)T}\boldsymbol{h}^{(2)} + \boldsymbol{b}(3))$$


输入
$\boldsymbol{x}$
隐层1
$\boldsymbol{h}^{(1)}$
隐层2
$\boldsymbol{h}^{(2)}$
输出层
$\hat{y}$
$\boldsymbol{W}(1),\boldsymbol{b}(1)$
$\boldsymbol{W}(2),\boldsymbol{b}(2)$
$\boldsymbol{W}(3),\boldsymbol{b}(3)$
第一层参数
2x3+2
第二层参数
3x2+3
第三层参数
3x3+3

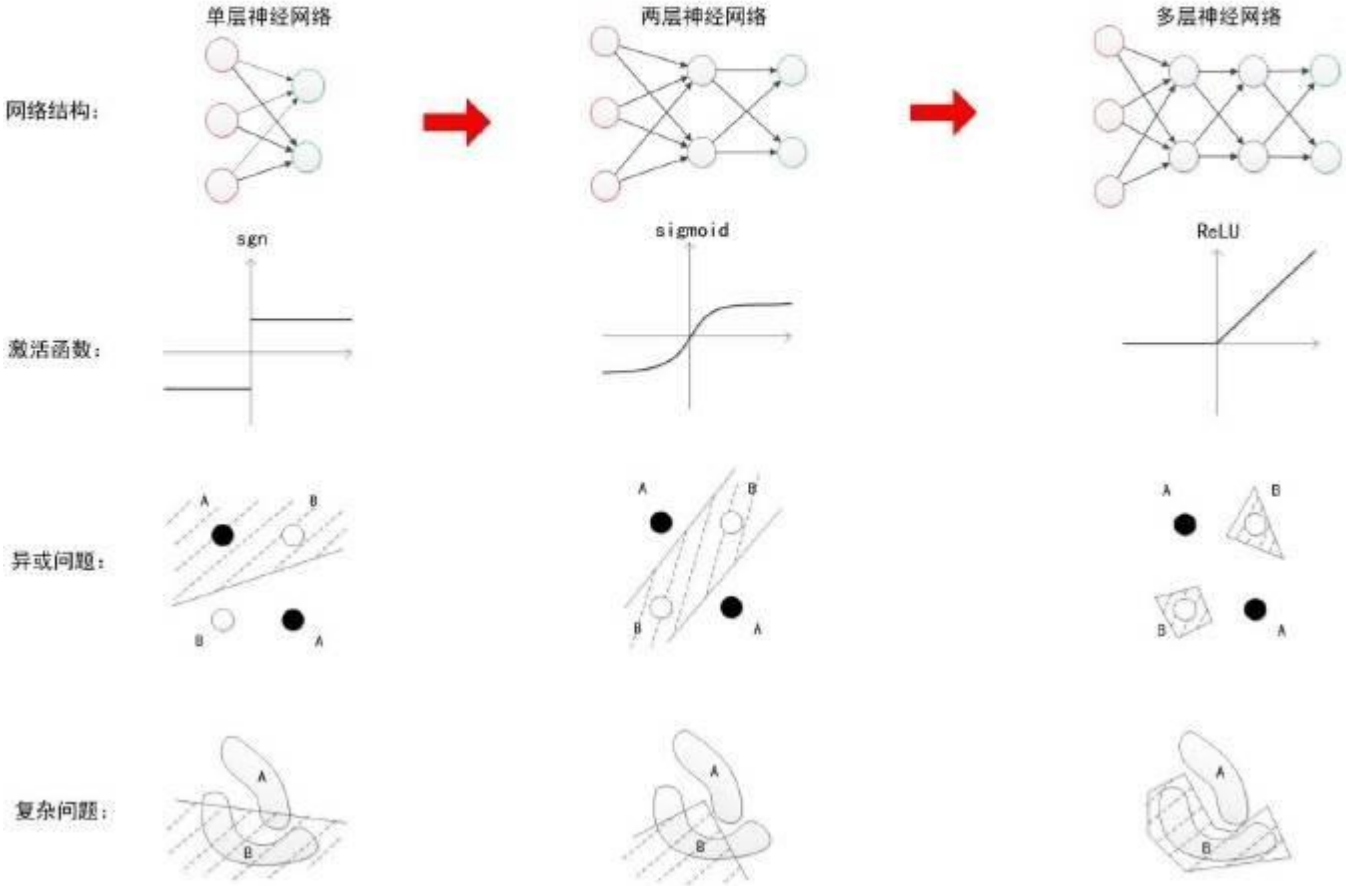
需 6+6+9+8=29个参数

- 随着网络的层数增加，每一层对于前一层次的抽象表示更深入，每一层神经元学习到的是前一层神经元更抽象的表示

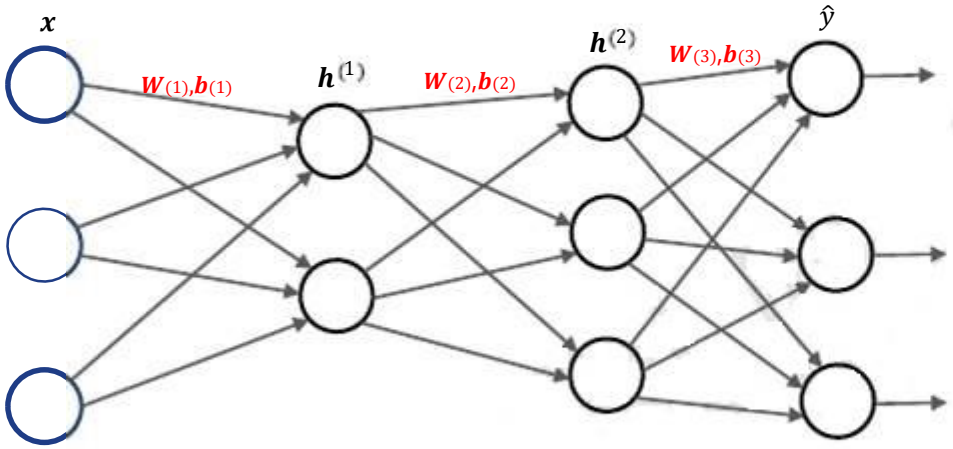
- 通过抽取更抽象的特征来对事物进行区分，从而获得更好的区分与分类能力



从单层神经网络，到两层神经网络，再到多层神经网络，随着网络层数的增加，以及激活函数的调整，神经网络拟合非线性分界不断增强。



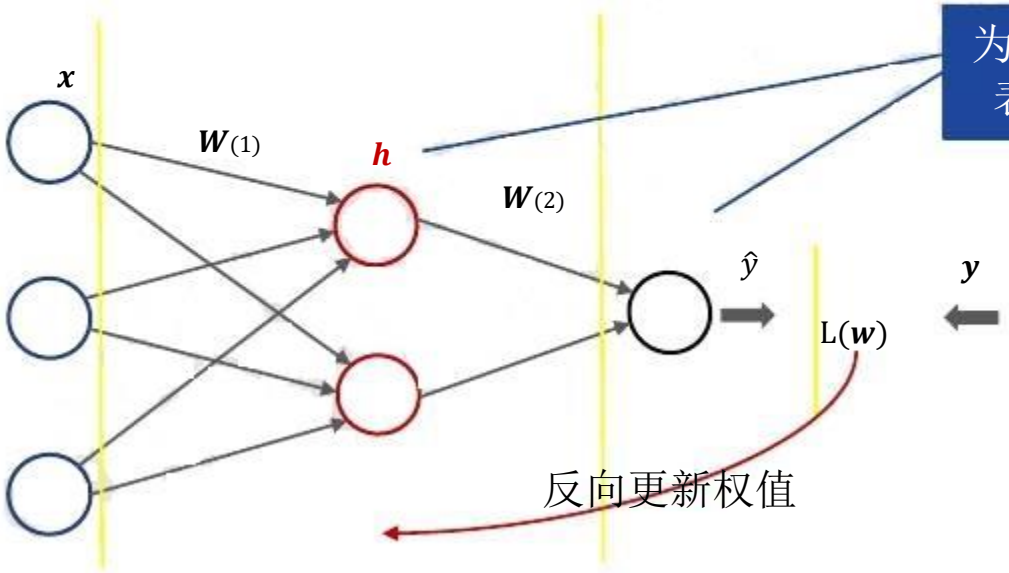
神经网络的模型训练



- 模型训练的目的，就是调整参数使得模型计算值$\hat{y}$尽可能的与真实值y逼近

神经网络训练

- 正向传播（推断）是根据输入，经过权重、激活函数计算出隐层，将输入的特征向量从低级特征逐步提取为抽象特征，直到得到最终输出结果的过程。



- 反向传播是根据正向传播的输出结果和期望值计算出损失函数，再通过链式求导，最终从网络后端逐步修改权重使输出和期望值的差距变到最小的过程。

BP神经网络

BP学习算法可分为前向传播预测与反向传播学习两个过程。

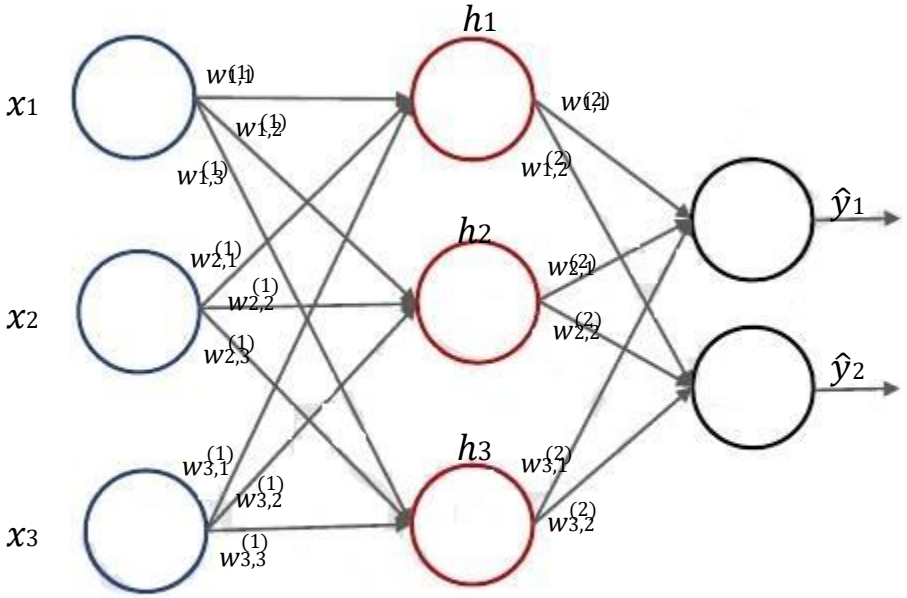
要学习的各参数值一般先作随机初始化。取训练样本输入网络，逐层前向计算输出，在输出层得到预测值，此为前向传播预测过程。

