智能计算系统

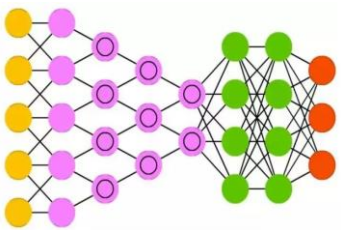
第二章 神经网络基础

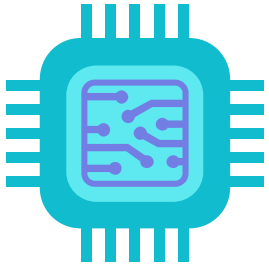
中国科学院计算技术研究所

陈云霁 研究员

cyj@ict.ac.cn

Driving Example




输入 ▶ 建模 ▶ 实现 ▶ 运行 ▶ 输出


神经网络基础


深度学习

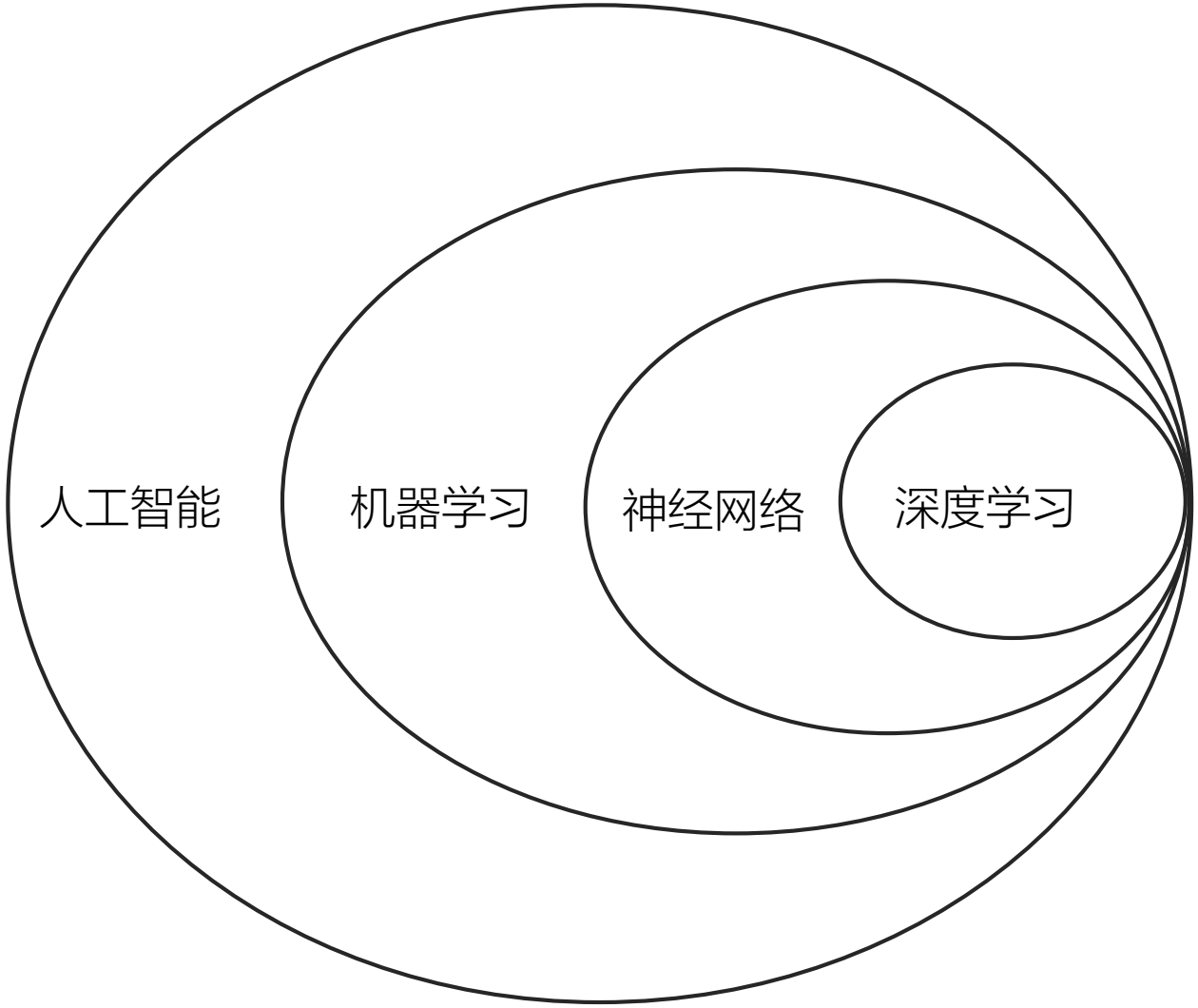
第二章将学习到搭建一个神经网络需要的基本知识，为深入理解深度学习打下基础

提纲

- 从机器学习到神经网络
- 正向传播与反向传播
- 神经网络设计原则
  - 网络的拓扑结构
  - 激活函数
  - 损失函数
- 过拟合与正则化
- 交叉验证
- 小结

包含关系

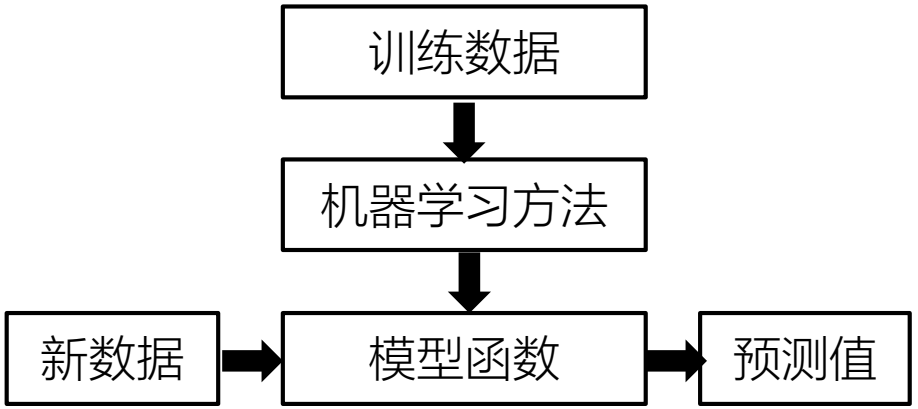
- 人工智能
- 机器学习
- 神经网络
- 深度学习



机器学习相关概念

- 机器学习是对能通过经验自动改进的计算机算法的研究（Mitchell）
- 机器学习是用数据或以往的经验，以此提升计算机程序的能力（Alpaydin）
- 机器学习是研究如何通过计算的手段、利用经验来改善系统自身性 能的一门学科（周志华）

典型机器学习过程



如何学习？如何理解？如何学会？

从最简单的线性模型开始，直至搭建出一个完整的神经网络架构

符号说明

输入数据： x

真实值（实际值）：y

计算值（模型输出值）：$\hat{y}$

模型函数： $H(x)$

激活函数： $G(x)$

损失函数： $L(x)$

标量：斜体小写字母 *a*、*b*、*c*

向量：黑斜体小写字母 **a**、**b**、**c**

矩阵：黑斜体大写字母**A**、**B**、**C**

线性回归

问题引入：假设房屋销售中心有这样一组关于房屋面积和房屋位置与销售价格的数据，用x_1表示房屋面积，x_2表示房屋楼层，y表示售价（万元）。

x_1	50	47	60	55	...	65
x_2	2	1	4	3	...	10
...					...	
y	50	42	80	52	...	?

设计一个回归程序进行预测

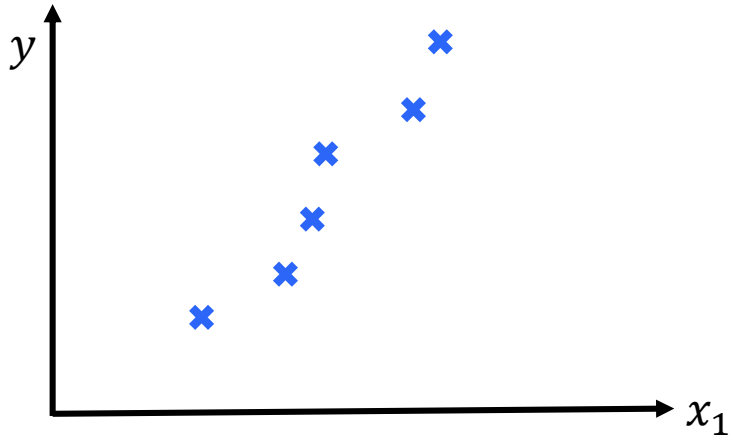
房屋面积x_1，与售价y的关系

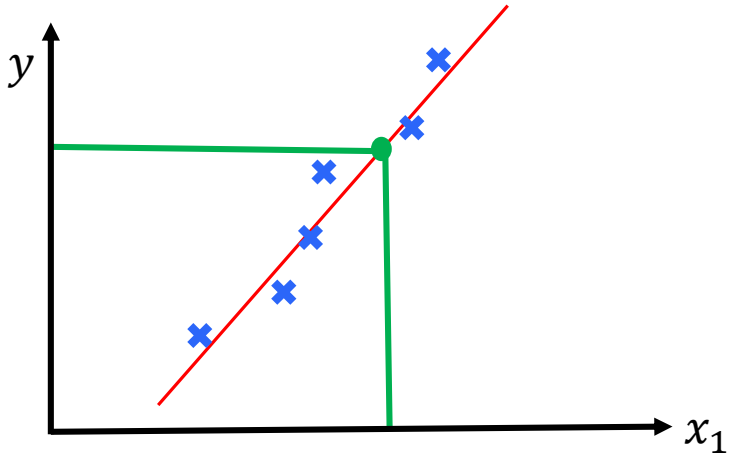
x_1	50	47	60	55	...	65
y	50	42	80	52	...	?

如果给出一个新的房屋面积，在销售记录中没有的，如何确定在该面积下的房屋售价y的值呢?

如预测$x_1 = 65$时的售价?

解决办法->寻找x_1和y的之间的关系，使用已有记录数据进行训练



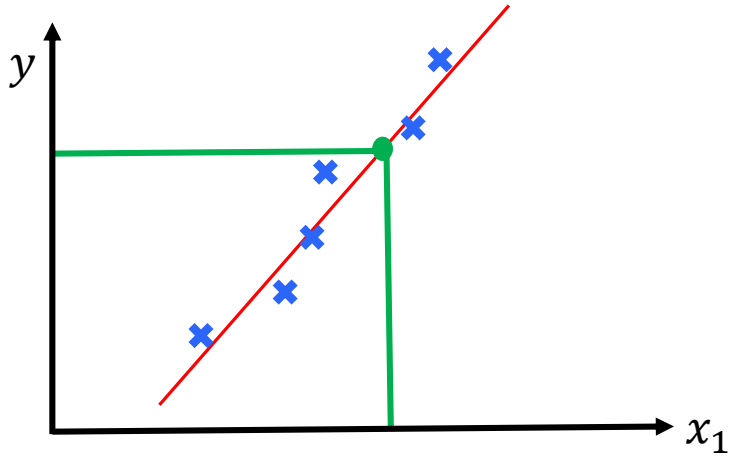


单变量线性回归模型（一元回归模型）

- 给定一个点的集合，能够通过一条曲线进行拟合，如果拟合出的线是一条直线，那就称为线性回归。

单变量线性模型

$$h_w(x) = w_0 + wx$$



x：Feature；$\mathbf{h}(x)$：hypothesis

多变量线性回归模型

假设影响售价y的特征不仅仅只有房屋面积x_1，还有楼层x_2，房屋朝向x_3，x_4，…，x_n

单变量线性模型

$$h_w(x) = w_0 + wx$$

多变量线性模型

$$h_w(x) = w_0 + w_1x_1 + w_2x_2$$ 2个特征

$$h_w(x) = \sum_{i=0}^{n} w_i x_i = \hat{\boldsymbol{w}}^T \boldsymbol{x},$$ n个特征

$$\hat{\boldsymbol{w}} = [w_1; w_2; w_0], \boldsymbol{x} = [x_1; x_2; x_0], x_0 = 1$$

线性函数拟合得好不好?

