第七章 深度学习背景与基础知识

1.1 深度学习的背景

1.2 神经网络结构与优化

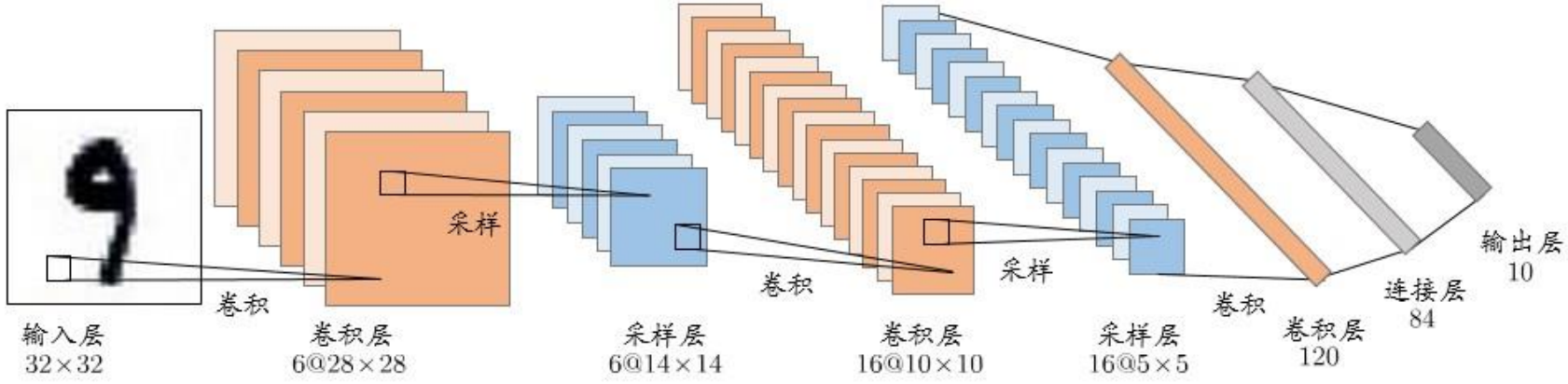
1.3 深度神经网络的训练

什么是深度学习?

典型的深度学习模型就是层数很深（很多）的神经网络

（例如微软研究院2015年在ImageNet竞赛获胜使用152层网络）

深度学习：研究深度神经网络结构设计和优化及训练方法等相关问题的理论以及在不同领域的应用



LeNet-5 (LeCun et al., 1998)

CNN: Convolutional NN [LeCun and Bengio, 1995; LeCun et al. , 1998]

深度神经网络模型的优势

1. 神经网络结构的非线性拟合能力
2. 增加深度可以获得指数增长的表示能力

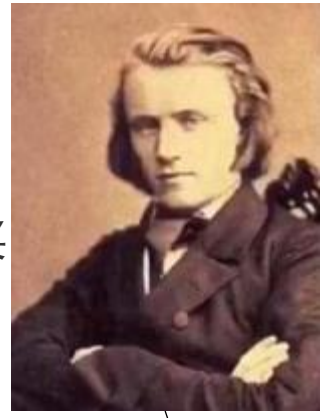
厉害!

3. 基于min-batch的训练方式适用于大规模问题
4. 从实践角度来讲具有相对标准化的设计和训练步骤

好用!

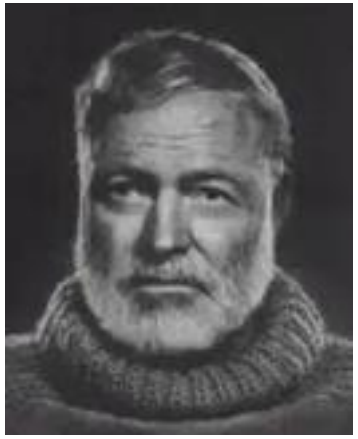
颜值+实力

普朗克（1858~1947），
德国物理学家，
量子力学的创始人
1918年诺贝尔物理学奖

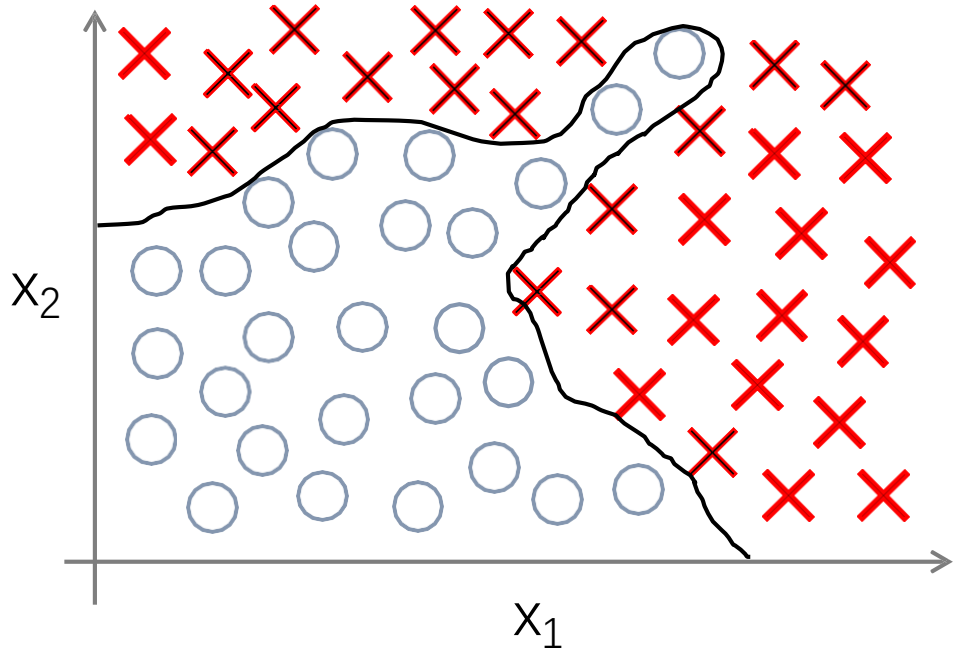


勃拉姆斯（1833~1897）
德国浪漫主义作曲家
与巴赫、贝多芬并称为
德国音乐史中的“3B”

海明威（1899-1961）
20世纪最著名的小说家之一，由《老人与海》获诺贝尔文学奖



非线性分类



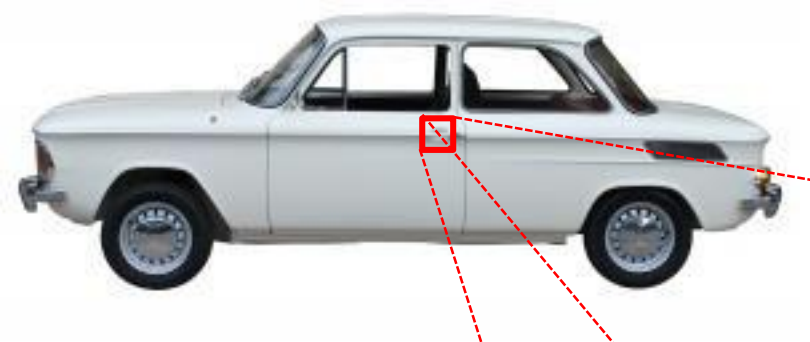
x_1 =size
x_2 =# bedrooms
x_3 =# floors
x_4 =age
$\cdots$
x_{100}

$$f_{w,b}(x) = b + w_1x_1 + w_2x_2 + w_3x_1x_2 + w_4x_1^2x_2 + w_5x_1x_2^2 + w_6x_1^3x_2 + \cdots$$

对$d = 100$个特征，组成的两次方的特征有5000个$(\frac{d^2}{2})$ 更高次数的特征数目会更多。

这是什么?