根据预测值与实际值的误差再从输出层开始逐层反向调节各层的参数，此为反向传播学习过程。

经过多样本的多次前向传播预测和反向传播学习过程，最终得到网络各参数的最终值。

正向传播

$\boldsymbol{x}$ $(\boldsymbol{W}(1),\boldsymbol{b}(1))$ $\boldsymbol{h}$ $(\boldsymbol{W}(2),\boldsymbol{b}(2))$ $\hat{y}$

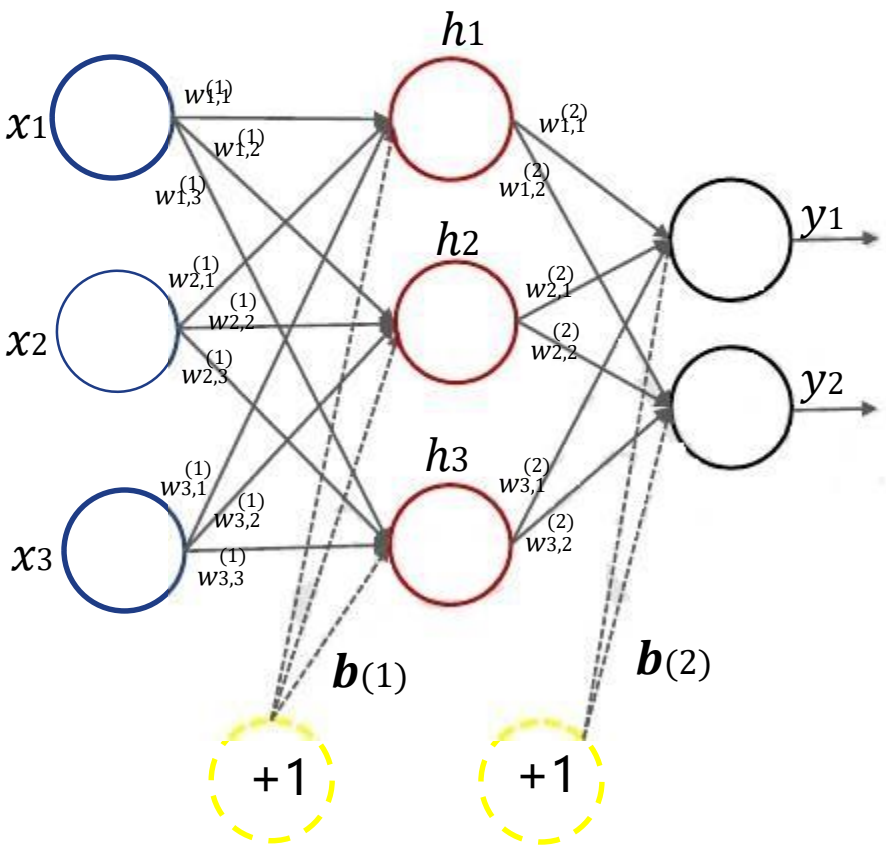


输入 $\boldsymbol{x} = [x_1;\ x_2;\ x_3]$

输入 $\boldsymbol{h} = [h_1;\ h_2;\ h_3]$

权重 $\boldsymbol{W}(1) = \begin{bmatrix} w_{1,1}^{(1)} & w_{1,2}^{(1)} & w_{1,3}^{(1)} \\ w_{2,1}^{(1)} & w_{2,2}^{(1)} & w_{2,3}^{(1)} \\ w_{3,1}^{(1)} & w_{3,2}^{(1)} & w_{3,3}^{(1)} \end{bmatrix}$

权重 $\boldsymbol{W}(2) = \begin{bmatrix} w_{1,1}^{(2)} & w_{1,2}^{(2)} \\ w_{2,1}^{(2)} & w_{2,2}^{(2)} \\ w_{3,1}^{(2)} & w_{3,2}^{(2)} \end{bmatrix}$



- 使用 sigmoid 函数作为激活函数

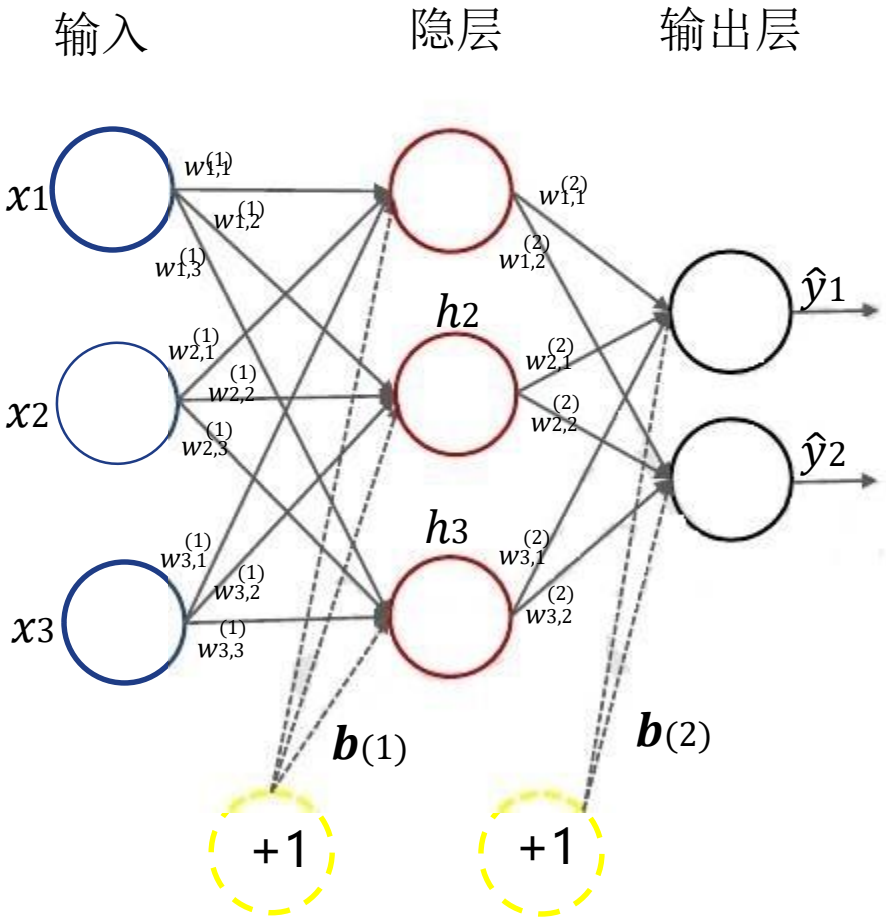
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

前向传输：输入到隐层

$$\boldsymbol{v} = \boldsymbol{W}^{(1)T}\boldsymbol{x}$$

$$= \begin{bmatrix} w_{1,1}^{(1)} & w_{2,1}^{(1)} & w_{3,1}^{(1)} \\ w_{1,2}^{(1)} & w_{2,2}^{(1)} & w_{3,2}^{(1)} \\ w_{1,3}^{(1)} & w_{2,3}^{(1)} & w_{3,3}^{(1)} \end{bmatrix} \begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} + \boldsymbol{b}(1)$$

$$\boldsymbol{h} = \frac{1}{1+e^{-v}}$$



- 输入：包含神经元 x_1，x_2，x_3
- 隐层：包含 h_1, h_2, h_3
- 输出层：包含 $\hat{y}_1$， $\hat{y}_2$

- 输入和隐层之间

  偏置：$\boldsymbol{b}(1)$

  权重：$\boldsymbol{W}(1)$

- 隐层和输出层之间

  偏置：$\boldsymbol{b}(2)$

  权重：$\boldsymbol{W}(2)$

示例

假定输入数据 $x_1 = 0.02$ 、$x_2 = 0.04$ 、$x_3 = 0.01$

固定偏置$\boldsymbol{b}^{(1)} = [0.4;0.4;0.4]$、$\boldsymbol{b}^{(2)} = [0.7;0.7]$

期望输出$y_1 = 0.9$、$y_2 = 0.5$

未知权重 $\boldsymbol{W}^{(1)} = \begin{bmatrix} w_{1,1}^{(1)} & w_{1,2}^{(1)} & w_{1,3}^{(1)} \\ w_{2,1}^{(1)} & w_{2,2}^{(1)} & w_{2,3}^{(1)} \\ w_{3,1}^{(1)} & w_{3,2}^{(1)} & w_{3,3}^{(1)} \end{bmatrix}$, $\boldsymbol{W}^{(2)} = \begin{bmatrix} w_{1,1}^{(2)} & w_{1,2}^{(2)} \\ w_{2,1}^{(2)} & w_{2,2}^{(2)} \\ w_{3,1}^{(2)} & w_{3,2}^{(2)} \end{bmatrix}$

目的是为了能得到 $y_1 = 0.9$、$y_2 = 0.5$的期望的值，需计算出合适的 $\boldsymbol{W}^{(1)}$、$\boldsymbol{W}^{(2)}$ 的权重值

- 初始化权重值

$$\boldsymbol{W}^{(1)} = \begin{bmatrix} w_{1,1}^{(1)} & w_{1,2}^{(1)} & w_{1,3}^{(1)} \\ w_{2,1}^{(1)} & w_{2,2}^{(1)} & w_{2,3}^{(1)} \\ w_{3,1}^{(1)} & w_{3,2}^{(1)} & w_{3,3}^{(1)} \end{bmatrix} = \begin{bmatrix} 0.25 & 0.15 & 0.30 \\ 0.25 & 0.20 & 0.35 \\ 0.10 & 0.25 & 0.15 \end{bmatrix}$$

$$\boldsymbol{W}^{(2)} = \begin{bmatrix} w_{1,1}^{(2)} & w_{1,2}^{(2)} \\ w_{2,1}^{(2)} & w_{2,2}^{(2)} \\ w_{3,1}^{(2)} & w_{3,2}^{(2)} \end{bmatrix} = \begin{bmatrix} 0.40 & 0.25 \\ 0.35 & 0.30 \\ 0.01 & 0.35 \end{bmatrix}$$