模型预测值$\hat{y}$与真实值y之间存在误差 $\varepsilon = y - \hat{y} = y - \hat{\boldsymbol{w}}^T \boldsymbol{x}$

ε 满足$\mathrm{N}(0, \sigma^2)$的高斯分布: $p(\varepsilon) = \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{(\varepsilon)^2}{2\sigma^2})$

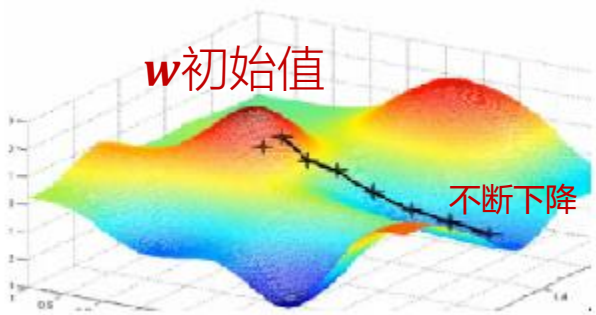
$$p(y|\boldsymbol{x}; \hat{\boldsymbol{w}}) = \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{(y - \hat{\boldsymbol{w}}^T \boldsymbol{x})^2}{2\sigma^2})$$

通过求最大似然函数，得到预测值与真实值之间误差尽量小的目标函数

损失函数 $$L(\hat{\boldsymbol{w}}) = \frac{1}{2}\sum_{j=1}^{m}(h_w(\boldsymbol{x}_j) - \boldsymbol{y}_j)^2$$

目标: 求出参数$\hat{\boldsymbol{w}}$，使得损失函数$L(\hat{\boldsymbol{w}})$取值最小

寻找参数$\hat{w}$->使得$L(\hat{w})$ 最小



梯度下降法寻找参数

- 初始先给定一个$\hat{w}$，如$\mathbf{0}$向量或随机向量
- 沿着梯度下降的方向进行迭代，使更新后的$L(\hat{w})$ 不断变小

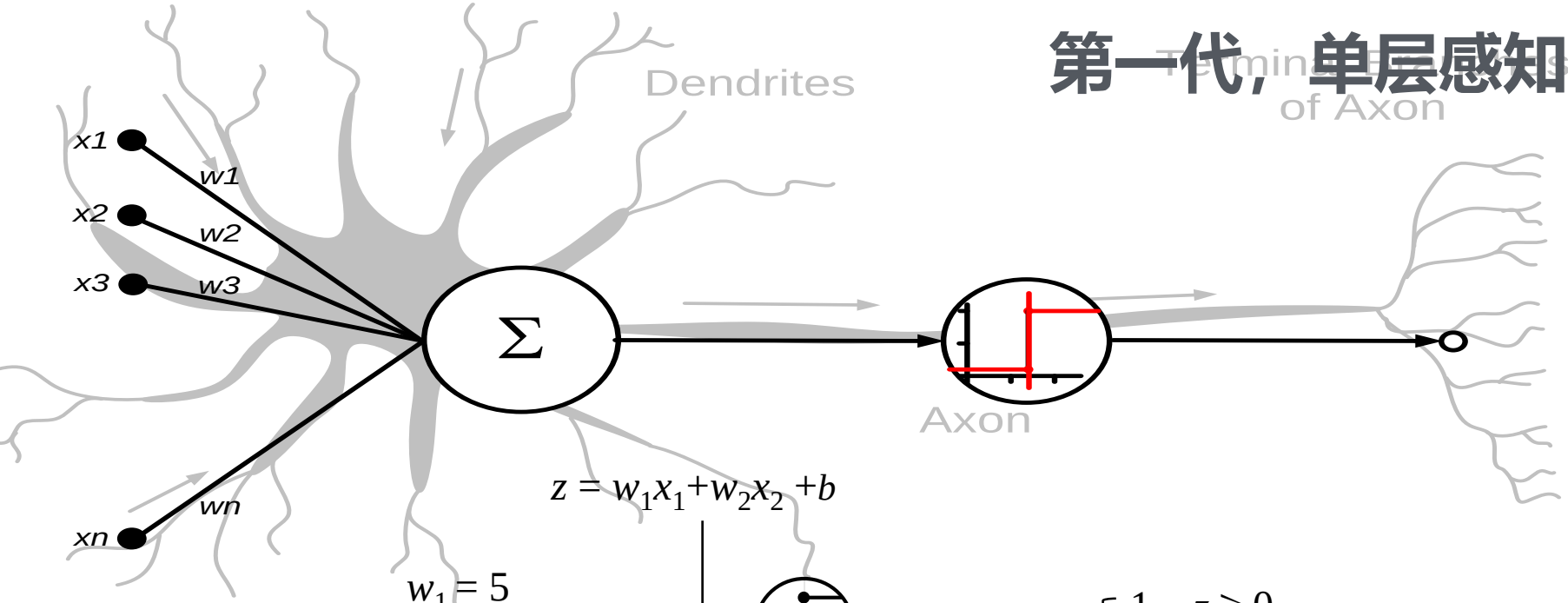
$$\hat{w} = \hat{w} - \alpha \frac{\partial L(\hat{w})}{\partial \hat{w}}$$

α 称为学习率或步长

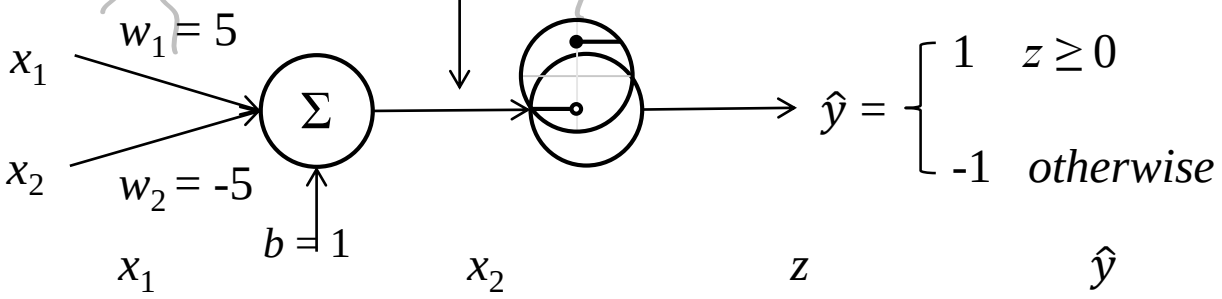
迭代至找到使得$L(\hat{w})$最小的$\hat{w}$值停止，从而得到回归模型参数

一个神经元的单层感知机

第一代，单层感知机



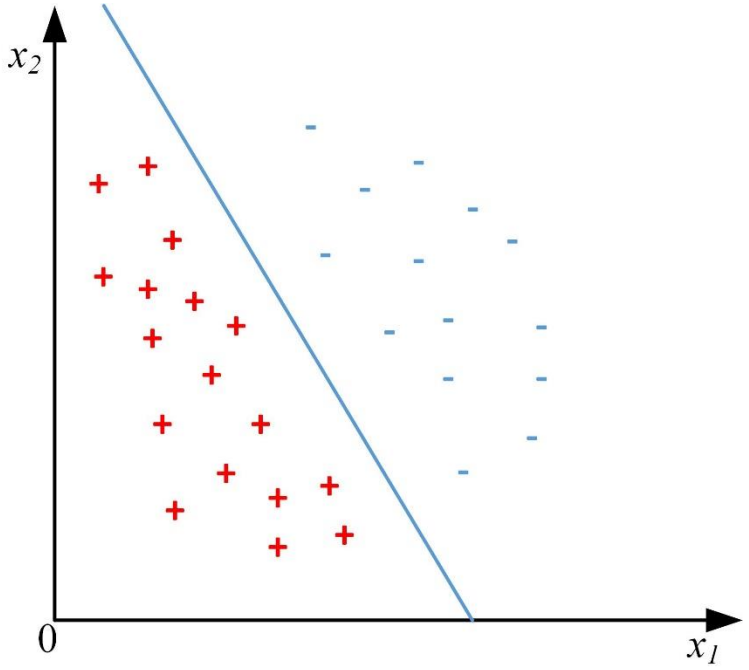
$z = w_1x_1 + w_2x_2 + b$



x_1	x_2	z	$\hat{y}$
1	-1	11	1
-1	1	-9	-1

感知机（Perceptron）模型

感知机模型$\mathrm{f(x)} = \mathrm{sign}(\boldsymbol{w}^T\boldsymbol{x} + b)$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。



$$\mathrm{f(x)} = \mathrm{sign}(\boldsymbol{w}^T\boldsymbol{x} + b)$$

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

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 $\mathrm{D} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$，其中，$\boldsymbol{x}_j \in \mathrm{R}^n, y_j \in \{+1, -1\}$。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 $L(\boldsymbol{w},b)=-\sum_{\boldsymbol{x}_j\in M}y_j(\boldsymbol{w}^T\boldsymbol{x}_j+b)$

感知机算法

$$\mathrm{f(x)} = sign(\boldsymbol{w}^T \boldsymbol{x} + b)$$

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

➡

损失函数

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

问题转化为寻找（$\boldsymbol{w}, b$）使得损失函数极小化的最优化问题

损失函数极小化的最优化问题可使用：随机梯度下降法

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

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

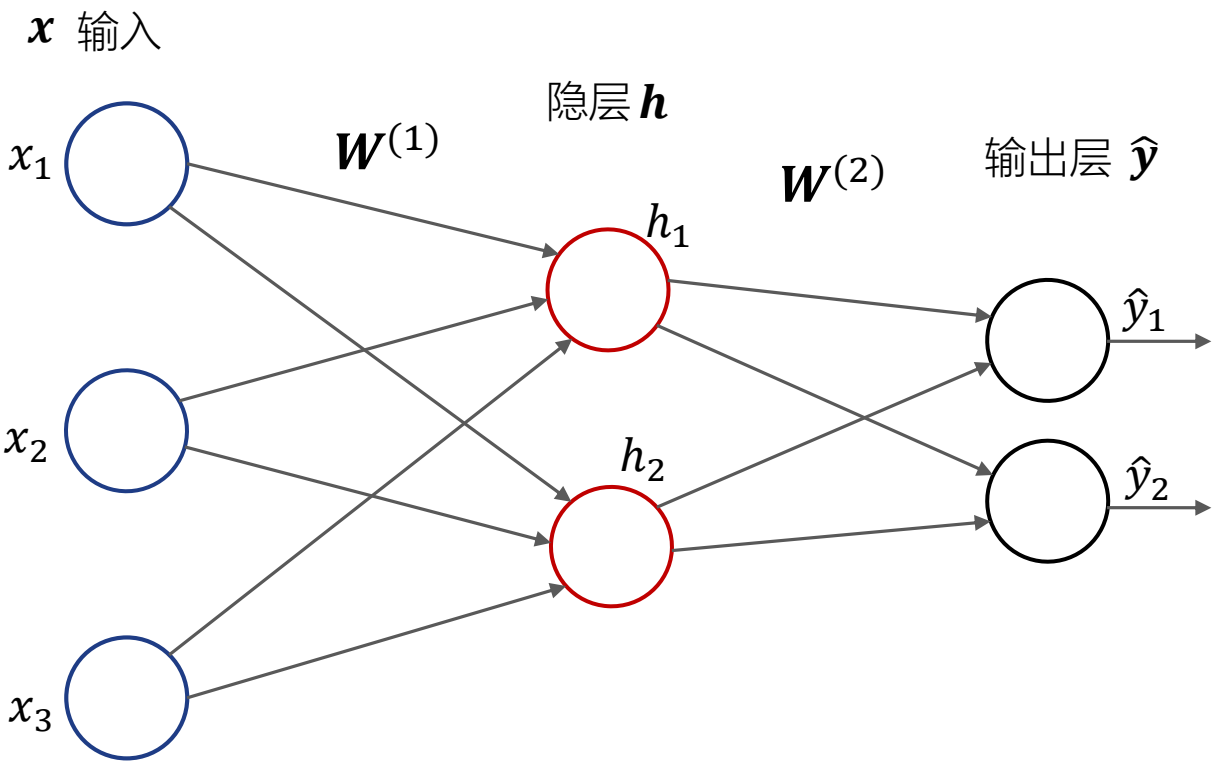
$$\nabla_b L(\boldsymbol{w}, b) = -\sum_{\boldsymbol{x}_j \in M} y_j \Rightarrow b \leftarrow b + \alpha y_j$$

