教育部产学合作协同育人项目资助
2022年北京交通大学《深度学习》课程

第四讲 深度前馈网络

主讲教师：丛润民

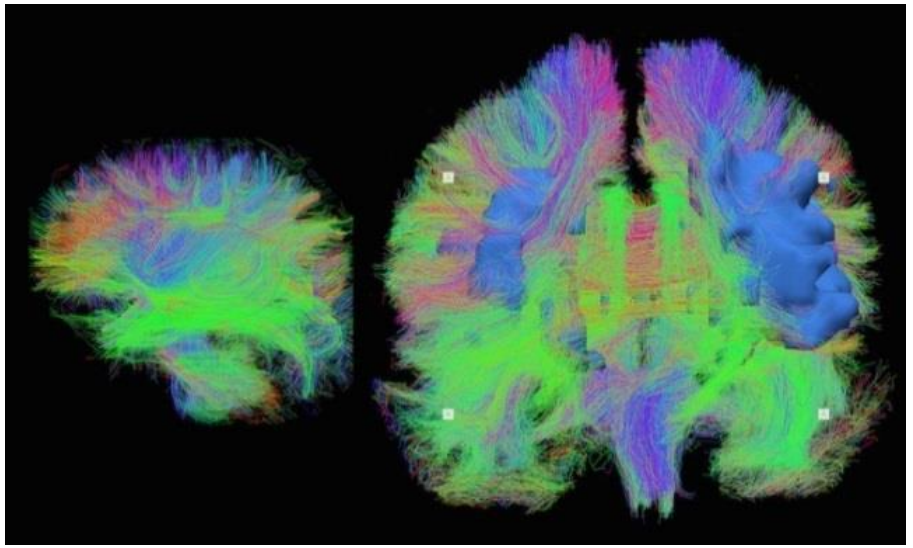
目录

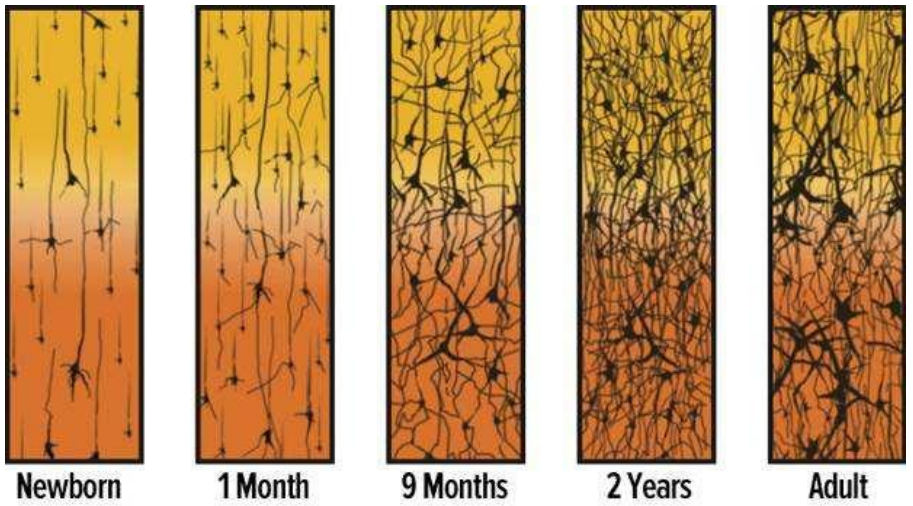
4.1 人工神经网络

- 人脑神经网络
- 人工神经元模型
- 人工神经网络

人脑神经网络

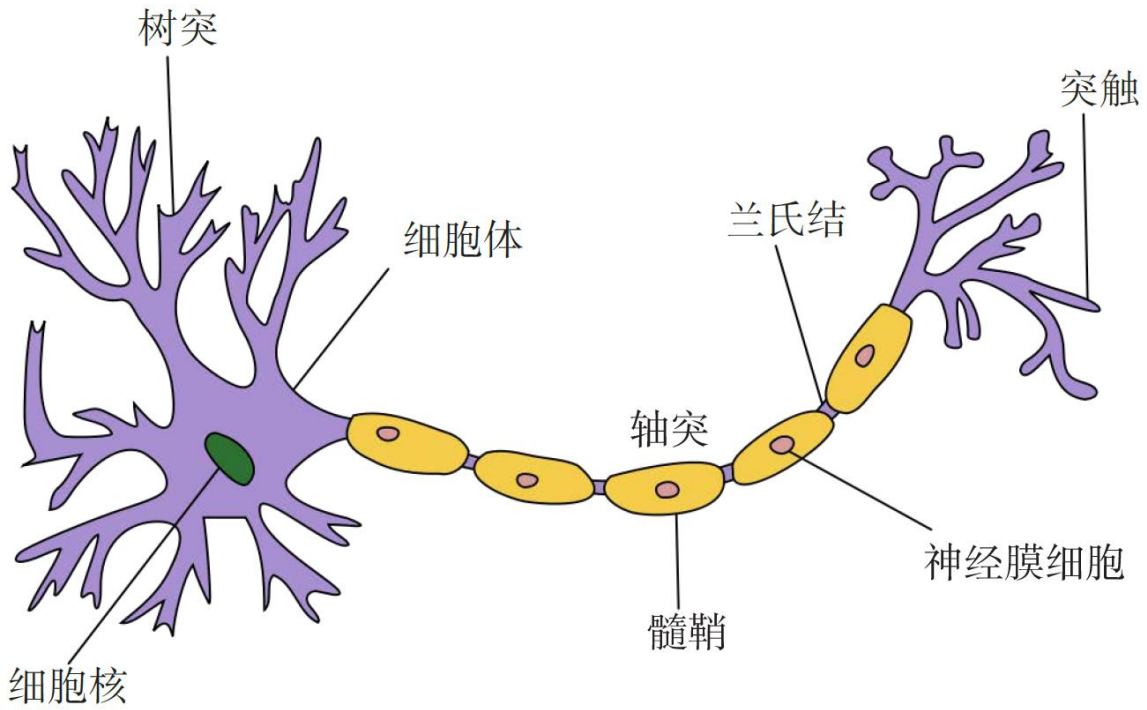
- 人类大脑由**神经元**、**神经胶质细胞**、**神经干细胞**和**血管**组成
- **神经元(neuron)**也叫神经细胞(nerve cell)，是人脑神经系统中最基本的单元
- 人脑神经系统包含近860亿个神经元
- 每个神经元有上千个突触与其他神经元相连
- 人脑神经元连接成巨大的复杂网络，总长度可达数千公里





人脑神经网络的发育过程

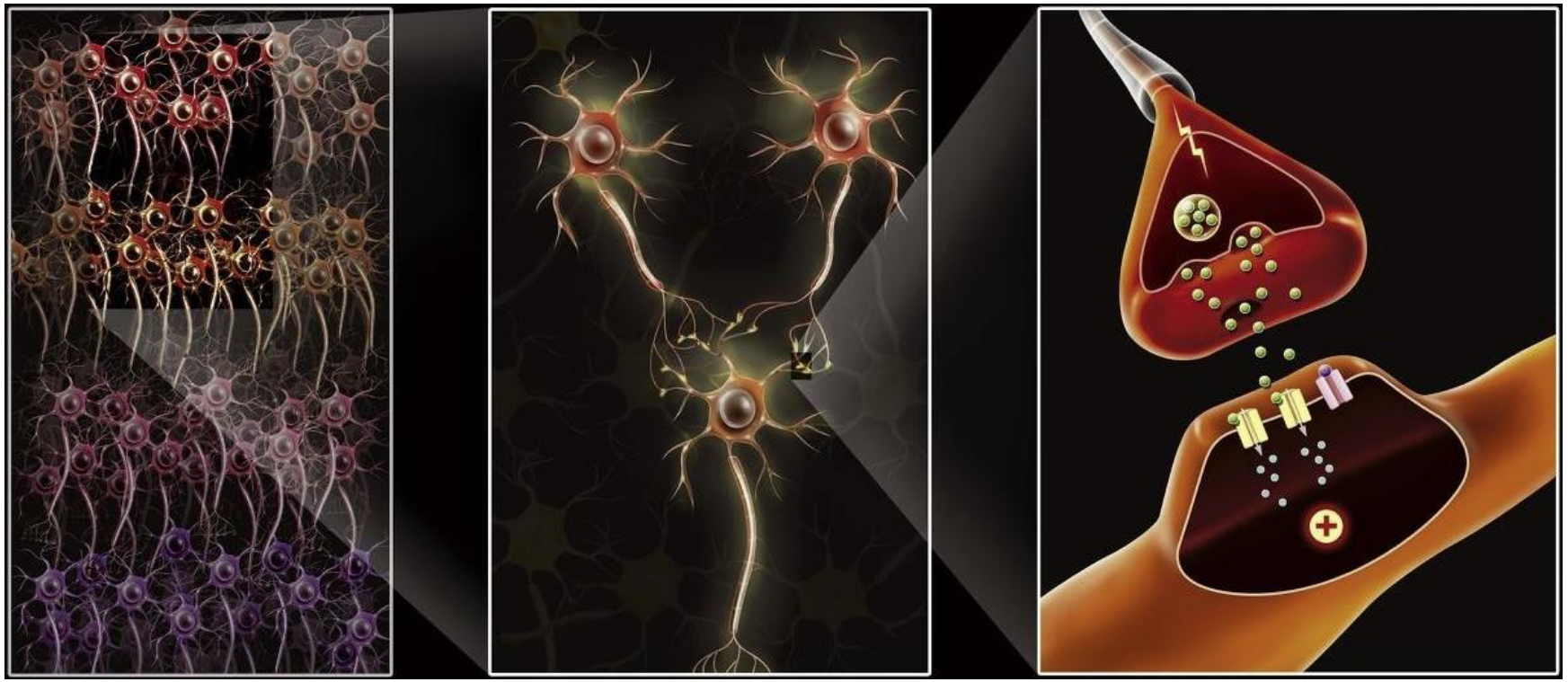
神经元结构



- **细胞体**：通过生物化学反应，引起细胞膜内外电位差发生改变，形成**兴奋**或**抑制**状态
- **细胞突起**：由细胞体延伸出来，又可分为树突和轴突：
  - ✓ **树突**：可接收刺激并将兴奋传入细胞体，每个神经元可以有一个或多个树突
  - ✓ **轴突**：可把自身兴奋状态从细胞体传给另一个神经元，每个神经元只有一个轴突

神经元之间的信息传递

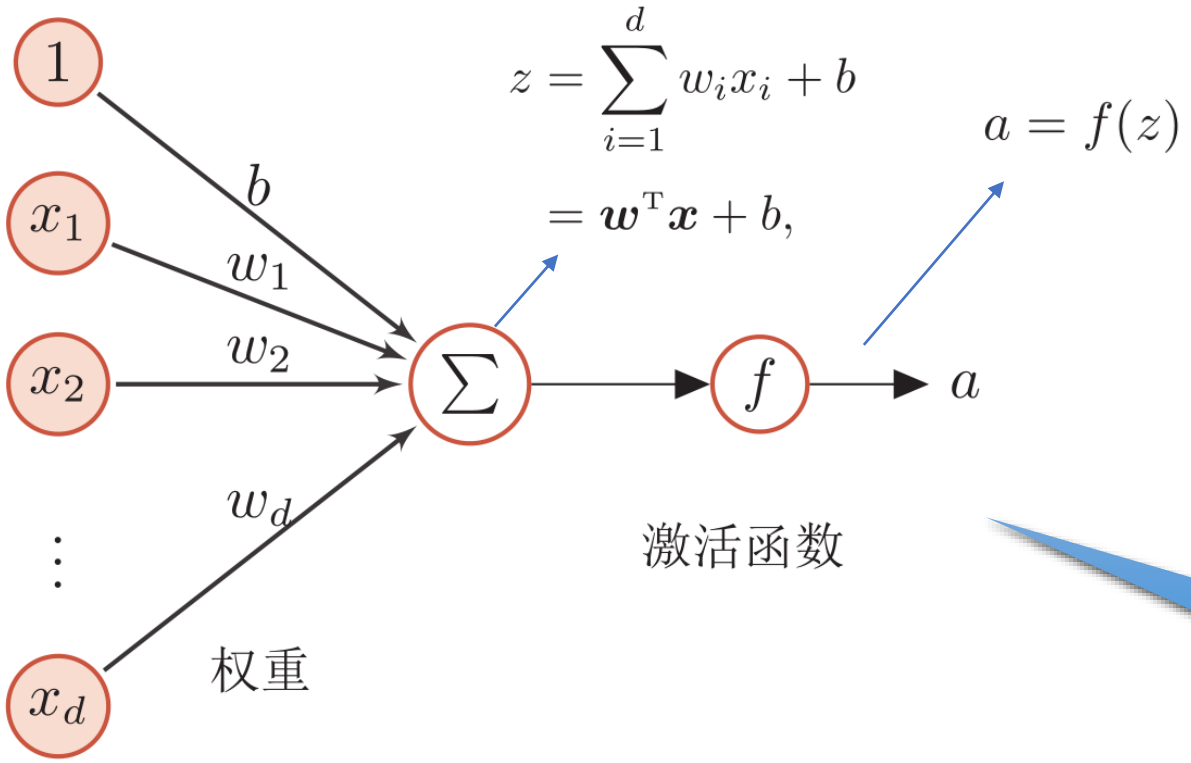
- 每个神经元与其他神经元相连，当它“兴奋”时，就会向相连的神经元发送化学物质，从而改变这些神经元内的电位；
- 如果神经元的电位超过一定“阈值”，它就会被激活，即“兴奋”起来，然后向其他神经元发送化学物质。



