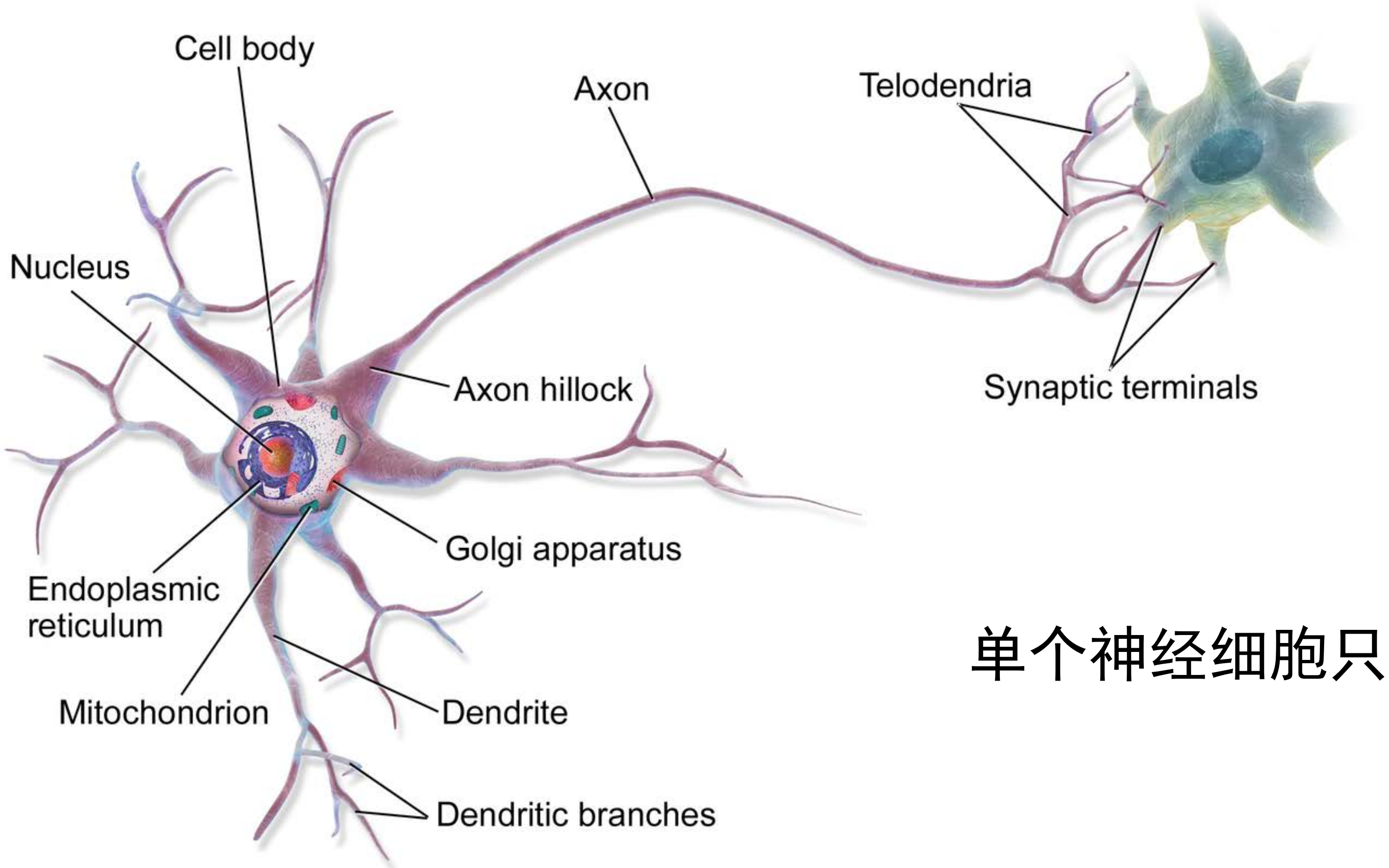
Lecture4
神经网络

生物神经元



单个神经细胞只有两种状态：兴奋和抑制

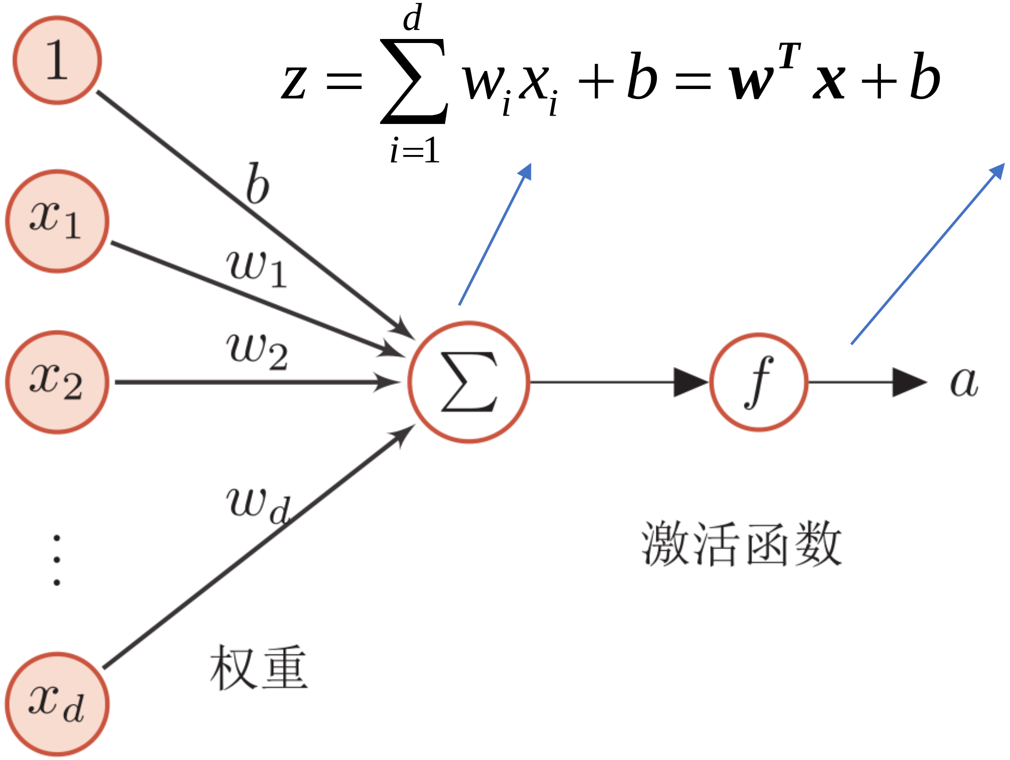
人工神经元

- M-P 神经元模型 [McCulloch and Pitts,1943]

  神经元接到来自前置d个神经元传递过来的输入信号，这些输入信号通过带权重的连接进行传递，神经元接收到的总输入值将与神经元的阀值进行比较，然后通过“激活函数”处理产生神经元的输出。