- 输入到隐层计算

$$\boldsymbol{v}=\begin{bmatrix} v1 \\ v2 \\ v3 \end{bmatrix} = \boldsymbol{W}^{(1)^T}\boldsymbol{x} + \boldsymbol{b}^{(1)} = \begin{bmatrix} 0.25 & 0.25 & 0.10 \\ 0.15 & 0.20 & 0.25 \\ 0.30 & 0.35 & 0.15 \end{bmatrix} \begin{bmatrix} 0.02 \\ 0.04 \\ 0.01 \end{bmatrix} + \begin{bmatrix} 0.4 \\ 0.4 \\ 0.4 \end{bmatrix} = \begin{bmatrix} 0.416 \\ 0.4135 \\ 0.4215 \end{bmatrix}$$

$$\boldsymbol{h} = \begin{bmatrix} h1 \\ h2 \\ h3 \end{bmatrix} = \frac{1}{1+e^{-v}} = \begin{bmatrix} \frac{1}{1+e^{-0.416}} \\ \frac{1}{1+e^{-0.4135}} \\ \frac{1}{1+e^{-0.4215}} \end{bmatrix} = \begin{bmatrix} 0.6025 \\ 0.6019 \\ 0.6038 \end{bmatrix}$$

- 隐层到输出层计算

$$\boldsymbol{z} = \begin{bmatrix} z1 \\ z2 \end{bmatrix} = \boldsymbol{W}^{(2)T}\boldsymbol{h} + \boldsymbol{b}^{(2)} = \begin{bmatrix} 0.40 & 0.35 & 0.01 \\ 0.25 & 0.30 & 0.35 \end{bmatrix} \begin{bmatrix} 0.6025 \\ 0.6019 \\ 0.6038 \end{bmatrix} + \begin{bmatrix} 0.7 \\ 0.7 \end{bmatrix} = \begin{bmatrix} 1.1577 \\ 1.2425 \end{bmatrix}$$

$$\hat{y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \end{bmatrix} = \frac{1}{1+e^{-z}} = \begin{bmatrix} \frac{1}{1+e^{-1.1577}} \\ \frac{1}{1+e^{-1.2425}} \end{bmatrix} = \begin{bmatrix} 0.7609 \\ 0.7760 \end{bmatrix}$$

距离期望输出 $y_1 = 0.9$、$y_2 = 0.5$ 还有差距，通过反向传播修改权重

模型计算输出

$$\begin{bmatrix}\hat{y}_1\\ \hat{y}_2\end{bmatrix}=\begin{bmatrix}0.7609\\ 0.7760\end{bmatrix}$$

期望输出

$$y_1=0.9$$

$$y_2=0.5$$

- 计算误差

$$L(\boldsymbol{W})=L_1+L_2==\frac{1}{2}(y_1-\hat{y}_1)^2+\frac{1}{2}(y_2-\hat{y}_2)^2$$

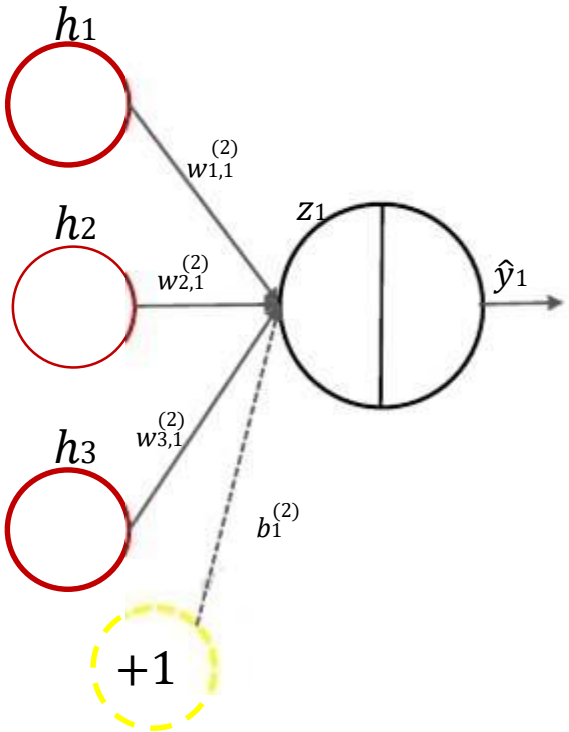
$$=\frac{1}{2}(0.7609-0.9)^2+\frac{1}{2}(0.7760-0.5)^2=0.0478$$

计算值与真实值之间还有很大的差距，如何缩小计算值与真实值之间的误差？

通过反向传播进行反馈，调节权重值

反向传播

- 隐层到输出层的权值$\boldsymbol{W}^{(2)}$的更新



$$L(\boldsymbol{W}) = L_1 + L_2 == \frac{1}{2}(y_1 - \hat{y}_1)^2 + \frac{1}{2}(y_2 - \hat{y}_2)^2$$

- 以 $w_{2,1}^{(2)}$(记为 ω) 参数为例子，计算 ω 对整体误差的影响有多大，可以使用整体误差对 ω 参数求偏导

- 根据偏导数的链式法则推导

$$\frac{\partial L(\boldsymbol{W})}{\partial \omega} = \frac{\partial L(\boldsymbol{W})}{\partial \hat{y}_1} \frac{\partial \hat{y}_1}{\partial z_1} \frac{\partial z_1}{\partial \omega}$$

$$L(\boldsymbol{W}) = \frac{1}{2}(y_1 - \hat{y}_1)^2 + \frac{1}{2}(y_2 - \hat{y}_2)^2$$

$$\frac{\partial L(\boldsymbol{W})}{\partial \hat{y}_1} = -(y_1 - \hat{y}_1) = -(0.9 - 0.7609) = -0.1391$$

$$\hat{y}_1 = \frac{1}{1 + e^{-z_1}}$$

$$\frac{\partial \hat{y}_1}{\partial z_1} = \hat{y}_1(1 - \hat{y}_1) = 0.7609 * (1 - 0.7609) = 0.1819$$

- 根据偏导数的链式法则推导

$$\frac{\partial L(\boldsymbol{W})}{\partial \omega} = \frac{\partial L(\boldsymbol{W})}{\partial \hat{y}_1} \frac{\partial \hat{y}_1}{\partial z_1} \frac{\partial z_1}{\partial \omega}$$

$$z_1 = w_{1,1}^{(2)} \times h_1 + \omega \times h_2 + w_{3,1}^{(2)} \times h_3 + b_1^{(2)}$$

$$\frac{\partial z_1}{\partial \omega} = h_2 = 0.6019$$

$$\frac{\partial L(\boldsymbol{W})}{\partial \omega} = -(y_1 - \hat{y}_1) \times \hat{y}_1(1 - \hat{y}_1) \times h_2$$
$$= -0.1391 \times 0.1819 \times 0.6019 = -0.0152$$

➢ 更新$w_{2,1}^{(2)}$（记为ω）的权重

初始值

$$\boldsymbol{W}^{(2)} = \begin{bmatrix} w_{1,1}^{(2)} & w_{1,2}^{(2)} \\ w_{2,1}^{(2)} & w_{2,2}^{(2)} \\ w_{3,1}^{(2)} & w_{3,2}^{(2)} \end{bmatrix} = \begin{bmatrix} 0.40 & 0.25 \\ 0.35 & 0.30 \\ 0.01 & 0.35 \end{bmatrix}$$

$$\frac{\partial L(\boldsymbol{W})}{\partial \omega} = -0.0152$$

$$\omega = \omega - \alpha \times \frac{\partial L(\boldsymbol{W})}{\partial \omega} = 0.35 - (-0.0152) = 0.3652$$

➢同理，可以计算新的$\boldsymbol{W}^{(2)}$的其他元素的权重值

- 反向传播的作用是将神经网络的输出误差反向传播到神经网络的输入端，并以此来更新神经网络中各个连接的权重

- 当第一次反向传播法完成后，网络的模型参数得到更新，网络进行下一轮的正向传播过程，如此反复的迭代进行训练，从而不断缩小计算值与真实值之间的误差。

其他代表性神经网络

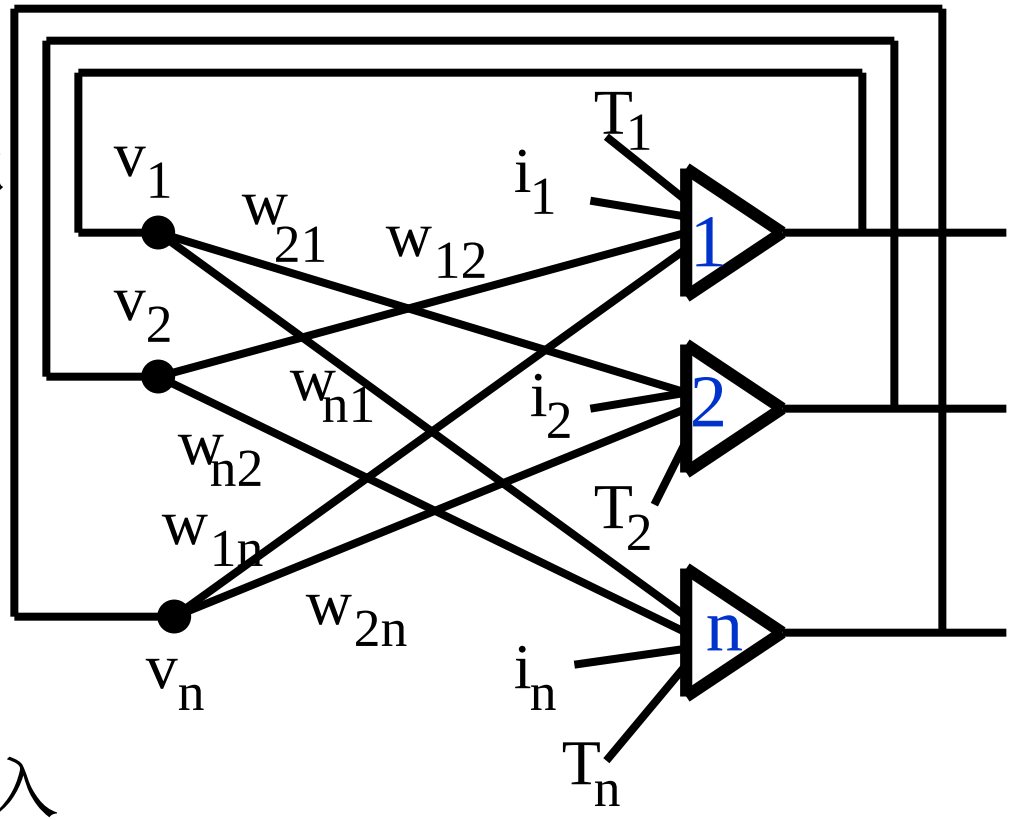
——Hopfield反馈神经网络

- 1982年，美国物理学家Hopfield提出的单层全互联含有对称突触连接的反馈网络是最典型的反馈网络模型。
- Hopfield用能量函数的思想形成了一种新的计算方法，阐明了神经网络与动力学的关系，并用非线性动力学的方法来研究这种网络的特性。
- 把神经网络看作一种非线性的动力学系统，并特别注意其稳定性研究的学科，被称为神经动力学(Neurodynamics)