人工神经元

- M-P神经元模型（McCulloch and Pitts，1943）

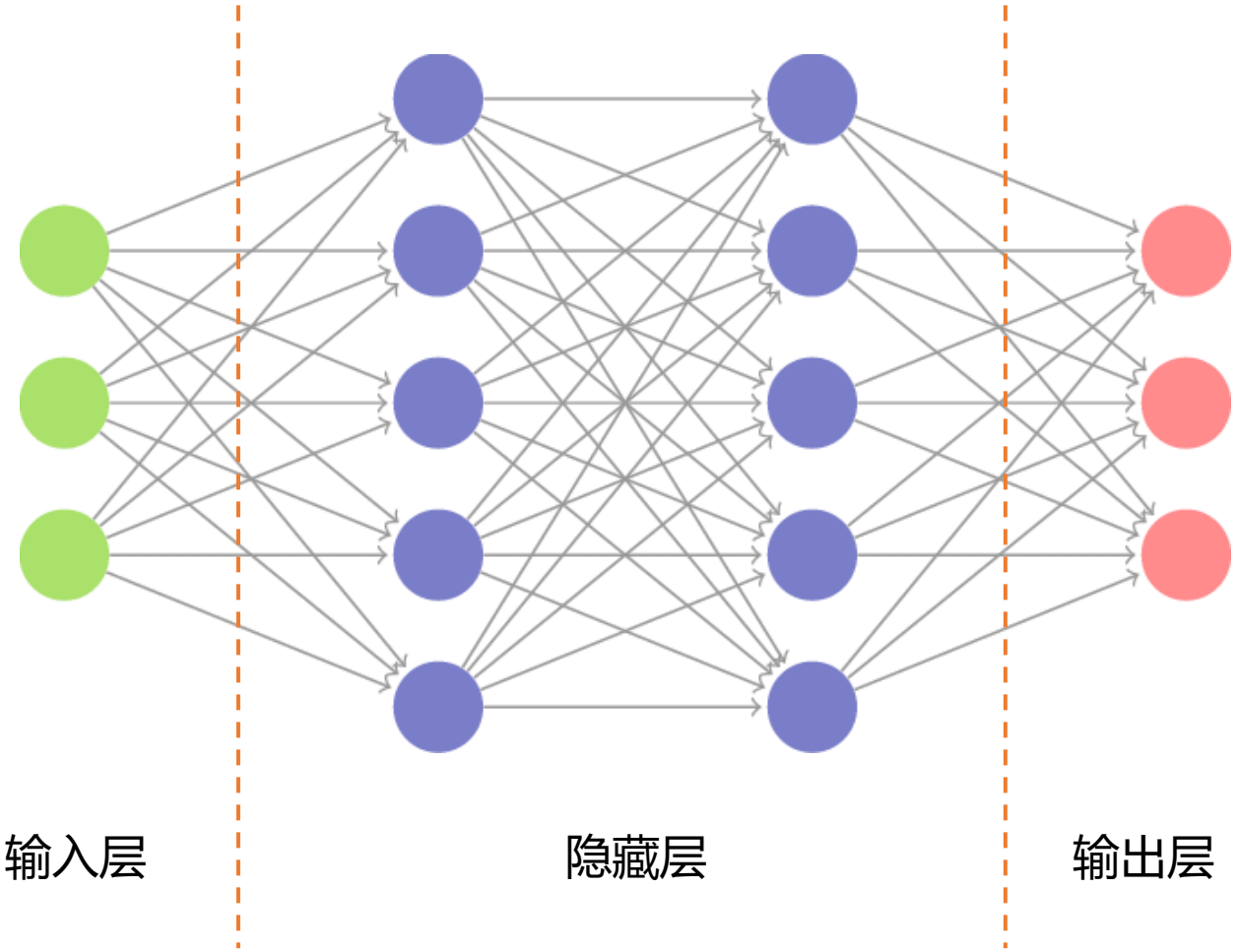
神经元接收到来自其他d个神经元传递过来的输入信号，这些输入信号通过带权重的连接进行传递，神经元接收到的总输入值将与神经元的阈值（bias）进行比较，然后通过“激活函数”处理产生神经元的输出。



一个简单的线性模型!

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**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?

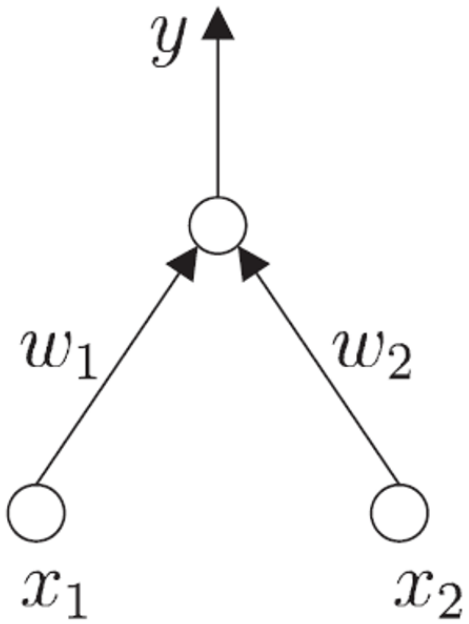


4.2 一个解决异或问题的简单网络

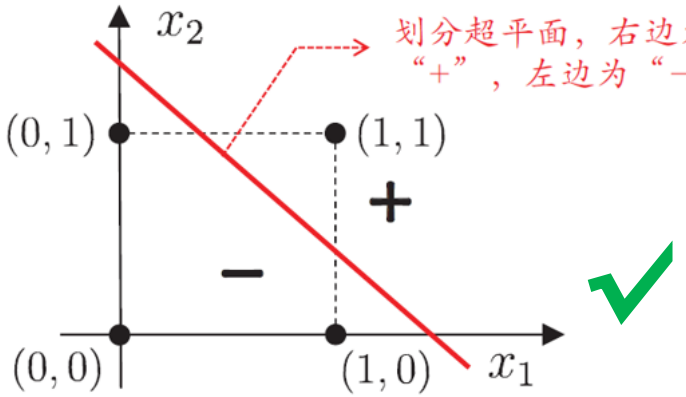
- 感知器回顾
- 双层感知器解决异或问题

感知器求解异、或、非及异或问题

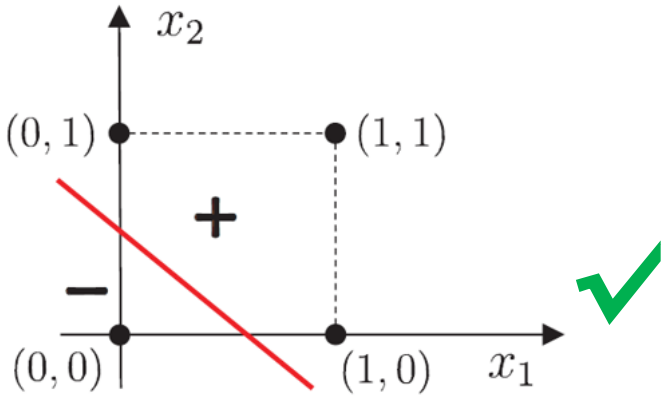
- 输入为$[x_1; x_2]$的单层单个神经元（输入层不计入层数），采用阶跃激活函数。



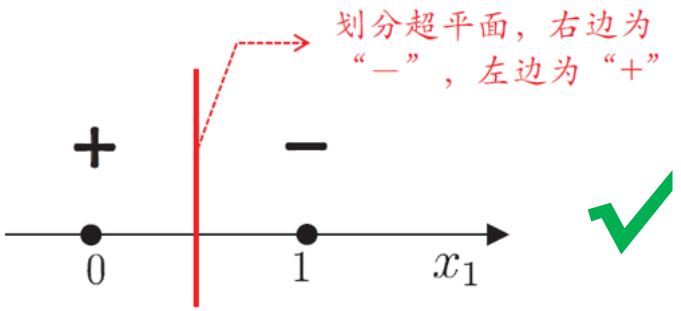
感知器



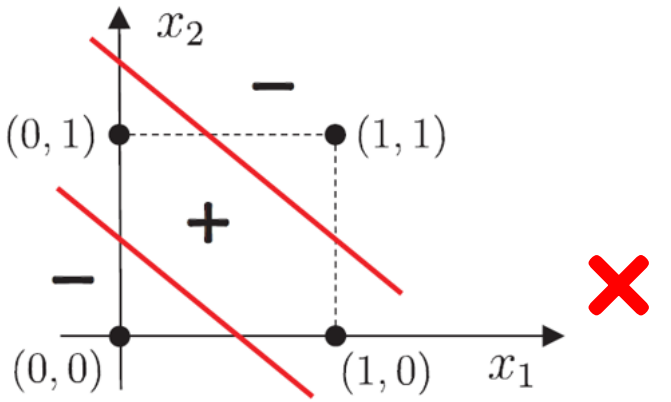
(a) “与”问题 $(x_1 \wedge x_2)$



(b) “或”问题 $(x_1 \vee x_2)$



(c) “非”问题 $(\neg x_1)$



(d) “异或”问题 $(x_1 \oplus x_2)$

双层感知器 —— 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：

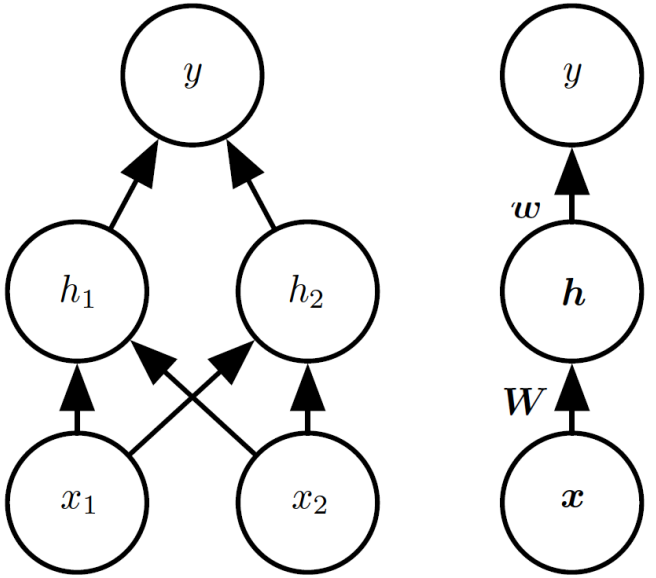
    $$\boldsymbol{h} = f^{(1)}(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c})$$

  - ✓ **输出层**包含一个神经元：

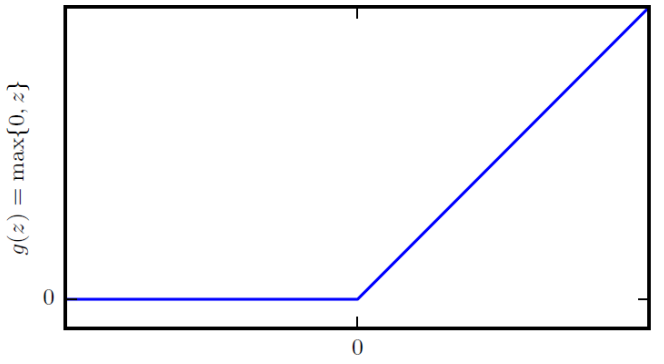
    $$y = f^{(2)}(\boldsymbol{h}; \boldsymbol{w}, b)$$

  - ✓ 隐藏层采用线性整流激活函数(ReLU)，则整个模型为：

    $$f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = f^{(2)}(f^{(1)}(\boldsymbol{x}))$$
    $$= \boldsymbol{w}^{\mathrm{T}} \max\{0, \boldsymbol{W}^{\mathrm{T}}\boldsymbol{x} + \boldsymbol{c}\} + b$$



双层感知器



ReLU函数 $g(z) = \max\{0, z\}$

双层感知器 —— 一个简单的神经网络

- 给出异或问题的一个解：

$$\boldsymbol{W}=\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}, \boldsymbol{c}=\begin{bmatrix}0\\ -1\end{bmatrix}, \boldsymbol{w}=\begin{bmatrix}0\\ -2\end{bmatrix}, b=0$$

模型处理流程如下：

① 输入4个样本的矩阵表示为：

$$\boldsymbol{X}=\begin{bmatrix}0 & 0\\ 0 & 1\\ 1 & 0\\ 1 & 1\end{bmatrix}$$

② 乘以第一层权重矩阵，得到：

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

③ 加上偏置向量$\boldsymbol{c}$，得到：

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

在这个空间中，所有样本都处在一条斜率为1的直线上。

④ 使用整流线性变换，得到：

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

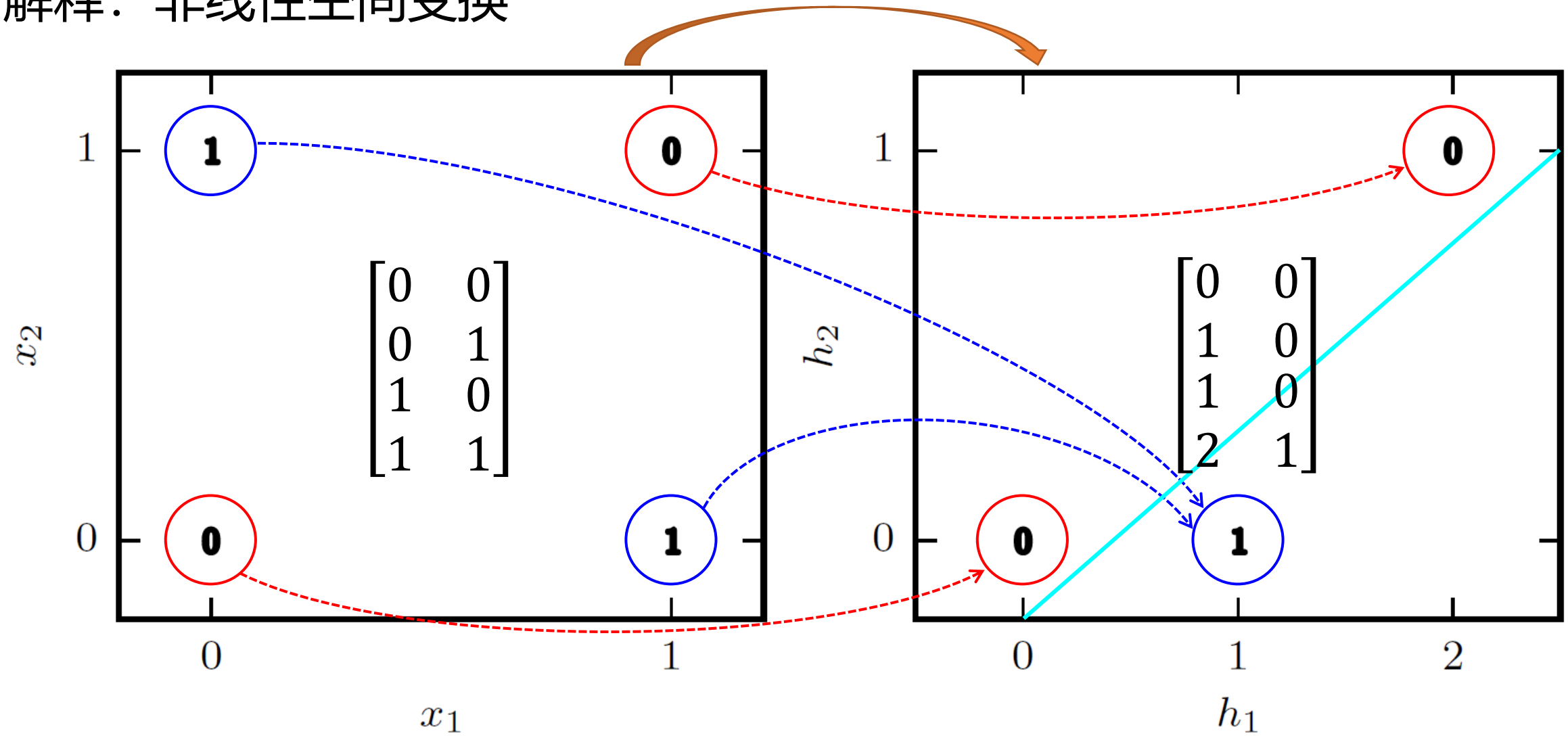
在这个空间中，所有样本不再处在同一条直线上了。

⑤ 乘以第二层权重向量$\boldsymbol{w}$，得到：

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

双层感知器 —— 一个简单的神经网络

- 解释：非线性空间变换



原始的x空间　　学习得到的隐藏h空间

4.3 神经网络结构

- 万能近似定理
- 为什么要深度
- 常见神经网络结构
- 结构设计的其他考虑

万能近似定理

■ 万能近似定理（Universal Approximation Theorem）

令$\varphi(\cdot)$是一个非常数、有界、单调递增的连续函数，I_d是一个d维的单位超立方体$[0,1]^d$，$C(I_d)$是定义在I_d上的连续函数集合。对于任一函数$f \in C(I_d)$，存在一个整数m和一组实数$v_i, b_i \in \mathbb{R}$以及实数向量$\boldsymbol{w}_i \in \mathbb{R}^d, i = 1, \ldots, m$，以至于我们可以定义函数

$$F(\boldsymbol{x}) = \sum_{i=1}^{m} v_i \varphi(\boldsymbol{w}_i^{\mathrm{T}} \boldsymbol{x} + b_i)$$