$$z=\sum_{i=1}^{d} w_i x_i + b = \boldsymbol{w}^T \boldsymbol{x} + b$$

$$a=f(z)$$



人工神经网络

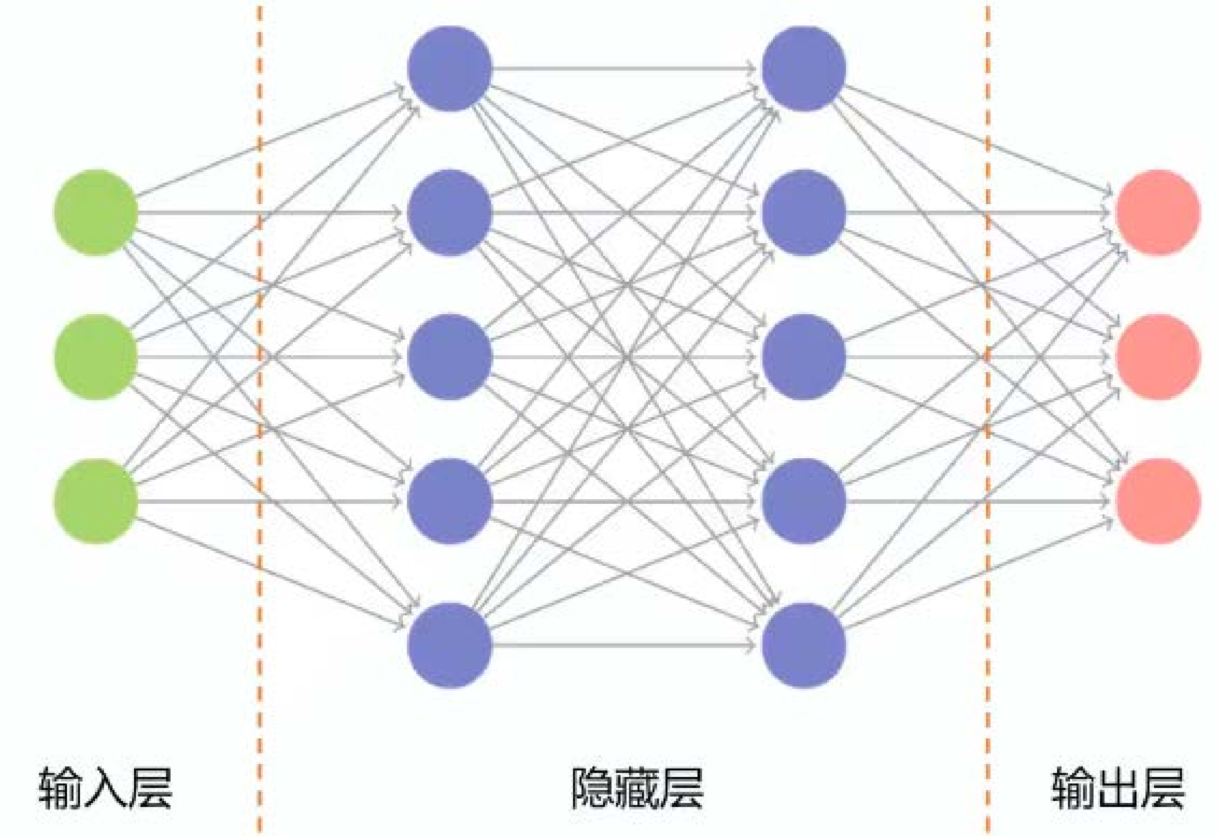
■ 把许多人工神经元按一定的层次结构连接起来，就形成了人工神经网络。

■ 人工神经网络的三大要素：

✓ 节点 —— 采用什么激活函数？

✓ 连边 —— 权重（参数）是多少？

✓ 连接方式 —— 如何设计层次结构？

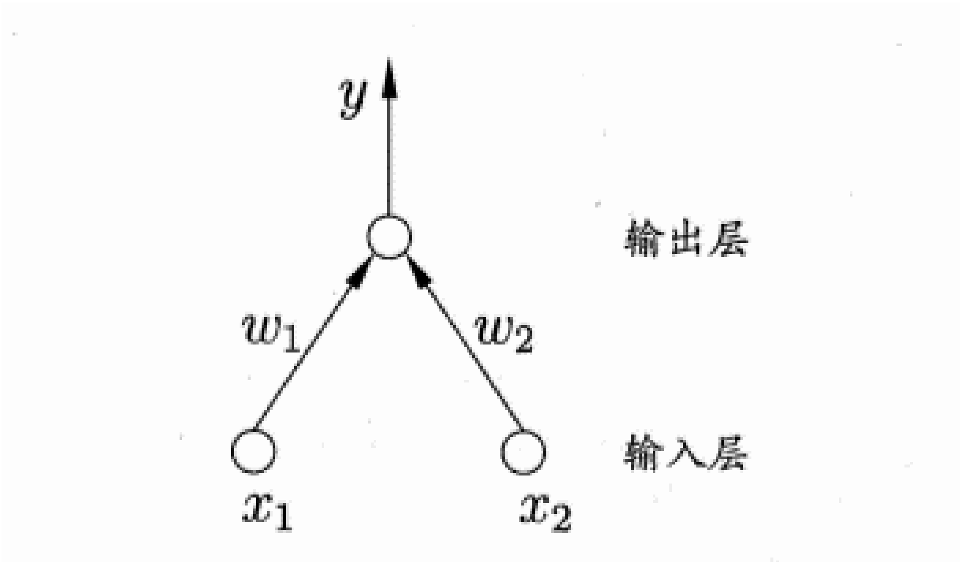


一个解决异或问题的简单网络

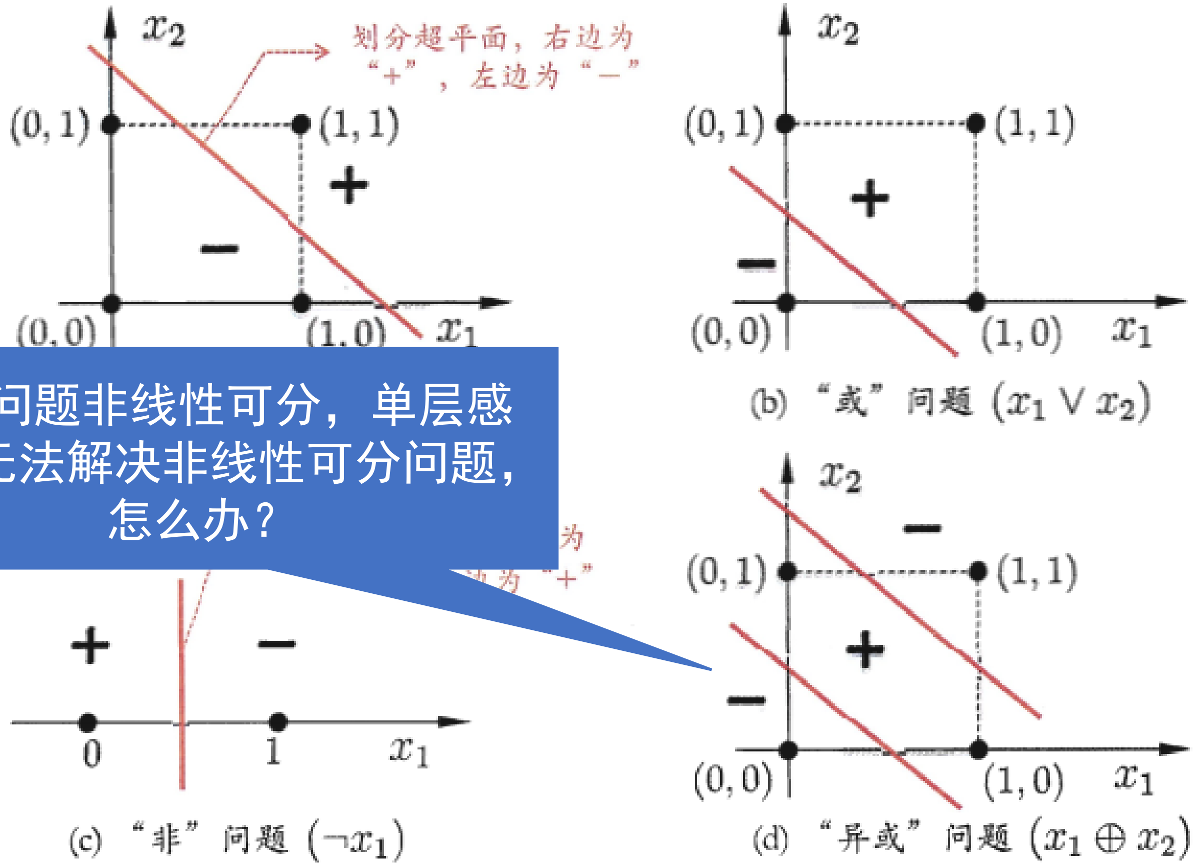
- 感知机回顾
- 双层感知机解决异或问题

感知机求解异、或、非及异或问题

- 输入为[$x_1;x_2$]的单层单个神经元（输入层不计入层数），采用阶跃激活函数。



感知机



异或问题非线性可分，单层感知机无法解决非线性可分问题，怎么办？

双层感知机 —— 一个简单的神经网络

- 输入仍为$[x_1; x_2]$，让网络包含两层

✓ 隐藏层包含两个神经元：

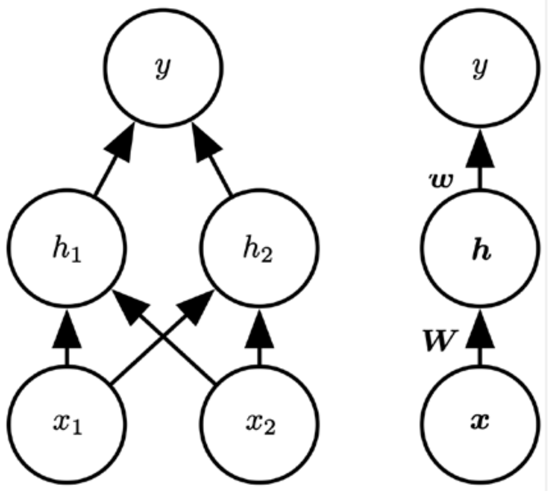
$$\boldsymbol{h} = f^{(1)}(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c})$$