离散Hopfield网络模型

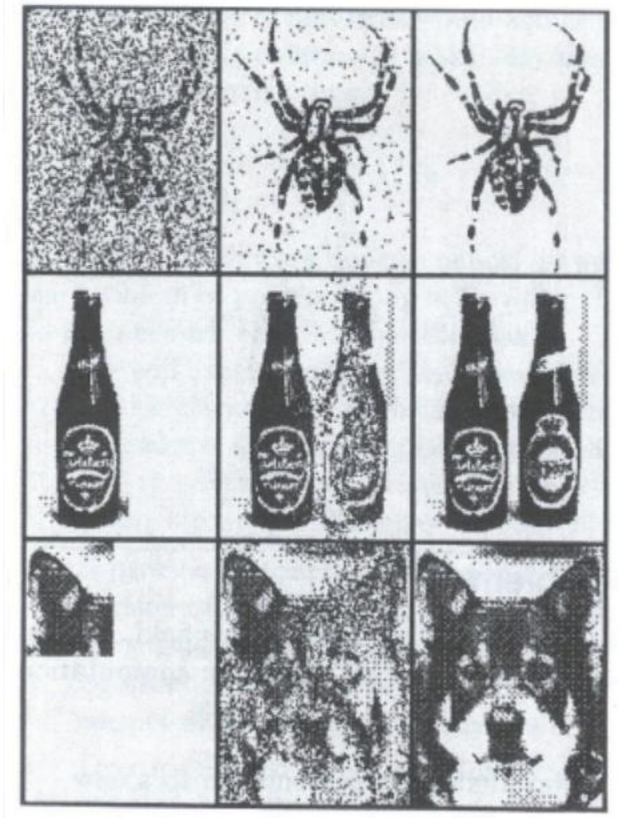
1. 单层网络, n 个神经元
2. T_i -- 神经元i的阈值

3. w_{ij} --神经元j 到i的权值
4. v_j --神经元j的输出（状态）
5. i_i -- 神经元i的外部输入



Hopfield网络模型

联想记忆功能：将最右侧三张图输入网络中可以利用参数权值进行存储。如果再输入左侧这种只有部分内容的图片，网络可以从记忆中取出完整的图像。



神经网络的模型训练

训练完了结果就是不准，怎么办？

- 调整网络拓扑结构
- 选择合适的激活函数
- 选择合适的损失函数

神经网络的拓扑调节

神经网络的结构一般为：输入×隐层×输出层

输入：神经元个数=特征维度

输出层：神经元个数=分类类别数

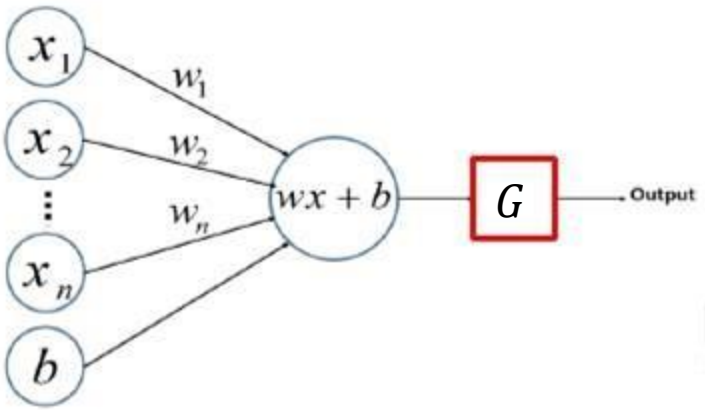
给定训练样本后，输入和输出层节点数便已确定

隐层：

- 隐层的数量？
- 隐层神经元的个数？

- 隐层的设计：
  - 隐层节点的作用是提取输入特征中的隐藏规律，每个节点都赋予一定权重
  - 隐层节点数太少，则网络从样本中获取信息的能力就越差，无法反映数据集的规律；隐层节点数太多，则网络的拟合能力过强，可能拟合数据集中的噪声部分，导致模型泛化能力变差。

选择合适的激活函数



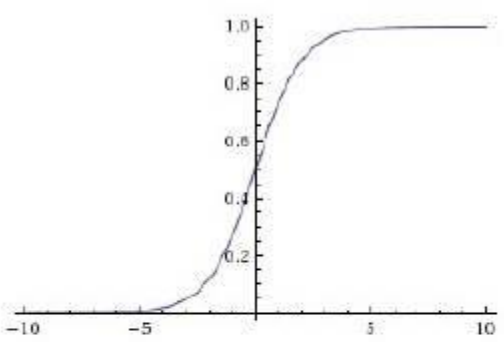
- 在神经元中，输入的数据通过加权求和后，还被作用了一个函数G，这个函数G就是激活函数（Activation Function）。
- 激活函数给神经元引入了非线性因素，使得神经网络可以任意逼近任何非线性函数，因此神经网络可以应用到众多的非线性模型中
- 激活函数需具备的性质
  - 可微性： 当优化方法是基于梯度的时候，这个性质是必须的。
  - 输出值的范围： 当激活函数输出值是有限的时候，基于梯度的优化方法会更加稳定，因为特征的表示受有限权值的影响更显著；当激活函数的输出是无限的时候，模型的训练会更加高效，不过在这种情况小，一般需要更小的学习率。

sigmoid函数

数学表达式

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

几何图像



- Sigmoid 是最常见的非线性激活函数
- 能够把输入的连续实值变换为0和1之间的输出；如果是非常大的负数，那么输出就变为0；如果是非常大的正数，输出就变为1。

- 非0均值的输出，导致w计算的梯度始终都是正的
- 计算机进行指数运算速度慢
- 饱和性问题及梯度消失现象

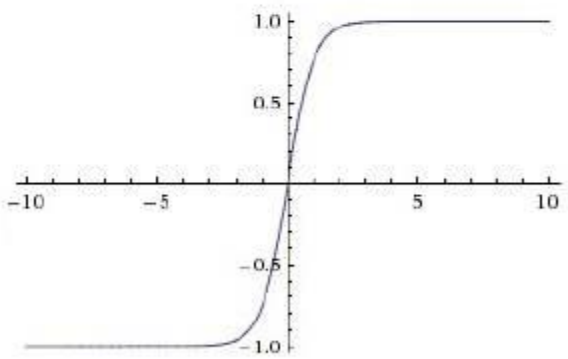
tanh函数

Sigmoid函数存在神经元会产生非0均值的输出的问题

寻找解决办法

$$\tanh(x) = \frac{\sinh(x)}{\cosh x} \quad \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$\tanh(x) = 2sigmoid(2x) - 1$$



- 与sigmoid相比，tanh是0均值的
- 在输入很大或是很小的时候，输出几乎平滑，梯度很小，不利于权重更新

梯度消失和梯度爆炸

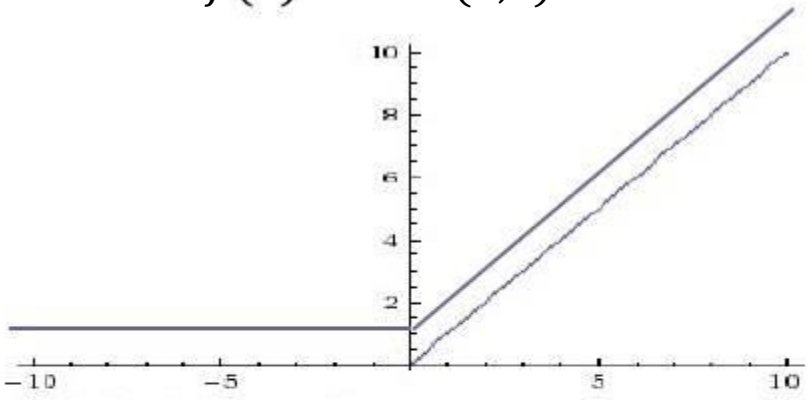
在校对误差反向传播的过程中，如果偏导数较小，在多次连乘之后，校对误差会趋近于0，导致梯度也趋近于0，前面层的参数无法有效更新，称之为梯度消失。梯度消失会使得增加再多的层也无法提高效果，甚至反而会降低。相反，如果偏导数较大，则会在反向传播的过程中呈指数级增长，导致溢出，无法计算，网络不稳定，称之为梯度爆炸。

ReLU函数

tanh 函 数 虽 然 解决了sigmoid 函数存在非0均值输出的问题，但仍然没改变梯度消失问题

寻找解决办法

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



- ReLU 能够在x>0时保持梯度不衰减，从而缓解梯度消失问题
- ReLU死掉。如果学习率很大，反向传播后的参数可能为负数，导致下一轮正向传播的输入为负数。当输入是负数的时候，ReLU是完全不被激活的，这就表明一旦输入到了负数，ReLU就会死掉
- 输出范围是无限的