作为函数f的近似实现，即

$$|F(\boldsymbol{x}) - f(\boldsymbol{x})| < \epsilon, \forall \boldsymbol{x} \in I_d$$

其中$\epsilon > 0$是一个很小的正数。

万能近似定理应用到神经网络

- 根据万能近似定理，对于具有线性输出层和至少一个使用“挤压”性质的激活函数的隐藏层组成的神经网络，只要其隐藏层神经元的数量足够多，它就可以以任意精度来近似任何一个定义在实数空间中的有界闭集函数。

- **神经网络**可以作为一个**“万能”函数**来使用，可以用来进行复杂的特征转换，或逼近一个复杂的条件分布。

$$\hat{y} = g(\varphi(\boldsymbol{x}), \theta)$$

分类器　　神经网络

如果$g(\cdot)$为Logistic或Softmax分类器，那么$g(\cdot)$也可以视为神经网络的最后一层。

为什么要深度

- **单隐层网络**可以近似任何函数，但其**规模可能巨大**
  - ✓ 在最坏的情况下，需要**指数级的隐藏单元**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个单元的深度整流网络可以描述的线性区域的数量为

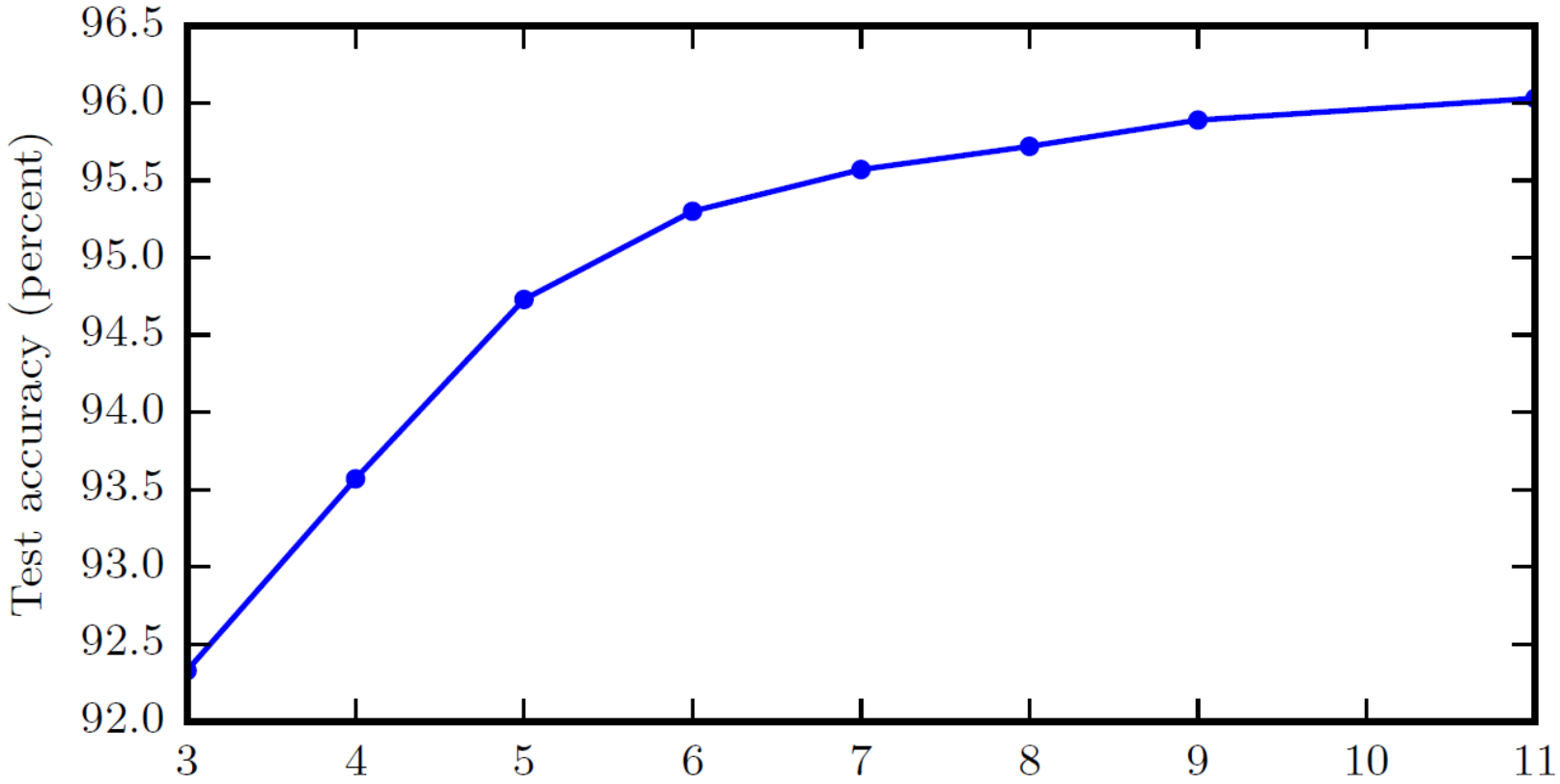
$$O\left(\binom{n}{d}^{d(l-1)} n^d\right)$$

意味着，描述能力为深度的**指数级**[Montufar et al, 2014]。

深度的影响

■ 更深层的网络具有更好的泛化能力

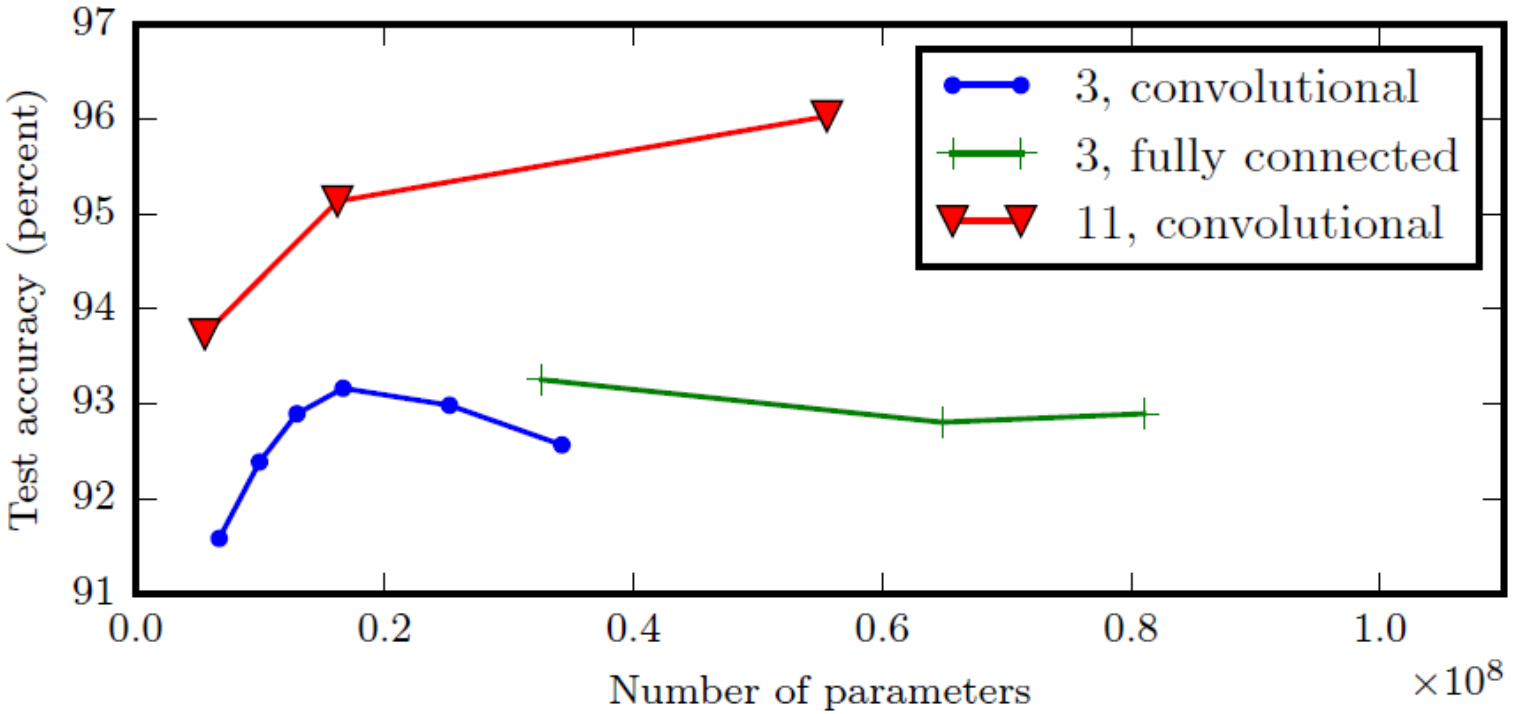
✓ [Goodfellow et al., 2014]手写体数字识别的实验结果



模型的性能随着随着深度的增加而不断提升。

深度与参数数量的关系

- **参数数量的增加未必一定会带来模型效果的提升**
  - ✓ [Goodfellow et al., 2014]手写体数字识别的实验结果



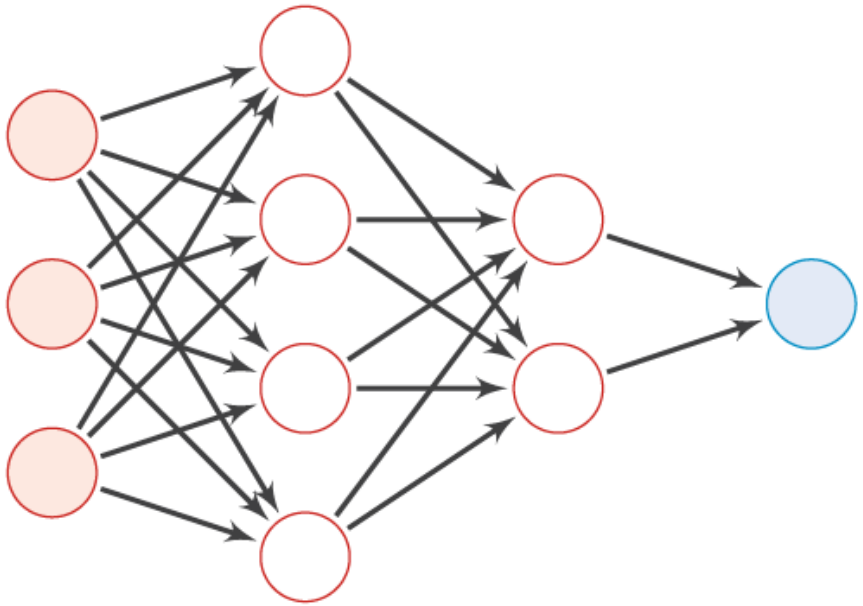
更深的模型往往表现更好，不仅仅是因为模型更大。

想要学得的函数应该由许多更简单的函数复合在一起而得到。

常见的神经网络结构

前馈网络

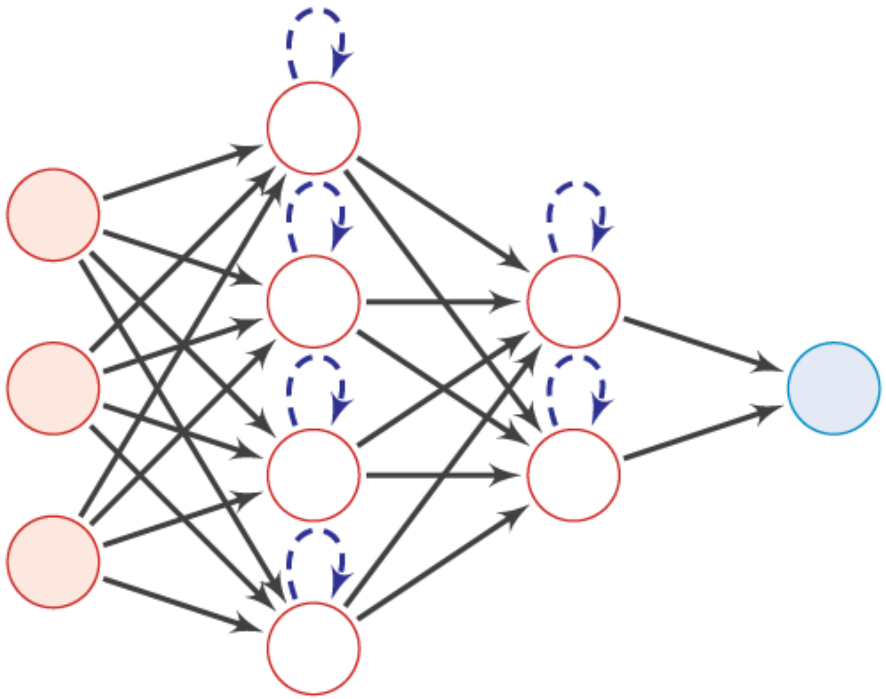
- ✓ 各个神经元按照接收信息的先后分成不同的组，每一组可看作一个神经层
- ✓ 每一层中的神经元接收来自前一层神经元的输出，并输出给下一层神经元
- ✓ 整个网络中信息朝一个方向传播，没有反向的信息传播，可以用一个有向无环图表示
- ✓ 前馈网络包括全连接前馈神经网络和卷积神经网络