这张图片是你看到的:



但是数码相机看到的是个矩阵:

194	210	201	212	199	213	215	195	178	158	182	209
180	189	190	221	209	205	191	167	147	115	129	163
114	126	140	188	176	165	152	140	170	106	78	88
87	103	115	154	143	142	149	153	173	101	57	57
102	112	106	131	122	138	152	147	128	84	58	66
94	95	79	104	105	124	129	113	107	87	69	67
68	71	69	98	89	92	98	95	89	88	76	67
41	56	68	99	63	45	60	82	58	76	75	65
20	43	69	75	56	41	51	73	55	70	63	44
50	50	57	69	75	75	73	74	53	68	59	37
72	59	53	66	84	92	84	74	57	72	63	42
67	61	58	65	75	78	76	73	59	75	69	50

计算机视觉: 车辆检测





小汽车

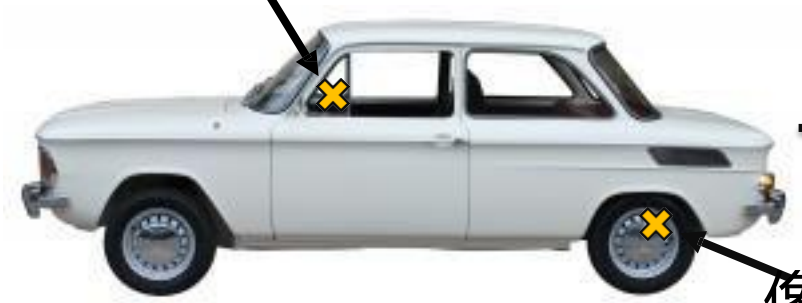





不是小汽车

测试:

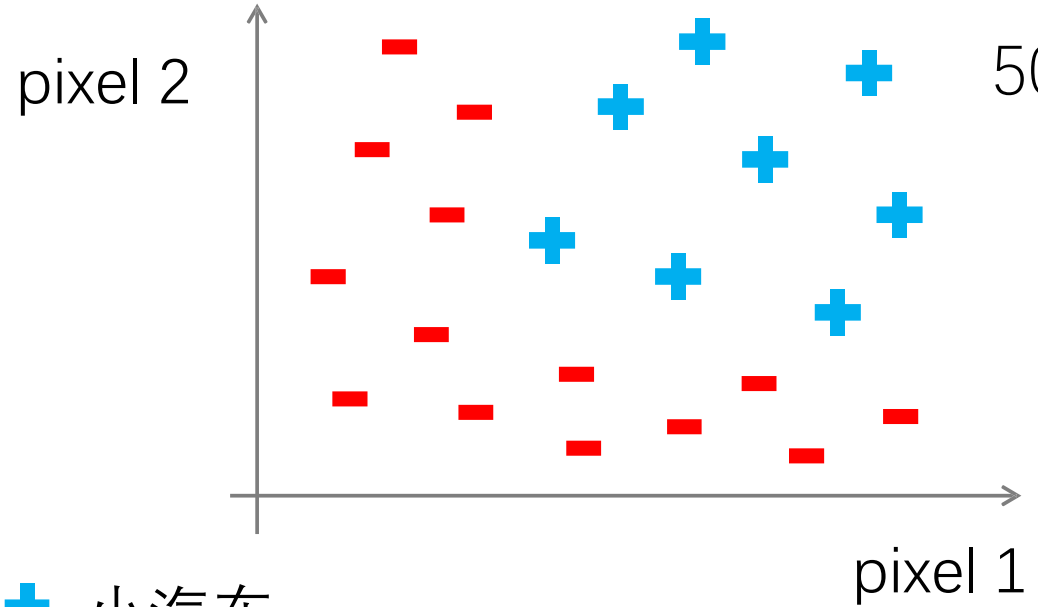
这是小汽车吗?



学习算法

$n = 2500$

50 x 50 像素图片→ 2500 像素（灰度图）
(7500如果是 RGB)



$$x = \begin{bmatrix} \text{像素 1 灰度强度} \\ \text{像素 2 灰度强度} \\ \vdots \\ \text{像素 2500 灰度强度} \end{bmatrix}$$

[0, 255]

二次方特征($x_i \times x_j$): ≈3 百万

为什么需要神经网络?

现实中很多数据集是线性不可分的，需要非线性分类器。

通过特征组合的方式获得高阶特征会产生大量特征。

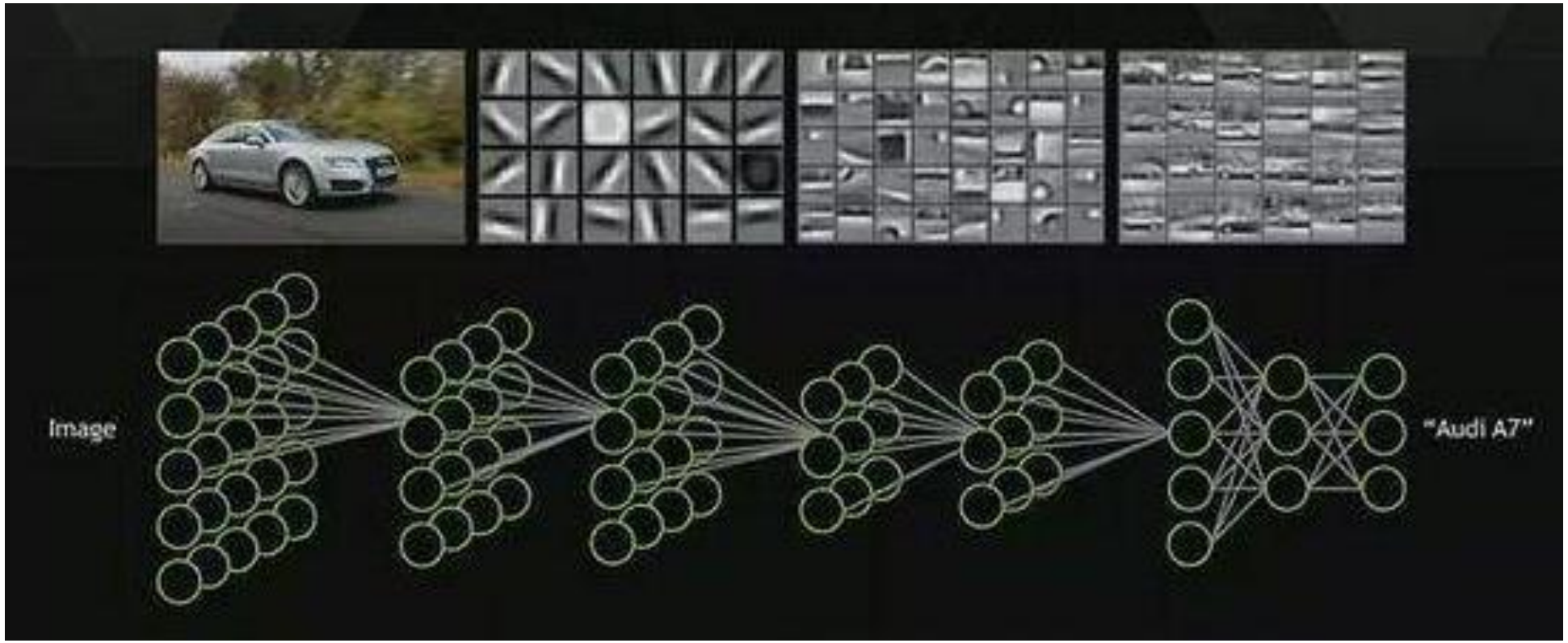
普通的对率回归模型不能有效地处理这么多特征

神经网络

为什么要深度?