✓ 输出层包含一个神经元：

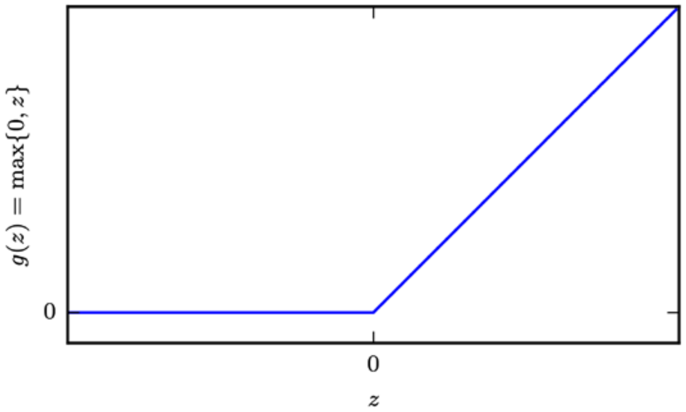
$$y = f^{(2)}(\boldsymbol{h}; \boldsymbol{w}, b)$$

✓ 隐藏层采用线性整流激活函数（ReLU），则整个模型为：

$$\begin{aligned} f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) &= f^{(2)}\left(f^{(1)}(\boldsymbol{x})\right) \\ &= \boldsymbol{w}^{\boldsymbol{T}} \max\{0, \boldsymbol{W}^{\boldsymbol{T}}\boldsymbol{x} + \boldsymbol{c}\} + b \end{aligned}$$



双层感知机



ReLU**函数** $g(z) = \max\{0, z\}$

双层感知机 —— 一个简单的神经网络

■ 给出异或问题的一个解:

$$W=\begin{pmatrix}1&1\\1&1\end{pmatrix},c=\begin{pmatrix}0\\-1\end{pmatrix}^T,w=\begin{pmatrix}1\\-2\end{pmatrix},b=0$$

模型处理流程如下:

① 输入4个样本的矩阵表示为:

$$X=\begin{pmatrix}0&0\\0&1\\1&0\\1&1\end{pmatrix}$$

② 乘以第一层权重矩阵,得到:

$$XW=\begin{pmatrix}0&0\\1&1\\1&1\\2&2\end{pmatrix}$$

③ 加上偏置向量 c, 得到:

$$XW+c=\begin{pmatrix}0&-1\\1&0\\1&0\\2&1\end{pmatrix}$$

④ 使用ReLU激活函数,得到:

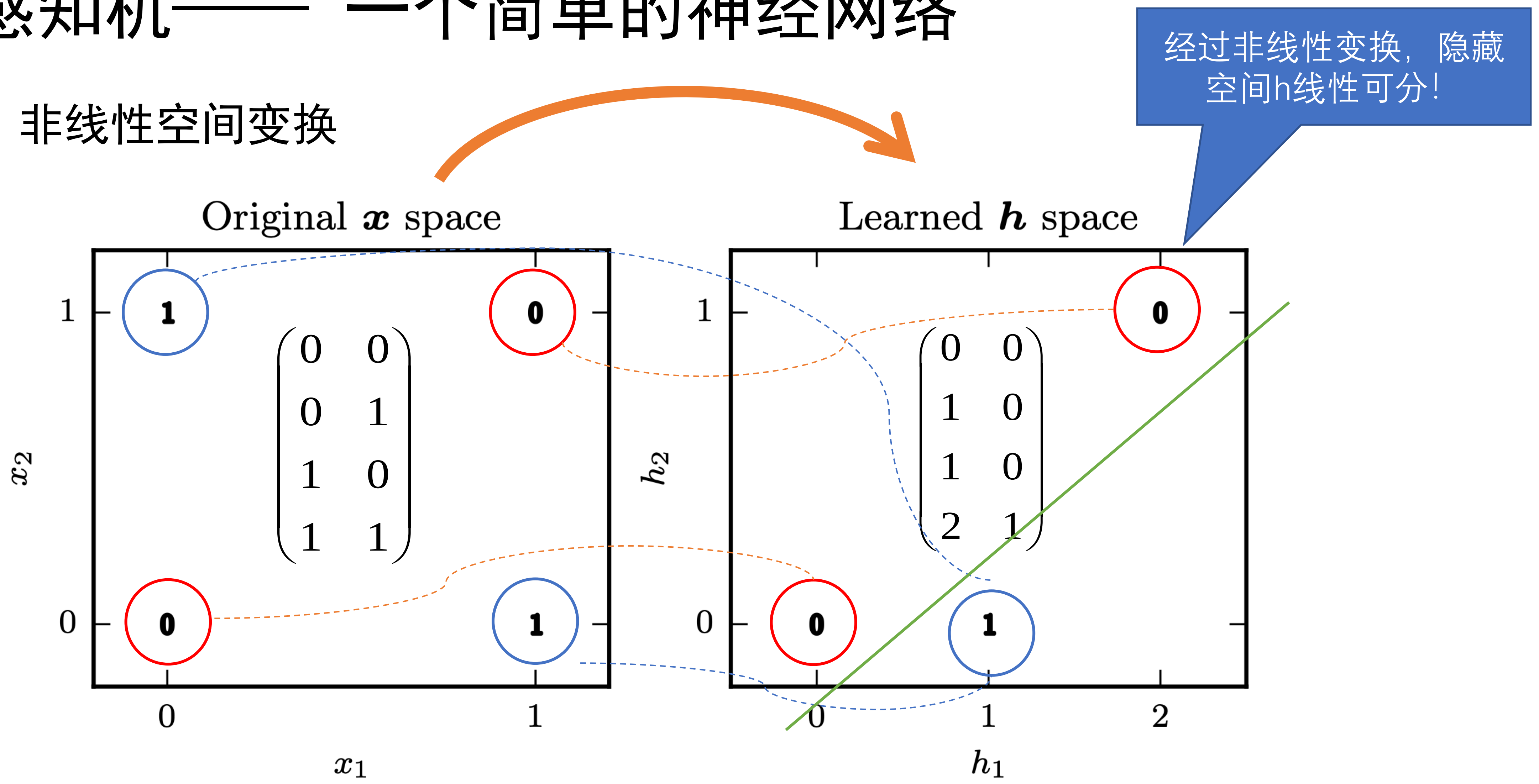
$$\max\{0,XW+c\}=\begin{pmatrix}0&0\\1&0\\1&0\\2&1\end{pmatrix}$$

⑤ 乘以第二层权重向量w, 得到:

$$y=\begin{pmatrix}0\\1\\1\\0\end{pmatrix}$$

双层感知机—— 一个简单的神经网络

- 解释：非线性空间变换



原始的空间x

学习得到的隐藏空间h

神经网络的结构

- 为什么要加深度
- 常见神经网络结构

为什么要加深度

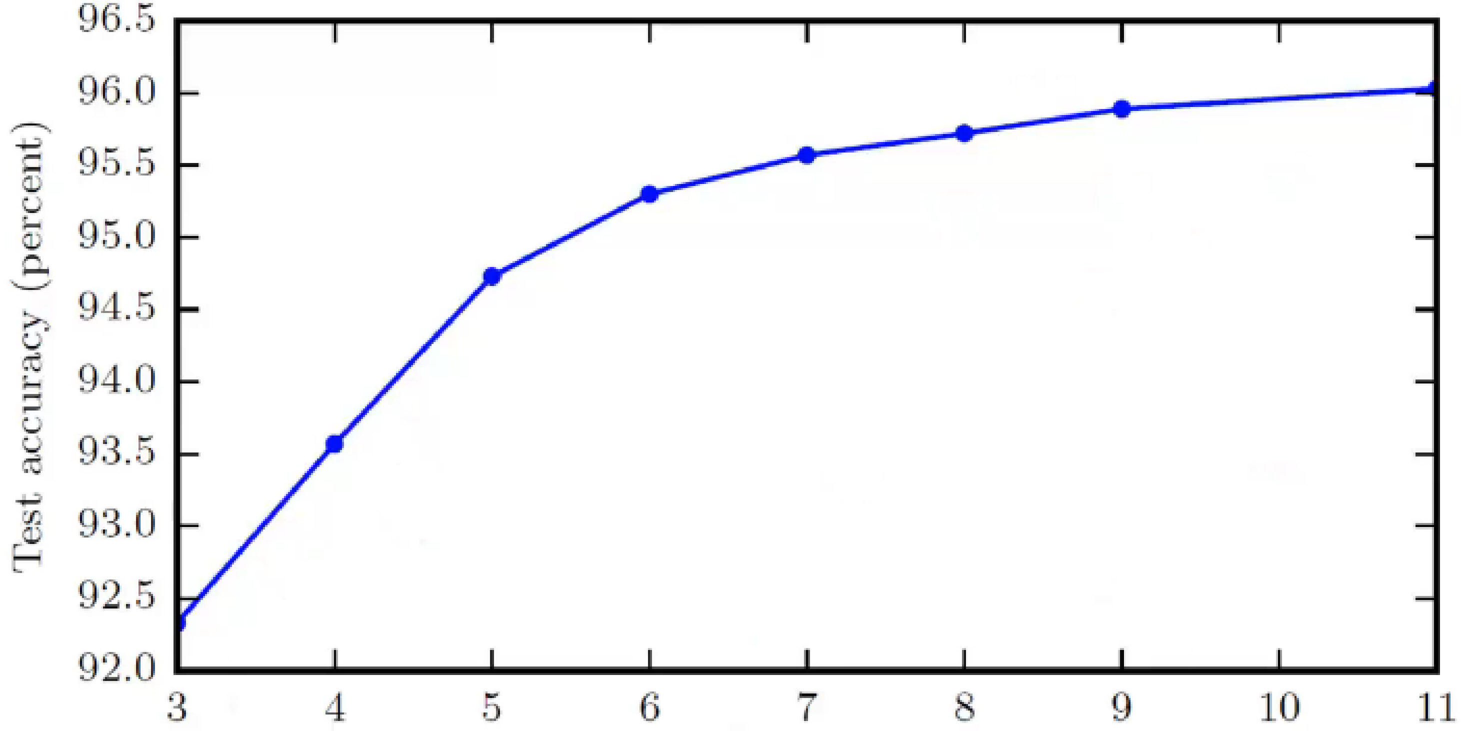
- 数学理论表明，单隐层网络可以近似任何函数，但其规模可能巨大
  - ✓ 在最坏情况下，需要指数级别的隐藏单元才能近似某个函数[Barron,1993]
- 随着深度的增加，网络的表示能力呈指数增加
  - ✓ 具有d个输入、深度为l、每个隐藏层具有n个单元的深度网络可以描述的线性区域的数量为