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

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

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。
- 全连接的两层神经网络模型也称为多层感知机

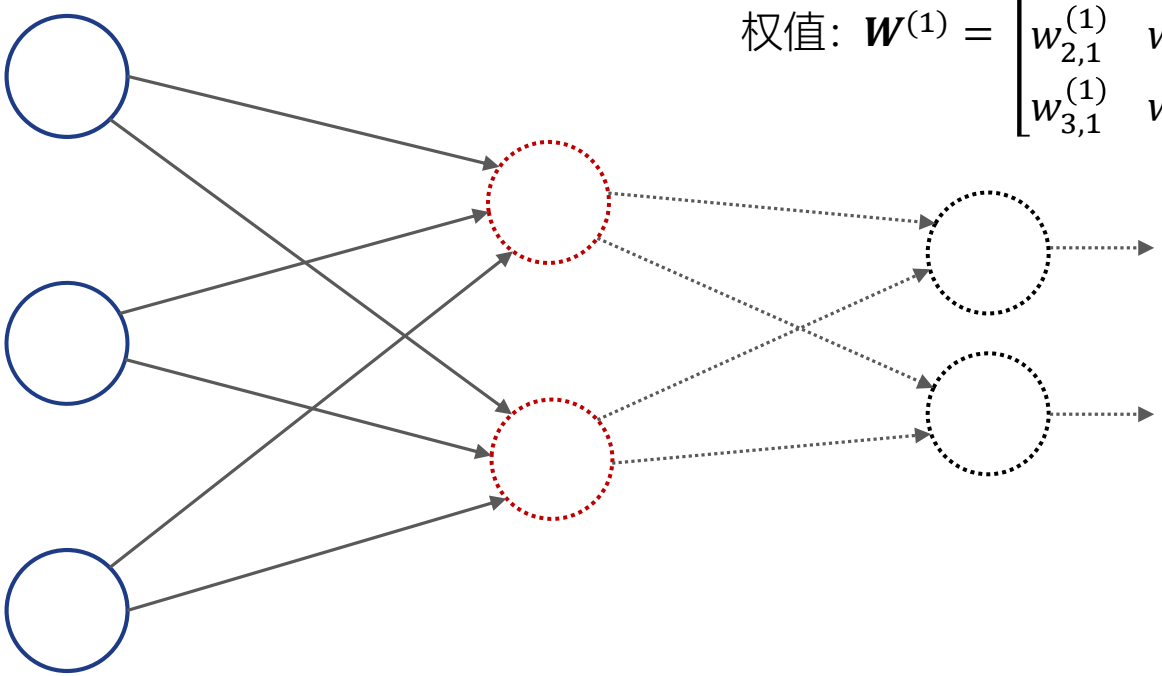


输入样本：$\boldsymbol{x} = [x_1; x_2; x_3]$

输入

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

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

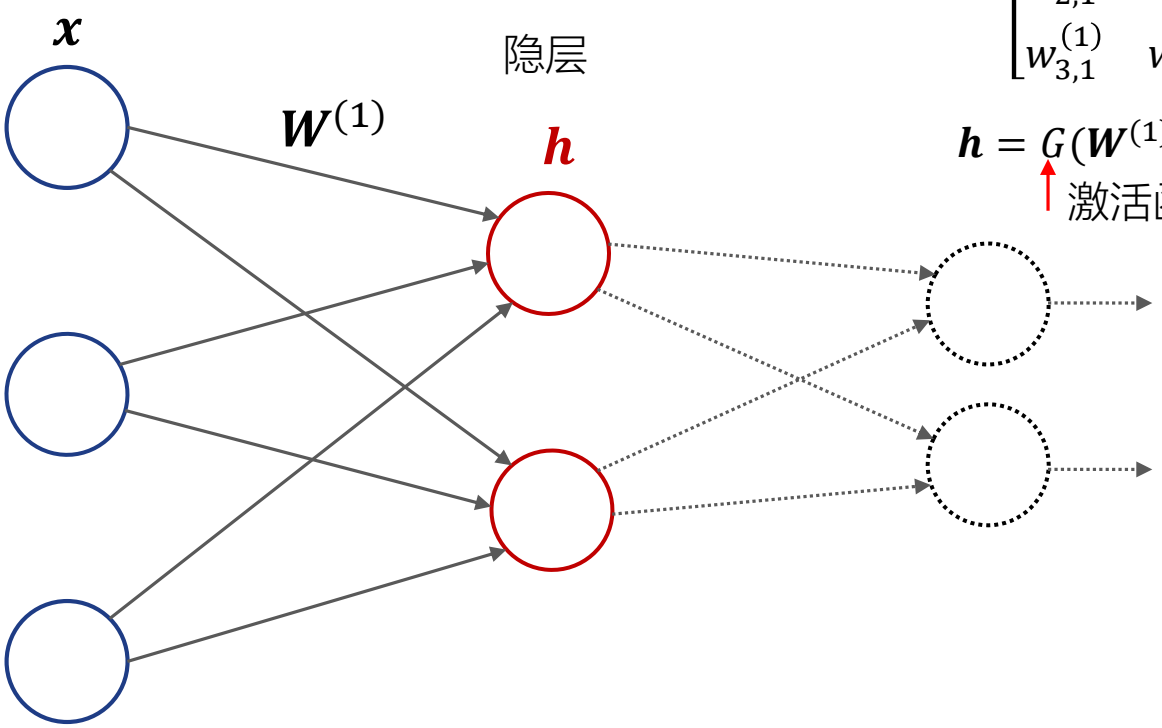


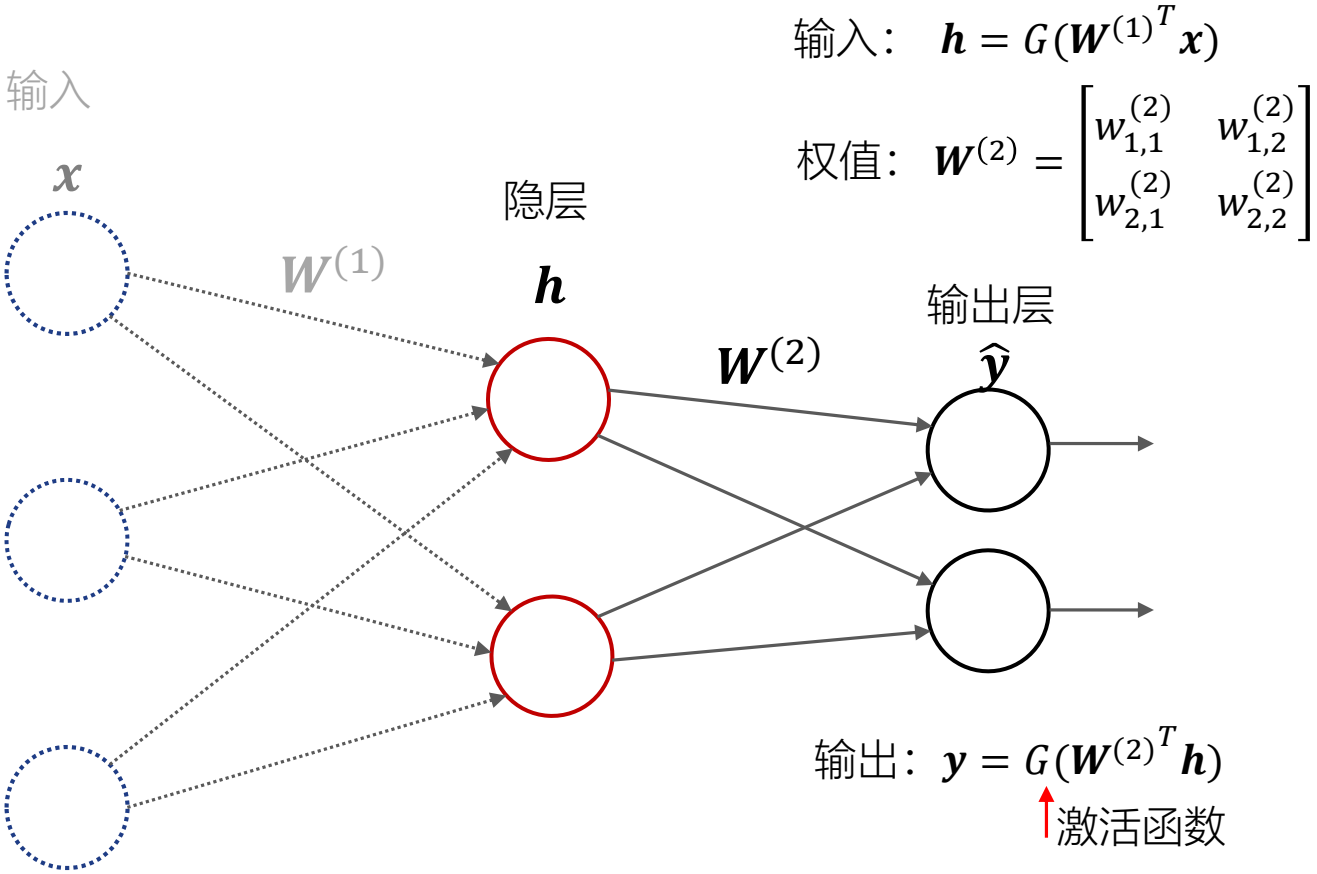
输入：$\boldsymbol{x} = [x_1; x_2; x_3]$

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

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

↑ 激活函数

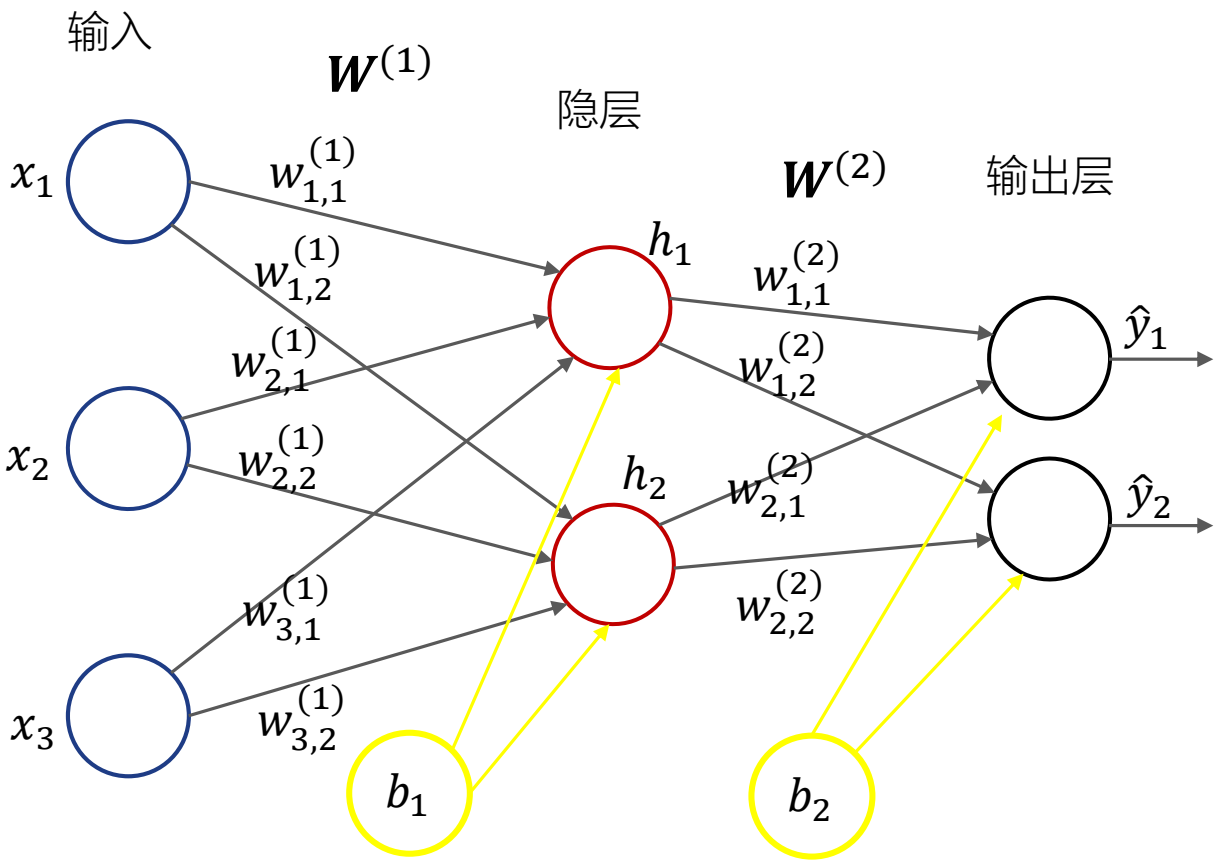



输入：
$\boldsymbol{h} = G(\boldsymbol{W}^{(1)^T}\boldsymbol{x})$
权值：
$\boldsymbol{W}^{(2)} = \begin{bmatrix} w_{1,1}^{(2)} & w_{1,2}^{(2)} \\ w_{2,1}^{(2)} & w_{2,2}^{(2)} \end{bmatrix}$
输入
$\boldsymbol{x}$
$\boldsymbol{W}^{(1)}$
隐层
$\boldsymbol{h}$
$\boldsymbol{W}^{(2)}$
输出层
$\hat{\boldsymbol{y}}$
输出：
$\boldsymbol{y} = G(\boldsymbol{W}^{(2)^T}\boldsymbol{h})$
激活函数

偏置节点

$$\boldsymbol{h} = G(\boldsymbol{W}^{(1)^T}\boldsymbol{x} + b_1)$$

$$\hat{\boldsymbol{y}} = G(\boldsymbol{W}^{(2)^T}\boldsymbol{h} + b_2)$$



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

神经网络

- 神经网络是通过已知的样本预测未知数据的模型

  x_1, x_2, x_3 已知，用模型计算值 $\hat{y}$预测未知属性 y

  → $y = f(w_1x_1 + w_2x_2 + w_3x_3)$　w_1, w_2, w_3 权值

- 如何由感知机设计出一个实用的神经网络?