通过加深层数获得强大的非线性拟合能力

深层特征由浅层特征组合得到，具有不同层次的抽象能力



图片来自网络

深度学习最重要的特点：表示和具体任务的联合优化

传统做法：

Feature Engineering

人工设计特征 → 学习分类

深度学习：

Representation learning

学习特征 ⇔ 学习分类

所谓 end-to-end Learning

大数据、
高性能计算设备

深度学习方法面临的主要技术难点

深度增加/宽度增大提升模型复杂度

→ 增加过拟合风险

拟合能力太强，容易过拟合，比如记住训练样本和它们对应的标签

→增加训练难度

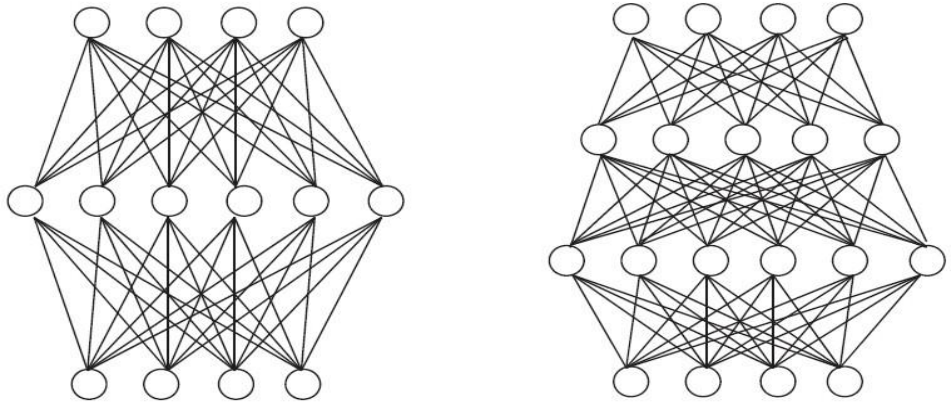
误差梯度在多隐层内传播时，往往会发散而不能收敛到稳定状态

深度学习的兴起

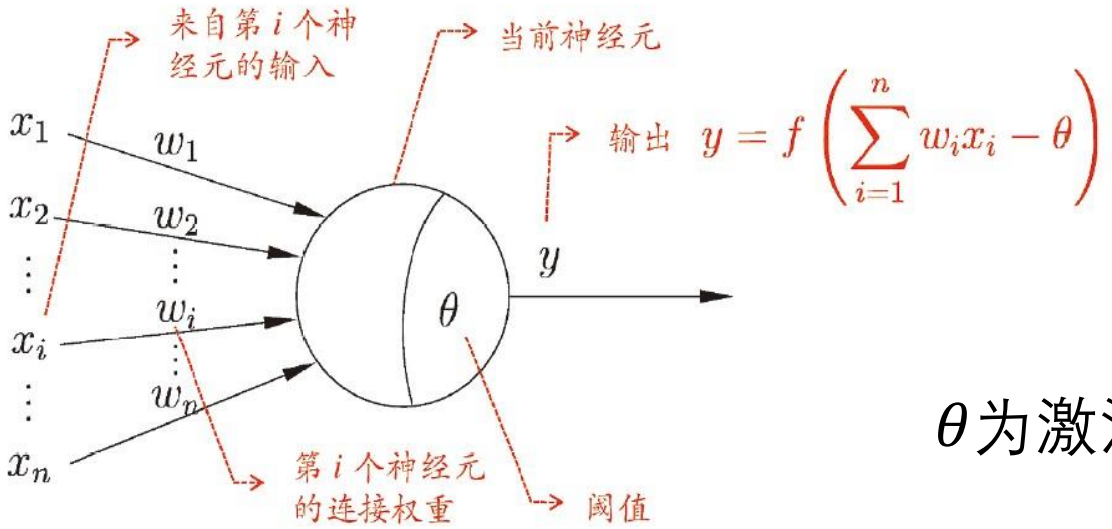
神经网络如感知机在上世纪五六十年代就被提出，八九十年代繁荣，之后进入冰河期。

- 2006年, Hinton发表了深度学习的 Nature 文章
- 2012年, Hinton 组参加ImageNet 竞赛, 使用 CNN 模型以超过第二名10个百分点的成绩夺得当年竞赛的冠军
- 伴随云计算、大数据时代的到来，计算能力的大幅提升，使得深度学习模型在计算机视觉、自然语言处理、语音识别等众多领域都取得了较大的成功

什么是神经网络?



(a) 单隐层前馈网络　　(b) 双隐层前馈网络



神经网络模型=神经元模型+结构+参数

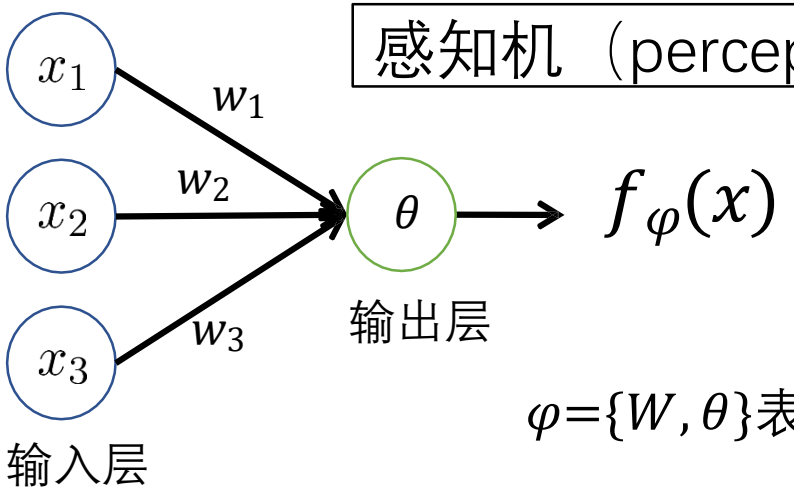
神经元（neuron）也被称为节点(node)

神经网络中的每个神经元是 一个学习模型。

左边的神经元模型为：输入的线性组合通过激活函数输出。

θ为激活该神经元的阈值，或者偏置(bias)

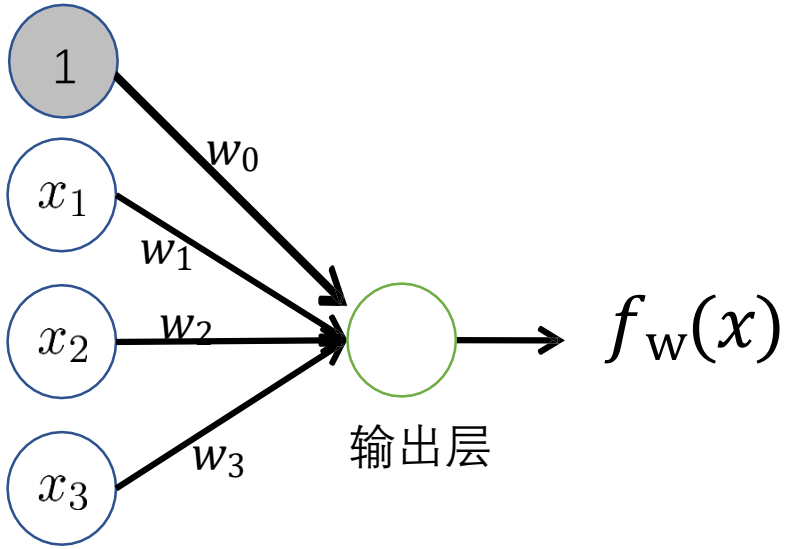
感知机（perceptron）：多个输入到一个输出



$\varphi=\{W,\theta\}$表示模型的所有参数：包括权重(weight) W和偏置(bias)θ

$$f_\varphi(x)=f(\sum_{i=1}^{3}w_ix_i-\theta)$$

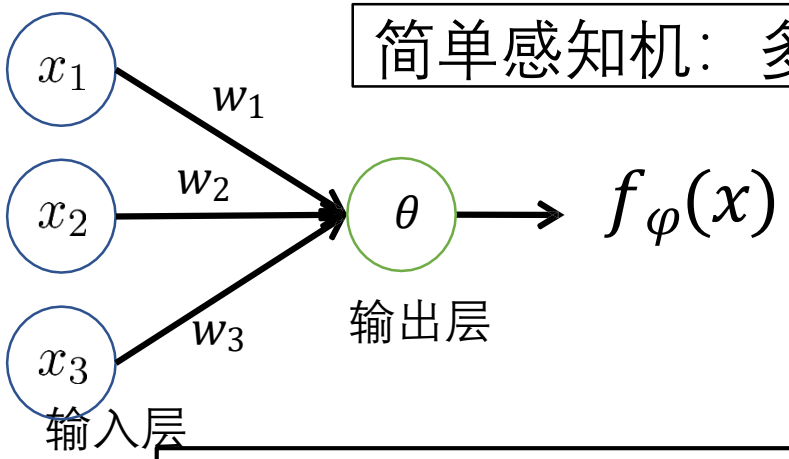
偏置前面的正负不重要



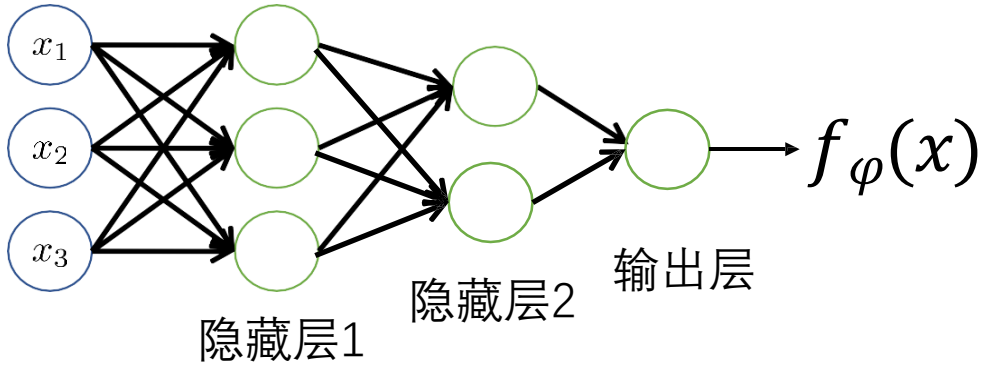
把偏置统一吸收到权重，对应恒定输入1

$$f_w(x)=f(\sum_{i=0}^{3}w_ix_i)=f(\boldsymbol{w}^T\boldsymbol{x})$$

简单感知机：多个输入到一个输出



多层感知机（MLP）：包含隐层的前馈网络

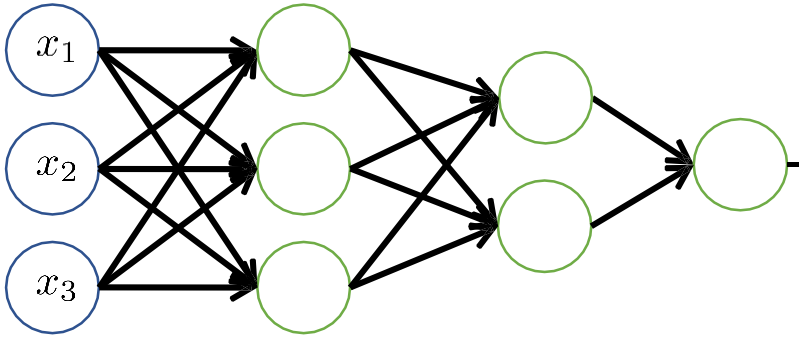


多层前馈网络有强大的表示能力：

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

- 前馈网络：神经元之间不存在同层连接也不存在跨层连接
- 输入层只负责输入，不做功能性处理，只有隐层和输出层神经元才是“功能单元”(functional unit)