PReLU/Leaky ReLU 函数

ReLU 在 x<0 时，ReLU完全不被激活

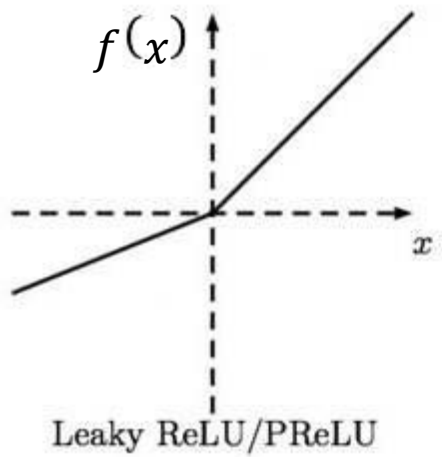
改进

出现了ReLU的改进版本：

Leaky ReLU

$f(x) = \max(\alpha x, x)$，$\alpha \in (0,1)$

- 负数区域内，Leaky ReLU有一个很小的斜率，可以避免ReLU死掉的问题



Leaky ReLU/PReLU

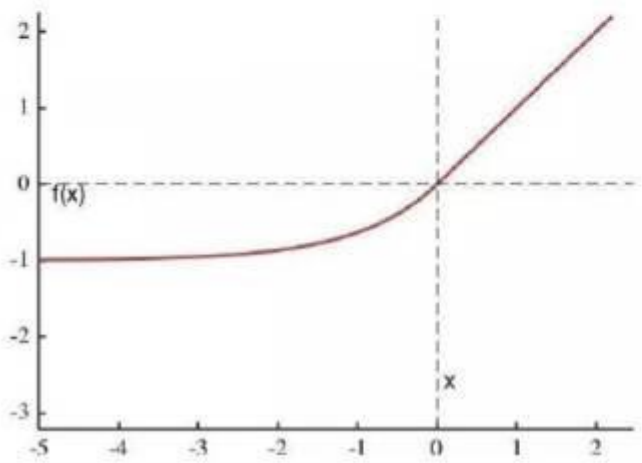
PReLU定义类似

- α为可调参数，每个通道有一个α，反向传播训练得到

ELU函数(Exponential Linear Unit)

融合sigmoid和ReLU

$$\text{ELU: } f(x) = \begin{cases} x & ,x > 0 \\ \alpha\,(e^{x} - 1) & ,x \le 0 \end{cases}$$



- α是可调参数，控制着ELU在负值区间的饱和位置
- ELU的输出均值接近于零，所以收敛速度更快
- 右侧线性部分使得ELU能够缓解梯度消失，而左侧软饱能够让ELU对输入变化或噪声更鲁棒，避免神经元死掉

选择恰当的损失函数

损失函数 $L = f(\hat{y},y)$，$\hat{y}$是模型预测值，是神经网络模型参数 W的函数，记作 $\hat{y} = Hw(x)$

从$\boldsymbol{w}$角度看，损失函数可以记为 $L(\boldsymbol{w}) = f(Hw(x),y)$

常用损失函数

➢均方差损失函数

- 均方差损失函数是神经网络优化常用的损失函数

$$L = \frac{1}{2}(y - \hat{y})2$$ ← 以一个神经元的均方差损失函数为例

假设使用sigmoid函数作为激活函数，则$\hat{y} = \sigma(z)$，其中$z = wx + b$，

$$\frac{\partial L}{\partial w} = (y - \hat{y})\,\sigma'(z)\,x \qquad \sigma'(z) = \big(1 - \sigma(z)\big) \cdot \sigma(z)$$

$$\frac{\partial L}{\partial b} = (y - \hat{y})\sigma'(z)$$

所求的与 $\frac{\partial L}{\partial w}$ 和 $\frac{\partial L}{\partial b}$ 梯度中都含有$\sigma'(z)$，当神经元输出接近1时，梯度将趋于0，出现梯度消失，导致神经网络反向传播时参数更新缓慢，学习效率下降。

均方差损失函数+Sigmoid激活函数出现问题—>如何解决？

➢引入交叉熵损失函数

交叉熵损失+Sigmoid激活函数可以解决输出层神经元学习率缓慢的问题。

➢交叉熵损失函数

- 交叉熵损失函数能够有效克服使用sigmoid函数时，均方差损失函数出现的参数更新慢的问题

交叉熵损失函数：

$$L = -\frac{1}{m}\sum_{x\in D}\sum_{i} y_i \ln(\hat{y_i})$$

其中，m 为训练样本的总数量，i 为分类类别

- 以二分类为例，交叉熵损失函数为：

$$L = -\frac{1}{m}\sum_{x\in D}(y\ln(\hat{y}) + (1-y)\ln(1-\hat{y}))$$

➢ 以二分类为例，则使用Sigmoid激活函数时的交叉熵损失函数为：

$$L = -\frac{1}{m}\sum_{x\in D}(y\ln(\hat{y}) + (1-y)\ln(1-\hat{y}))$$

$$\hat{y} = \sigma(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-(w^T x+b)}}$$

$$\frac{\partial L}{\partial w} = -\frac{1}{m}\sum_{x\in D}\left[\frac{y}{\sigma(z)} - \frac{1-y}{1-\sigma(z)}\right]\cdot\frac{\partial\sigma(z)}{\partial w} = -\frac{1}{m}\sum_{x\in D}\left[\frac{y}{\sigma(z)} - \frac{1-y}{1-\sigma(z)}\right]\cdot\sigma'(z)\cdot x = \frac{1}{m}\sum_{x\in D}\frac{\sigma'(z)\cdot x}{\sigma(z)(1-\sigma(z))}\cdot(\sigma(z)-y)$$

$$\sigma'(z) = (1-\sigma(z))\cdot\sigma(z)$$

$$\frac{\partial L}{\partial w} = -\frac{1}{m}\sum_{x\in D}(\sigma(z)-y)\cdot x$$

同理得：$$\frac{\partial L}{\partial b} = -\frac{1}{m}\sum_{x\in D}(\sigma(z)-y)$$

sigmoid的导数被约掉，这样最后一层的梯度中就没有$\sigma'(z)$

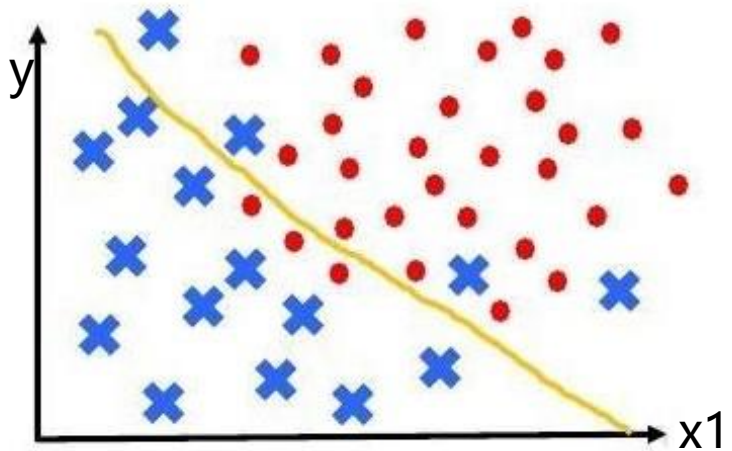
神经网络中损失函数的特性

- 同一个算法的损失函数不是唯一的
- 损失函数是参数(w,b)的函数
- 损失函数可以评价网络模型的好坏，损失函数越小说明模型和参数越符合训练样本(x, y)
- 损失函数是一个标量
- 选择损失函数时，挑选对参数(w,b)可微的函数（全微分存在，偏导数一定存在）
- 损失函数又称为代价函数、目标函数

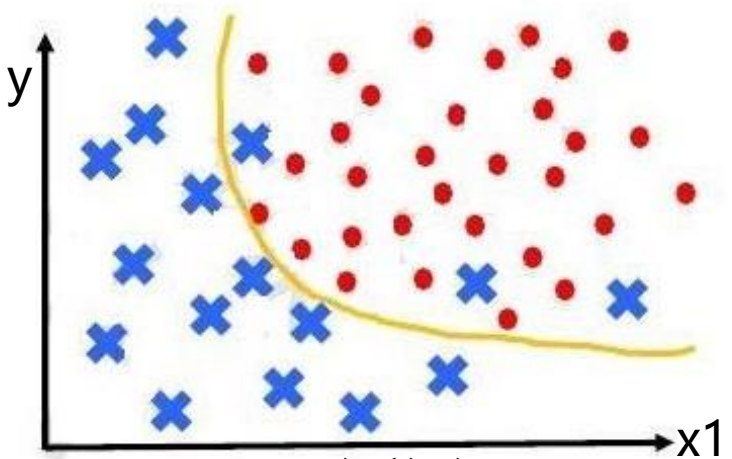
过拟合与泛化

- 机器学习不仅要求数据在训练集上求得一个较小的误差，在测试集上也要表现好。因为模型最终是要部署到没有见过训练数据的真实场景。提升模型在测试集上的预测效果叫做泛化。
- 过拟合（overfitting）指模型过度接近训练的数据，模型的泛化能力不足。具体表现为在训练数据集上测试的误差很低，但在验证数据集上的误差很大。
- 神经网络的层数增加，参数也跟着增加，表示能力大幅度增强，极容易出现过拟合现象。