$$\mathrm{O}\left(\binom{n}{d}^{d(l-1)} n^{d}\right)$$

意味着，描述能力为深度的指数级[Montufar etal,2014]。

深度的影响

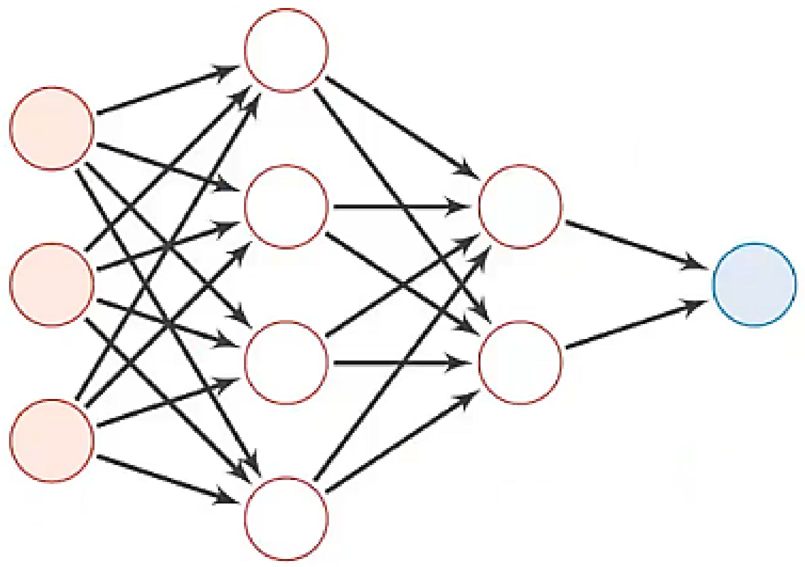
- 更深层的网络具有更好的泛化能力
  - ✓ [Goodfellow et al.,2014]手写数字识别的实验结果