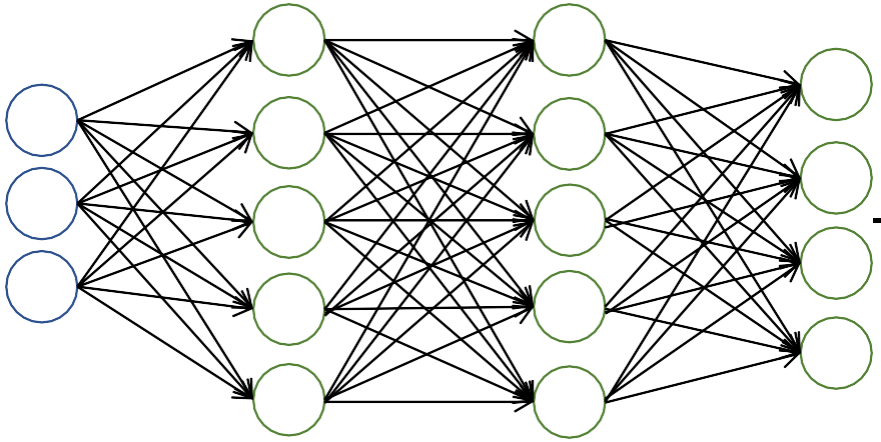
输出层：二分类vs多分类



标签={行人，汽车}

$y = 0$或 1

标签={行人，汽车，摩托车，卡车}



$$\boldsymbol{y} \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$f_\Theta(x) \in \mathbb{R}^4$

输出是一个l维向量（l是类的个数）

非0的那个元素对应所属类别

$f_\Theta(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$，行人

$f_\Theta(x) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$，汽车

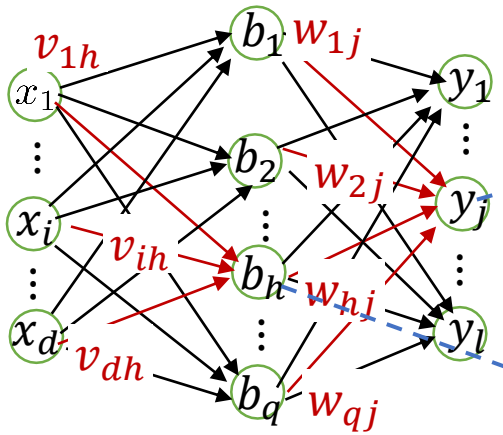
$f_\Theta(x) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$，摩托车

$f_\Theta(x) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$ 卡车

多层前馈网络

前一层的输出是后一层的输入

输入层　隐藏层　输出层



$$y_j = f(\beta_j) = f(\sum_{h=1}^{q} w_{hj} b_h - \theta_j)$$

第j个输出层神经元的输出

第j个输出层神经元的偏置

第h个隐层神经元的输出

$$b_h = f(\alpha_h) = f(\sum_{i=1}^{d} v_{ih} x_i - \gamma_h)$$

第h个隐层神经元的偏置

关于激活函数

1.激活函数必须非线性

2.最后一层的激活函数根据具体任务来确定

模型参数量

给定训练集

$$D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\},\ x_i \in \mathbb{R}^d,\ y_i \in \mathbb{R}^l$$

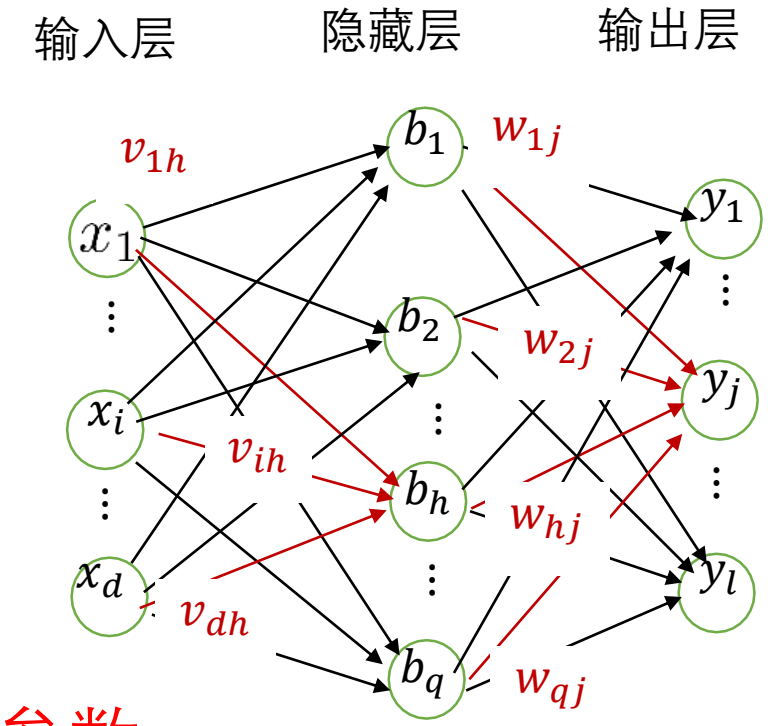
用一个包含q个神经元的单隐层网络，把d维数据分到l个类。

网络结构：

输入：d个节点

隐层：q个神经元

输出层：l个神经元



一共需要估计的参数个数：

$d \times q + q \times l$个权重，以及$q + l$个偏置。

网络结构确定后，训练的任务就是估计参数

误差逆传播算法(BP)

1.设计网络结构
2.选择一个损失函数
3.设置学习率

So easy!

- 主要步骤包括

  初始化网络参数（权重和偏置）

  对读入的每一个样本

  forward (向前计算每层的输出作为下一层的输入)