常见的神经网络结构

记忆网络（反馈网络）

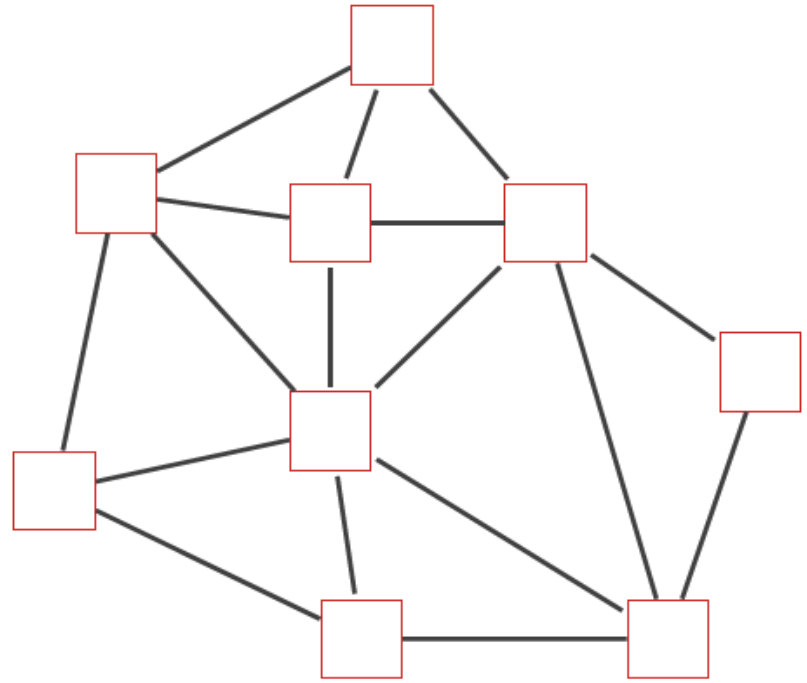
- ✓ 神经元不但可以接收其他神经元的信息，也可以接收自己的历史信息
- ✓ 神经元具有记忆功能，在不同的时刻具有不同的状态
- ✓ 信息传播可以是单向或者双向传递，可用一个有向循环图或无向图来表示
- ✓ 记忆网络包括循环神经网络、Hopfield网络、玻尔兹曼机、受限玻尔兹曼机等



常见的神经网络结构

图网络

- ✓ 图网络是定义在图结构数据上的神经网络
- ✓ 图中的每个节点都是由一个或者一组神经元构成
- ✓ 节点之间的连接可以是有向的，也可以是无向的
- ✓ 每个节点可以接收来自相邻节点或者自身的信息
- ✓ 图网络是前馈网络和记忆网络的方法，包含许多不同的实现方式，如图卷积网络、图注意力网络、消息传递网络等



其他结构设计方面的考虑

除了深度和宽度之外，神经网络的结构还具有其他方面的多样性。

■ 改变层与层之间的连接方式

- ✓ 前一层的每个单元仅与后一层的一个小单元子集相连
- ✓ 可以极大地减少参数的数量
- ✓ 具体的连接方式高度依赖于具体的问题

■ 增加跳跃连接

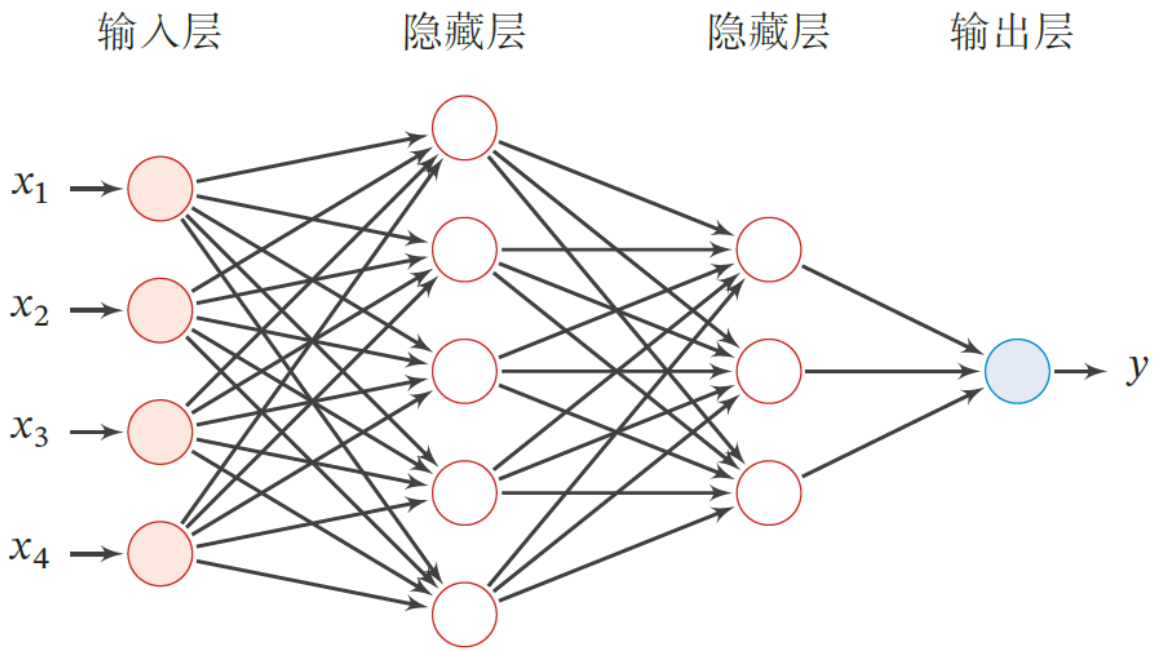
- ✓ 从第i层与第$i+2$层甚至更高层之间建立连接
- ✓ 使得梯度更容易从输出层流向更接近输入的层，利于模型优化

4.4 前馈神经网络

- 结构与表示
- 隐藏单元
- 输出单元
- 参数学习

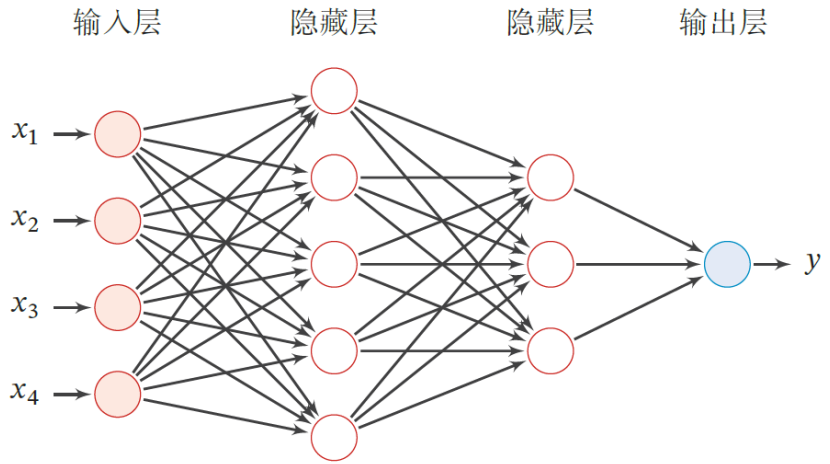
前馈神经网络的结构

- 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(Multi-Layer Perceptron, MLP)，但这个叫法并不十分合理（激活函数通常并不是感知器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，整个网络中无反馈，可用一个有向无环图表示



前馈神经网络形式化表示

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：

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

或者：

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

- 如此，通过逐层传递，得到最后的输出$\boldsymbol{a}^{(L)}$，整个网络可以看作一个复合函数$\boldsymbol{\phi}(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{b})$：

$$\begin{array}{ccccccccccc} \boldsymbol{a}^{(0)} & \to & \boldsymbol{z}^{(1)} & \to & \boldsymbol{a}^{(1)} & \to & \boldsymbol{z}^{(2)} & \to \cdots \to & \boldsymbol{a}^{(L-1)} & \to & \boldsymbol{z}^{(L)} & \to & \boldsymbol{a}^{(L)} \\ \| & & & & & & & & & & & & \| \\ \boldsymbol{x} & & & & & & & & & & & & \boldsymbol{\phi}(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{b}) \end{array}$$

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。

Sigmoid型函数

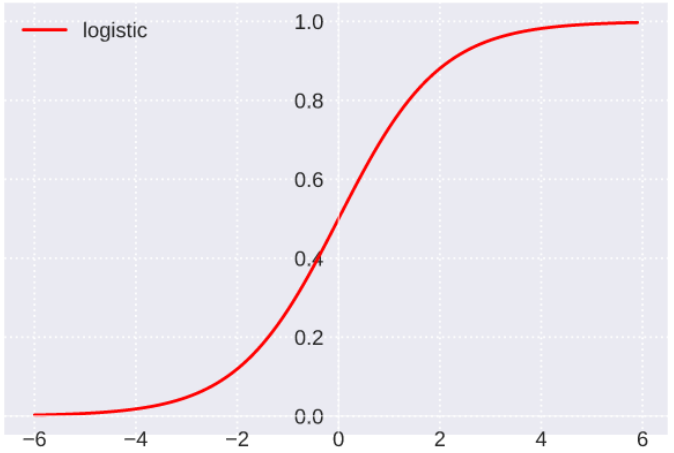
Sigmoid型函数指一类S型曲线函数，为**两端饱和函数**。

- **Logistic函数**

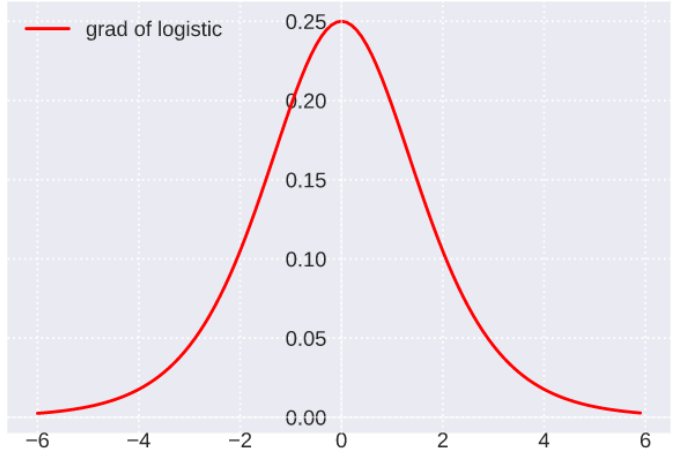
$$\sigma(x) = \frac{1}{1+\exp(-x)}$$

✓ 具有“挤压”功能

✓ 输出可看作概率分布



Logistic函数



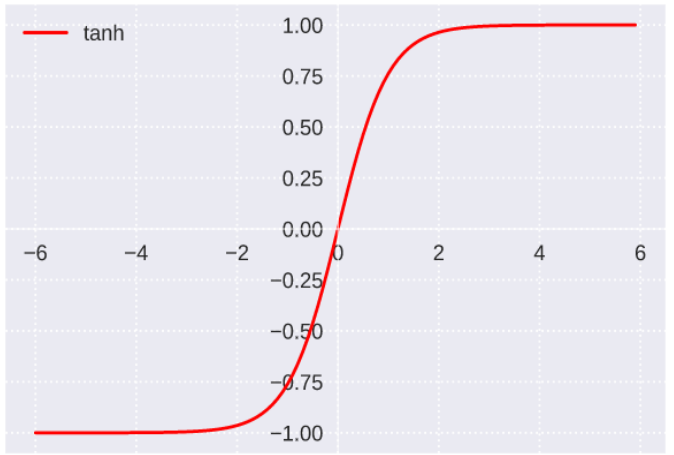
Logistic函数的导数

- **Tanh函数**

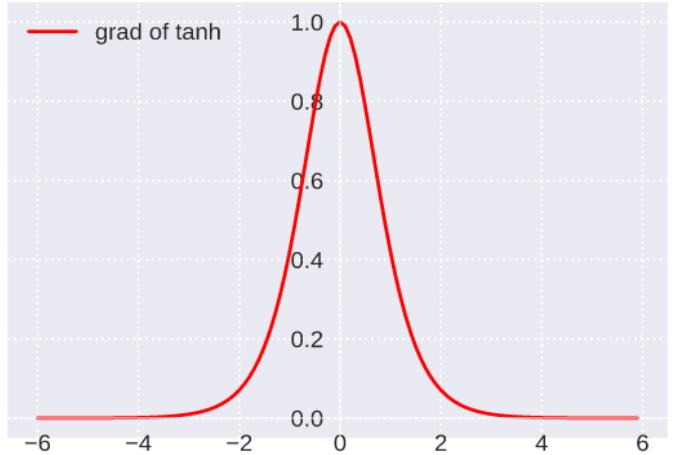
$$\tanh(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$$

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

✓ 零中心化，可提升收敛速度



Tanh函数



Tanh函数的导数

Sigmoid型函数

- **Hard-Logistic函数**

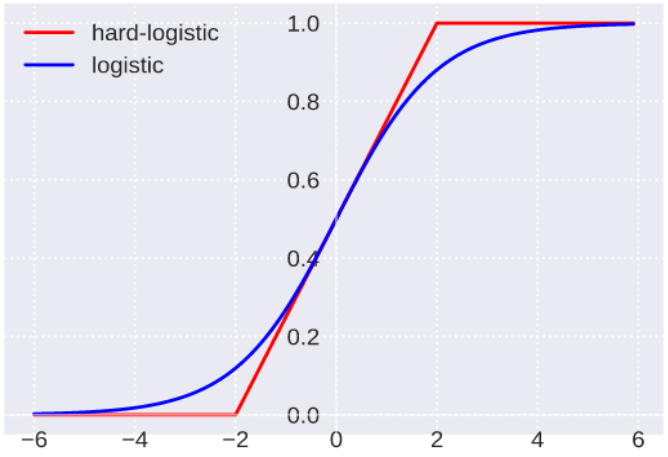
$$\text{hard}-\text{logistic}(x)$$
$$= \max(\min(0.25x + 0.5, 1), 0)$$