模型的性能随着深度的增加而不断的提升

常见的神经网络结构

■ 前馈网络



✓ 各个神经元按照接收信息的先后分成不同得组，每一组可看做一个神经网络层

✓ 每一层中的神经元接收来自前一层神经元的输出，并输出给下一个神经元

✓ 整个网络中朝一个方向传播，没有反向的信息传播，可以用一个有向无环图表示

常见的神经网络结构

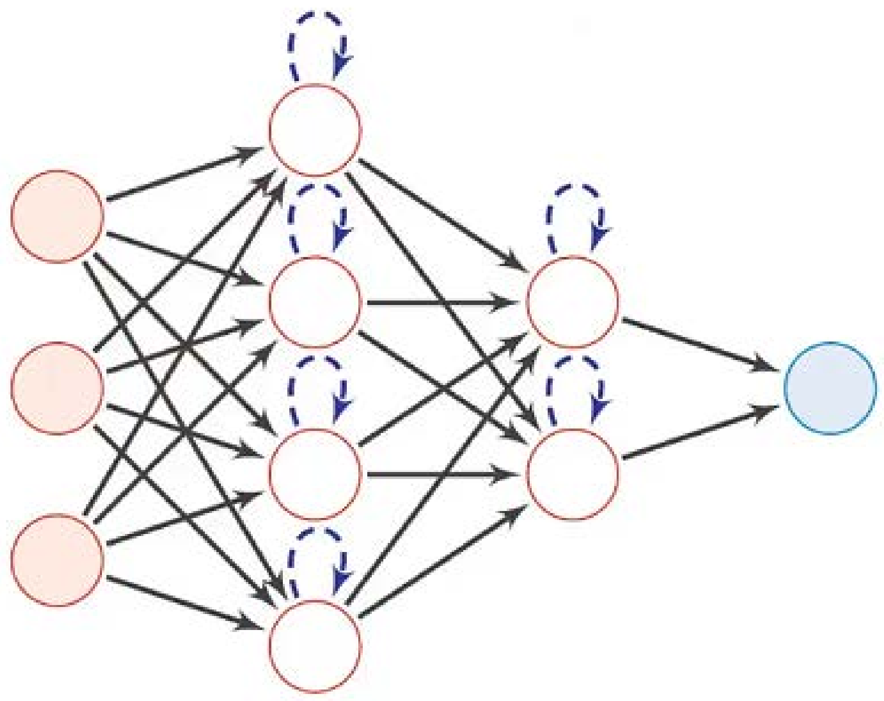
■ 记忆网络（反馈网络）

✓ 神经元不但可以接受其他神经元的信息，也可以接收自己的历史信息

✓ 神经元具有记忆功能，在不同时刻具有不同的状态

✓ 信息传播可以是单向或者双向传递，可以用一个有向循环图或者无向图来表示

✓ 记忆网络包括循环神经网络（RNN）、Hopfield网络、Boltzmann机等

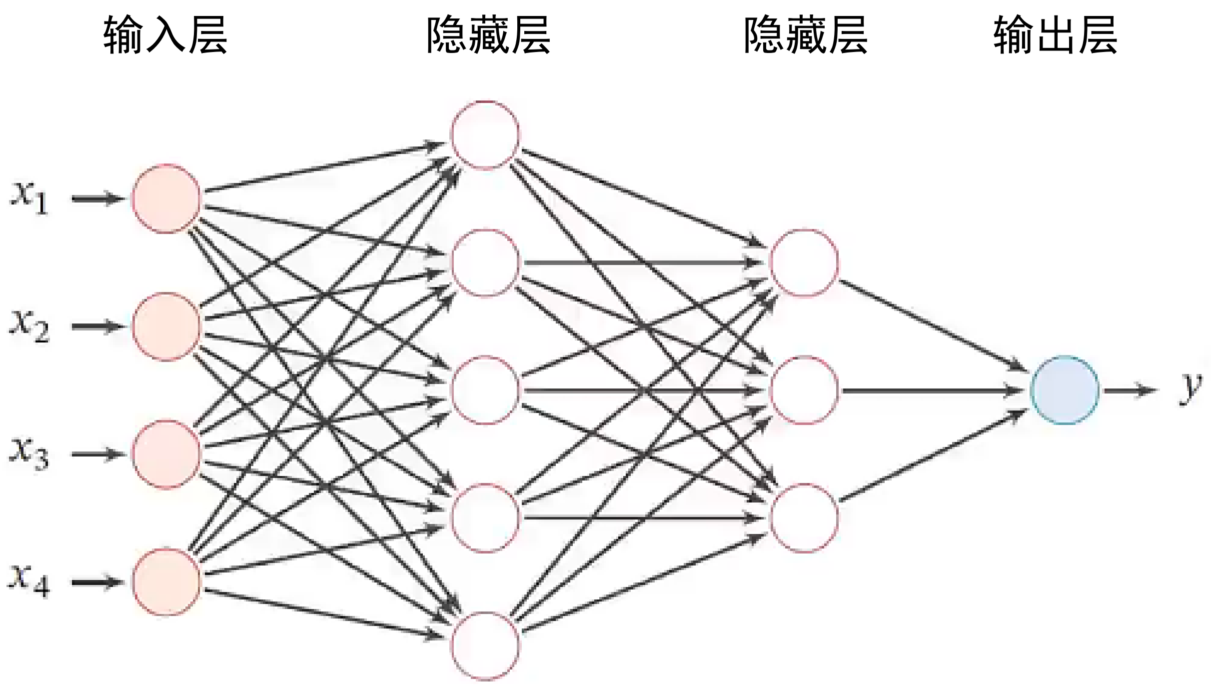


前馈神经网络

- 结构与表示
- 隐藏单元
- 输出单元
- 参数学习

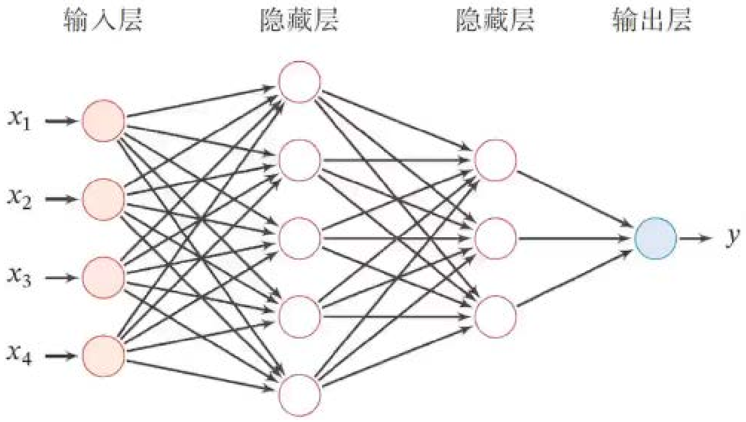
前馈神经网络的结构

- 前馈神经网络(Feedforward Neural Network, FNN)是最早发明的简单人工神经网络，也经常被称为多层感知器(MLP)（激活函数通常并不是感知机所采用的不连续阶跃函数）
- 第0层为输入层，最后一层为输出层，其他中间层称为隐藏层
- 信号从输入层向输出层单向传播，整个网络无反馈，可用一个有向无环图表示



前馈神经网络的形式化表示

■ 前馈神经网络的符号表示



记号	含义
L	神经网络的层数
M_l	第 l 层神经元的个数（$1 \le l \le L$）
$f_l(\cdot)$	第 l 层神经元的激活函数
$\boldsymbol{W}^{(l)} \in \mathbb{R}^{M_l \times M_{l-1}}$	第 $l-1$ 层到第 l 层的权重矩阵
$\boldsymbol{b}^{(l)} \in \mathbb{R}^{M_l}$	第 $l-1$ 层到第 l 层的偏置
$\boldsymbol{z}^{(l)} \in \mathbb{R}^{M_l}$	第 l 层神经元的净输入（净活性值）
$\boldsymbol{a}^{(l)} \in \mathbb{R}^{M_l}$	第 l 层神经元的输出（活性值）

■ 前馈神经网络的信息传递

✓ 令 $\boldsymbol{a}^{(0)} = \boldsymbol{x}$，信息通过以下公式不断迭代传播：

$$\boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)}$$

$$\boldsymbol{a}^{(l)} = f_l(\boldsymbol{z}^{(l)})$$

✓ 以上公式也可合并写成：

$$\boldsymbol{a}^{(l)} = f_l(\boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)})$$

✓ 如此，通过逐层传递，得到最后的输出：$\boldsymbol{a}^{(L)}$，整个网络可以看成一个复合函数 $\varphi(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{b})$

$$\begin{array}{ccc} \boldsymbol{a}^{(0)} \to \boldsymbol{z}^{(1)} \to \boldsymbol{a}^{(1)} \to \boldsymbol{z}^{(2)} \cdots \to \boldsymbol{a}^{(L-1)} \to \boldsymbol{z}^{(L)} \to & \boldsymbol{a}^{(L)} \\ \uparrow & \downarrow \\ \boldsymbol{x} \longrightarrow & \varphi(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{b}) \end{array}$$

隐藏单元——激活函数

- 隐藏单元的设计是一个非常活跃的研究领域，但是目前还没有很明确的指导原则
- 激活函数的性质要求
  - ✓ 连续并可导(允许少数点的上不可导)的非线性函数。可导的激活函数可以直接利用数值优化的方法来学习网络参数。
  - ✓ 激活函数及其导函数要尽可能的简单，有利于提高网络计算效率。
  - ✓ 激活函数的导函数的值域要在一个合适的区间内，不能太大也不能太小，否则会影响训练的效率和稳定性。

常见的激活函数

- Sigmoid函数

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

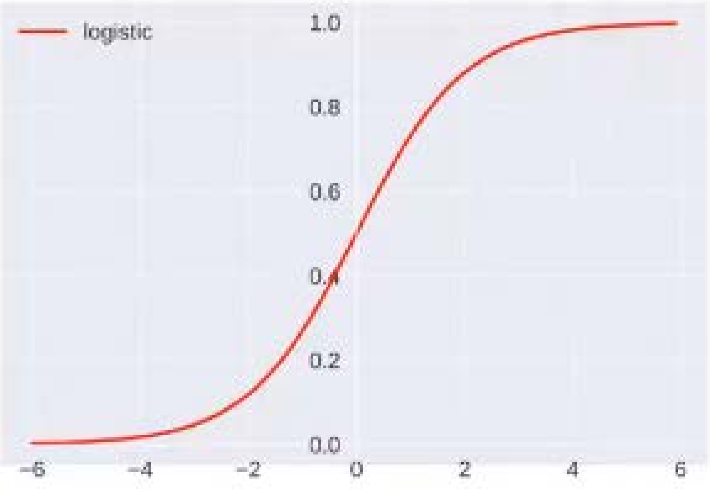
✓ 具有“挤压”功能

✓ 输出可看作概率分布

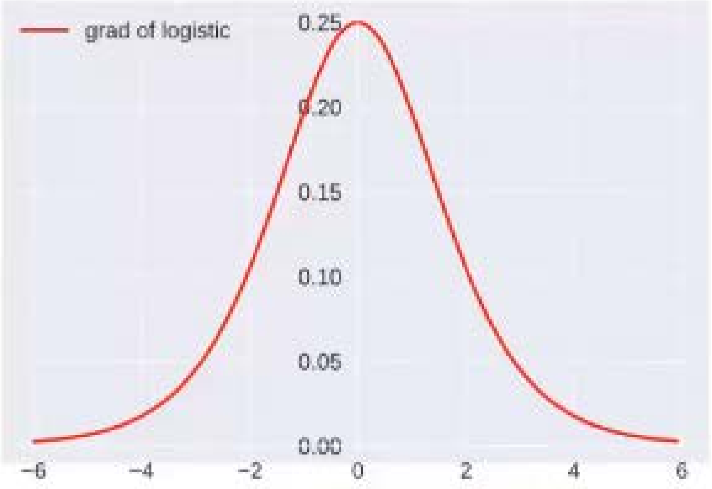
- Tanh函数

$$\tanh(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$
$$=2\sigma(2x)-1$$