$$y_j^l = f(\sum_{h=1}^{q} w_{hj}^l y_h^{l-1} - \theta_j^l)$$

y_j^l表示第l个隐藏层中第j个节点的输出。

  根据最后一层的输出和期望输出，计算损失$loss = l(\boldsymbol{y}, \hat{\boldsymbol{y}})$

  backward（向后传播损失的梯度，更新网络参数$\varphi = \{W, \theta\}$ ）

$$\varphi \leftarrow \varphi - \eta \frac{\partial l}{\partial \varphi}$$

学习率η是一个重要的超参数

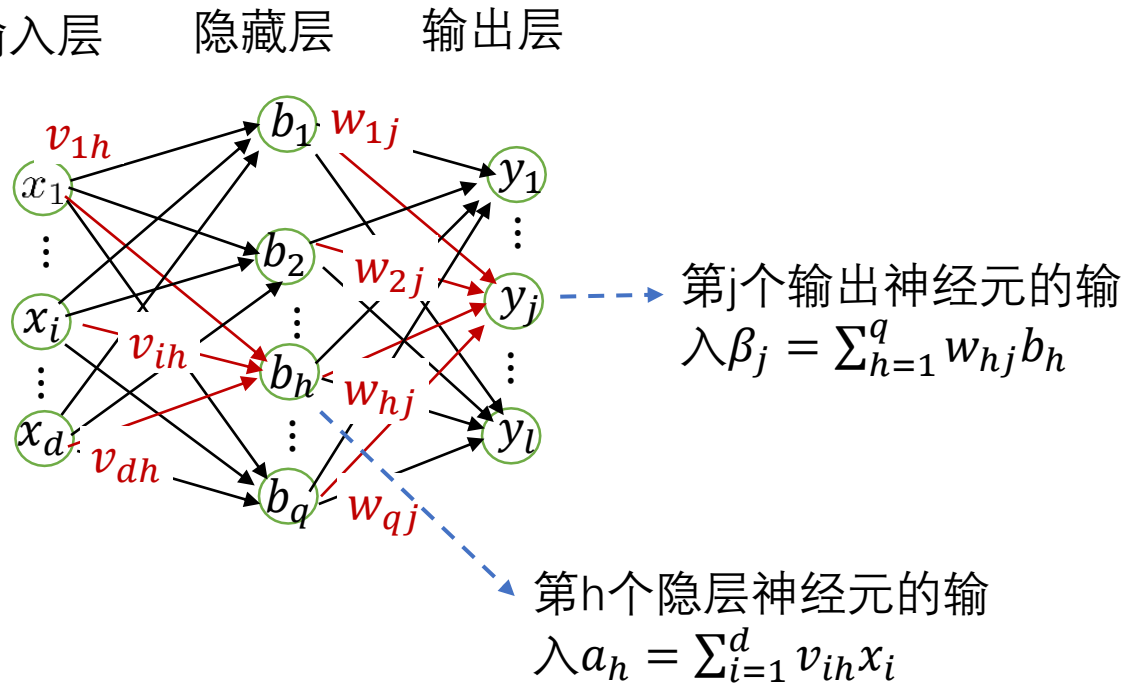
BP 算法推导

对于训练例 $(\boldsymbol{x}_k, \boldsymbol{y}_k)$, 假定网络的实际输出为 $\hat{\boldsymbol{y}}_k = (\hat{y}_1^k, \hat{y}_2^k, \ldots, \hat{y}_l^k)$

$$\hat{y}_j^k = f(\beta_j - \theta_j)$$

则网络在 $(\boldsymbol{x}_k, \boldsymbol{y}_k)$ 上的均方误差为:

$$E_k = \frac{1}{2}\sum_{j=1}^{l}(\hat{y}_j^k - y_j^k)^2$$



BP 是一个迭代学习算法, 在迭代的每一轮中采用广义感知机学习规则

$$v \leftarrow v + \boxed{\triangle v}.$$

BP 算法推导 (续)

BP 算法基于**梯度下降**策略，以目标的负梯度方向对参数进行调整

以 w_{hj} 为例

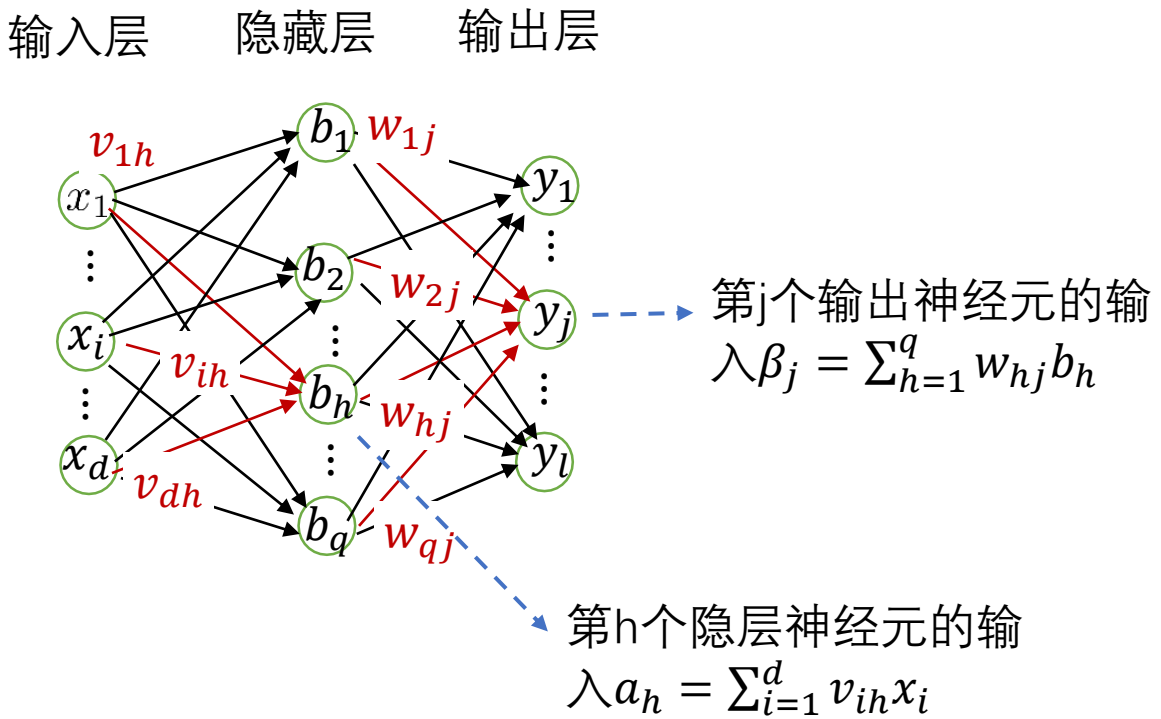
对误差 E_k，给定学习率 η，有：

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

注意到 w_{hj} 先影响到 β_j，

再影响到 $\hat{y}_j^k$，然后才影响到 E_k，有：

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



⇐ “链式法则”

BP 算法推导 (续)

假定功能单元均使用Sigmoid激活函数

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

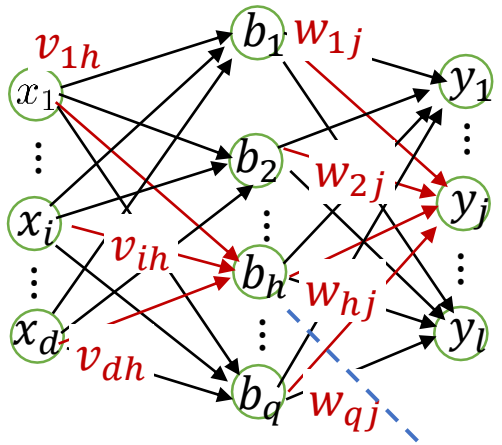
$$= b_h$$

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

$$= -(\hat{y}_j^k - y_j^k) f'(\beta_j - \theta_j)$$

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

输入层　隐藏层　输出层



第j个输出神经元的输入 $\beta_j = \sum_{h=1}^{q} w_{hj} b_h$

第h个隐层神经元的输入 $a_h = \sum_{i=1}^{d} v_{ih} x_i$

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

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

再注意到 $\hat{y}_j^k = f(\beta_j - \theta_j)$

于是，

$$\Delta w_{hj} = -\eta \frac{\partial E_k}{\partial w_{hj}} = \eta g_j b_h \quad (2)$$

BP 算法推导 (续)