  输入的节点数与特征的维度匹配，输出层的节点数与目标的维度匹配，设计者只需要设计隐层的节点数和层数

- 隐层节点数设计方法 - Grid Search（网格搜索）

  预先设定若干个可选值，通过切换这几个值来观察整个模型的预测效果，选择效果最好的值作为最终选择

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

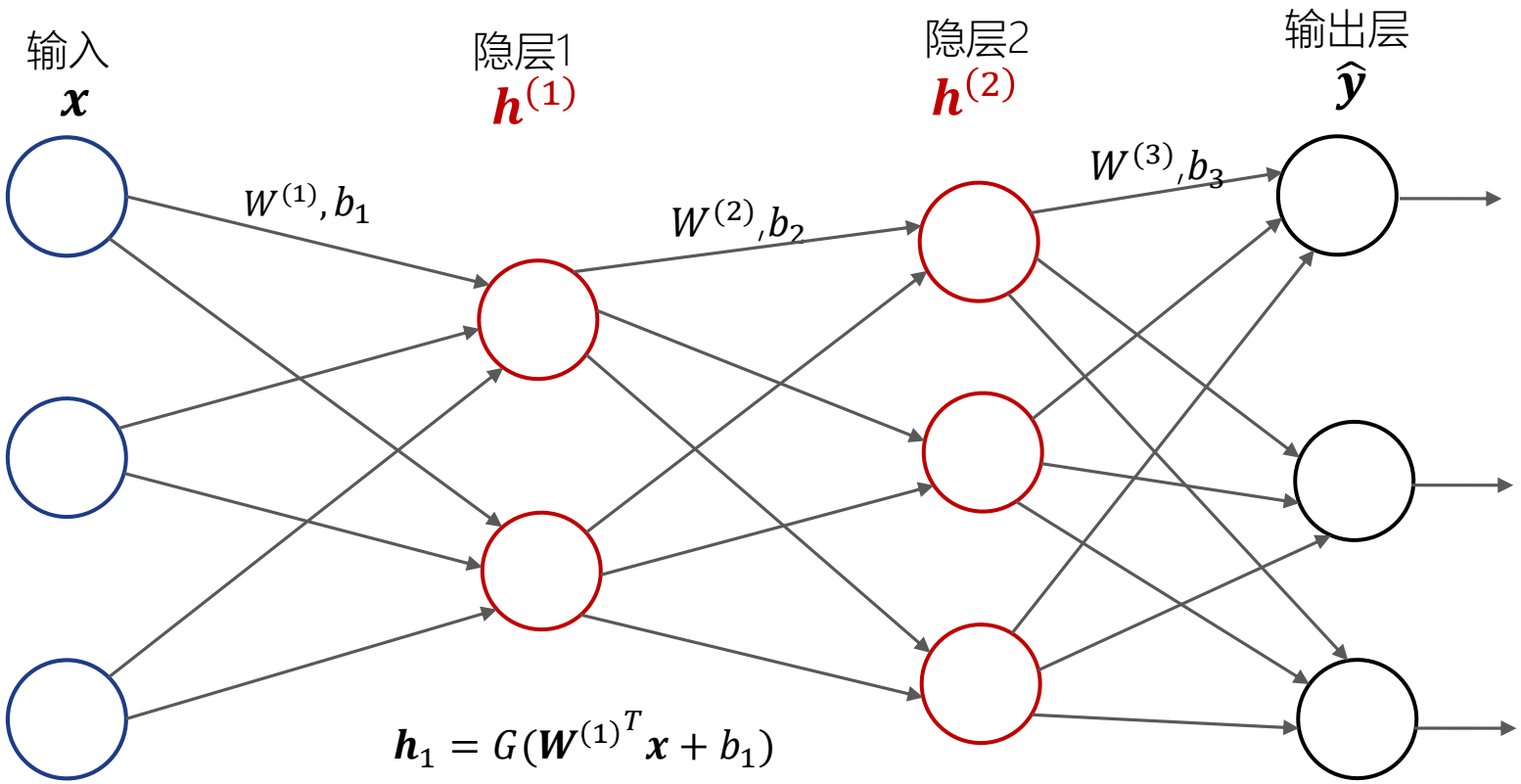
Geoffery Hinton

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

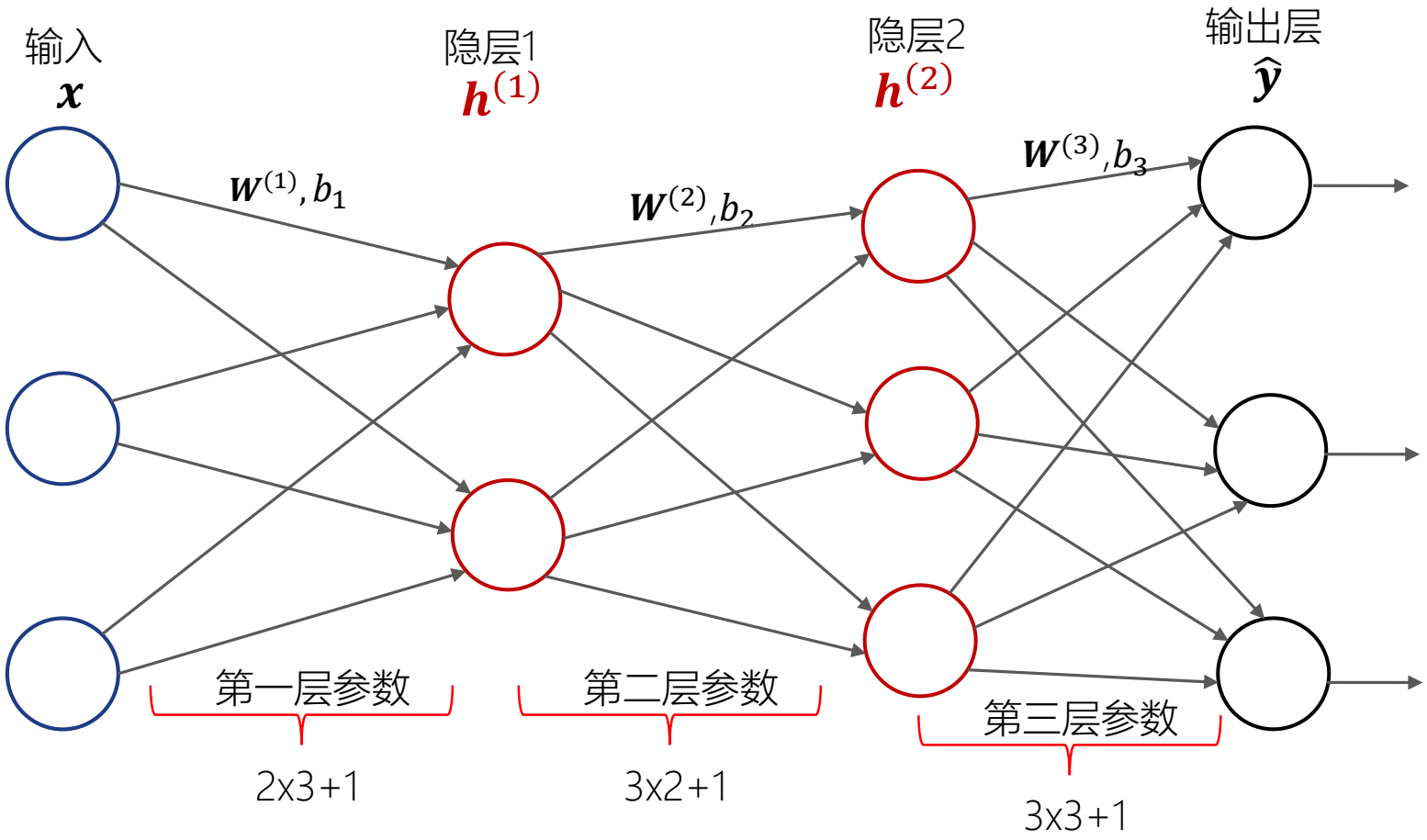


推导公式

$$\boldsymbol{h}_1 = G(\boldsymbol{W}^{(1)^T}\boldsymbol{x} + b_1)$$

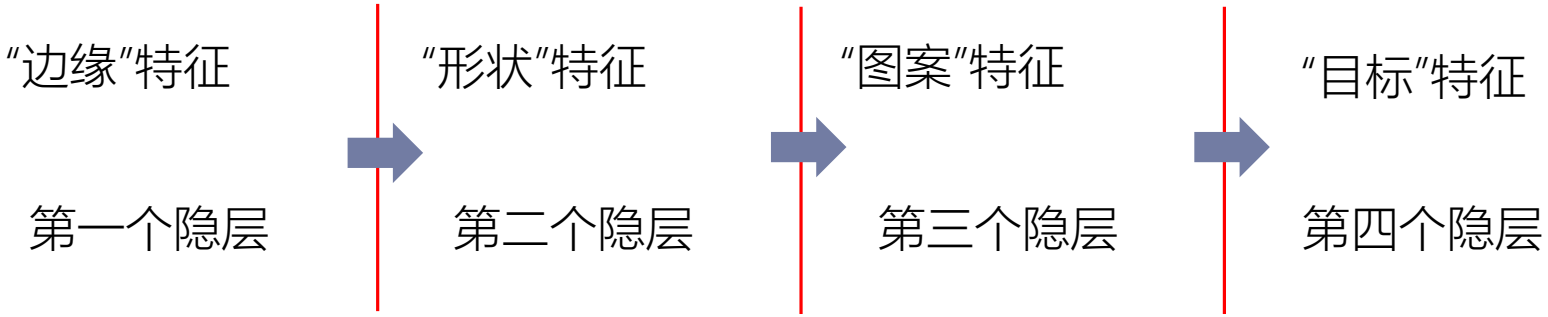
$$\boldsymbol{h}_2 = G(\boldsymbol{W}^{(2)^T}\boldsymbol{h_1} + b_2)$$

$$\hat{\boldsymbol{y}} = G(\boldsymbol{W}^{(3)^T}\boldsymbol{h_2} + b_3)$$

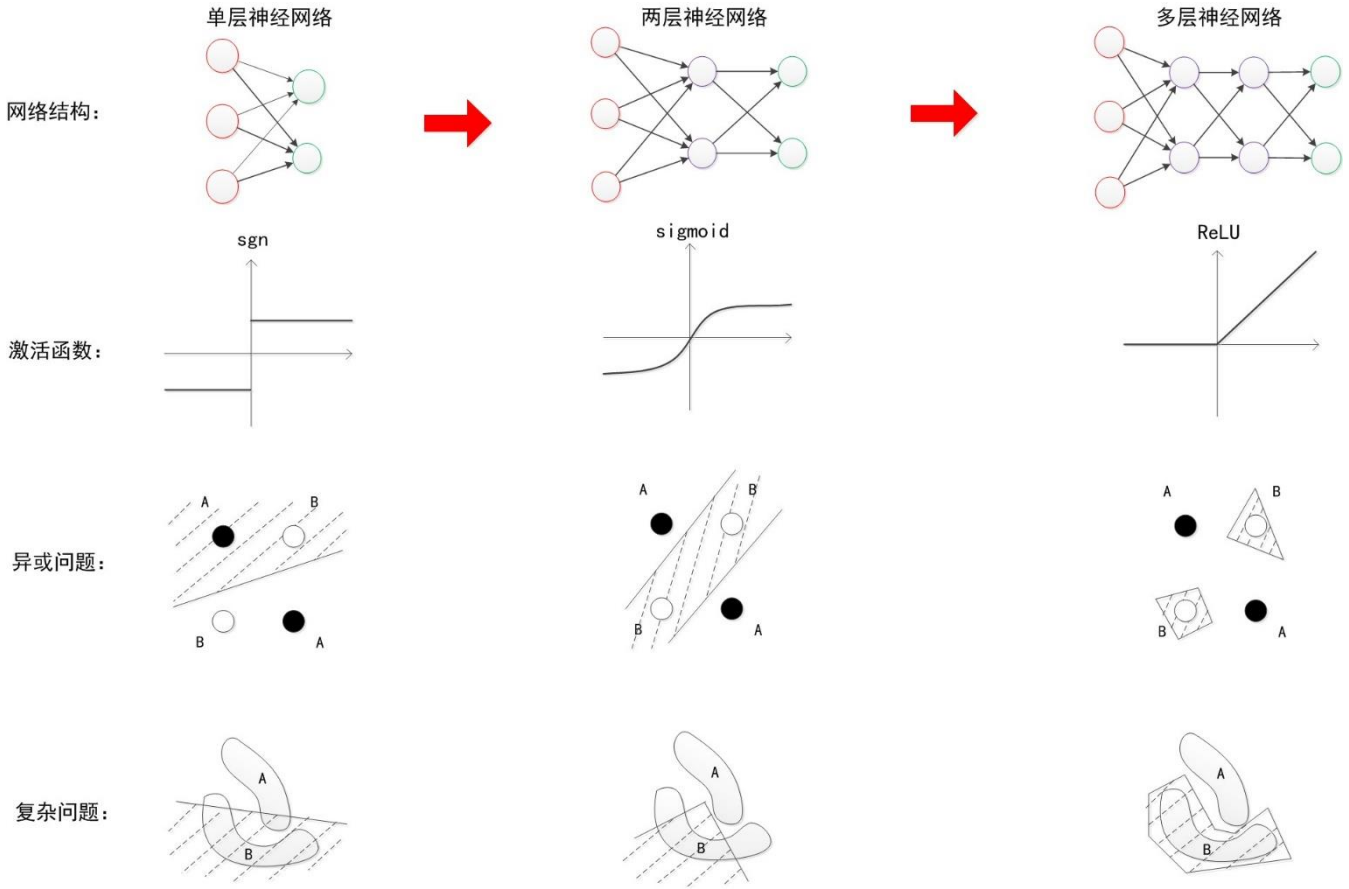

输入
$\boldsymbol{x}$
隐层1
$\boldsymbol{h}^{(1)}$
隐层2
$\boldsymbol{h}^{(2)}$
输出层
$\hat{\boldsymbol{y}}$
$\boldsymbol{W}^{(1)}, b_1$
$\boldsymbol{W}^{(2)}, b_2$
$\boldsymbol{W}^{(3)}, b_3$
第一层参数
第二层参数
第三层参数
2x3+1
3x2+1
3x3+1