- **Hard-Tanh函数**

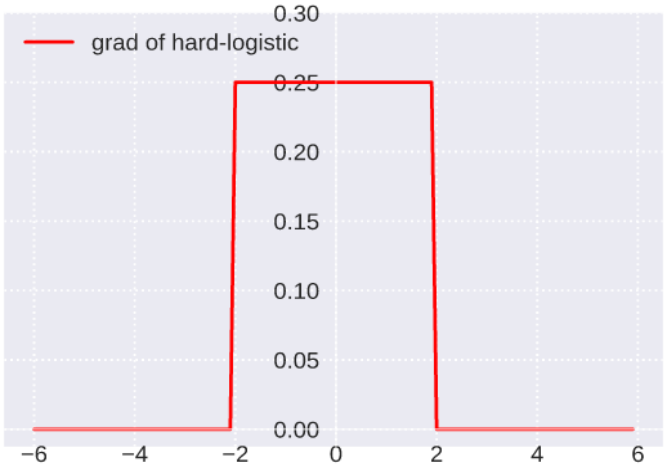
$$\text{hard}-\tanh(x)$$
$$= \max(\min(x, 1), -1)$$

✓ 这两个函数是对Logistic和Tanh函数的分段近似

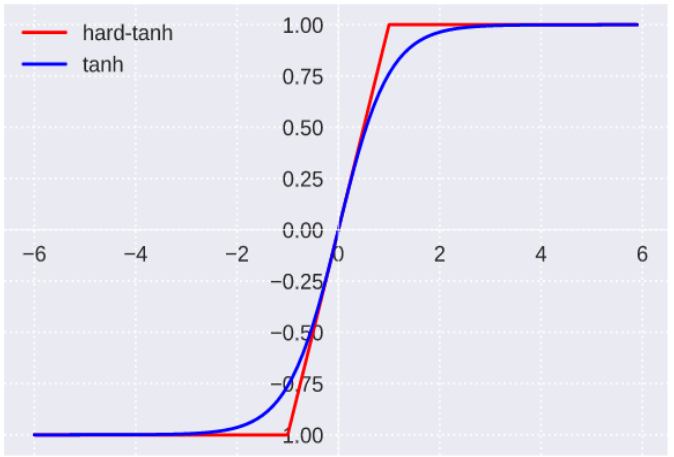
✓ 与Logistic和Tanh函数相比，降低了计算开销



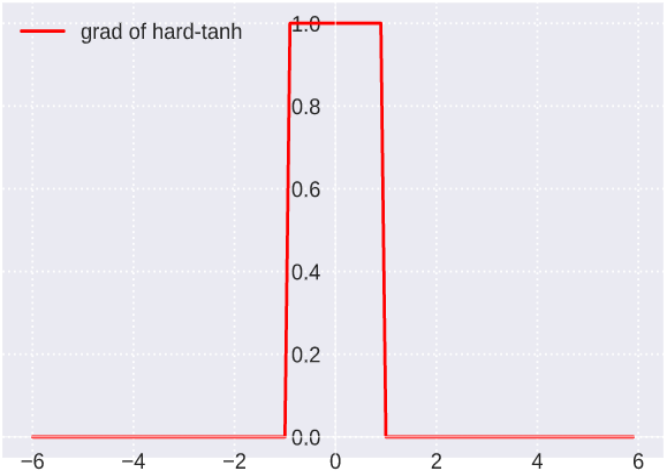
Hard-Logistic函数



Hard-Logistic函数的导数



Hard-Tanh函数



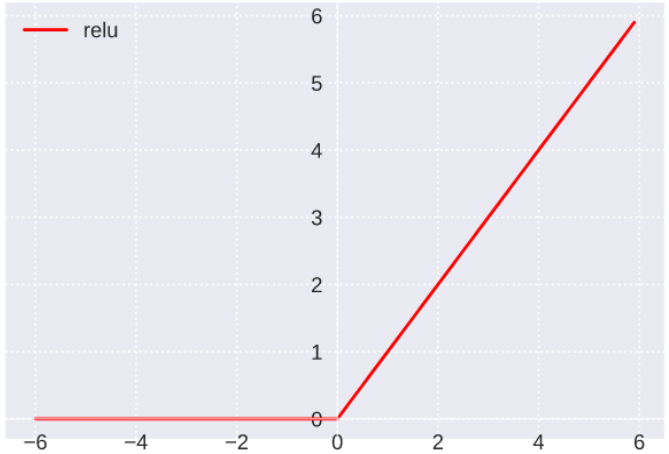
Hard-Tanh函数的导数

整流线性单元(ReLU)函数及其扩展

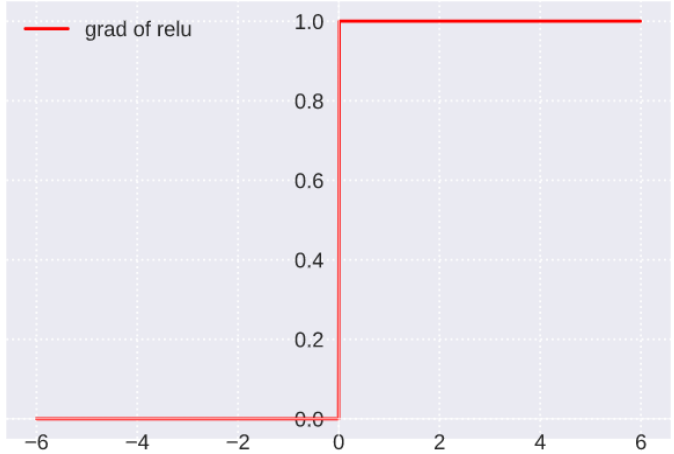
ReLU

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

- ✓ 是目前最常用的激活函数
- ✓ 具有**单侧抑制、宽兴奋边界**的生物学合理性
- ✓ 可缓解梯度消失问题
- ✓ 缺点：有可能导致神经元的死亡



ReLU函数



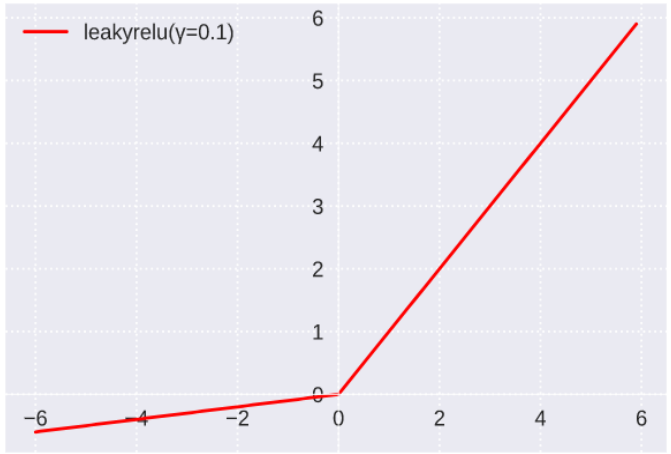
ReLU函数的导数

带泄露的ReLU（Leaky ReLU）

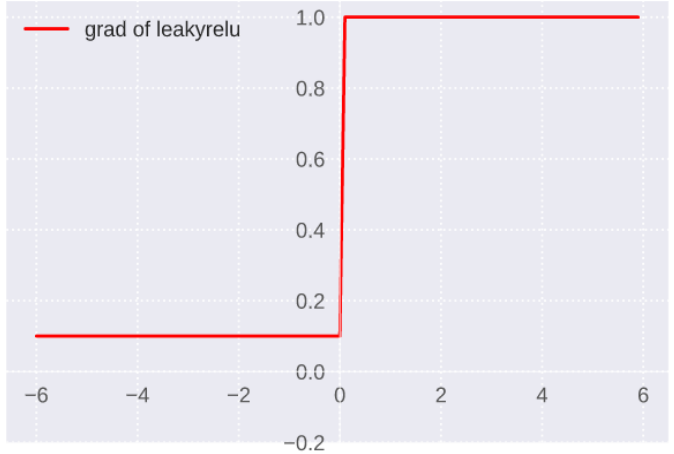
$$\mathrm{LeakyReLU}(x) = \begin{cases} x, & x \geq 0 \\ \gamma x, & x < 0 \end{cases}$$

$$= \max(0, x) + \gamma \min(0, x)$$

- ✓ 在$x < 0$时也保持一个很小的梯度，避免永远不能被激活的情况
- ✓ γ为超参



Leaky ReLU函数



Leaky LeLU函数的导数

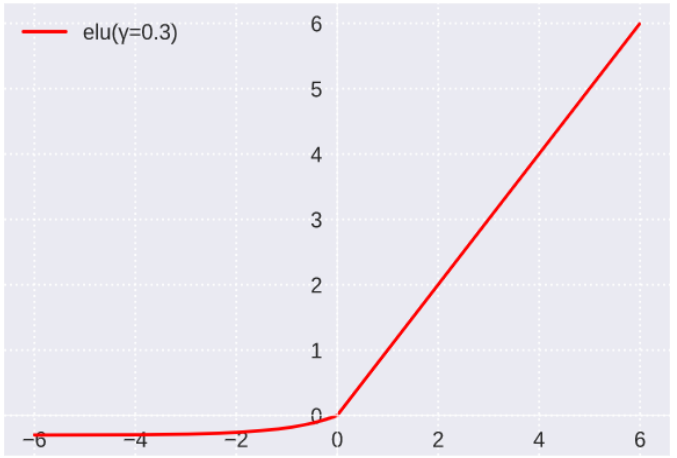
整流线性单元(ReLU)函数及其扩展

指数线性单元ELU

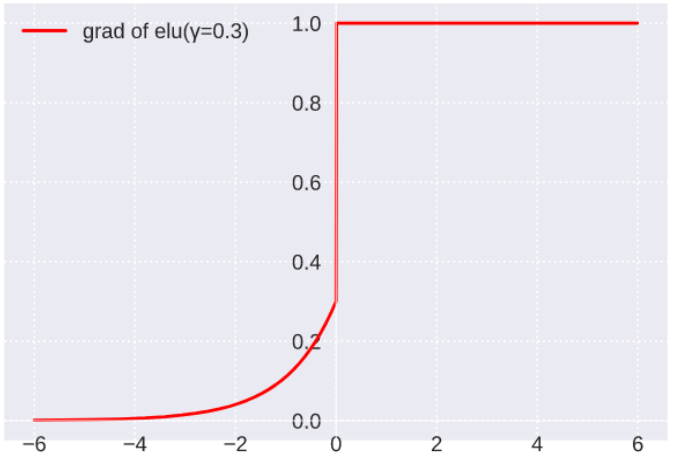
$$\mathrm{ELU}(x) = \begin{cases} x, & x \geq 0 \\ \gamma(\exp(x) - 1), & x < 0 \end{cases}$$

$$= \max(0, x) + \min(0, \gamma(\exp(x) - 1))$$

- ✓ 近似零中心化
- ✓ γ为超参



ELU函数

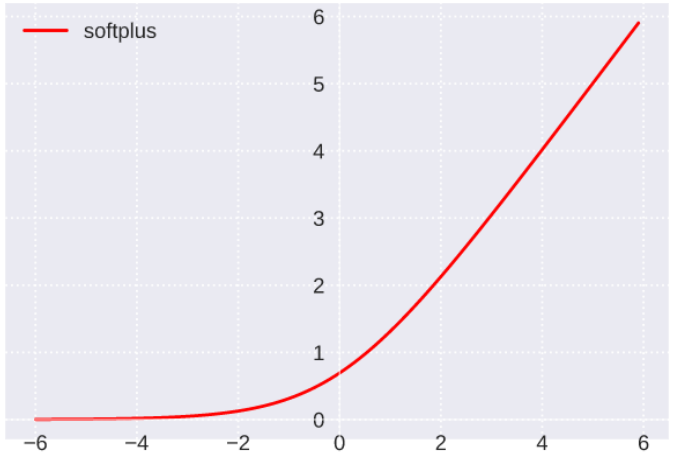


ELU函数的导数

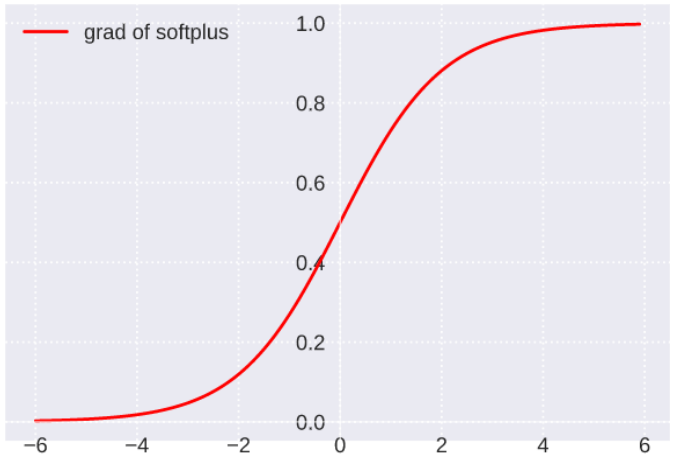
Softplus

$$\mathrm{Softplus}(x) = \log(1 + \exp(x))$$

- ✓ 可以看成ReLU的平滑版本
- ✓ 其导数刚好为Logistic函数
- ✓ 具有单侧抑制、宽兴奋边界的特性
- ✓ 但不具有稀疏性



Softplus函数



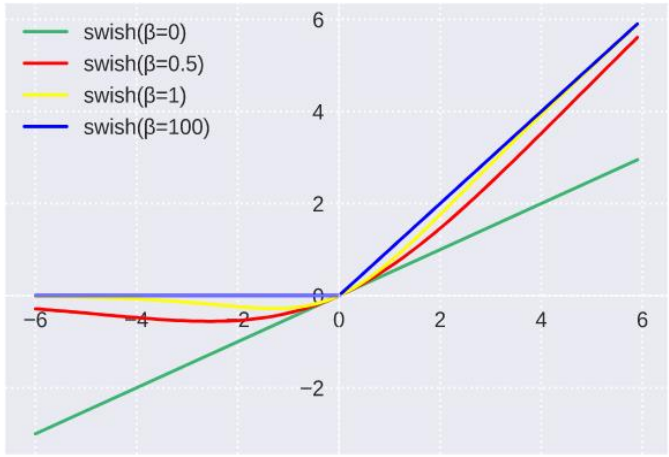
Softplus函数的导数

其他激活函数

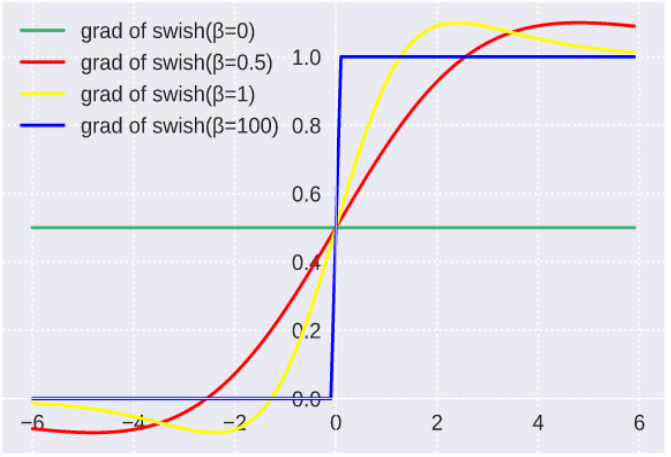
Swish函数

$$\text{swish}(x) = x\sigma(\beta x) = \frac{x}{1+\exp(-\beta x)}$$

- ✓ 当$\beta = 0$时为线性函数
- ✓ 当$\beta = 1$时在$x > 0$为近似线性
- ✓ 当$\beta \to \infty$时近似ReLU函数
- ✓ 可看成线性函数和ReLU之间的非线性插值函数



