《神经网络与深度学习》

前馈神经网络

周晓飞
自动化学院

神经网络（Neural Network）

- 神经网络最早是作为一种主要的连接主义模型。20世纪80年代后期，最流行的一种连接主义模型是分布式并行处理（Parallel Distributed Processing，PDP）网络，其有3个主要特性：
  - （1）信息表示是分布式的（非局部的）；
  - （2）记忆和知识是存储在单元之间的连接上；
  - （3）通过逐渐改变单元之间的连接强度来学习新的知识。

- 早期关注生物学合理性，引入误差反向传播来改进其学习能力之后，神经网络也越来越多地应用在各种机器学习任务上。

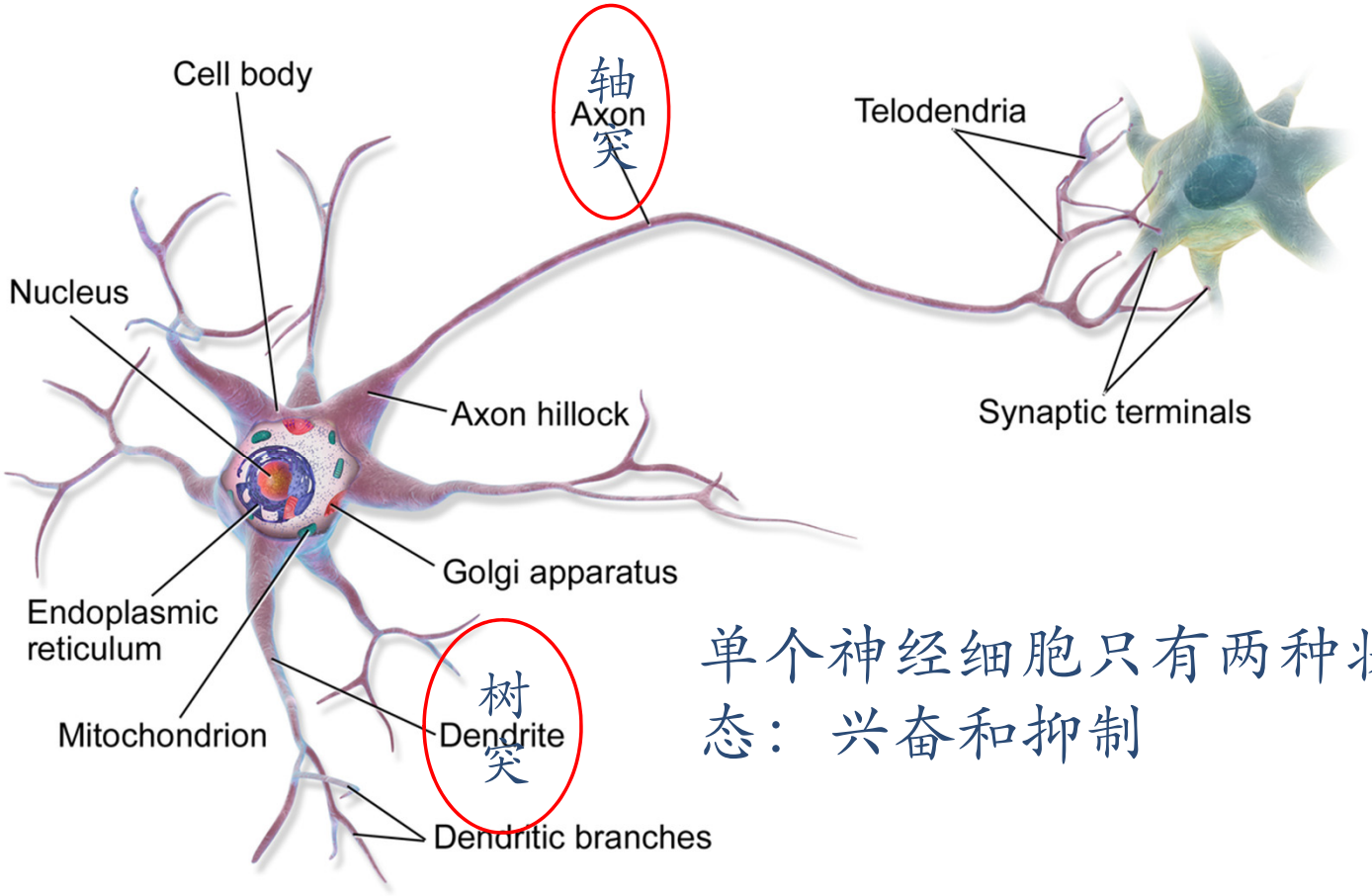
人工神经网络

- 人工神经网络主要由大量的神经元以及它们之间的有向连接构成。因此考虑三方面：

- 神经元的激活规则
  - 主要是指神经元输入到输出之间的映射关系，一般为非线性函数。
- 网络的拓扑结构
  - 不同神经元之间的连接关系。
- 学习算法
  - 通过训练数据利用梯度下降算法来学习神经网络的参数。