需 6+6+9+3=24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



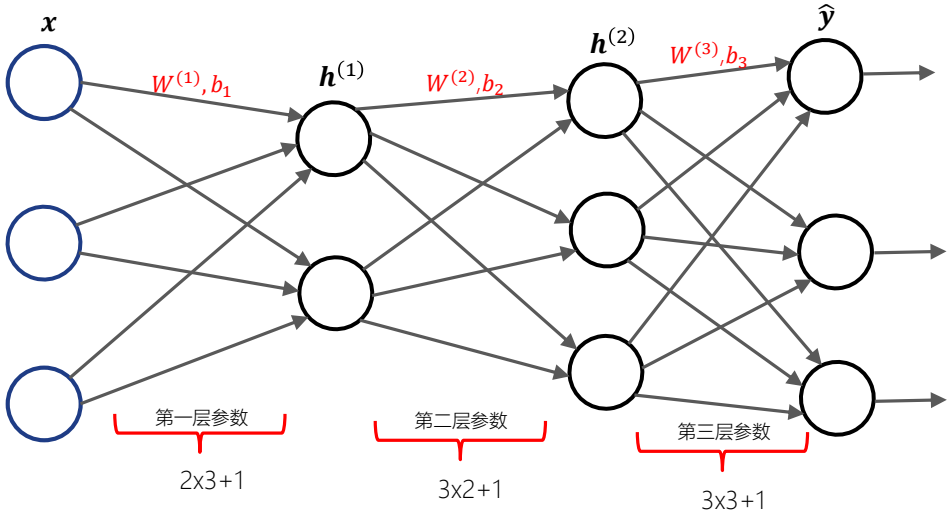
从单层神经网络，到两层神经网络，再到多层神经网络，随着网络层数的增加，以及激活函数的调整，神经网络拟合非线性分界不断增强。



提纲

- 从机器学习到神经网络
- 正向传播与反向传播
- 神经网络设计原则
  - 网络的拓扑结构
  - 激活函数
  - 损失函数
- 过拟合与正则化
- 交叉验证
- 小结

神经网络的模型训练



- 模型训练的目的，就是使得参数尽可能的与真实的模型逼近，模型计算值$\hat{\boldsymbol{y}}$与$\boldsymbol{y}$无限接近

$\hat{\boldsymbol{y}}$：样本的预测值

$\boldsymbol{y}$：真实值

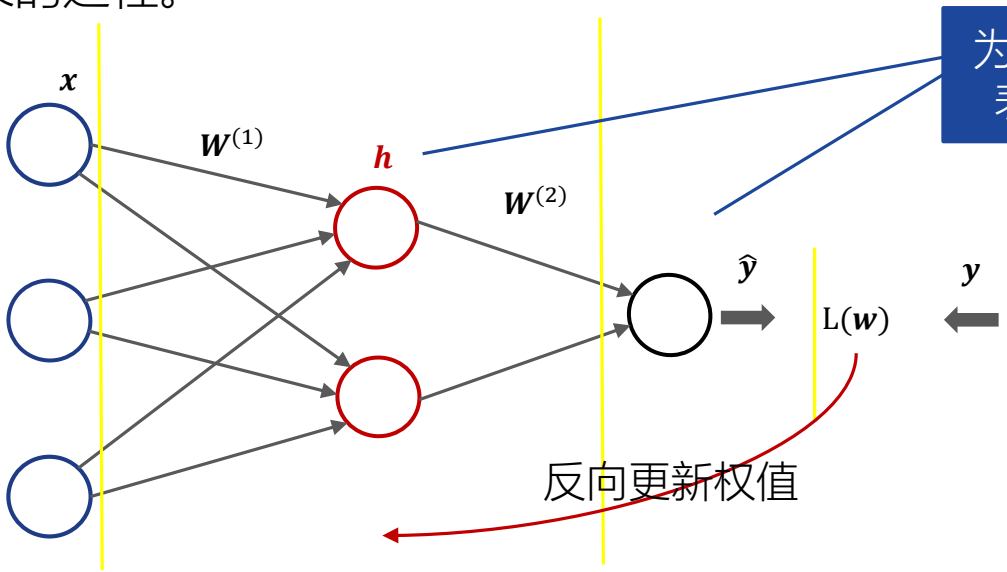
神经网络的模型训练

数据在神经网络中如何进行数据传输?

- 正向传播
- 反向传播

正向传播与反向传播

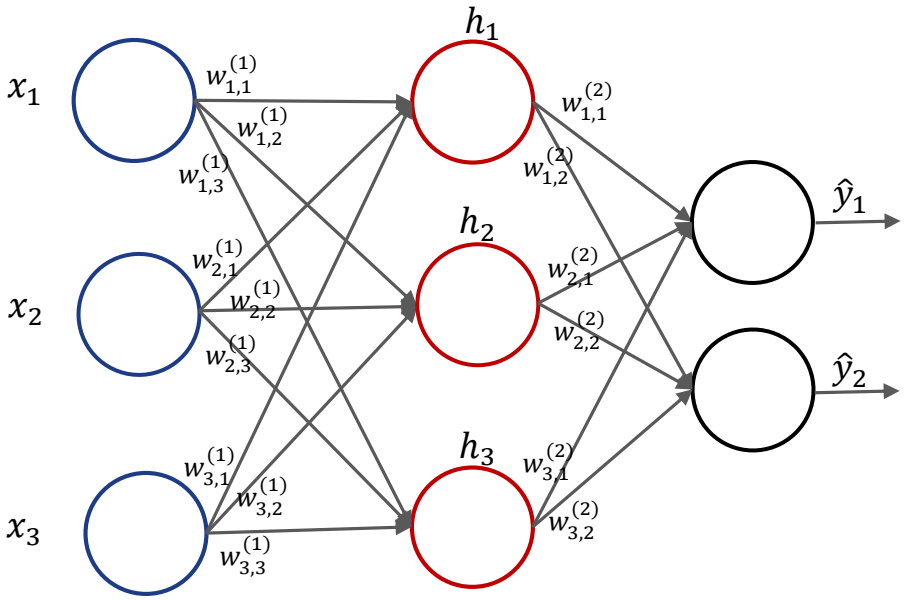
- 正向传播（前向传播/正向计算）是根据输入，经过权重、激活函数计算出隐层，将输入的特征向量从低级特征逐步提取为抽象特征，直到得到最终输出结果的过程。



- 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。

正向传播

$\boldsymbol{x}$ $(\boldsymbol{W}^{(1)}, b_1)$ $\boldsymbol{h}$ $(\boldsymbol{W}^{(2)}, b_2)$ $\hat{\boldsymbol{y}}$

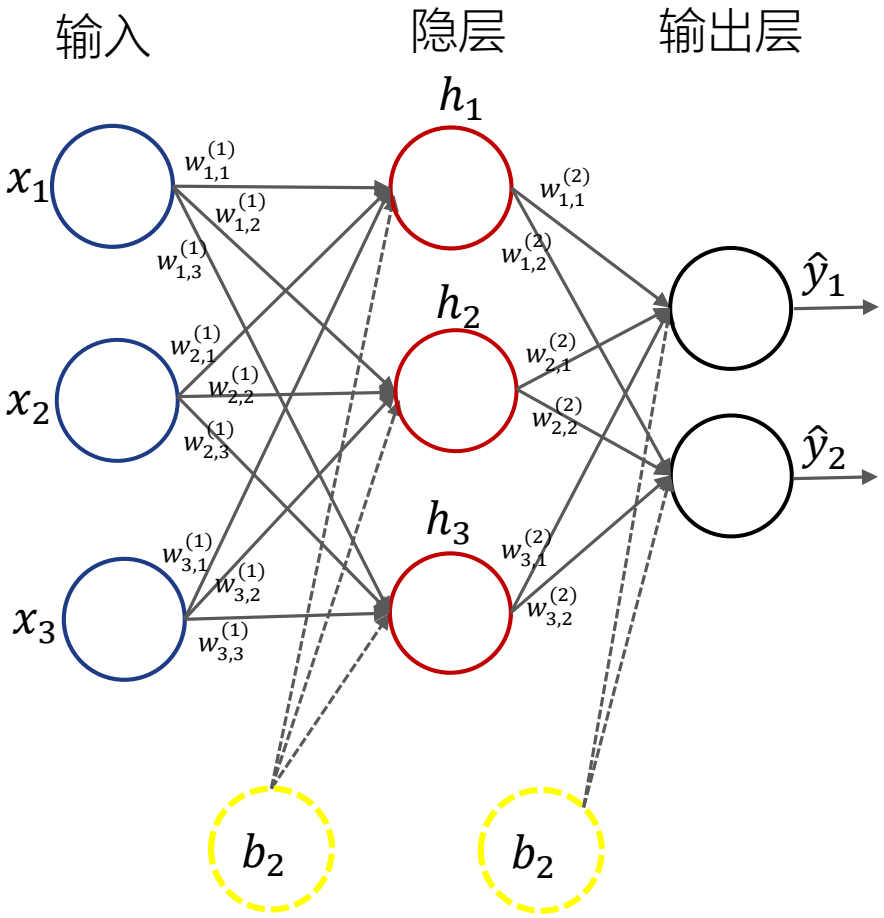


输入 $\boldsymbol{x} = [x_1; \quad x_2; \quad x_3]$

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

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



- 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

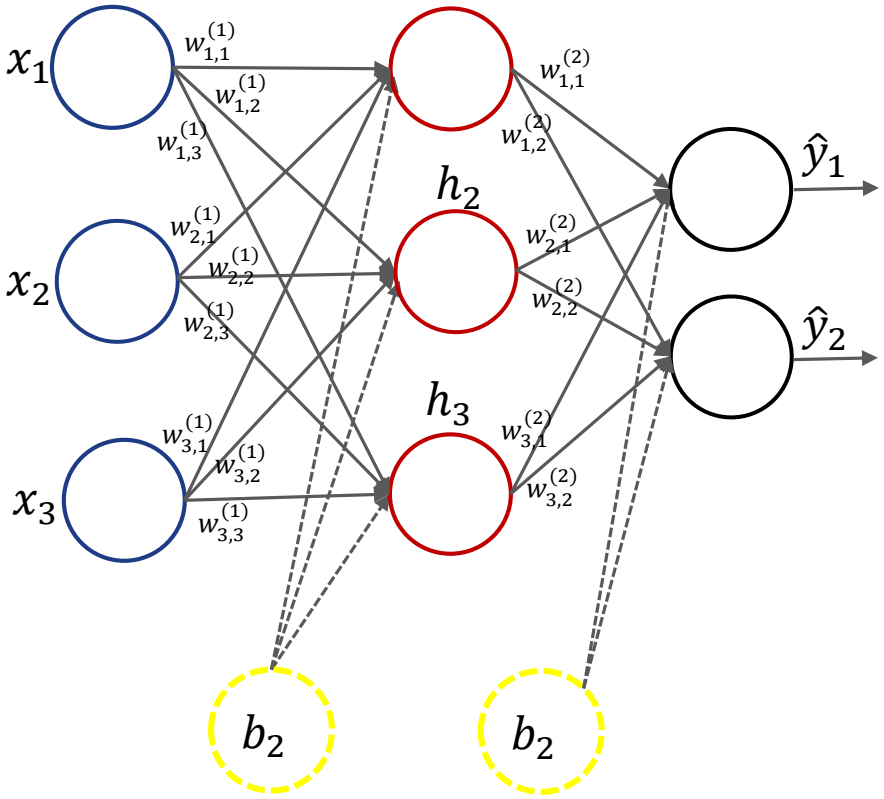
  偏置：b_1

  权重：$\boldsymbol{W}^{(1)}$

- 隐层和输出层之间

  偏置：b_2

  权重：$\boldsymbol{W}^{(2)}$



- 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

$$\boldsymbol{v} = \boldsymbol{W}^{(1)^T}\boldsymbol{x} = \begin{bmatrix} w_{1,1}^{(1)} & w_{2,1}^{(1)} & w_{3,1}^{(1)} \\ w_{1,2}^{(1)} & w_{2,2}^{(1)} & w_{3,2}^{(1)} \\ w_{1,3}^{(1)} & w_{2,3}^{(1)} & w_{3,3}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + b_1$$

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

示例

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

偏置 $b_1 = 0.4$、$b_2 = 0.7$

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

$$\boldsymbol{v}=\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \boldsymbol{W}^{(1)^T}\boldsymbol{x} + b_1 = \begin{bmatrix} 0.25 & 0.25 & 0.10 \\ 0.15 & 0.20 & 0.25 \\ 0.30 & 0.35 & 0.15 \end{bmatrix}\begin{bmatrix} 0.02 \\ 0.04 \\ 0.01 \end{bmatrix}+0.4 = \begin{bmatrix} 0.416 \\ 0.4135 \\ 0.4215 \end{bmatrix}$$

$$\boldsymbol{h} = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} = \frac{1}{1+e^{-\boldsymbol{v}}} = \begin{bmatrix} \frac{1}{1+e^{-0.416}} \\ \frac{1}{1+e^{-0.4135}} \\ \frac{1}{1+e^{-0.4215}} \end{bmatrix} = \begin{bmatrix} 0.6025 \\ 0.6019 \\ 0.6038 \end{bmatrix}$$

- 隐层到输出层计算

$$\boldsymbol{z} = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \boldsymbol{W}^{(2)^T}\boldsymbol{h} + b_2 = \begin{bmatrix} 0.40 & 0.35 & 0.01 \\ 0.25 & 0.30 & 0.35 \end{bmatrix} \begin{bmatrix} 0.6025 \\ 0.6019 \\ 0.6038 \end{bmatrix} + 0.7 = \begin{bmatrix} 1.1577 \\ 1.2425 \end{bmatrix}$$