Swish函数

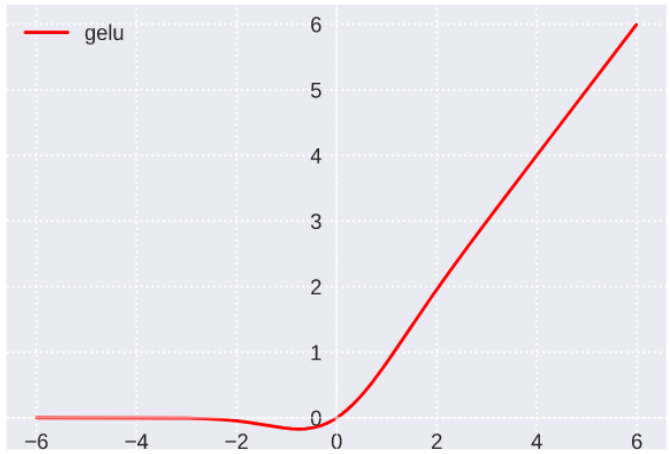


Swish函数的导数

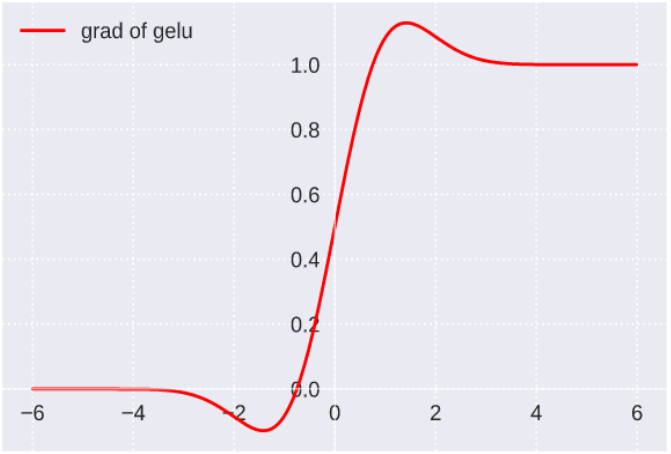
高斯误差线性单元（GELU）

$$\text{GELU}(x) = xP(X \le x) \approx x\sigma(1.702x)$$

- ✓ $P(X \le x)$为高斯分布的累积分布函数
- ✓ 可使用Logistic或者Tanh来近似计算
- ✓ 当使用Logistic近似时，相当于一种特殊的Swish函数



GELU函数

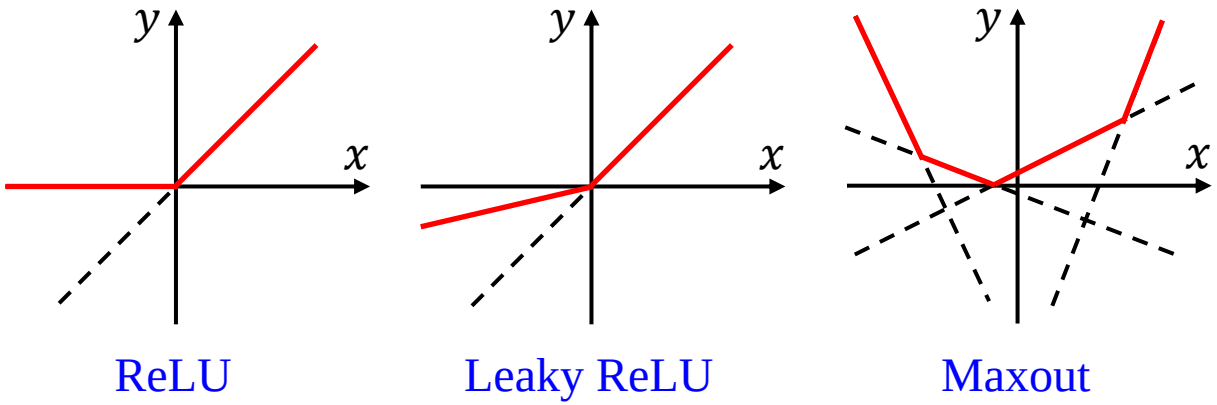


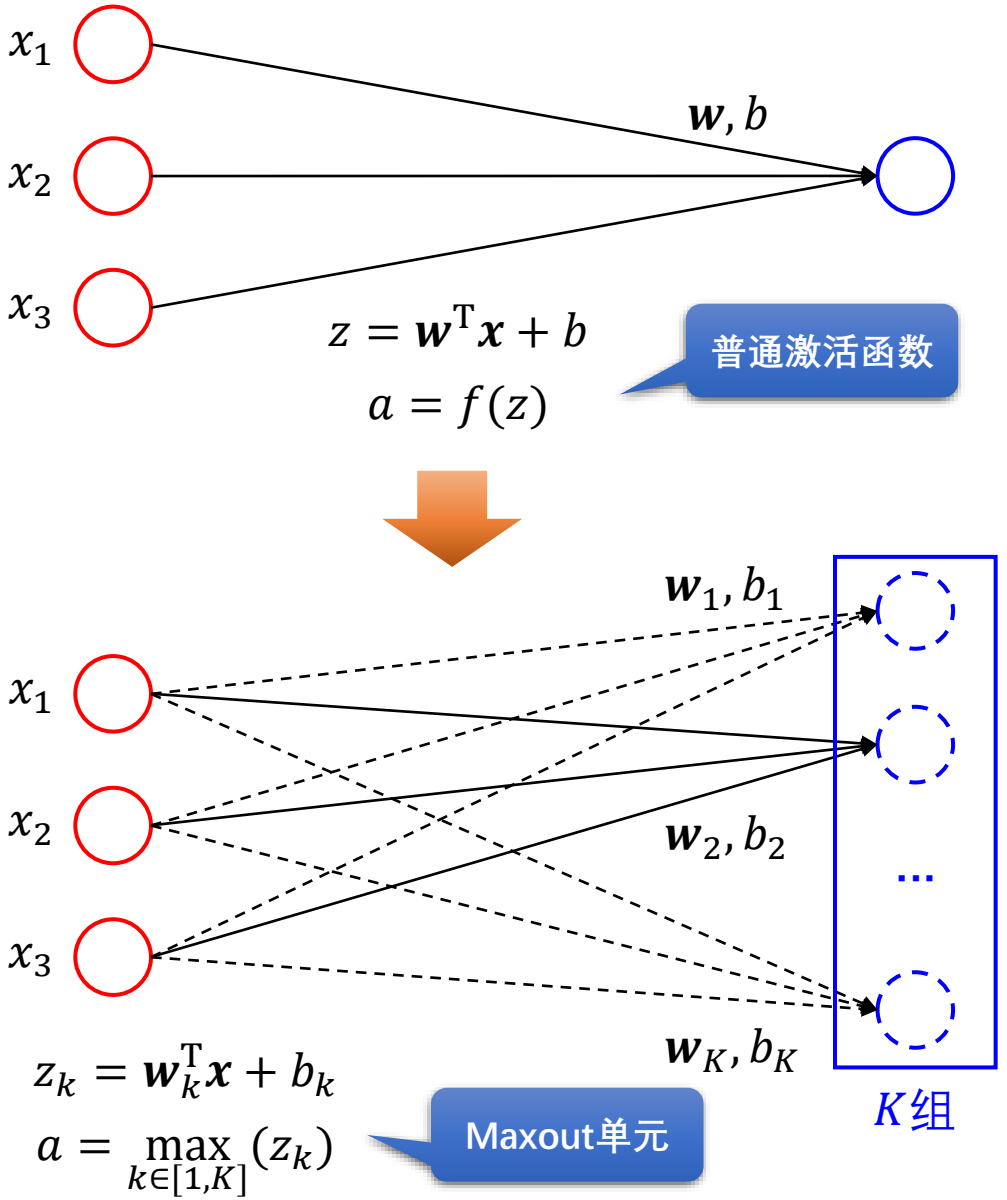
GELU函数的导数

其他激活函数

Maxout单元

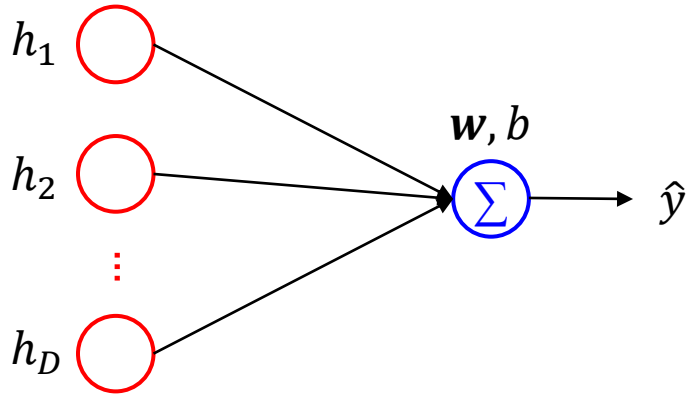
- ✓ Maxout是对整流线性单元的进一步扩展
- ✓ Maxout将神经元的净输入z分为K组，每一组均为一个线性函数
- ✓ Maxout可以看作任意凸函数的分段线性近似





线性输出单元

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



- ✓ 线性输出单元经常用于产生**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，最大化似然（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：

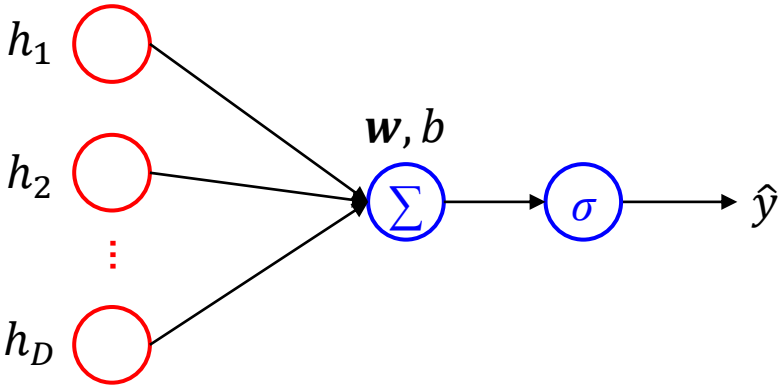
$$L(y, \hat{y}) = \frac{1}{N}\sum_{n=1}^{N}\left\|\hat{y}^{(n)} - y^{(n)}\right\|^2$$

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

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



- ✓ 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：

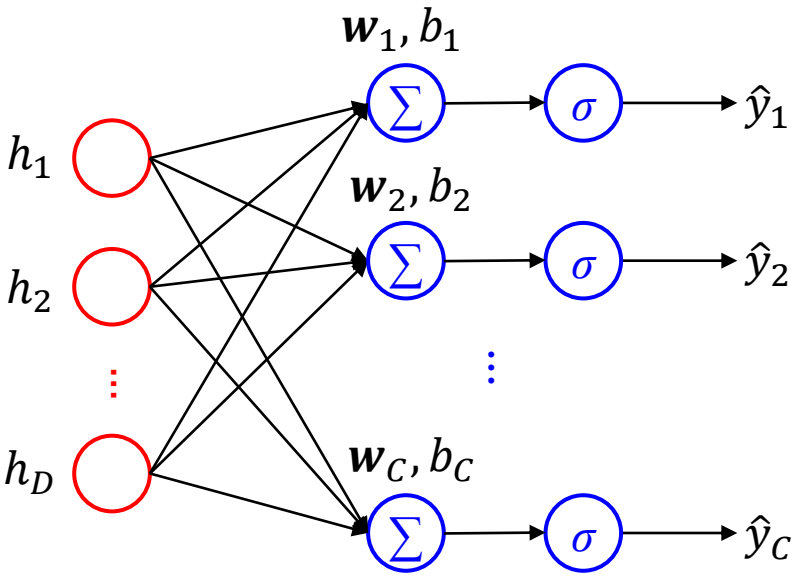
$$L(y, \hat{y}) = -\frac{1}{N}\sum_{n=1}^{N}\left(y^{(n)}\log\hat{y}^{(n)} + (1-y^{(n)})\log(1-\hat{y}^{(n)})\right)$$

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

输出单元

Softmax单元

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



- ✓ 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：

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

其中$\boldsymbol{y}^{(n)} = [y_1^{(n)}, y_2^{(n)}, \ldots, y_C^{(n)}]^{\mathrm{T}}$为真实标签向量，$\hat{\boldsymbol{y}}^{(n)} = [\hat{y}_1^{(n)}, \hat{y}_2^{(n)}, \ldots, \hat{y}_C^{(n)}]^{\mathrm{T}}$为预测标签概率向量，$N$为样本数，$C$为标签数。

前馈神经网络参数学习

学习准则

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

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

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$D = \{(\boldsymbol{x}^{(n)}, y^{(n)})\}_{n=1}^N$，每个样本的特征$\boldsymbol{x}^{(n)}$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：

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

其中$\boldsymbol{W}$和$\boldsymbol{b}$表示网络的全部参数，λ为超参，正则化项$\|\boldsymbol{W}\|_F^2$为Frobenius范数的平方：

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

前馈神经网络参数学习

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：

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

$$= \boldsymbol{W}^{(l)} - \alpha \left(\frac{1}{N} \sum_{n=1}^{N} \left(\frac{\partial L\left(\boldsymbol{y}^{(n)}, \hat{\boldsymbol{y}}^{(n)}\right)}{\partial \boldsymbol{W}^{(l)}} \right) + \lambda \boldsymbol{W}^{(l)} \right)$$

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

$$= \boldsymbol{b}^{(l)} - \alpha \left(\frac{1}{N} \sum_{n=1}^{N} \left(\frac{\partial L\left(\boldsymbol{y}^{(n)}, \hat{\boldsymbol{y}}^{(n)}\right)}{\partial \boldsymbol{b}^{(l)}} \right) \right)$$

其中α为学习率。

- ✓ 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地计算梯度

4.5 反向传播算法

- 微分链式法则
- 反向传播算法

矩阵微分

矩阵微分（Matrix Differentiation）是多元微积分的一种表达方式，使用矩阵和向量来表示因变量每个成分关于自变量每个成分的偏导数。

- **向量关于标量的偏导数**

$$\frac{\partial \boldsymbol{y}}{\partial x} = \left[\frac{\partial y_1}{\partial x}, \cdots, \frac{\partial y_N}{\partial x}\right] \in \mathbb{R}^{1\times N}$$