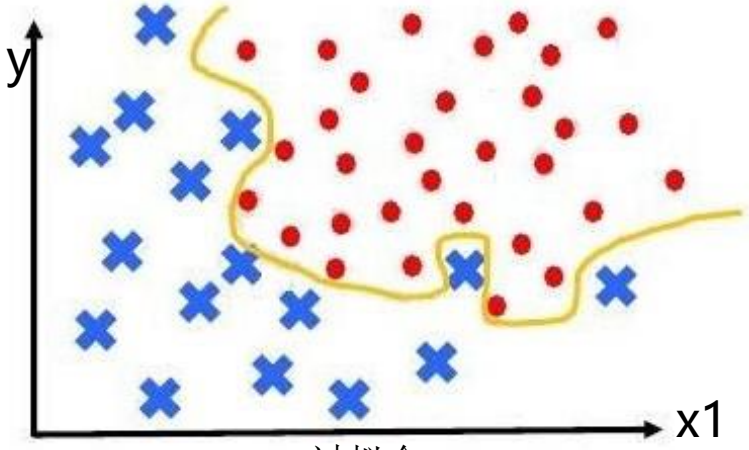
欠拟合和过拟合



欠拟合



合适拟合



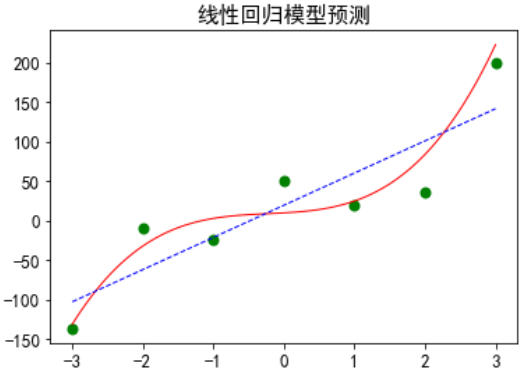
过拟合

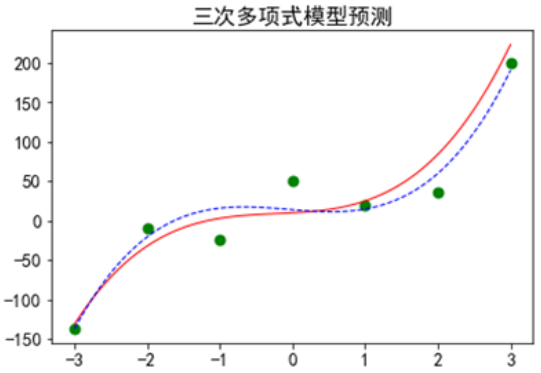
欠拟合：训练考虑的维度太少，拟合函数无法满足训练集，误差较大。

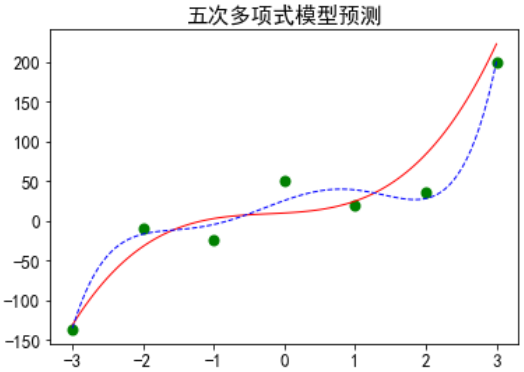
过拟合，训练考虑的维度太多，使得拟合的函数很完美的接近训练数据集，但泛化能力差，对新数据预测能力不足。

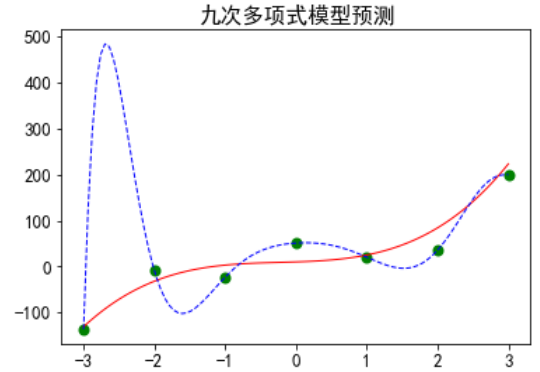
欠拟合、过拟合示例

模型在训练样本上产生的误差叫训练误差（training error。在测试样本上产生的误差叫测试误差（test error）。





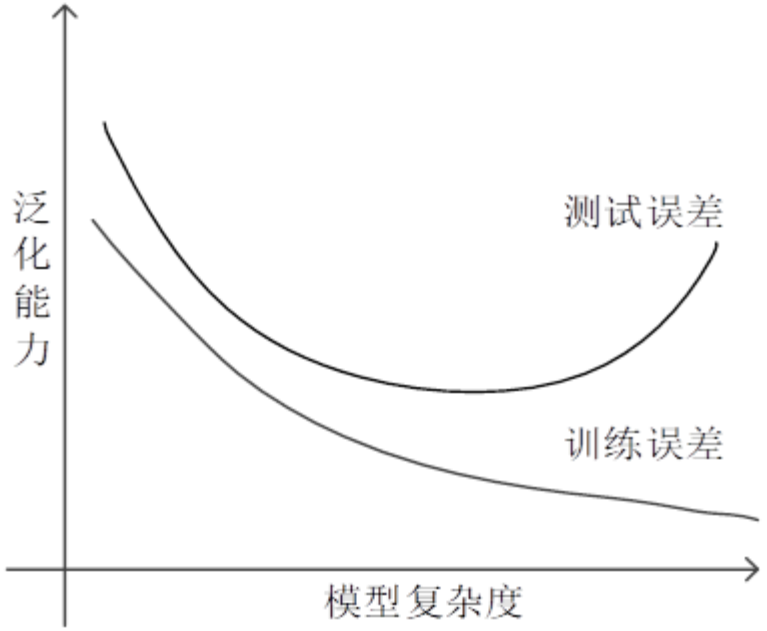




	线性回归模型	三次多项式模型	五次多项式模型	九次多项式模型
训练误差	2019	534	209	4
测试误差	578	247	1232	38492
和	2597	781	1441	38496

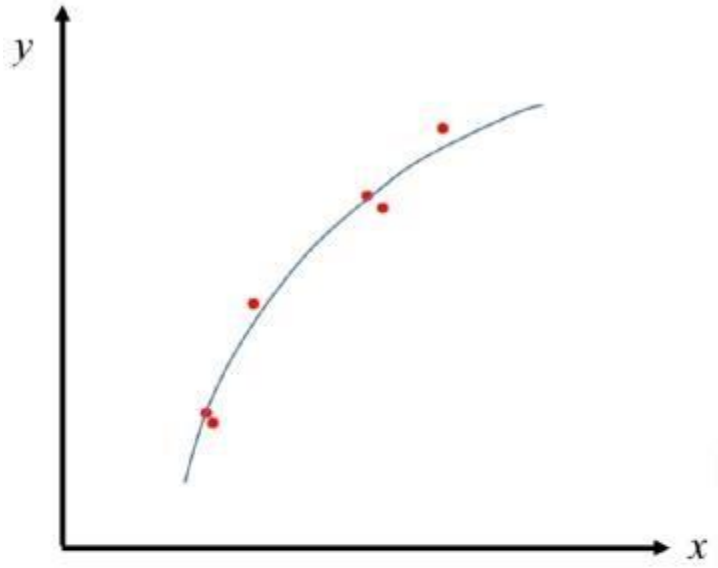
泛化能力与模型复杂度

衡量模型好坏的是测试误差，它标志了模型对未知新实例的预测能力，因此一般追求的是测试误差最小的那个模型。模型对新实例的预测能力称为泛化能力，模型在新实例上的误差称为泛化误差。

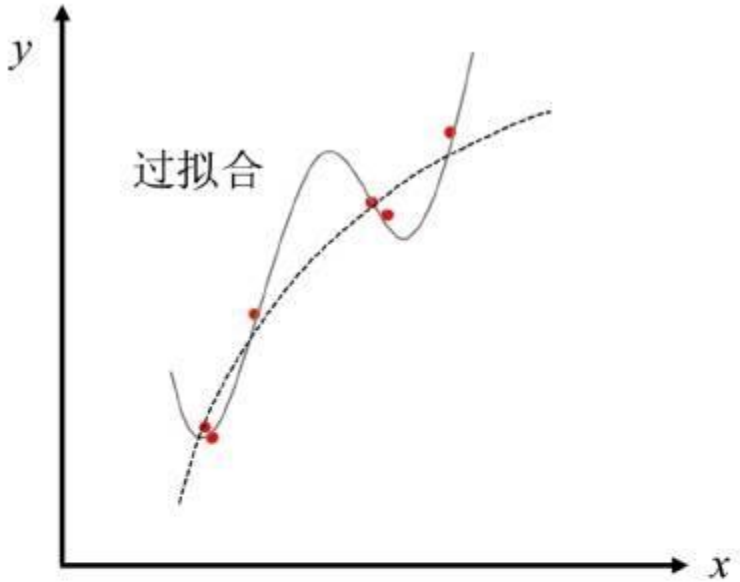


能够求解问题的模型往往不只一个。一般来说，只有合适复杂程度的模型才能最好地反映出训练集中蕴含的规律，取得最好的泛化能力。

正则化思路



拟合函数 $w_0+w_1x+w_2x2$



$w_0+w_1x+w_2x^2+w_3x^3+w_4x^4$

目标函数

$$L(\boldsymbol{w}) = \frac{1}{2m}\sum_{i=1}^{m}||y_i - \hat{y}_i||^2$$

加上惩罚项使 w_3、w_4 足够小

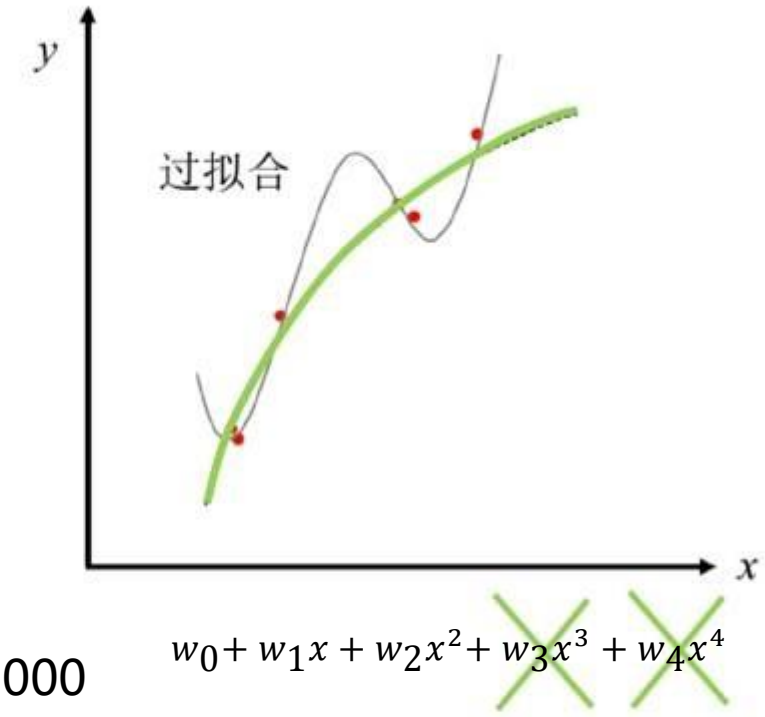
$$L(\boldsymbol{w}) = \frac{1}{2m}\sum_{i=1}^{m} ||y_i - \hat{y}_i||^2$$

$$\min_{\boldsymbol{w}} \frac{1}{2m}\sum_{i=1}^{m} ||y_i - \hat{y}_i||^2 + C1 * w_3^2 + C2 * w_4^2$$

C1、C2可取常数，如取1000

$$\min_{\boldsymbol{w}} \frac{1}{2m}\sum_{i=1}^{m} ||y_i - \hat{y}_i||^2 + 1000\, w_3^2 + 1000\, w_4^2$$

要使目标函数最小，则应有 $w_3 \approx 0$、$w_4 \approx 0$



$$\min_{w} \frac{1}{2m}\sum_{i=1}^{m} ||y_i - \hat{y}_i||^2 + C1 * w_3^2 + C2 * w_4^2$$