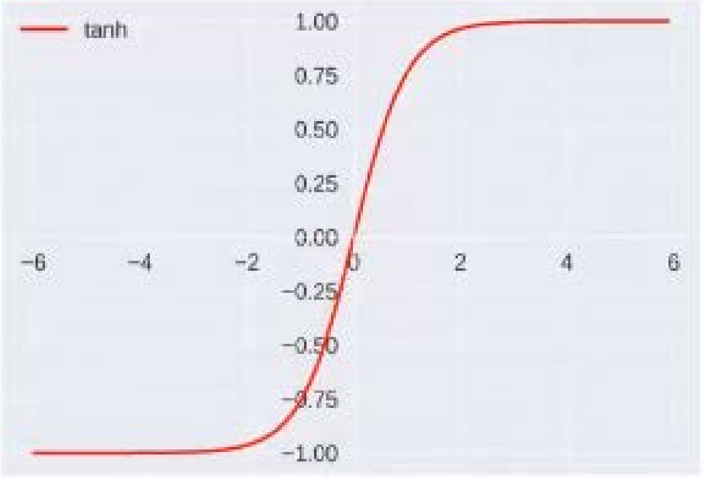
✓ 零中心化，可提升收敛速度



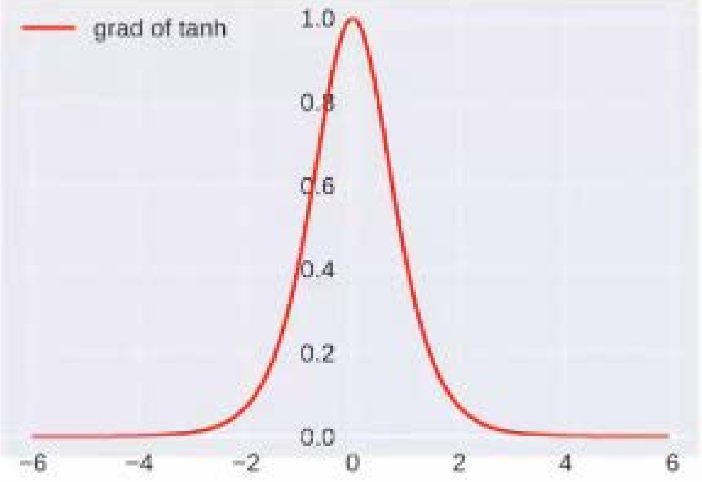
Sigmoid函数



Sigmoid函数的导数



Tanh函数



Tanh函数的导数

常见的激活函数

■ ReLu函数

$$\mathrm{ReLu}(x)=\begin{cases} x & x>=0 \\ 0 & x<0 \end{cases} = \max(0,x)$$

✓ 目前最常用的激活函数

✓ 可缓解梯度消失的问题

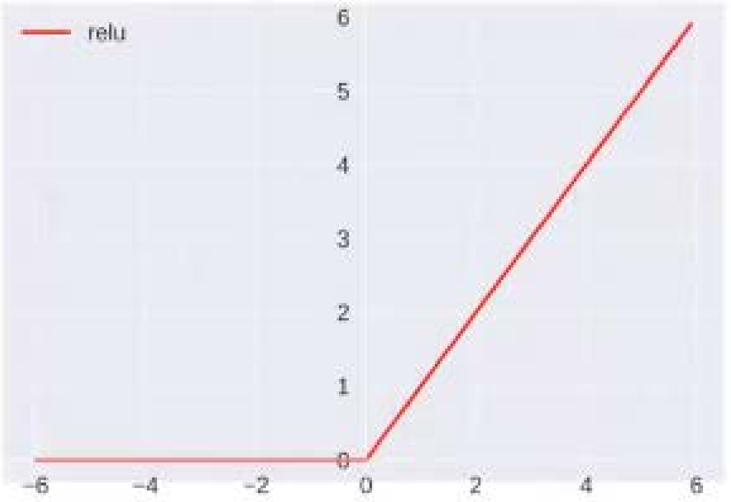
✓ 缺点：可能导致神经元的死亡

■ LeakReLu

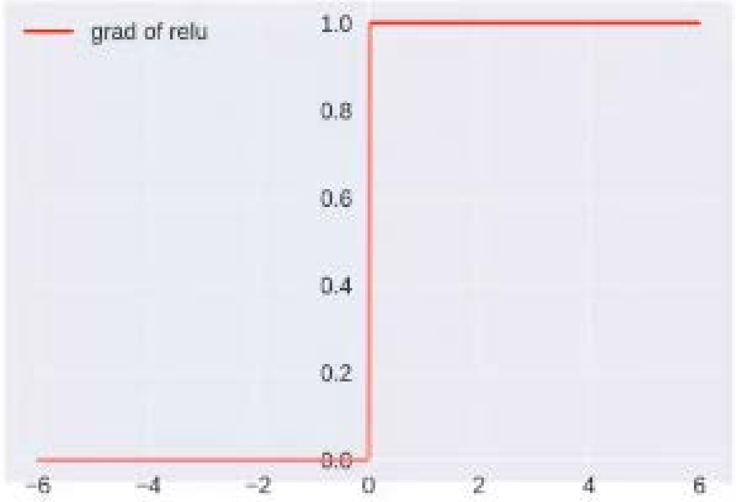
$$\mathrm{LeakReLu}(x)=\begin{cases} x & x>=0 \\ \gamma x & x<0 \end{cases}$$

✓ 在$x<0$时也保持一个很小的梯度，避免了永远不能激活的情况

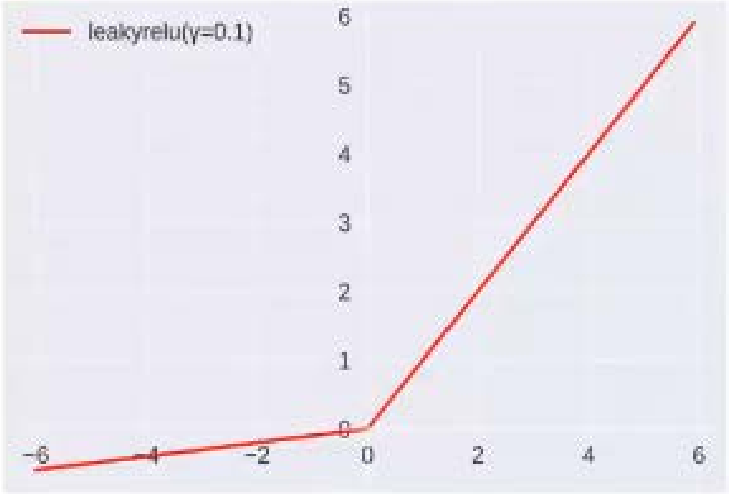
✓ γ为超参数



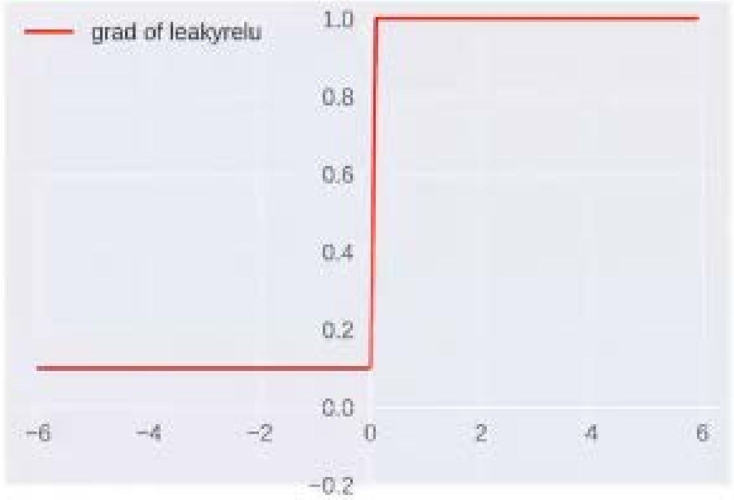
ReLu函数



ReLu函数的导数



LeakReLu函数

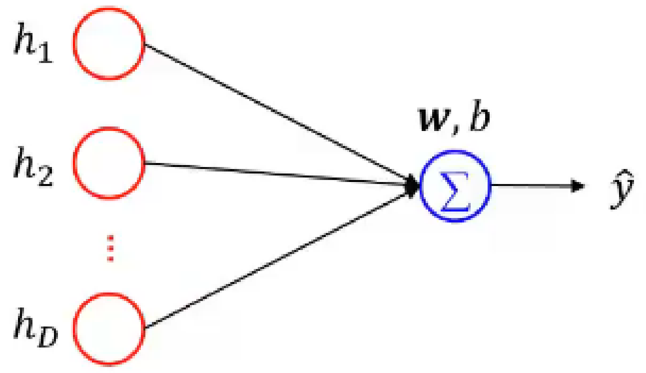


LeakReLu函数的导数

输出单元

■ 线性输出单元

$$\hat{y} = \mathbf{w}^T h + b$$



✓ 线性输出单元常用于产生条件高斯分布的均值。

✓ 适合连续值预测（回归）问题。

✓ 基于高斯分布，最大似然(最小化负对数似然)等价于最小化均方误差，因此线性输出单元可采用均方误差损失函数：

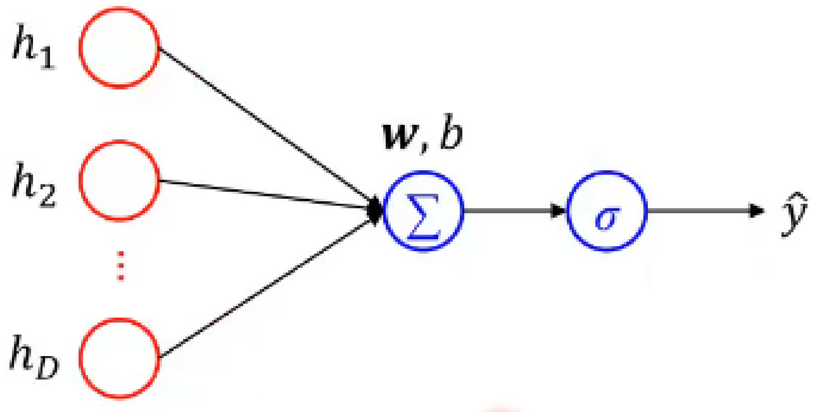
$$L(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N}\sum_{n=1}^{N} \| \hat{y}^{(n)} - y^{(n)} \|^2$$

其中$y^{(n)}$为真实值，$\hat{y}^{(n)}$为预测值，N为样本数。

输出单元

- Sigmoid单元

$$\hat{y}=\sigma(\boldsymbol{w}^T\boldsymbol{h}+b)=\frac{1}{1+\exp(-\boldsymbol{w}^T\boldsymbol{h}-b)}$$