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

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

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

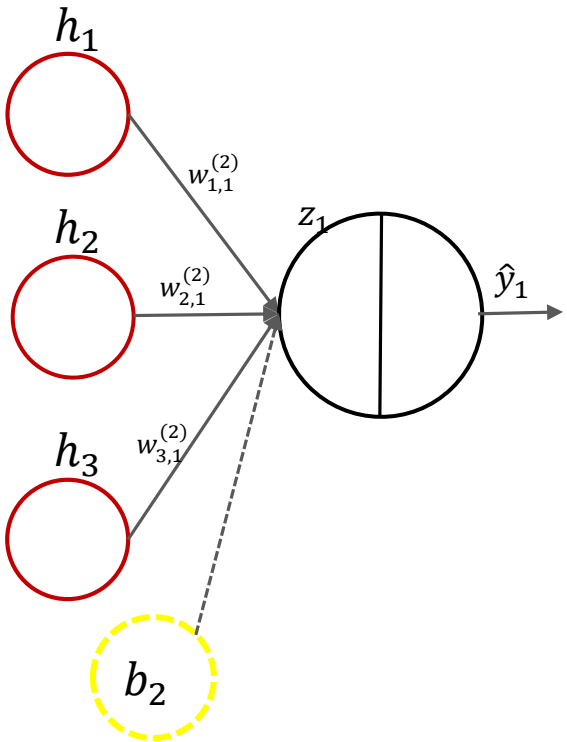
$$= \frac{1}{2}(0.7609 - 0.9)^2 + \frac{1}{2}(0.7760 - 0.5)^2 = 0.0478$$

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



$$L_1 = \frac{1}{2}\ (y_1 - \hat{y}_1)^{\ 2}$$

$$L(\boldsymbol{W}) = L_1 + L_2$$

- 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由ω 对 z_1的计算公式推导

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

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

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

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

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

- 根据偏导数的链式法则由ω 对 z_1的计算公式推导

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

$$z_1 = w_{1,1}^{(2)} \times h_1 + \omega \times h_2 + w_{3,1}^{(2)} \times h_3 + b_2$$

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

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

- 更新$w_{2,1}^{(2)}$（记为ω）的权重

初始值

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

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

$$\omega = \omega - \alpha \times \frac{\partial L(\boldsymbol{W})}{\partial \omega} = 0.35 - (-0.0152) = 0.3348$$

- 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元的输出误差反向传播到神经元的输入端，并以此来更新神经元输入端的权重

- 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。

提纲

- 从机器学习到神经网络
- 正向传播与反向传播
- 神经网络设计原则
  - 网络的拓扑结构
  - 激活函数
  - 损失函数
- 过拟合与正则化
- 交叉验证
- 小结

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

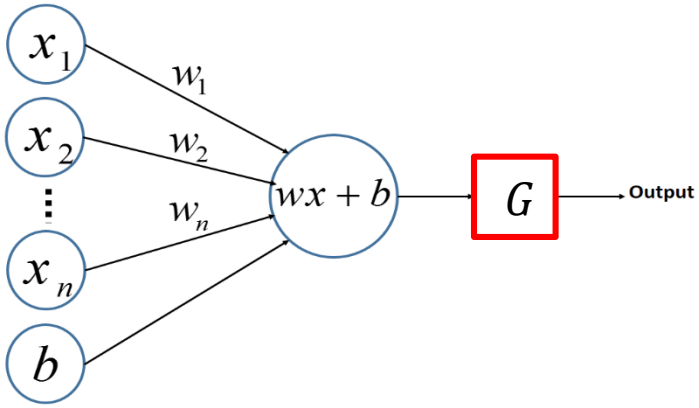
给定训练样本后，输入和输出层节点数便已确定

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



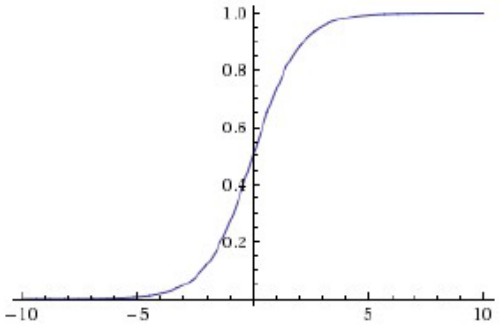
- 在神经元中，输入的数据通过加权求和后，还被作用了一个函数G，这个函数G就是激活函数（Activation Function）。
- 激活函数给神经元引入了非线性因素，使得神经网络可以任意逼近任何非线性函数，因此神经网络可以应用到众多的非线性模型中
- 激活函数需具备的性质
  - 可微性： 当优化方法是基于梯度的时候，这个性质是必须的。
  - 输出值的范围： 当激活函数输出值是有限的时候，基于梯度的优化方法会更加稳定，因为特征的表示受有限权值的影响更显著;当激活函数的输出是无限的时候，模型的训练会更加高效，不过在这种情况小，一般需要更小的学习率。

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

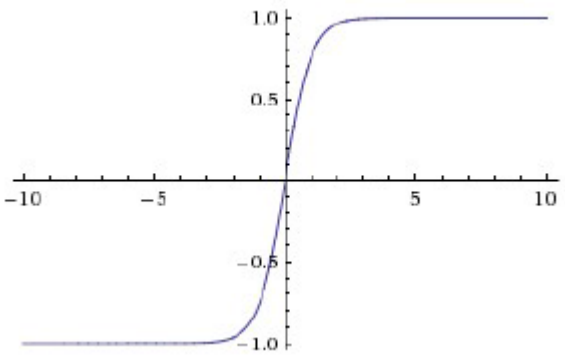
tanh函数

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

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

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



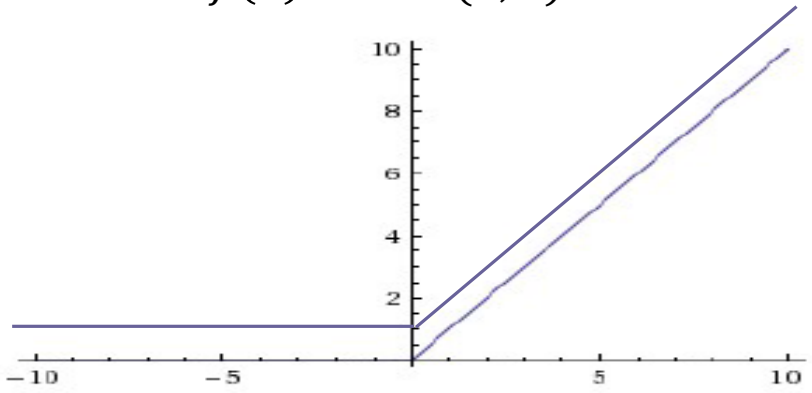
- 与sigmoid相比，tanh是0均值的
- 在输入很大或是很小的时候，输出几乎平滑，梯度很小，不利于权重更新

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

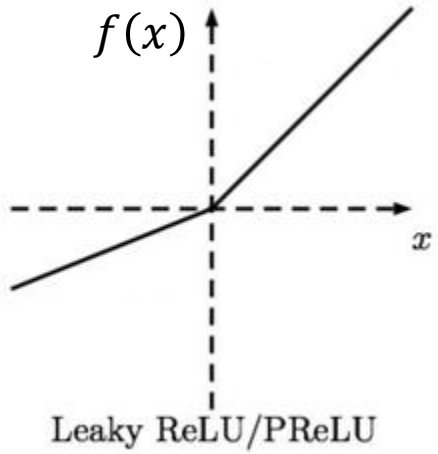
改进

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

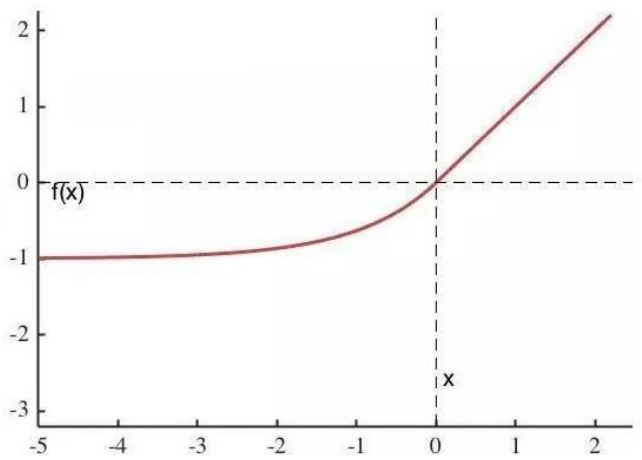
PReLU定义类似

- 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

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



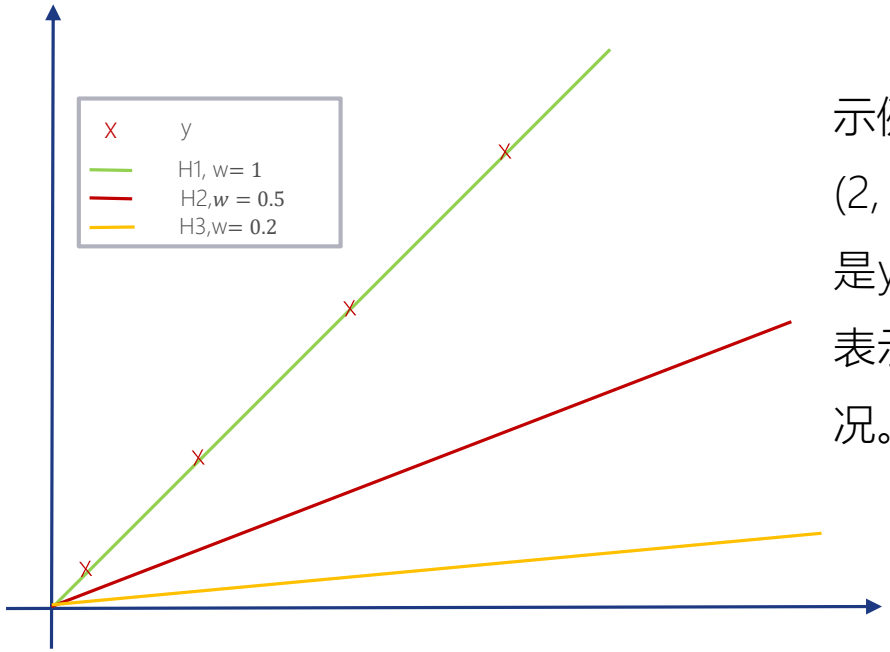
- α是可调参数，控制着ELU在负值区间的饱和位置
- ELU的输出均值接近于零，所以收敛速度更快
- 右侧线性部分使得ELU能够缓解梯度消失，而左侧软饱能够让ELU对输入变化或噪声更鲁棒，避免神经元死掉

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 $\hat{y} = H_w(x)$，损失函数记为 $L(\boldsymbol{w}) = f(H_w(x), y)$

损失函数与参数

参数w决定了$L(w)$的改变，进而影响了$L(w)$的取值，因此可以通过分析$L(w)$来指导参数优化



示例：有训练样本(x,y)：{(1，1)，(2，2)，(3，3)，(4，4)}，真实值是y=x这条直线上的点，右图表示了不同参数w下的拟合情况。

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，则使用Sigmoid激活函数时的交叉熵损失函数为：

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

- 以二分类为例，则使用Sigmoid激活函数时的交叉熵损失函数为：

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

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

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

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

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

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

提纲

- 从机器学习到神经网络
- 正向传播与反向传播
- 神经网络设计原则
  - 网络的拓扑结构
  - 激活函数
  - 损失函数
- 过拟合与正则化
- 交叉验证
- 小结

神经网络存在的问题及解决办法

随着神经网络层数的不断加深，多层神经网络会出现一个致命问题：

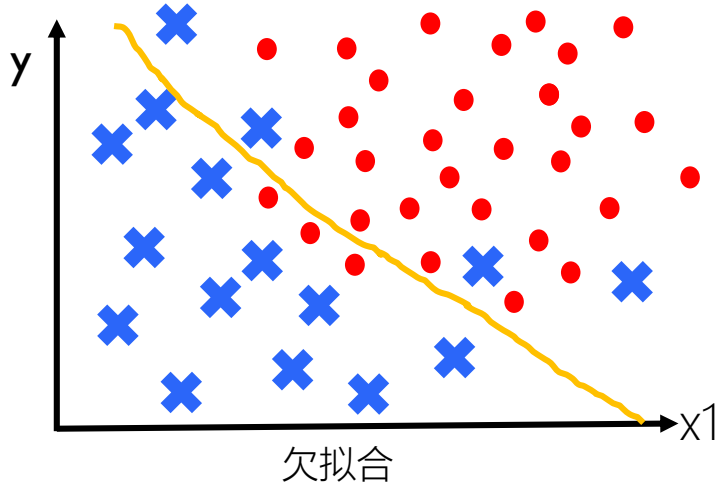
过拟合，范化能力差

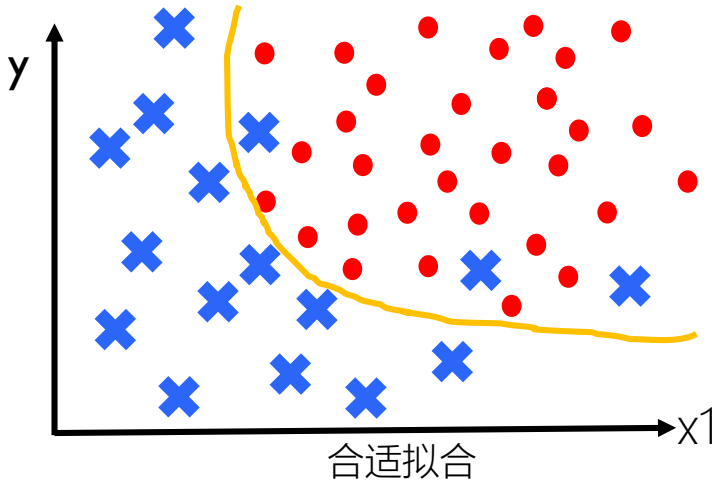
正则化

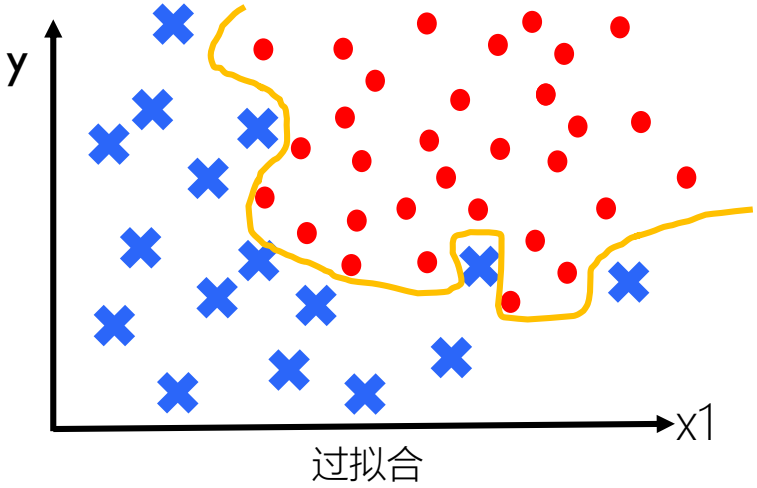
定义

- 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。
- 参数范数惩罚、稀疏化、Bagging集成、Dropout、提前终止、数据集扩增等正则化方法可以使网络中节点随机失活，降低参数对网络的依赖，有效抑制过拟合