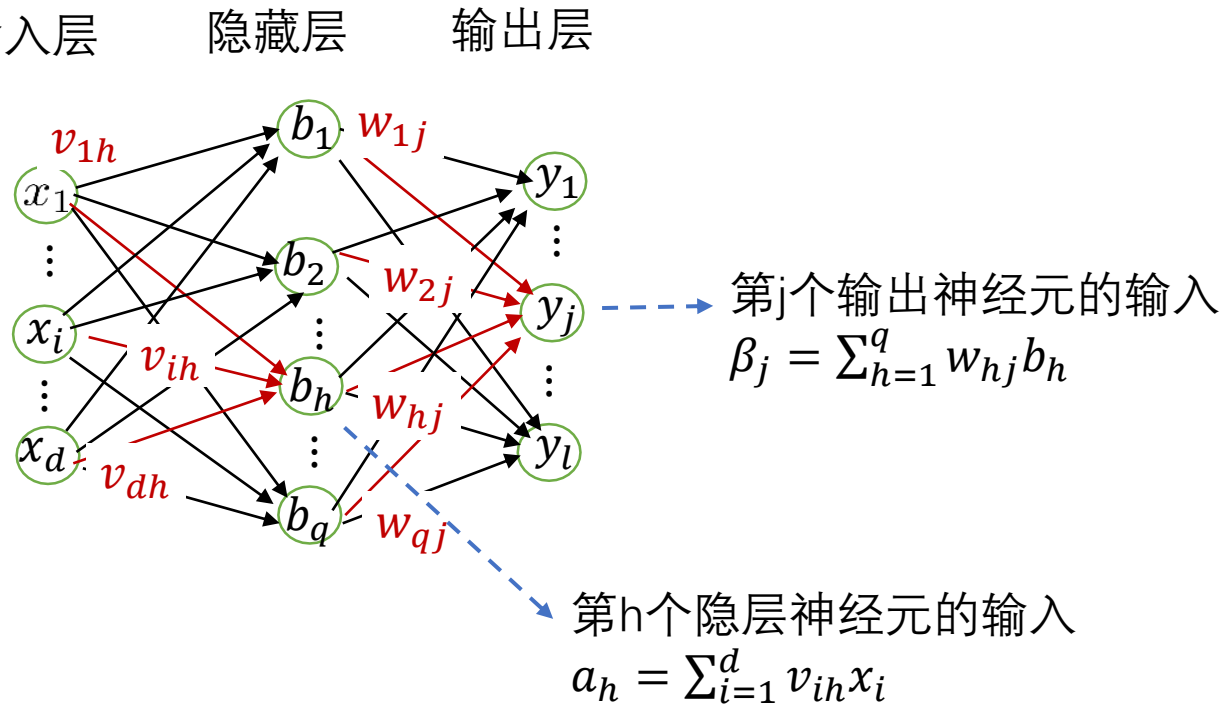
类似地，有：

$$\Delta\theta_j = -\eta g_j \quad (3)$$

$$\Delta v_{ih} = \eta e_h x_i \quad (4)$$

$$\Delta\gamma_h = -\eta e_h \quad (5)$$

其中：



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

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

$$= b_h(1 - b_h) \sum_{j=1}^{l} w_{hj} g_j \quad (6)$$

学习率 $\eta \in (0,1)$ 不能太大、不能太小

BP 算法

输入：训练集D=$\{(\boldsymbol{x}_k, \boldsymbol{y}_k)\}_{k=1}^{n}$，学习率 η。
输出：模型参数，权重$w_{hj} \in \boldsymbol{W}$, $v_{ih} \in \boldsymbol{V}$和阈值$\theta_j \in \Theta, \gamma_h \in \Gamma$
过程：
1. 在[0,1]范围内随机初始化参数
2. Repeat
3. for all $(\boldsymbol{x}_k, \boldsymbol{y}_k) \in$ D do
4. 根据当前参数，前向传播计算当前样本的预测输出$\hat{y}_k$;
5. 根据式（1）计算出输出神经元梯度项g_j;
6. 根据式（6）计算出隐藏神经元梯度项e_h;
7. 根据梯度下降公式以及（2）-（5），更新所有参数 w_{hj}，v_{ih}，θ_j, 以及γ_h。
8. End for
9. Until 达到结束条件

随机梯度下降Stochastic Gradient Descend

随机梯度下降

- 每次针对单个训练样例更新权值与偏置
- 参数更新频繁，不同样例可能抵消，需要多次迭代

梯度下降

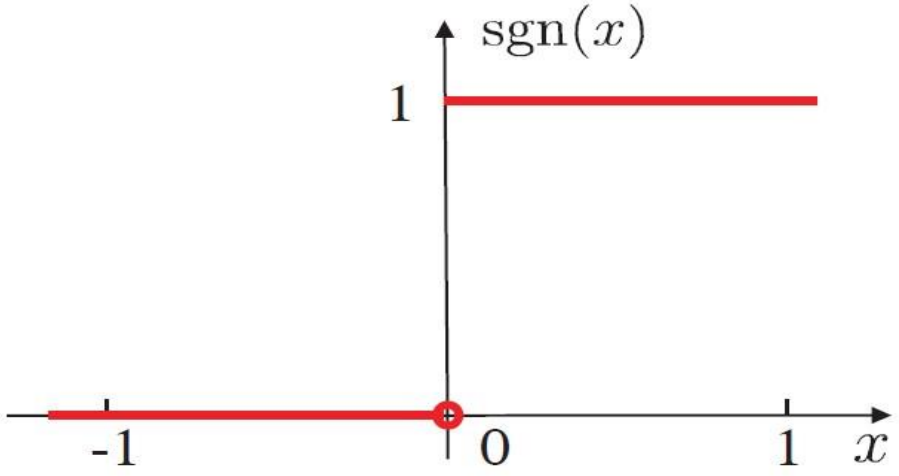
- 其优化目标是最小化整个 训练集上的累计误差
- 读取整个训练集一遍才对 参数进行更新，参数更新 频率较低

在很多任务中，累计误差下降到一定程度后，进一步下降会非常缓慢，这时随机梯度下降往往会获得较好的解，尤其当训练集非常大时效果更明显。

实际采用mini-batch的形式读入，基于每个mini-batch内的样本损失更新参数。mini-batch的大小可以取64， 128， 256等。

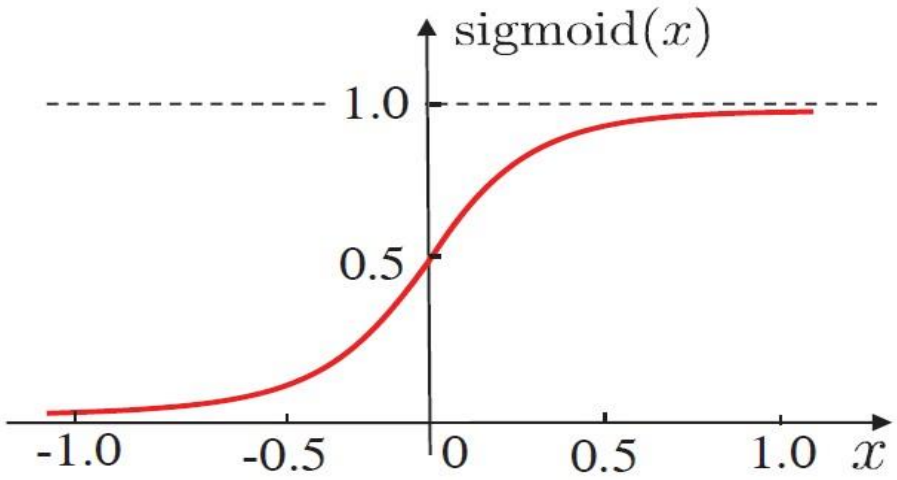
激活函数：非线性函数

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



$$\text{sgn}(x) = \begin{cases} 1, & \text{if } x \geq 0; \\ 0, & \text{if } x < 0. \end{cases}$$

(a) 阶跃函数



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

对率函数

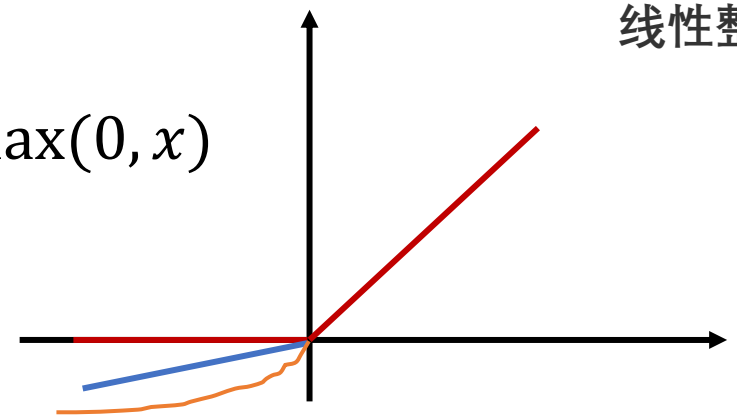
(b) Sigmoid 函数

激活函数

- ReLU: 由于sigmoid存在梯度消失问题(课外拓展)，在层数很多时激活函数ReLU用的更多

线性整流函数（Rectified Linear Unit, **ReLU**）

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

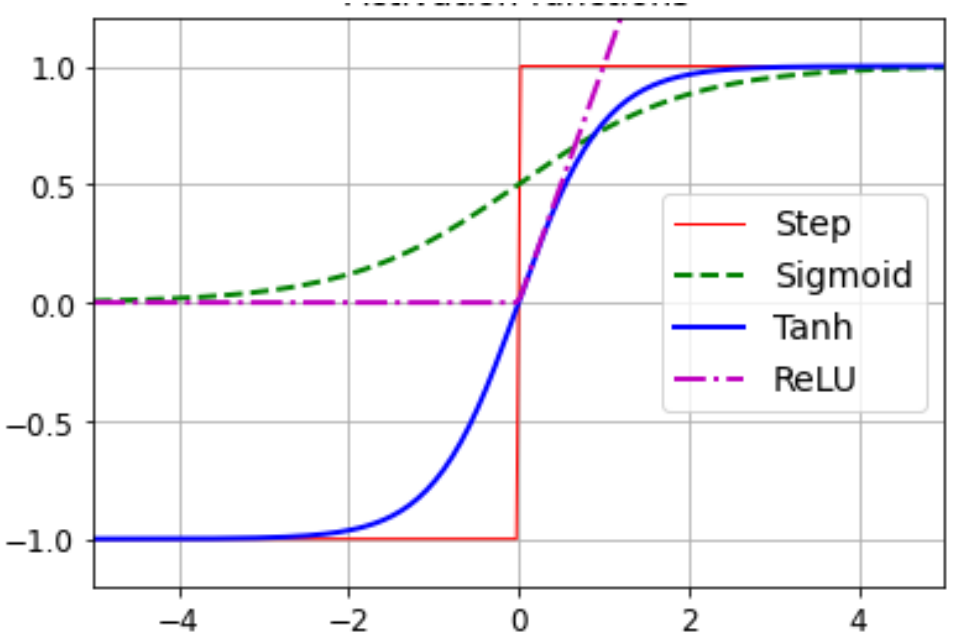


其他版本: Leaky ReLu、ELU等

- tanh(hyperbolic tangent):