✓ Sigmoid输出单元常用于输出Bernoulli分布

✓ 适合二分类问题

✓ Sigmoid输出单元可采用交叉熵损失函数：

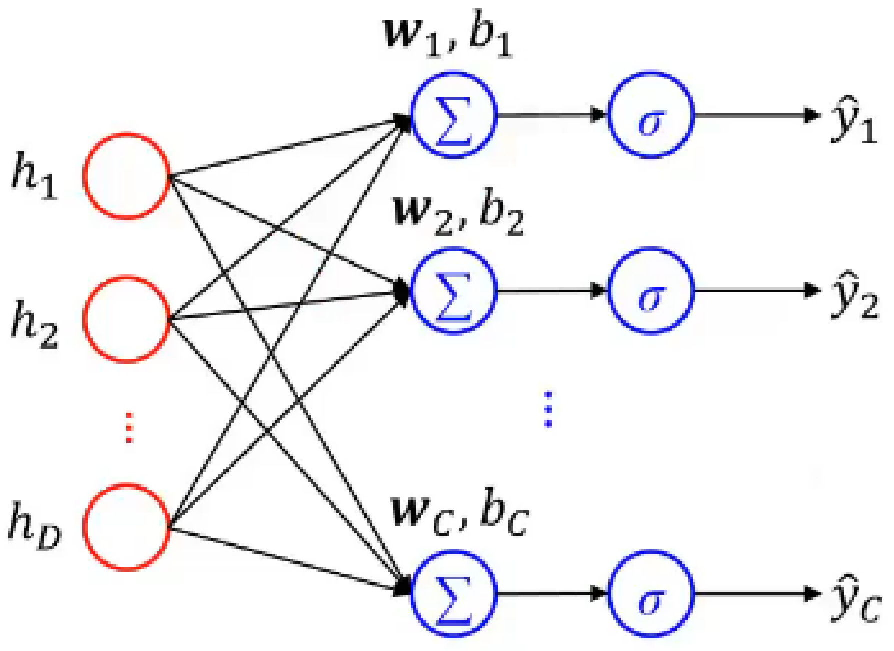
$$L(\boldsymbol{y},\hat{\boldsymbol{y}})=-\frac{1}{N}\sum_{n=1}^{N}(y^{(n)}\log\hat{y}^{(n)}-(1-y^{(n)})\log(1-\hat{y}^{(n)}))$$

其中$y^{(n)}$为真实值，$\hat{y}^{(n)}$为预测值，N为样本数。

输出单元

■ Softmax单元

$$\hat{\boldsymbol{y}}_c = \mathrm{softmax}(\boldsymbol{w}^T\boldsymbol{h}+\boldsymbol{b}) = \frac{\exp(\boldsymbol{w}_c^T\boldsymbol{h}+\boldsymbol{b}_c)}{\sum_{j=1}^{C}\exp(\boldsymbol{w}_j^T\boldsymbol{h}+\boldsymbol{b}_j)}$$



✓ Softmax输出单元常用于输出Multinoulli分布

✓ 适合多分类问题。

✓ Softmax输出单元可采用交叉熵损失函数：

$$L(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{y}^{(n)})^T \log \hat{\boldsymbol{y}}^{(n)}$$

其中$\boldsymbol{y}^{(n)} = \left[y_1^{(n)}, y_2^{(n)}, \cdots, y_C^{(n)}\right]^T$为真实标记向量值，$\hat{\boldsymbol{y}}^{(n)} = \left[\hat{y}_1^{(n)}, \hat{y}_2^{(n)}, \cdots, \hat{y}_C^{(n)}\right]^T$为预测标记概率向量，$N$为样本数，$C$为标记数。

前馈神经网络的参数学习

■ 学习准则

✓ 假设神经网络使用交叉熵损失，对于一个样本$(\boldsymbol{x}, y)$，其损失函数为:

$$L(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\boldsymbol{y}^T \log \hat{\boldsymbol{y}}$$

其中$\boldsymbol{y} \in \{0,1\}^C$为标签$y$对应的one-hot向量表示。

✓ 给定一个训练集$D = \{(\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)})\}_{n=1}^N$，每个样本的特征向量$\boldsymbol{x}^{(n)}$经过前馈神经网络的输出为$\hat{\boldsymbol{y}}^{(n)}$，模型在数据集$D$的结构化风险函数为：

$$R(\boldsymbol{W}, \boldsymbol{b}) = -\frac{1}{N}\sum_{n=1}^{N} L(\boldsymbol{y}^{(n)}, \hat{\boldsymbol{y}}^{(n)}) + \frac{1}{2}\lambda \|\boldsymbol{W}\|_F^2$$

其中W和b表示网络的全部参数，λ为超参数，正则化项为矩阵Frobenius范数的平方

$$\|\boldsymbol{W}\|_F^2 = \sum_{l=1}^{L}\sum_{i=1}^{M_l}\sum_{j=1}^{M_{l-1}} (w_{ij}^{(l)})^2$$

前馈神经网络的参数学习

- 梯度下降
  - ✓ 基于学习准则和训练样本，网络参数可以通过梯度下降法进行学习，在每次迭代中第l层的参数$\boldsymbol{W}^{(l)}$和$\boldsymbol{b}^{(l)}$更新方式为：

$$\boldsymbol{W}^{(l)} \leftarrow \boldsymbol{W}^{(l)} - \alpha \frac{\partial R(\boldsymbol{W}, \boldsymbol{b})}{\partial \boldsymbol{W}^{(l)}}$$

$$\boldsymbol{b}^{(l)} \leftarrow \boldsymbol{b}^{(l)} - \alpha \frac{\partial R(\boldsymbol{W}, \boldsymbol{b})}{\partial \boldsymbol{b}^{(l)}}$$

其中α为学习率(learning rate)。

  - ✓ 通过链式法则可以逐一对每个参数求偏导，但是效率低下
  - ✓ 在神经网络的训练中经常使用反向传播算法来高效计算梯度

反向传播算法

反向传播算法

给定一个样本$(\boldsymbol{x}, y)$，假设神经网络的输出为$\hat{y}$，损失函数为$L(y, \hat{y})$，采用梯度下降法计算损失函数关于每个参数的偏导数。

- 如何计算前馈神经网络中参数的偏导数——反向传播(Back Propagation, BP)算法

  考虑求第l层中参数$\boldsymbol{W}^{(l)}$和$\boldsymbol{b}^{(l)}$的偏导数，由于$\boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)}+\boldsymbol{b}^{(l)}$，根据链式法则：

$$\frac{\partial L(y,\hat{y})}{\partial w_{ij}^{(l)}} = \frac{\partial \boldsymbol{z}^{(l)}}{\partial w_{ij}^{(l)}} \frac{\partial L(y,\hat{y})}{\partial \boldsymbol{z}^{(l)}}$$

$$\frac{\partial L(y,\hat{y})}{\partial \boldsymbol{b}^{(l)}} = \frac{\partial \boldsymbol{z}^{(l)}}{\partial \boldsymbol{b}^{(l)}} \frac{\partial L(y,\hat{y})}{\partial \boldsymbol{z}^{(l)}}$$

表达式一样，只需计算一次

令$\boldsymbol{\delta}^{(l)} \triangleq \frac{\partial L(y,\hat{y})}{\partial \boldsymbol{z}^{(l)}}$为第$l$层的误差项

反向传播算法

① 求$\frac{\partial \boldsymbol{z}^{(l)}}{\partial w_{ij}^{(l)}}$，由$\boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)}+\boldsymbol{b}^{(l)}$：

$$\frac{\partial \boldsymbol{z}^{(l)}}{\partial w_{ij}^{(l)}} = \left[\frac{\partial z_1^{(l)}}{\partial w_{ij}^{(l)}}, \cdots, \frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l)}}, \cdots, \frac{\partial z_{M_l}^{(l)}}{\partial w_{ij}^{(l)}}\right]$$

$$= \left[0, \cdots, a_i^{(l-1)}, \cdots, 0\right] \in \mathbb{R}^{1\times M_l}$$

② 求$\frac{\partial \boldsymbol{z}^{(l)}}{\partial \boldsymbol{b}^{(l)}}$，由$\boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)}+\boldsymbol{b}^{(l)}$：

$$\frac{\partial \boldsymbol{z}^{(l)}}{\partial \boldsymbol{b}^{(l)}} = \boldsymbol{I}_{M_l} \in \mathbb{R}^{M_l\times M_l}$$

为$M_l \times M_l$的单位矩阵。

反向传播算法

③ 求$\boldsymbol{\delta}^{(l)} \triangleq \frac{\partial L(y,\hat{y})}{\partial \boldsymbol{z}^{(l)}}$，由$\boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)}+\boldsymbol{b}^{(l)}, \boldsymbol{a}^{(l)} = f_l\left(\boldsymbol{z}^{(l)}\right)$，根据链式法则:

$$\frac{\partial L(y,\hat{y})}{\partial \mathbf{z}^{(l)}} = \frac{\partial \boldsymbol{a}^{(l)}}{\partial \mathbf{z}^{(l)}} \frac{\partial \mathbf{z}^{(l+1)}}{\partial \boldsymbol{a}^{(l)}} \frac{\partial L(y,\hat{y})}{\partial \mathbf{z}^{(l+1)}}$$