欠拟合和过拟合



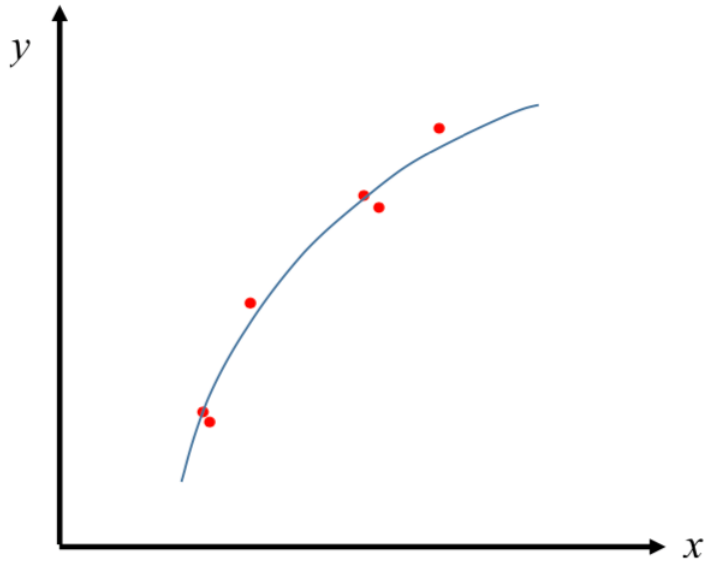




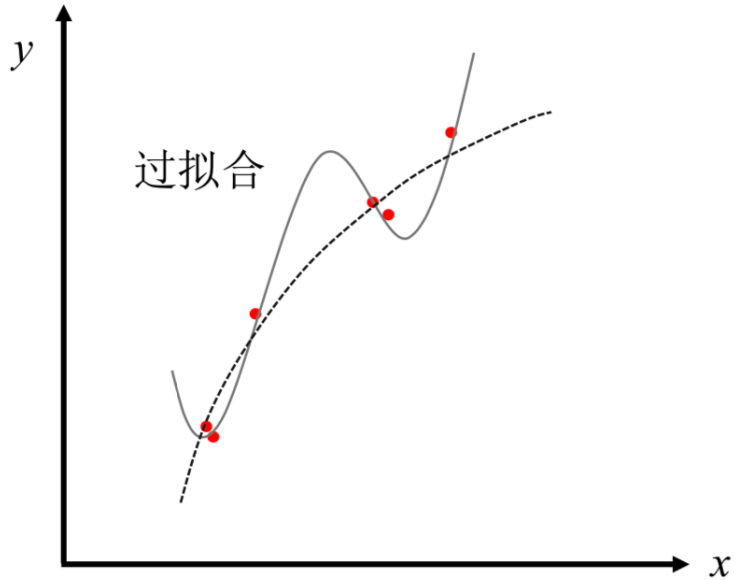
欠拟合：本质原因是训练的特征少，拟合函数无法满足训练集，误差较大。

过拟合，训练的特征维度多，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。

正则化思路



拟合函数 $w_0 + w_1x + w_2x^2$



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

加上惩罚项使 w_3、w_4足够小

目标函数

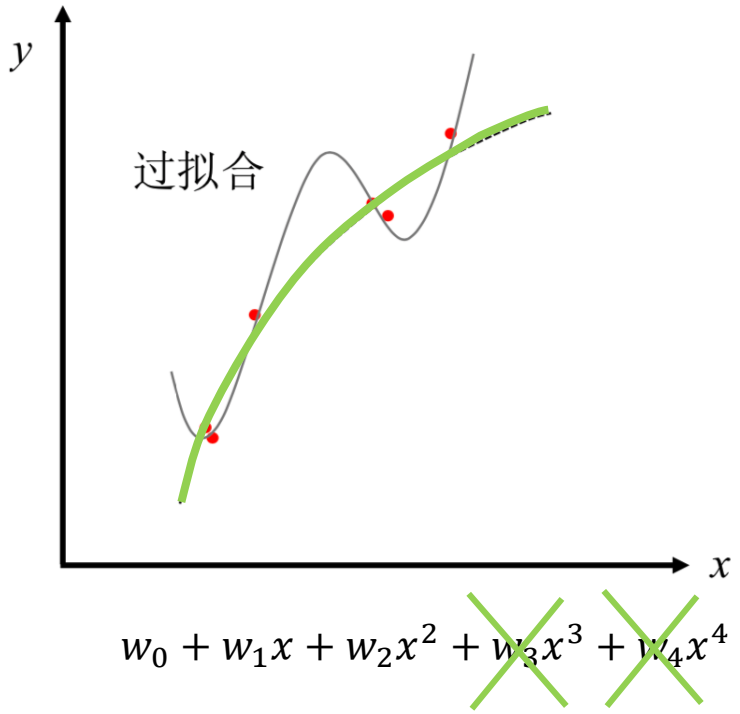
$$L(\boldsymbol{w}) = \frac{1}{2m}\sum_{i=1}^{m} ||\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i||^2$$

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

$$\min_{\boldsymbol{w}} \frac{1}{2m}\sum_{i=1}^{m} ||\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i||^2 + C_1 * w_3^2 + C_2 * w_4^2$$

C_1、C_2可取常数，如取1000

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



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

$$\min_{w} \frac{1}{2m} \sum_{i=1}^{m} ||\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i||^2 + C_1 * w_3^2 + C_2 * w_4^2$$

在代价函数中增加一个惩罚项，惩罚高阶参数，使其趋近于0

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

正则化项/惩罚项

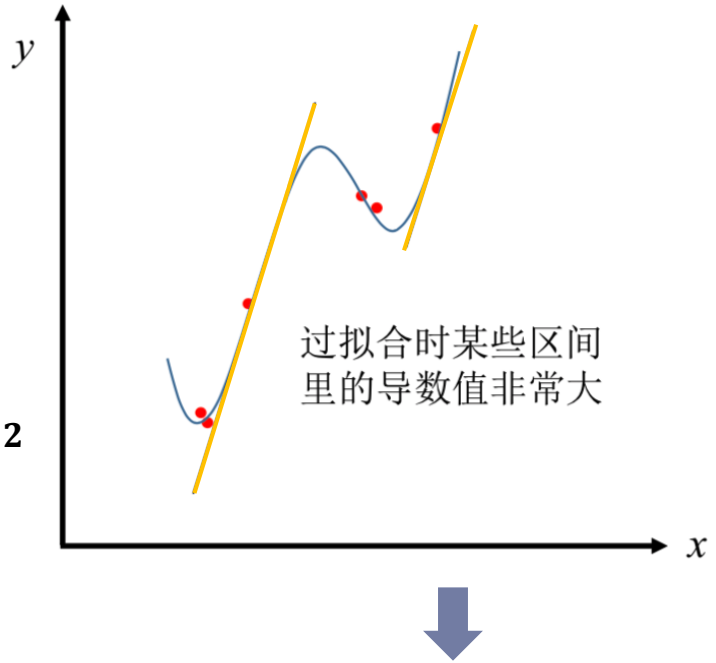
θ 为正则化参数，神经网络中的参数包括权重 $\boldsymbol{w}$ 和偏置 b，正则化过程仅对权重 w 进行惩罚，正则化项记为

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

正则化后的目标函数记为

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

L^2正则化



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

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

L^2 正则化如何避免overfitting?

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

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

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

通过L^2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

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

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

L^1正则化添加了一项符号函数$sign(\boldsymbol{w})$来影响梯度

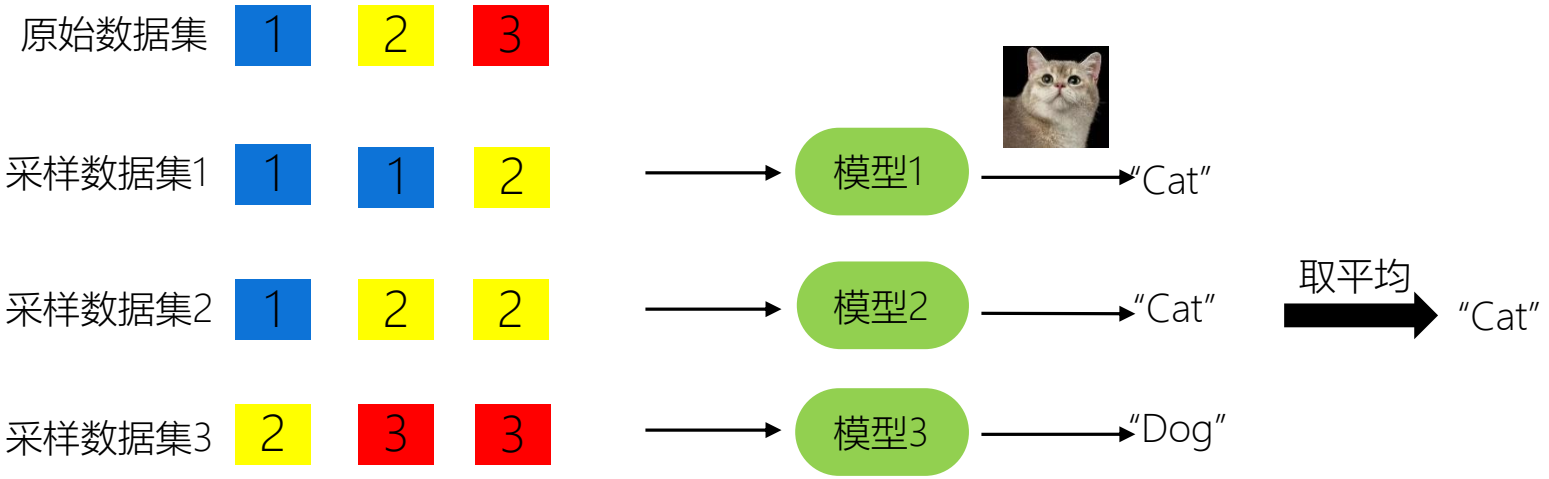
$$\nabla_w \tilde{L}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \nabla_w L(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) + \theta sign(\boldsymbol{w})$$

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(Bootstrap aggregating)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。