在损失函数中增加一个惩罚项，惩罚高阶参数，使其趋近于0

$$\min_{w} \frac{1}{2m}\sum_{i=1}^{m} ||y_i - \hat{y}_i||^2 + \theta \sum_{j=1}^{k} w_j^2$$

正则化项/惩罚项

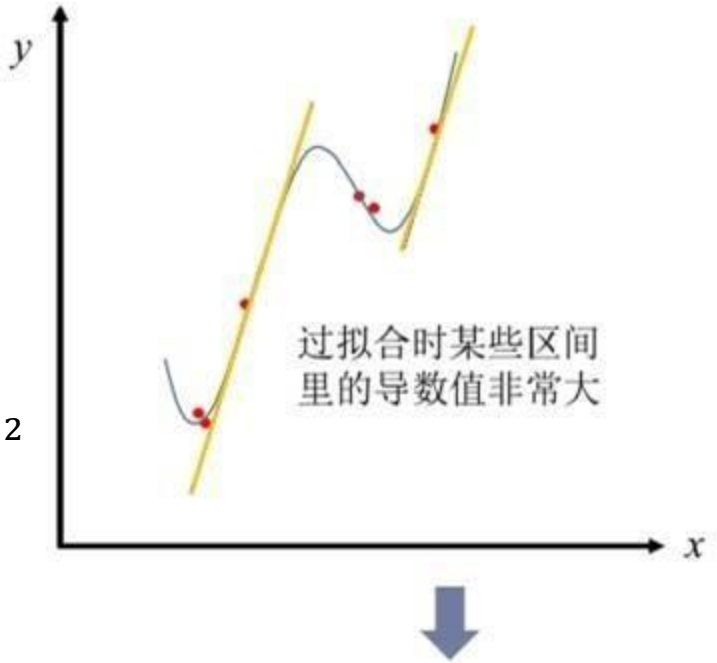
θ 为正则化参数，神经网络中的参数包括权重$\boldsymbol{w}$ 和偏置 b，正则化过程仅对权重 w 进行惩罚，正则化项记为

$$\Omega(\boldsymbol{w})$$

正则化后的损失函数记为

$$\tilde{L}(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y}) = \boldsymbol{L}(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y}) + \theta\Omega(\boldsymbol{w})$$

L^2正则化



$L2$正则化项 $\Omega(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2$

目标函数 $\tilde{L}(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y}) = L(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y}) + \frac{\theta}{2}||\boldsymbol{w}||^2$

$L2$ 正则化如何避免过拟合？

$$\nabla_w \tilde{L}(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y}) = \nabla_w L(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y}) + \theta\boldsymbol{w}$$

单步梯度更新权重 $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta(\nabla_w L(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y}) + \theta\boldsymbol{w})$

$$\boldsymbol{w} \leftarrow (\boldsymbol{1} - \eta\theta)\boldsymbol{w} - \eta\nabla_w L(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y})$$

通过$L2$正则化后，w权重值变小，网络的复杂度降低，对数据拟合的也更好。

L^1正则化

L_1正则化项是各个参数的绝对值之和

$$\boldsymbol{\Omega(w)} = \|\boldsymbol{w}\|_1 = \sum_i |w_i|$$

L_1正则化添加了一项符号函数$sign(\boldsymbol{w})$来影响梯度

目标函数 $\tilde{L}(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y}) = L(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y}) + \theta \|\boldsymbol{w}\|_1$

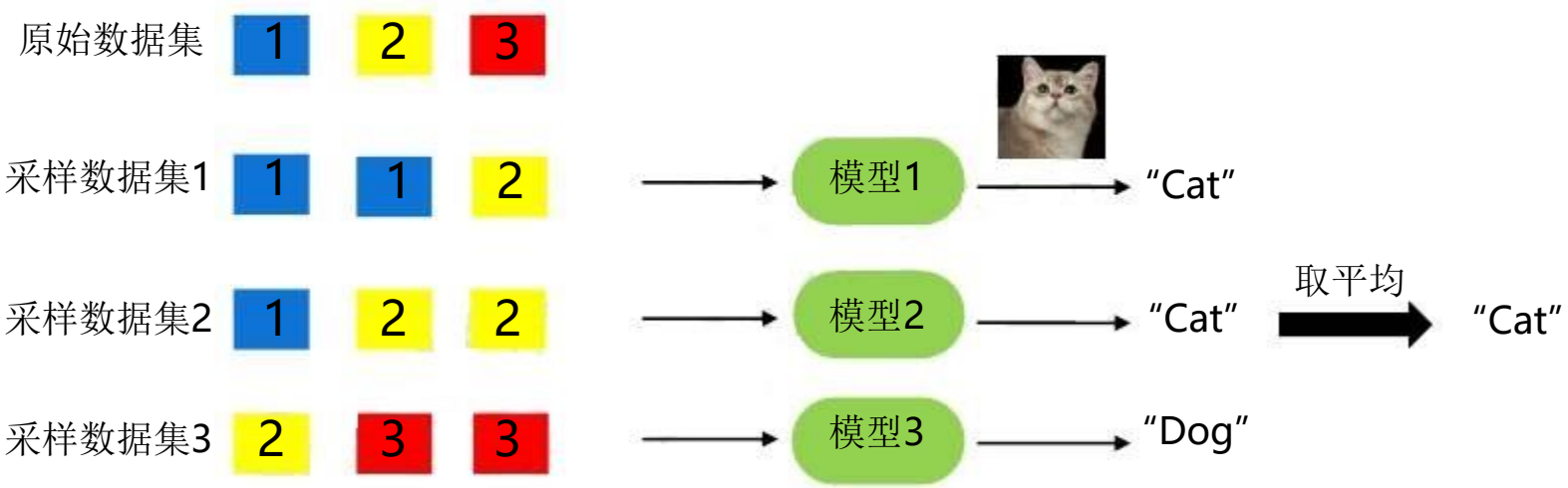
$$\nabla_w \tilde{L}(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y}) = \nabla_w L(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y}) + \theta sign(\boldsymbol{w})$$

L_1正则化通过加入一个符号函数，使得当w_i为正时，更新后的w_i变小，当w_i为负时，更新后的w_i变大，因此正则化后的效果就是让w_i接近0，这样网络中的权重就会接近0，从而也就减小了网络复杂度，防止了过拟合。

Bagging集成方法

- Bagging 训练不同的模型来共同决策测试样列的输出，不同的模型即使在同一个训练数据集上也会产生不同的误差。
- Bagging可以多次重复使用同一个模型、训练算法和目标函数进行训练。
- Bagging的数据集从原始数据集中重复采样获取，数据集大小与原始数据集保持一致。

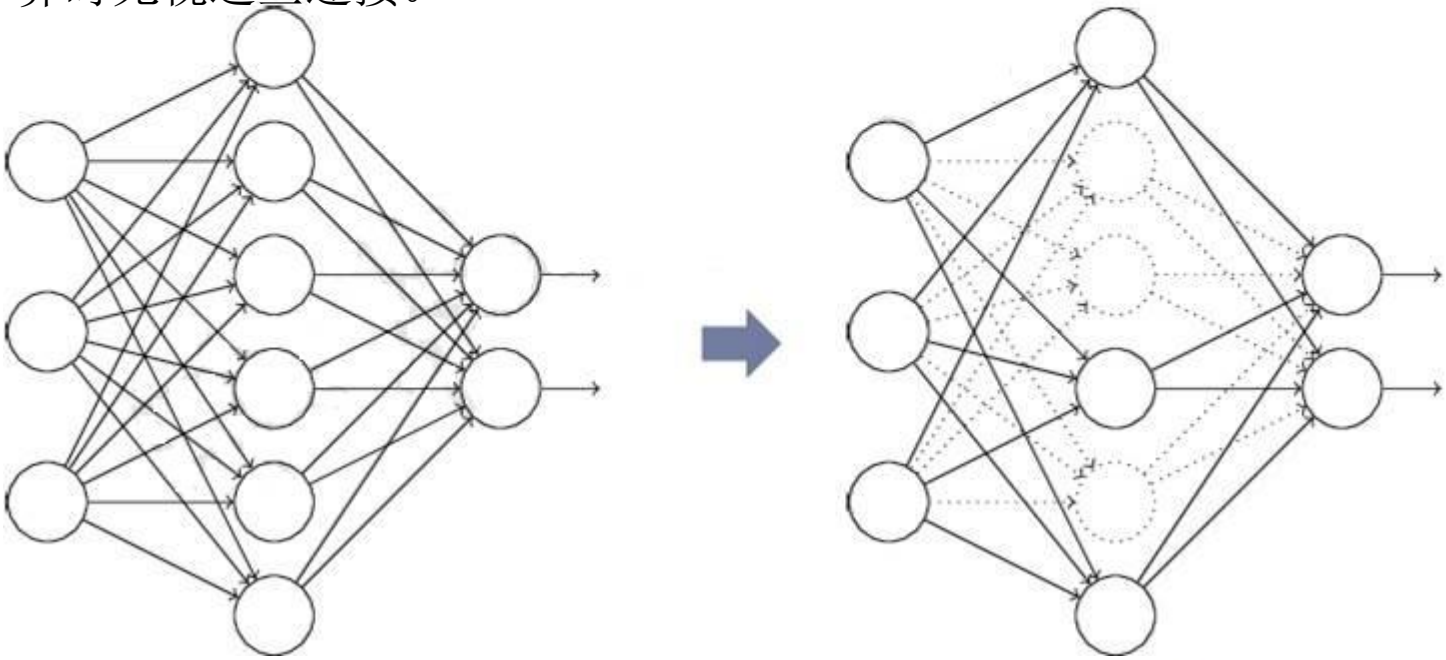
假设集成k=3个网络模型



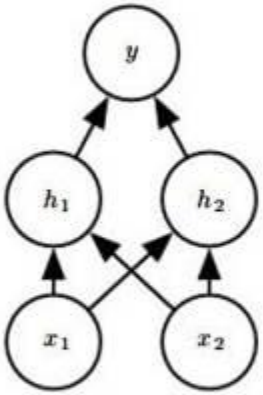
- 模型平均是减小泛化误差的一种可靠方法

Dropout 正则化

- L_2 和 L_1正则化是通过在目标函数中增加一项惩罚项，Dropout正则化是通过在训练时暂时修改神经网络来实现的。
- Dropout正则化思路：在训练过程中随机地“删除”一些隐层单元，在计算时无视这些连接。