- **标量关于向量的偏导数**

$$\frac{\partial y}{\partial \boldsymbol{x}} = \left[\frac{\partial y}{\partial x_1}, \cdots, \frac{\partial y}{\partial x_M}\right]^{\mathrm{T}} \in \mathbb{R}^{M\times 1}$$

- **向量关于向量的偏导数**

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_N}{\partial x_1} \\ \vdots & \vdots & \vdots \\ \frac{\partial y_1}{\partial x_M} & \cdots & \frac{\partial y_N}{\partial x_M} \end{bmatrix} \in \mathbb{R}^{M\times N}$$

微分链式法则

标量的微分链式法则

若$x \in \mathbb{R}$，$y = g(x) \in \mathbb{R}$，$z = f(y) \in \mathbb{R}$，则链式法则为：

$$\frac{dz}{dx} = \frac{dy}{dx}\frac{dz}{dy}$$

向量的微分链式法则

- ✓ 若$x \in \mathbb{R}$，$\boldsymbol{y} = g(x) \in \mathbb{R}^M$，$\boldsymbol{z} = f(\boldsymbol{y}) \in \mathbb{R}^N$，则链式法则为：

$$\frac{\partial \boldsymbol{z}}{\partial x} = \frac{\partial \boldsymbol{y}}{\partial x}\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{y}} \in \mathbb{R}^{1\times M}\mathbb{R}^{M\times N} = \mathbb{R}^N$$

- ✓ 若$\boldsymbol{x} \in \mathbb{R}^M$，$\boldsymbol{y} = g(\boldsymbol{x}) \in \mathbb{R}^N$，$\boldsymbol{z} = f(\boldsymbol{y}) \in \mathbb{R}^K$，则链式法则为：

$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{y}} \in \mathbb{R}^{M\times N}\mathbb{R}^{N\times K} = \mathbb{R}^{M\times K}$$

- ✓ 若$\boldsymbol{X} \in \mathbb{R}^{M\times N}$，$\boldsymbol{y} = g(\boldsymbol{X}) \in \mathbb{R}^N$，$z = f(\boldsymbol{y}) \in \mathbb{R}$，则链式法则为：

$$\frac{\partial z}{\partial x_{ij}} = \frac{\partial \boldsymbol{y}}{\partial x_{ij}}\frac{\partial z}{\partial \boldsymbol{y}} \in \mathbb{R}^{1\times N}\mathbb{R}^{N\times 1} = \mathbb{R}$$

反向传播算法

给定一个样本$(\boldsymbol{x}, \boldsymbol{y})$，假设神经网络输出为$\hat{\boldsymbol{y}}$，损失函数为$L(\boldsymbol{y}, \hat{\boldsymbol{y}})$，采用梯度下降法需要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：

$$\frac{\partial L(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial w_{ij}^{(l)}} = \frac{\partial \boldsymbol{z}^{(l)}}{\partial w_{ij}^{(l)}} \frac{\partial L(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial \boldsymbol{z}^{(l)}} \quad ①$$

$$\frac{\partial L(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial \boldsymbol{b}^{(l)}} = \frac{\partial \boldsymbol{z}^{(l)}}{\partial \boldsymbol{b}^{(l)}} \frac{\partial L(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial \boldsymbol{z}^{(l)}} \quad ②$$

③ $\boldsymbol{\delta}^{(l)} \triangleq \frac{\partial L(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial \boldsymbol{z}^{(l)}}$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$\boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)}$：

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

$$= \left[0, \dots, \frac{\partial\left(\sum_{k=1}^{M_{l-1}} w_{ik}^{(l)} a_k^{(l-1)} + b_i^{(l)}\right)}{\partial w_{ij}^{(l)}}, \dots, 0\right]$$

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

当$k = j$时，与$w_{ij}^{(l)}$有关

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

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

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

其中

$$\frac{\partial \boldsymbol{a}^{(l)}}{\partial \boldsymbol{z}^{(l)}} = \frac{\partial f_l(\boldsymbol{z}^{(l)})}{\partial \boldsymbol{z}^{(l)}} = \mathrm{diag}(f_l'(\boldsymbol{z}^{(l)})) \in \mathbb{R}^{M_l \times M_l}$$

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

所以

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

$$= f_l'(\boldsymbol{z}^{(l)}) \odot \left(\left(\boldsymbol{W}^{(l+1)}\right)^{\mathrm{T}}\boldsymbol{\delta}^{(l+1)}\right) \in \mathbb{R}^{M_l}$$

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。

其中⊙是点积，表示每个元素相乘。

反向传播算法

- **对于最后一层（输出层）的误差项** $\boldsymbol{\delta}^{(L)} \triangleq \frac{\partial L(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \boldsymbol{z}^{(L)}}$

由$\hat{\boldsymbol{y}} = \boldsymbol{a}^{(L)} = f_L(\boldsymbol{z}^{(L)})$，根据链式法则：

$$\begin{aligned}
\boldsymbol{\delta}^{(L)} \triangleq \frac{\partial L(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \boldsymbol{z}^{(L)}} &= \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z}^{(L)}} \frac{\partial L(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \hat{\boldsymbol{y}}} \\
&= \frac{\partial f_L(\boldsymbol{z}^{(L)})}{\partial \boldsymbol{z}^{(L)}} \frac{\partial L(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \hat{\boldsymbol{y}}} \\
&= \operatorname{diag}\left(f_L'\left(\boldsymbol{z}^{(L)}\right)\right) \frac{\partial L(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \hat{\boldsymbol{y}}} \\
&= f_L'\left(\boldsymbol{z}^{(L)}\right) \odot \frac{\partial L(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \hat{\boldsymbol{y}}} \in \mathbb{R}^{M_L}
\end{aligned}$$

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

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：

$$\begin{aligned}\frac{\partial L(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial w_{ij}^{(l)}} &= \frac{\partial \boldsymbol{z}^{(l)}}{\partial w_{ij}^{(l)}} \frac{\partial L(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \boldsymbol{z}^{(l)}} \\ &= \left[0,\ldots,a_j^{(l-1)},\ldots,0\right] \boldsymbol{\delta}^{(l)} \\ &= \left[0,\ldots,a_j^{(l-1)},\ldots,0\right] \left[\delta_1^{(l)},\ldots,\delta_i^{(l)},\ldots,\delta_{M_l}^{(l)}\right]^{\mathrm{T}} \\ &= \delta_i^{(l)} a_j^{(l-1)}\end{aligned}$$

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：

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

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

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

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

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

反向传播算法

算法4.1 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$\boldsymbol{W}$, $\boldsymbol{b}$;

repeat

  对训练集D中的样本随机重排序;

  For $n=1\ldots N$ **do**

    从训练集D中选取样本$(\boldsymbol{x}^{(n)},y^{(n)})$;

    前馈计算每一层的净输入$\boldsymbol{z}^{(l)}$和激活值$\boldsymbol{a}^{(l)}$，直到最后一层;

    反向传播计算每一层的误差$\boldsymbol{\delta}^{(l)}=f_l'(\boldsymbol{z}^{(l)})\odot\left((\boldsymbol{W}^{(l+1)})^{\mathrm{T}}\boldsymbol{\delta}^{(l+1)}\right)$; //最后一层的误差为$\boldsymbol{\delta}^{(L)}=f_L'(\boldsymbol{z}^{(L)})\odot\frac{\partial L(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial\hat{\boldsymbol{y}}}$

    //计算每一层的梯度

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

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

    //更新参数

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

    $\boldsymbol{b}^{(l)}=\boldsymbol{b}^{(l)}-\alpha\boldsymbol{\delta}^{(l)}$;

  end;

until 模型在验证集V上的错误率不再下降;

4.6 自动梯度计算

- 数值微分
- 符号微分
- 自动微分
- 计算图

自动梯度计算

手动使用链式法则计算每个参数的导数并编程实现非常繁琐且极易出错。可使用计算机实现参数的自动梯度计算，其方法可分为**数值微分**、**符号微分**和**自动微分**三类。

■ 数值微分（Numerical Differentiation）

用数值的方法来计算函数$f(x)$的导数，函数$f(x)$在点x处的导数定义为

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

在实际应用中，经常使用下面的方式来计算梯度，以减小截断误差

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x}$$

- ✓ Δx难以确定，太小会引起舍入误差，太大则增加截断误差
- ✓ 虽然实现非常简单，但实用性较差
- ✓ 计算复杂性高，因为需要为每个参数单独施加扰动，若参数数量为N，则复杂度为 $O(N^2)$

■ 符号微分（Symbolic Differentiation）

- ✓ 一种基于符号计算(代数计算)的自动求导方法，用计算机来求解带变量的数学表达式
- ✓ 变量被看作符号，不需要代入具体的值，输入和输出都是数学表达式
- ✓ 包括基于规则的化简、因式分解、微分、积分、解代数方程、解常微分方程等运算
- ✓ 例如：

输入z：$\ln x + x^2 y - \sin xy$

输出$\frac{\partial z}{\partial x}$：$\frac{1}{x} + 2xy - y\cos xy$

- ✓ 编译时间长
- ✓ 需要专门的数学计算语言
- ✓ 很难调试
- ✓ 对于深层复合函数，输出的表达式非常冗长，形成表达式膨胀（expression swell）

■ 自动微分（Automatic Differentiation）

✓ 一种介于数值微分和符号微分之间的方法

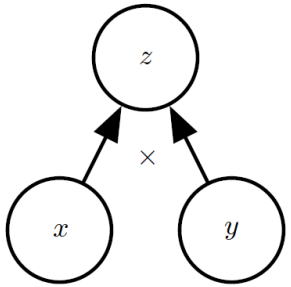
- 数值微分强调一开始直接代入数值近似求解，而符号微分强调直接对表达式进行求解，最后才代入数值；
- 自动微分将符号微分法应用于最基本的算子，比如常数、幂函数、指数函数、对数函数、三角函数等，将其代入数值，保留中间结果，最后再应用于整个函数

✓ 灵活性高

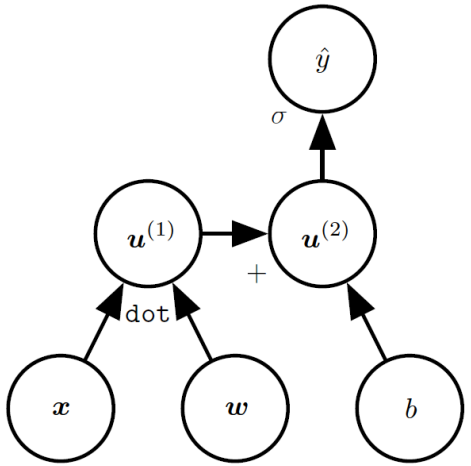
- 微分求解过程对用户是透明的
- 不需要专门的数学语言和编程
- 采用图的方式进行计算，可以做很多优化

■ 计算图（Computational Graph）

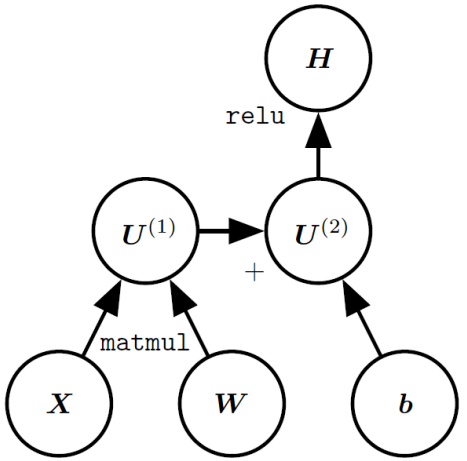
- ✓ 将复合函数分解为一系列基本操作，并以图的形式连接起来
- ✓ 是数学运算的图结构表示，每个非叶子节点代表一个基本操作，每个叶子节点代表一个输入变量或常量



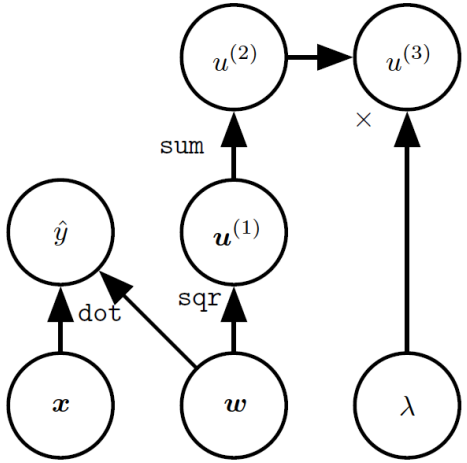
$z = xy$



$\hat{y} = \sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b)$



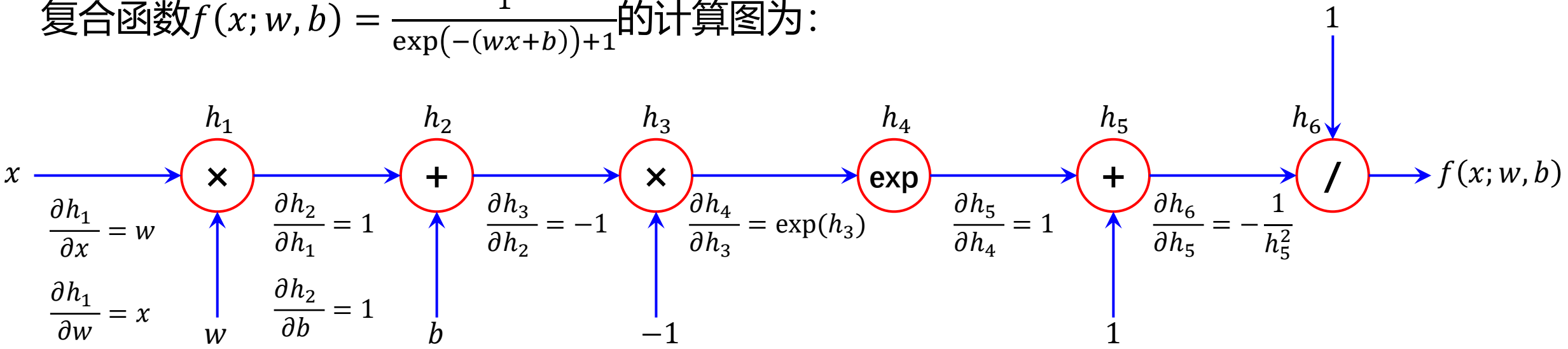
$\boldsymbol{H} = \max(0, \boldsymbol{W}\boldsymbol{X} + \boldsymbol{b})$