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

$\sigma(x)$为Sigmoid函数

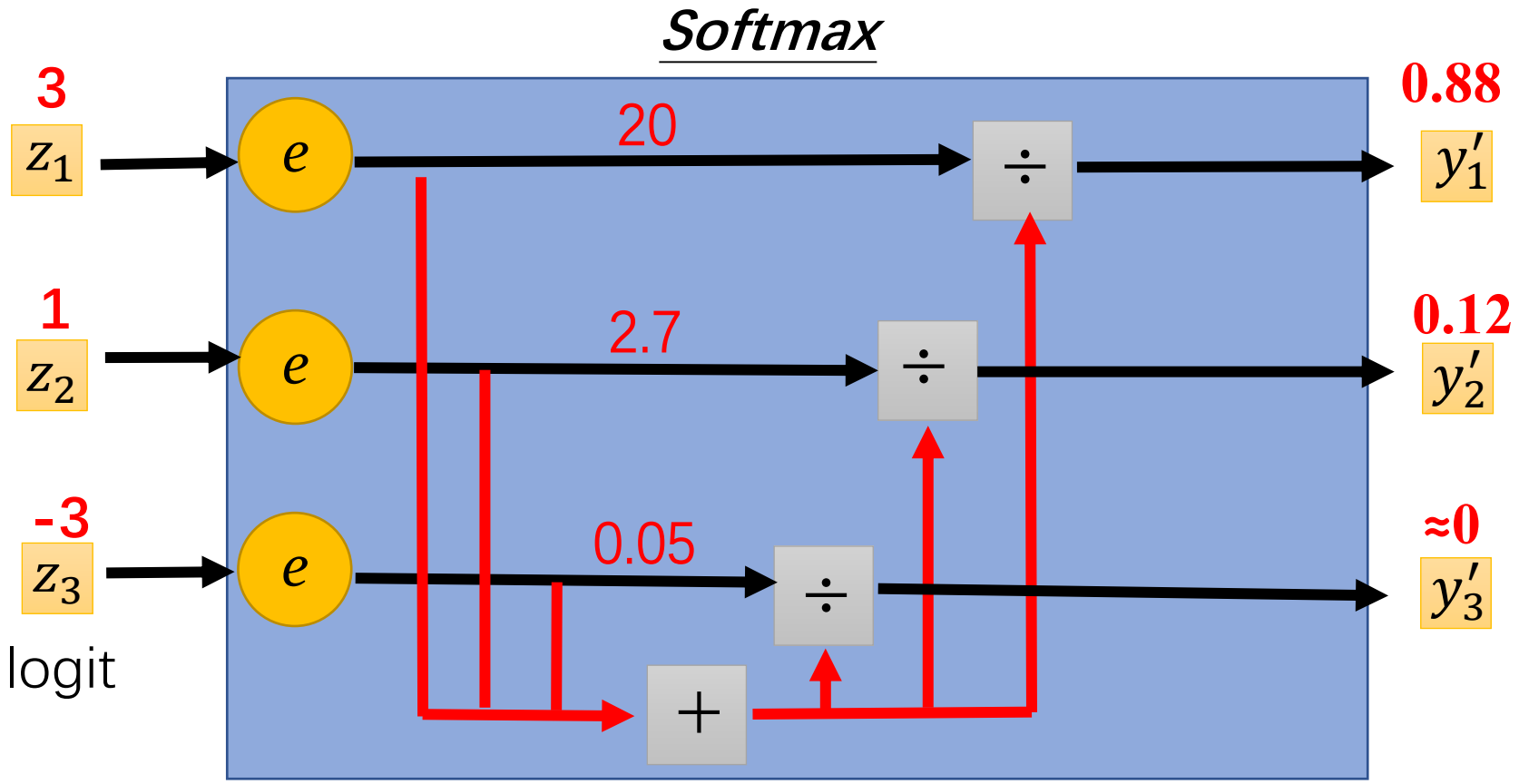


分类器最后一层的激活函数Softmax

$$y_i' = \frac{exp(z_i)}{\sum_j exp(z_i)}$$

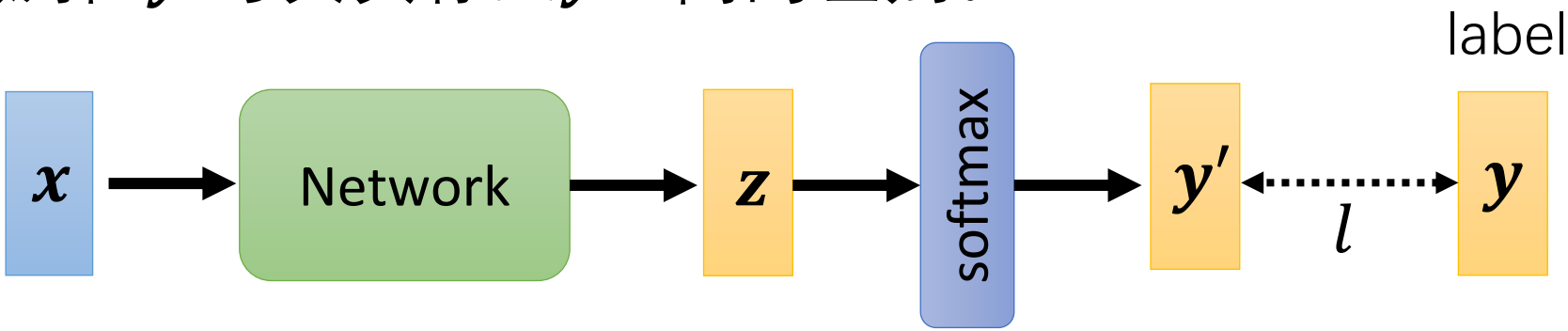
- $1 > y_i' > 0$
- $\sum_i y_i' = 1$

把输出z转成到类别的概率分布y'



损失函数

- 预测值$\boldsymbol{y'}$与真实标签$\boldsymbol{y}$之间的差别。



训练时，对一个batch内样本计算总损失：

$$\text{loss} = \sum_{x_i \in B} l(\boldsymbol{y_i}, \boldsymbol{y_i'})$$

损失函数

- 预测值$\boldsymbol{y}'$与真实标签$\boldsymbol{y}$之间的差别。
- 常用的损失函数(单个样本)有:

  均方误差（mean squared error）：$l = \|\boldsymbol{y} - \boldsymbol{y}'\|^2$，用于回归

  交叉熵（cross entropy）：$l = H(\boldsymbol{y}, \boldsymbol{y}') = -\sum_{c=1}^{C} y_c \log y'_c$，用于分类

当$\boldsymbol{y}$是$\boldsymbol{one-hot}$向量，只有一个元素为1(假设真实标签为第k类，则y_k=1)，得到$H(\boldsymbol{y}, \boldsymbol{y}') = -\log y'_k$。

正则(regularization)

深度神经网络由于模型复杂，非常容易过拟合

正则有利于缓减以上问题

- 常用方法主要包括

1. 损失函数中添加模型权重范数，如$\|\boldsymbol{w}\|^2 = \boldsymbol{w}^T\boldsymbol{w}$

   该损失称为l_2正则，产生的效果为”weight decay”

2. Drop out
3. 数据增强(data augmentation)
4. 提前终止(early stopping)

“正则”到底是什么？

模型权重范数

把权重范数(norm)作为额外惩罚项并入损失函数。(通常不对偏置做惩罚)

$$l'_w(x) = l_w(x) + \frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w}$$

求偏导得到梯度

$$\frac{\partial l'}{\partial w} = \frac{\partial l}{\partial w} + \lambda \boldsymbol{w}$$

不加惩罚项之前的梯度

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \frac{\partial l'_w}{\partial \boldsymbol{w}} \Rightarrow \boldsymbol{w} \leftarrow (1 - \lambda')\boldsymbol{w} - \eta \frac{\partial l_w}{\partial \boldsymbol{w}}$$