$= \boldsymbol{\delta}^{(l+1)}$

其中 $\frac{\partial \boldsymbol{a}^{(l)}}{\partial \mathbf{z}^{(l)}} = \frac{\partial f_l(\mathbf{z}^{(l)})}{\partial \mathbf{z}^{(l)}} = diag(f_l(\mathbf{z}^{(l)})) \in \mathbb{R}^{M_l \times M_l}$ $\frac{\partial \mathbf{z}^{(l+1)}}{\partial \boldsymbol{a}^{(l)}} = \left(\boldsymbol{W}^{(l+1)}\right)^T \in \mathbb{R}^{M_l \times M_{l+1}}$

第l层的误差项是第$l+1$层的误差项的加权和，然后再乘以该层的激活函数的梯度，这就是误差的反向传播。

所以 $\boldsymbol{\delta}^{(l)} \triangleq \frac{\partial L(y,\hat{y})}{\partial \mathbf{z}^{(l)}} = f_l^{'}\left(\mathbf{z}^{(l)}\right) \odot \left(\boldsymbol{W}^{(l+1)}\right)^T \boldsymbol{\delta}^{(l+1)} \in \mathbb{R}^{M_l}$

其中⊙是点积，表示每个元素相乘。

反向传播算法

计算出上面的三个偏导数之后。可得到第l层的梯度：

$$\frac{\partial L(y,\hat{y})}{\partial w_{ij}^{(l)}}=\frac{\partial \mathbf{z}^{(l)}}{\partial w_{ij}^{(l)}}\frac{\partial L(y,\hat{y})}{\partial \mathbf{z}^{(l)}}=\boldsymbol{\delta}_i^{(l)}\boldsymbol{a}_j^{(l-1)}$$

相当于向量$\boldsymbol{\delta}^{(l)}$和向量$\boldsymbol{a}^{(l-1)}$的外积的第$i,j$个元素，即：

$$\left[\frac{\partial L(y,\hat{y})}{\partial \boldsymbol{W}^{(l)}}\right]_{ij}=\left[\boldsymbol{\delta}^{(l)}\left(\boldsymbol{a}^{(l-1)}\right)^T\right]_{ij}$$

因此，$L(y,\hat{y})$关于第l层权重$\boldsymbol{W}^{(l)}$的梯度为：

$$\frac{\partial L(y,\hat{y})}{\partial \boldsymbol{W}^{(l)}}=\boldsymbol{\delta}^{(l)}\left(\boldsymbol{a}^{(l-1)}\right)^T\in\mathbb{R}^{M_l\times M_{l-1}}$$

同理可得 $L(y,\hat{y})$关于第l层偏置$\boldsymbol{b}^{(l)}$的梯度为：

$$\frac{\partial L(y,\hat{y})}{\partial \boldsymbol{b}^{(l)}}=\boldsymbol{\delta}^{(l)}\in\mathbb{R}^{M_l}$$

反向传播算法

使用反向传播算法的前馈神经网络随机梯度下降训练过程

输入：训练集$D=\{(\boldsymbol{x}^{(n)},y^{(n)})\}_{n=1}^{N}$，验证集$V$，学习率$\alpha$，正则化系数$\lambda$，网络层数$L$，神经元数量$\{M_l\}_{l=1}^{L}$.

输出：$\boldsymbol{W},\boldsymbol{b}$

```
随机初始化W, b;
repeat
    对训练集D中的样本随机重排序;
    For n = 1 ... N do
        从训练集D中选取样本(x^(n), y^(n));
        前馈计算每一层的净输入z^(l)和激活值a^(l)，直到最后一层;
        反向传播计算每一层的误差δ^(l) = f_l'(z^(l)) ⊙ ((W^(l+1))^T δ^(l+1));  //最后一层的误差为δ^(L) = f_L'(z^(L)) ⊙ ∂L(y,ŷ)/∂ŷ
        //计算每一层的梯度
        ∂L(y,ŷ)/∂W^(l) = δ^(l) (a^(l-1))^T;
        ∂L(y,ŷ)/∂b^(l) = δ^(l);
        //更新参数
        W^(l) ←= W^(l) − α (δ^(l) (a^(l-1))^T + λW^(l));
        b^(l) = b^(l) − αδ^(l);
    end;
until 模型在验证集V上的错误率不再下降;
```

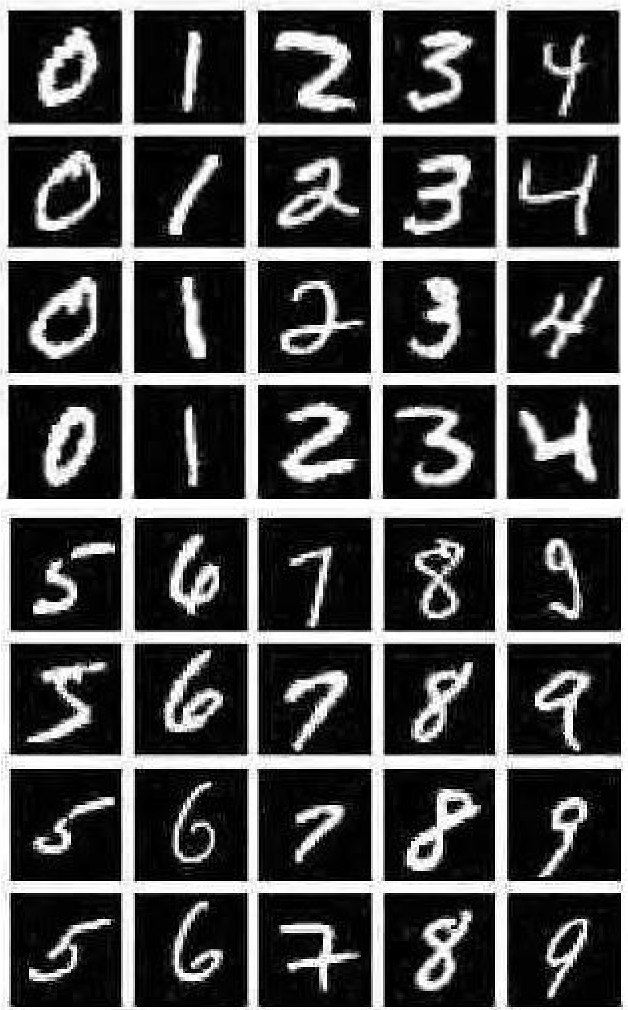
Where:

$$\boldsymbol{\delta}^{(l)}=f_l'(\boldsymbol{z}^{(l)})\odot\left(\left(\boldsymbol{W}^{(l+1)}\right)^{\mathrm{T}}\boldsymbol{\delta}^{(l+1)}\right);\quad \boldsymbol{\delta}^{(L)}=f_L'(\boldsymbol{z}^{(L)})\odot\frac{\partial L(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial\hat{\boldsymbol{y}}}$$

$$\frac{\partial L(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial\boldsymbol{W}^{(l)}}=\boldsymbol{\delta}^{(l)}\left(\boldsymbol{a}^{(l-1)}\right)^{\mathrm{T}};\quad \frac{\partial L(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial\boldsymbol{b}^{(l)}}=\boldsymbol{\delta}^{(l)};$$

$$\boldsymbol{W}^{(l)}\leftarrow=\boldsymbol{W}^{(l)}-\alpha\left(\boldsymbol{\delta}^{(l)}\left(\boldsymbol{a}^{(l-1)}\right)^{\mathrm{T}}+\lambda\boldsymbol{W}^{(l)}\right);\quad \boldsymbol{b}^{(l)}=\boldsymbol{b}^{(l)}-\alpha\boldsymbol{\delta}^{(l)};$$

作业三：基于ANN的MNIST图像分类



- 推荐编程环境：Anaconda+Jupyter notebook+PyTorch
- 搭建不同层数的人工神经网络并查看模型性能变化
- 可以使用Scikit-learn、PyTorch、TensorFlow等不同的库进行实现对比。

Sources

- https://www.youtube.com/watch?v=bfmFfD2RIcg
  Neural Network In 5 Minutes | What Is A Neural Network? 5mins

- https://www.youtube.com/watch?v=CqOfi41LfDw
  Neural Networks Part. 1: Inside the Black Box 19mins

- https://www.youtube.com/watch?v=oJNHXPs0XDk
  Neural Network Architectures & Deep Learning 9mins

- https://www.youtube.com/watch?v=3JQ3hYko51Y
  Neural Network 3D Simulation   3mins

- https://www.youtube.com/watch?v=f0t-OCG79-U
  Convolutional Neural Network Visualization 2mins