$\hat{y} = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x};\ u^{(3)} = \lambda \sum_i w_i^2$

■ 计算图实例

复合函数$f(x;w,b)=\frac{1}{\exp(-(wx+b))+1}$的计算图为：



$f(x;w,b)$关于参数w和b的导数可以通过计算图上的路径上的所有导数连乘得到：

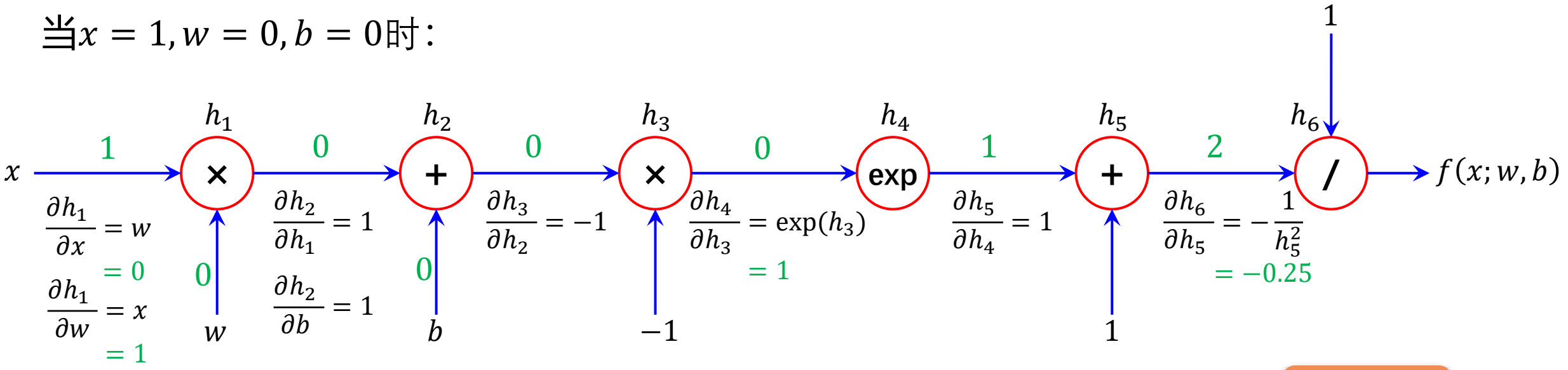
$$\frac{\partial f(x;w,b)}{\partial w}=\frac{\partial f(x;w,b)}{\partial h_6}\frac{\partial h_6}{\partial h_5}\frac{\partial h_5}{\partial h_4}\frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial w}$$

$$\frac{\partial f(x;w,b)}{\partial b}=\frac{\partial f(x;w,b)}{\partial h_6}\frac{\partial h_6}{\partial h_5}\frac{\partial h_5}{\partial h_4}\frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial b}$$

如果函数和参数之间有多条路径，可以将这些路径上的导数进行相加，得到最终的梯度。

■ 计算图实例

当$x=1, w=0, b=0$时：



$$\frac{\partial f(x;w,b)}{\partial w}\Big|_{x=1,w=0,b=0} = \frac{\partial f(x;w,b)}{\partial h_6}\frac{\partial h_6}{\partial h_5}\frac{\partial h_5}{\partial h_4}\frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial w}$$

反向模式

前向模式

$$= 1 \times -0.25 \times 1 \times 1 \times -1 \times 1 \times 1 = 0.25$$

$$\frac{\partial f(x;w,b)}{\partial b}\Big|_{x=1,w=0,b=0} = \frac{\partial f(x;w,b)}{\partial h_6}\frac{\partial h_6}{\partial h_5}\frac{\partial h_5}{\partial h_4}\frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial b}$$

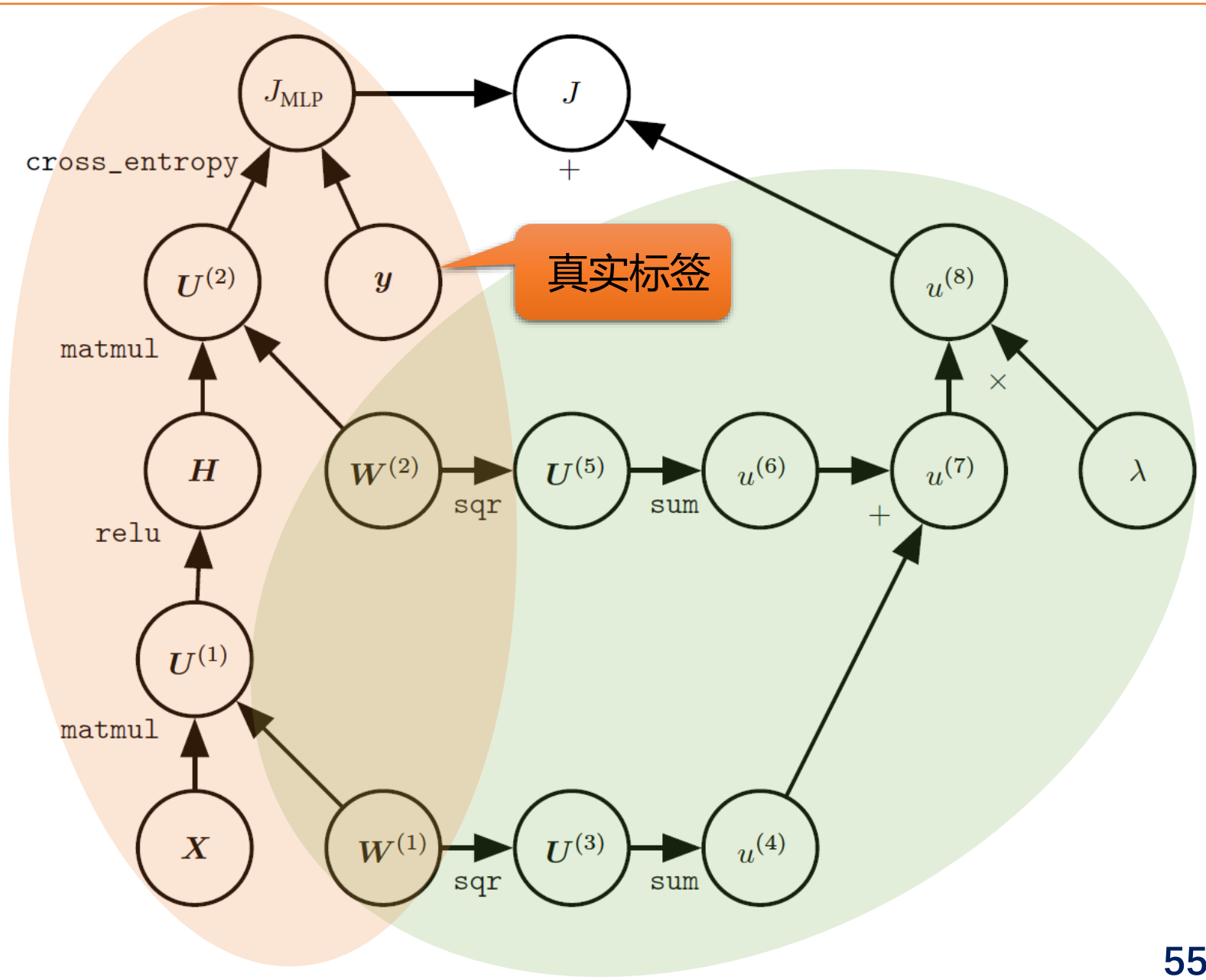
$$= 1 \times -0.25 \times 1 \times 1 \times -1 \times 1 = 0.25$$

计算图

■ 计算图实例

正则化单隐层MLP计算图：

$$J = J_{\text{MLP}} + \lambda \left(\sum_{i,j} \left(W^{(1)}_{i,j} \right)^2 + \sum_{i,j} \left(W^{(2)}_{i,j} \right)^2 \right)$$



静态图与动态图

- **静态计算图（Static Computational Graph）**

✓ 在编译时构建计算图，构建好后在程序运行时不能改变

✓ 在构建时可以进行优化、并行能力强

✓ 灵活性较差

- **动态计算图（Dynamic Computational Graph）**

✓ 在程序运行时动态构建计算图

✓ 不容易优化，当不同输入所使用的网络结构不一样时，难以并行计算

✓ 灵活性比较高

在当前深度学习框架中，Theano和Tensorflow采用的是静态计算图，而DyNet、Chainer和PyTorch采用的是动态计图。Tensorflow 2.0也开始支持动态计算图。

4.7 神经网络参数优化的主要问题

- 非凸优化
- 梯度消失

非凸优化

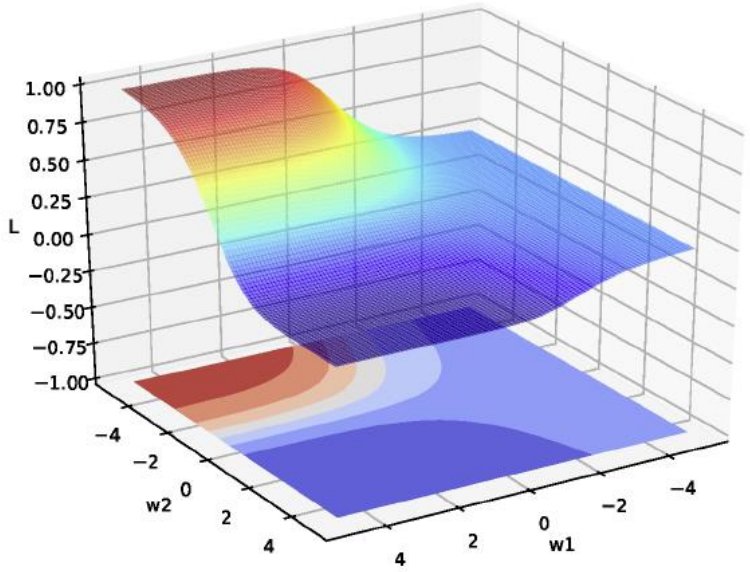
- **神经网络的优化问题是一个非凸优化问题**

以一个最简单的1-1-1结构的两层网络为例：

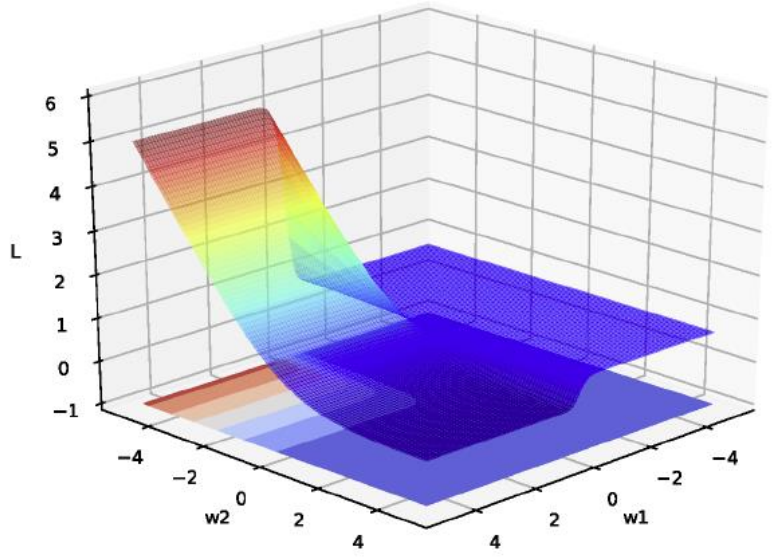
$$y = \sigma\big(w_2\sigma(w_1 x)\big)$$

其中w_1和w_2为网络参数，$\sigma(\cdot)$为Logistic激活函数。

给定一个样本(1,1)，平方误差损失和交叉熵损失与参数w_1和w_2的关系如下图：



(a) 平方误差损失



(b) 交叉熵损失

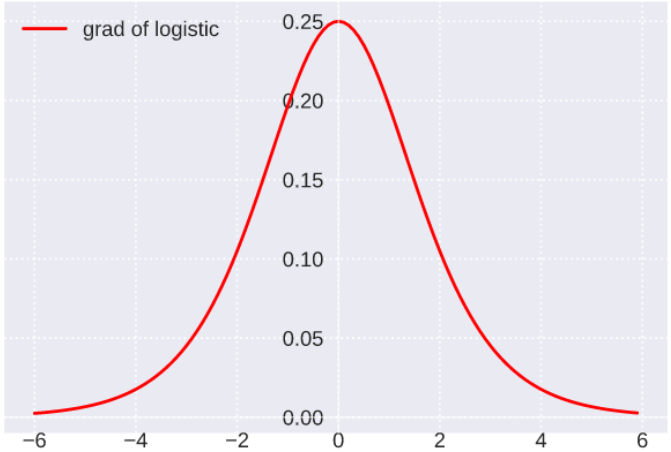
梯度消失

- **误差在反向传播过程中不断衰减甚至消失**

✓ 回顾误差反向传播的迭代公式：

$$\boldsymbol{\delta}^{(l)} = f_l'\left(\boldsymbol{z}^{(l)}\right) \odot \left(\left(\boldsymbol{W}^{(l+1)}\right)^{\mathrm{T}} \boldsymbol{\delta}^{(l+1)}\right)$$

误差在每一层传播时都要乘以该层激活函数的导数。



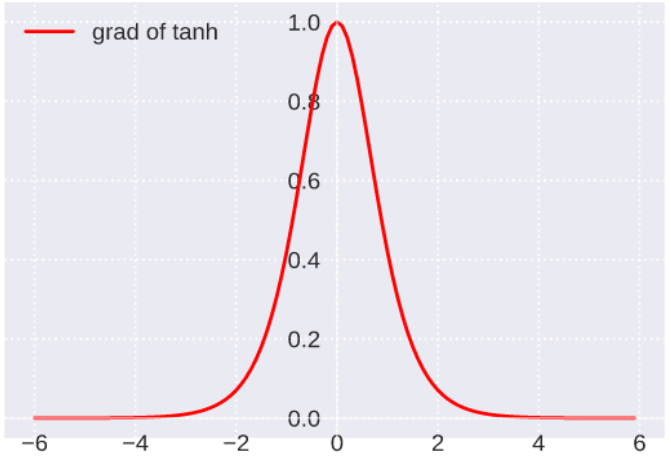
Logistic函数的导数

✓ 当采用Sigmoid型激活函数时，导数为：

$$\sigma'(x) = \sigma(x)\big(1-\sigma(x)\big) \in [0, 0.25]$$

$$\tanh'(x) = 1 - (\tanh(x))^2 \in [0, 1]$$

二者的导数的值域都小于或者等于1，在饱和区的导数更接近于0，这就会造成梯度的不断衰减，甚至消失，使得整个网络很难训练。



Tanh函数的导数