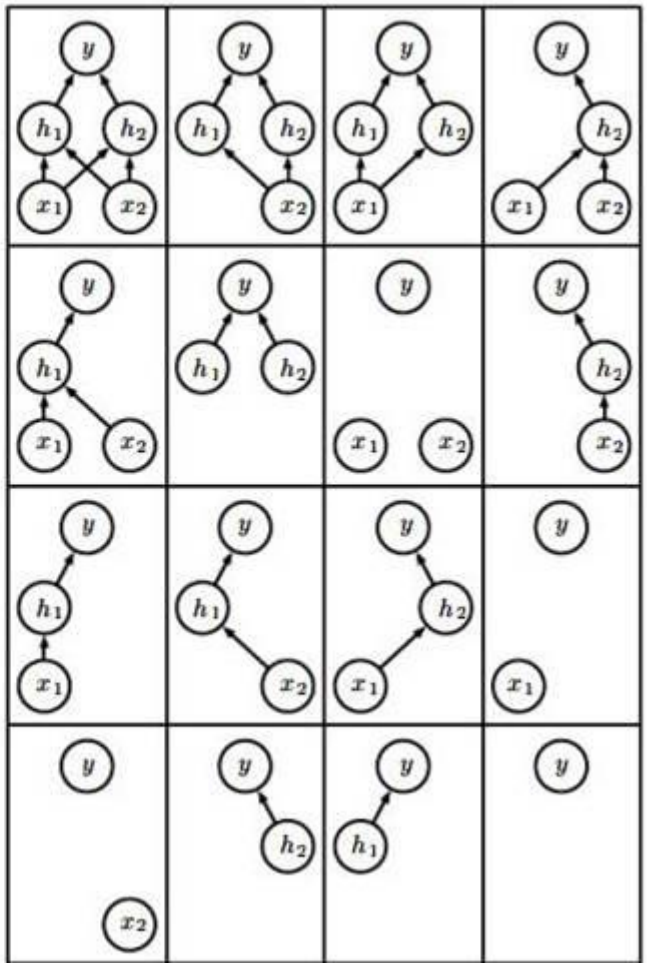


乘零的Dropout 算法



基础网络

从基础网络中丢弃不同的单元子集形成子网络



子网络集成

提前终止

提前终止是在模型迭代训练中，在模型对训练样本集收敛之前就停止迭代以防止过拟合的方法。

模型泛化能力评估的思路是将已有样本集划分为训练集和验证集，用训练集来训练模型，然后用验证集来验证模型的泛化能力。而提前终止提前引入验证集来验证模型的泛化能力，即在每一轮训练（一轮是指遍历所有训练样本一次）完后，就用验证集来验证泛化能力，如果n轮训练都没有使泛化能力没有提高，就停止训练。n是根据经验提前设定的参数。这种策略称为“No-improvement-in-n”，n常取10、20、30等值。

其他正则化方法

- 多任务学习

  多任务学习通过合并多个任务的样例来减少神经网络的泛化误差。

- 数据集增强

  使用更多的数据进行训练，可对原数据集进行变换形成新数据集添加到训练数据中。

- 参数共享

  强迫两个模型（监督模式下的训练模型和无监督模式下的训练模型）的某些参数相等，使其共享唯一的一组参数。

- 稀疏表示

  惩罚神经网络中的激活单元，稀疏化激活单元。

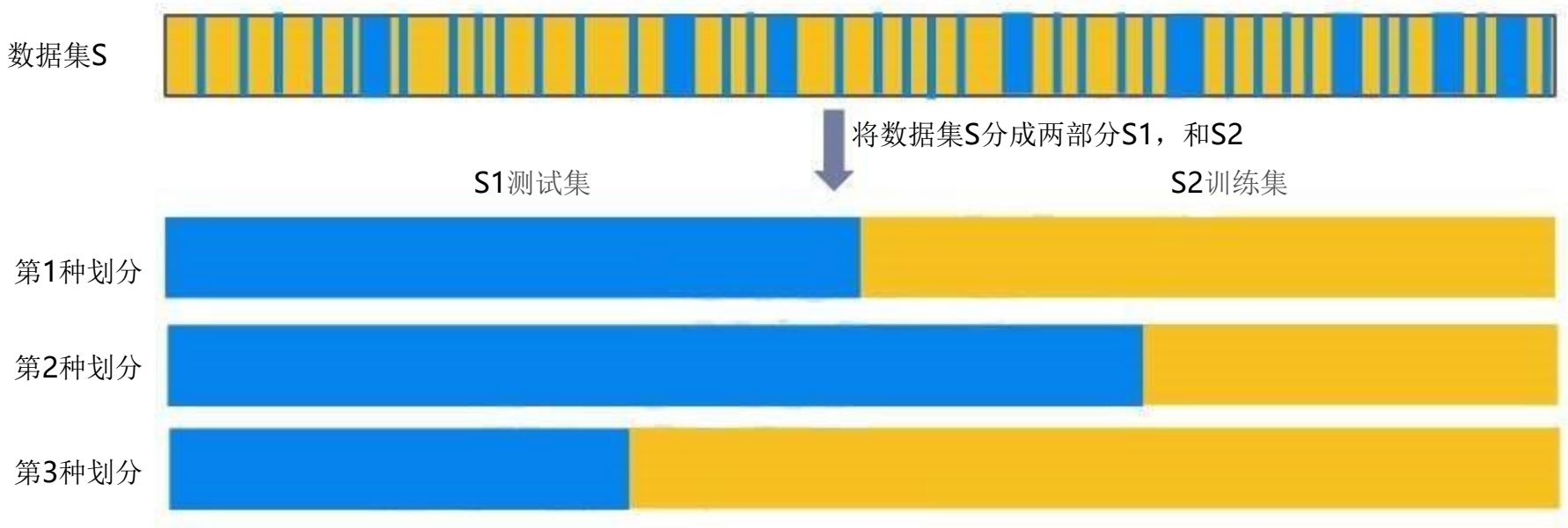
交叉验证

- 交叉验证的方式给每个样本作为测试集和训练集的机会，充分利用样本信息，保证了鲁棒性，防止过度拟合。

- 选择多种模型进行训练时，使用交叉验证能够评判各模型的鲁棒性。

最简单的验证方式

测试集和训练集

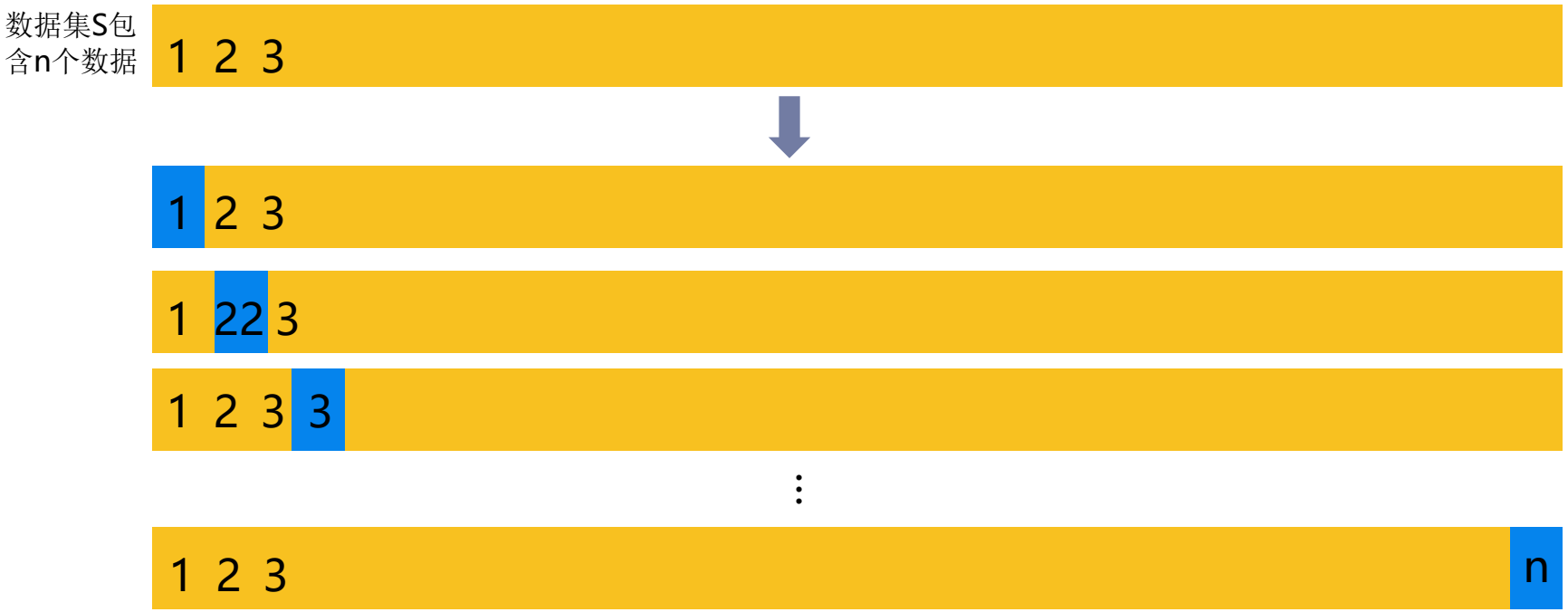


- 不同划分方式下，得到的MSE(Mean Squared Error)变动较大

缺点：最终模型与参数的选取将极大程度依赖于你对训练集和测试集的划分方法

只有部分数据参与了模型的训练

Leave-one-out cross-validation验证方法



每次取出一个数据作为测试集的唯一元素，而其他n-1个数据都作为训练集用于训练模型和调参。最终训练出n个模型，得到n个MSE。将这n个MSE取平均得到最终的test MSE。

缺点：计算量过大，耗费时间长

K-折交叉验证（k-fold cross validation）

不重复地每次取其中一份做测试集，用其他K-1份做训练集训练模型，之后计算该模型在测试集上的MSE_i，最后再将K次的MSE_i取平均得到最后的MSE

$$\text{MSE} = \frac{1}{K}\sum_{i=1}^{K} MSE_i$$

Leave-one-out cross-validation是一种特殊的K-fold Cross Validation（K=n）

优点：所有的样本都被作为了训练集和测试集，每个样本都被验证一次，

相比Leave-one-out cross-validation，计算成本降低，耗时减少