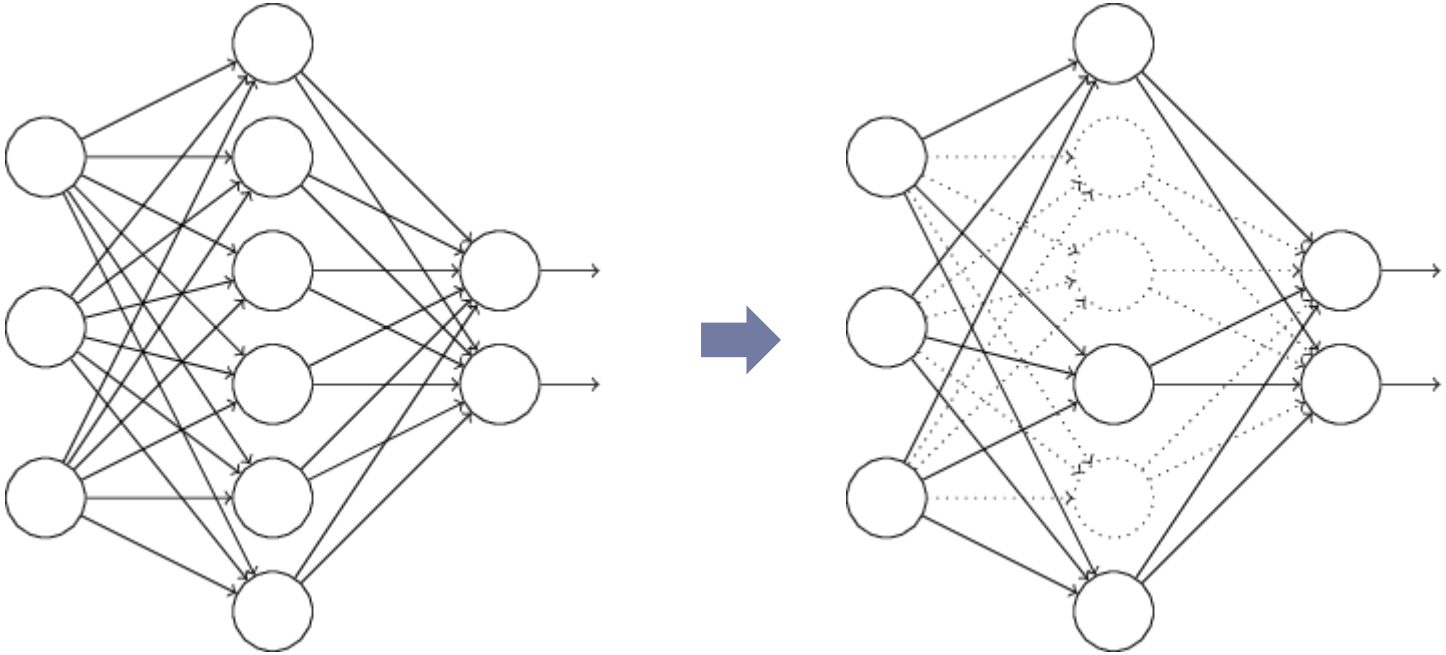
假设集成k=3个网络模型



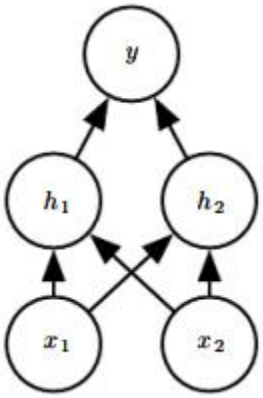
- 模型平均是减小泛化误差的一种可靠方法

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。

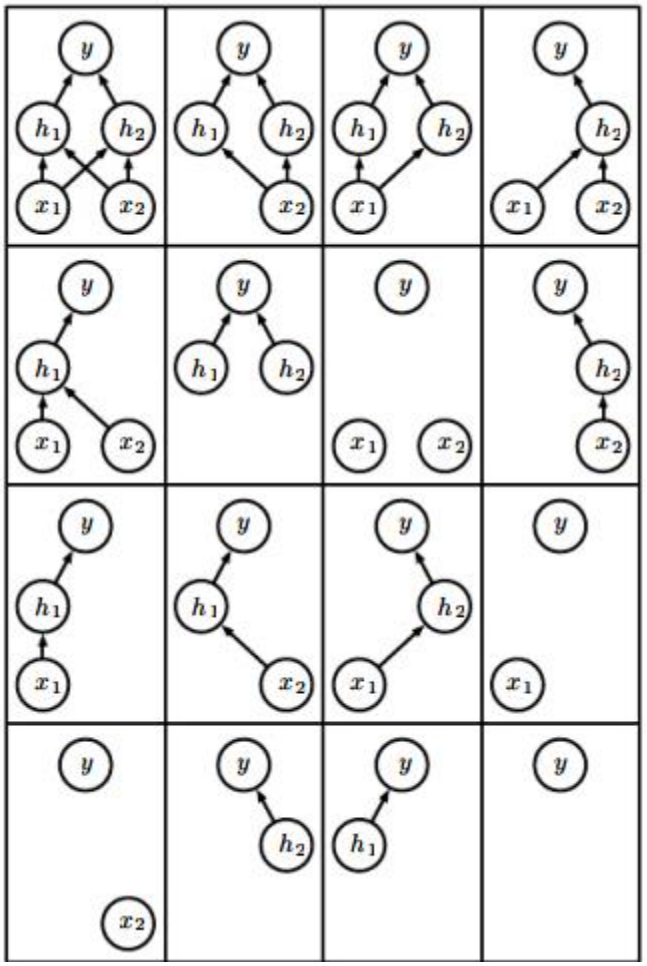


乘零的Dropout 算法

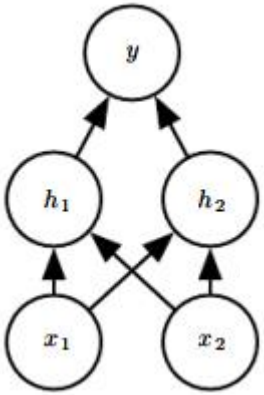


基础网络

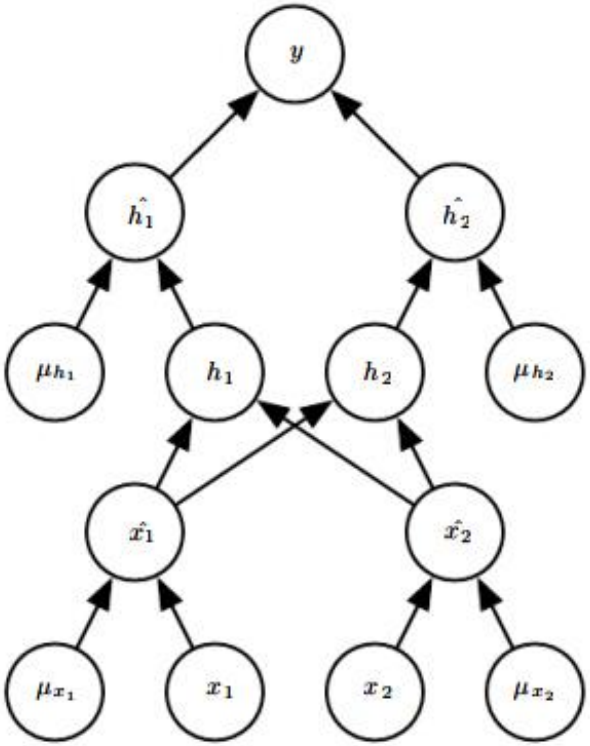
从基础网络中丢弃不同的单元子集形成子网络



子网络集成



基础网络



- 随机对掩码μ进行采样，输入单元的采样概率为0.8，隐藏单元的采样概率为0.5 。网络中的每个单元乘以相应的掩码后沿着网络的其余部分继续向前传播

其他正则化方法

- 提前终止

  当训练较大的网络模型时，能够观察到训练误差会随着时间的推移降低但验证集的误差会再次上升。因此，在训练过程中返回验证误差达最低的参数设置，就可以获得验证集误差更低的模型，这种策略称之为提前终止。

- 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。

提纲

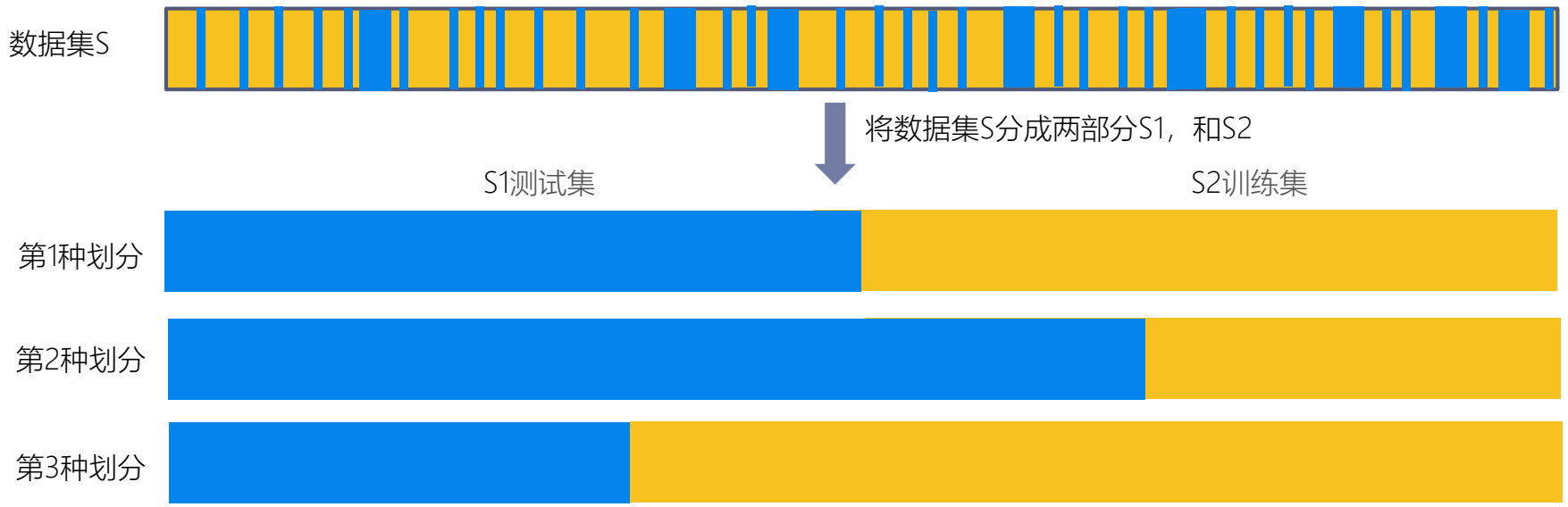
- 从机器学习到神经网络
- 正向传播与反向传播
- 神经网络设计原则
  - 网络的拓扑结构
  - 激活函数
  - 损失函数
- 过拟合与正则化
- 交叉验证
- 小结

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。
- 选择同一模型的不同参数时，使用交叉验证评价各参数组合下的模型稳定性

最简单的验证方式

测试集和训练集

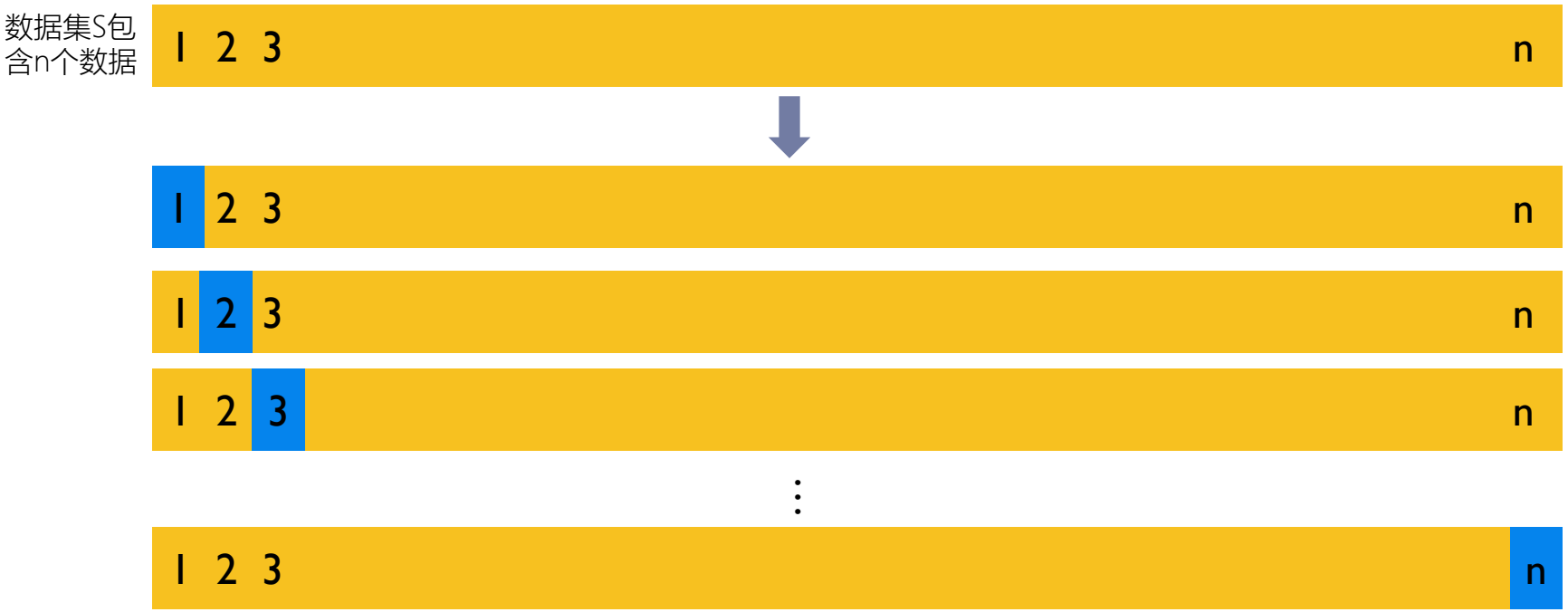


- 不同划分方式下，得到的MSE(Mean Squared Error)变动较大

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

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

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

小结

- 从机器学习到神经网络

  掌握单变量线性回归、多变量线性回归、感知机、神经元、激活函数、偏置等的概念和相关用法

- 正向传播、反向传播

  了解数据正向传播和反向传播的过程。

- 激活函数

  了解常用激活函数的优缺点

- 损失函数

  了解损失函数的作用和种类、掌握至少一种验证方式。

- 过拟合与正则化

  了解过拟合出现的原因并掌握防止过拟合的正则化方法

- 交叉验证

  掌握至少一种交叉验证的原理和方法

谢谢大家！