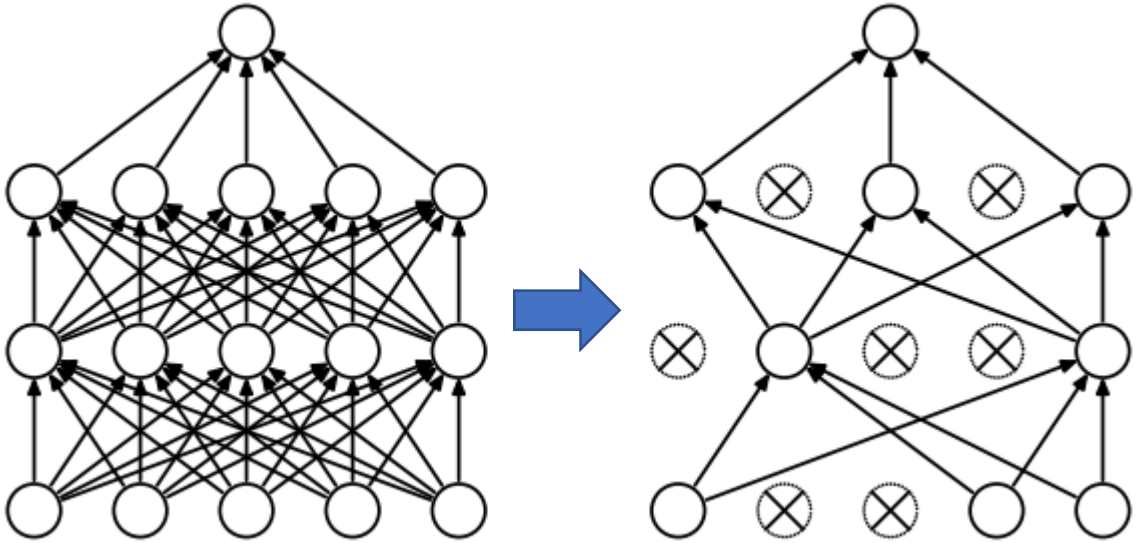
$$\lambda' = \eta\lambda$$

Dropout

- 对训练中每次迭代的每个样本，随机采样出一个宽度缩小的子网络进行损失的计算和更新。

具体做法：

以给定概率p通过随机丢弃每层(layer)中的部分节点以及与它们相连接的权重。



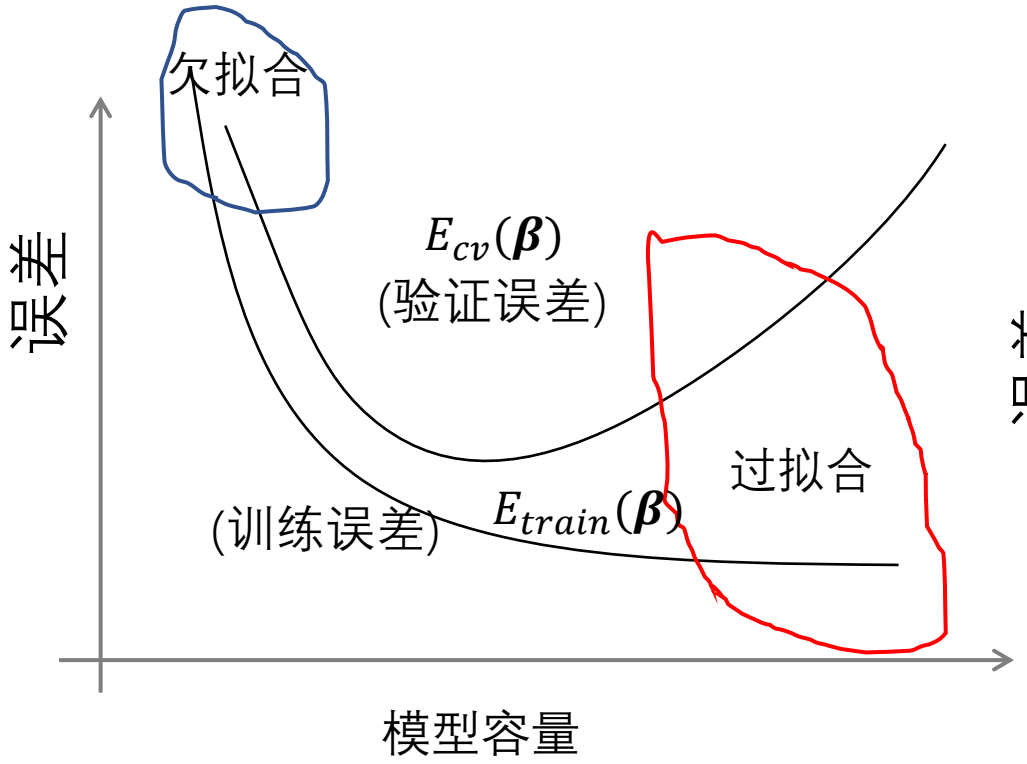
每次更新的是一部分权重，相当于动态剪枝。

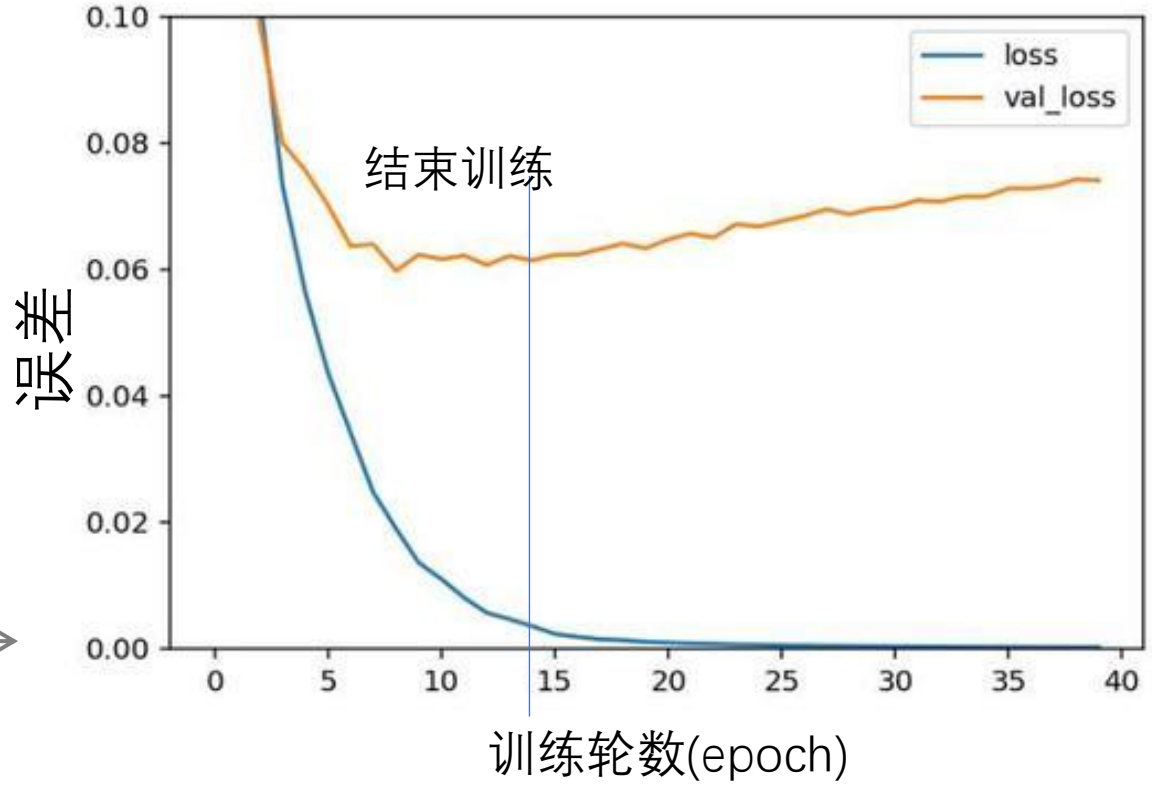
对于两个训练样本，其对应的模型不一样，但大多数参数共享。

在测试的时候，所有节点都被选中，但权重乘以p。

N. Srivastava et al. Dropout: A Simple Way to Prevent Neural Networks from Overfitting，Journal of Machine Learning Research 15 (2014) 1929-1958。

提前停止训练(early stopping)





通过设置patience参数。计算每个epoch结束后的验证误差，如果连续patience次都没有下降，则结束训练，并把停止时的参数作为最后模型参数。

数据增强

- 让机器学习模型泛化地更好的最好办法是使用更多的数据进行训练。
- 对已有训练样本进行数据增强是一种廉价的方式得到更多伪样本。
- 数据增强主要包括两种:

一种是普适的增强，比如对样本加入高斯噪声，用生成模型生成伪样本。

另外一种是适用于特定应用领域的，比如对图片进行随机裁剪，翻转，对句子里面的词进行随机替换等。