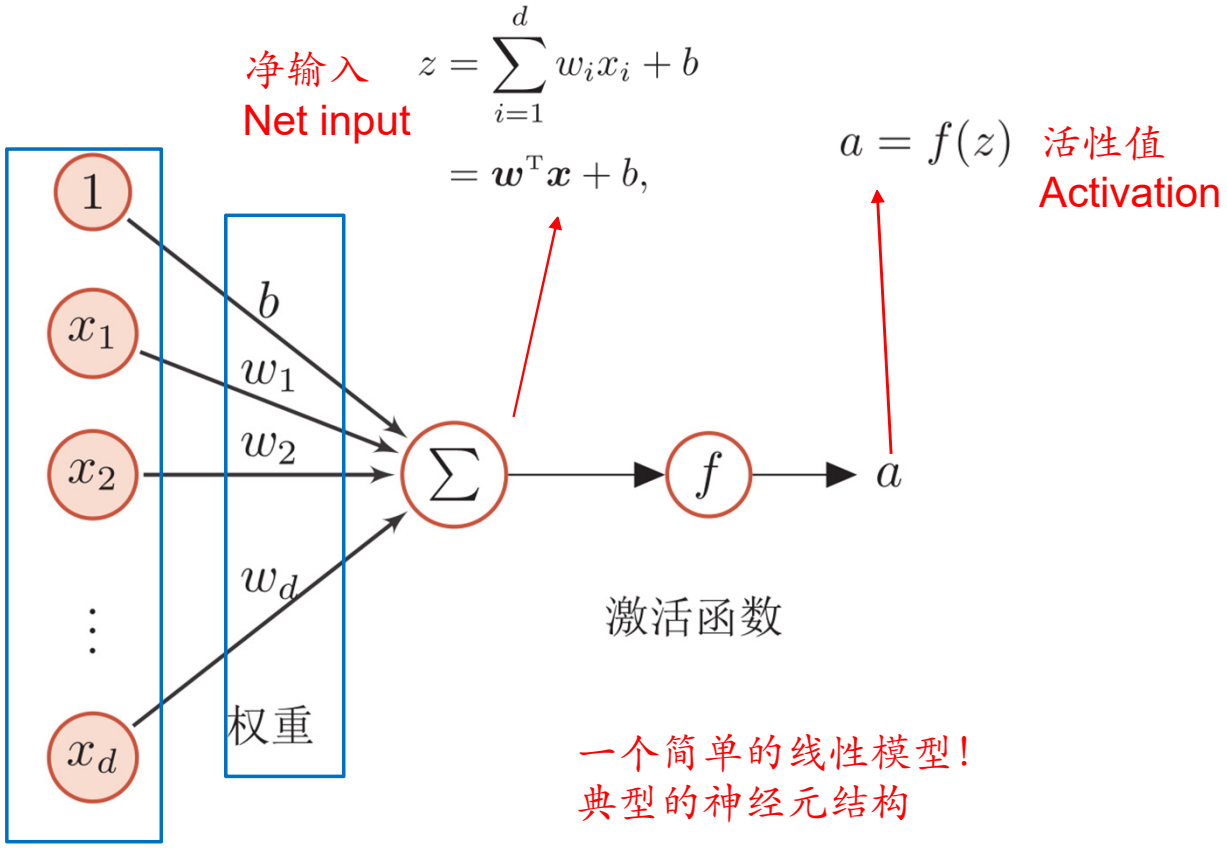
4.1 神经元

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


净输入
Net input
$z = \sum_{i=1}^{d} w_i x_i + b$
$= \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x} + b,$
$a = f(z)$
活性值
Activation
1
x_1
x_2
x_d
b
w_1
w_2
w_d
$\sum$
f
a
激活函数
权重
一个简单的线性模型!
典型的神经元结构

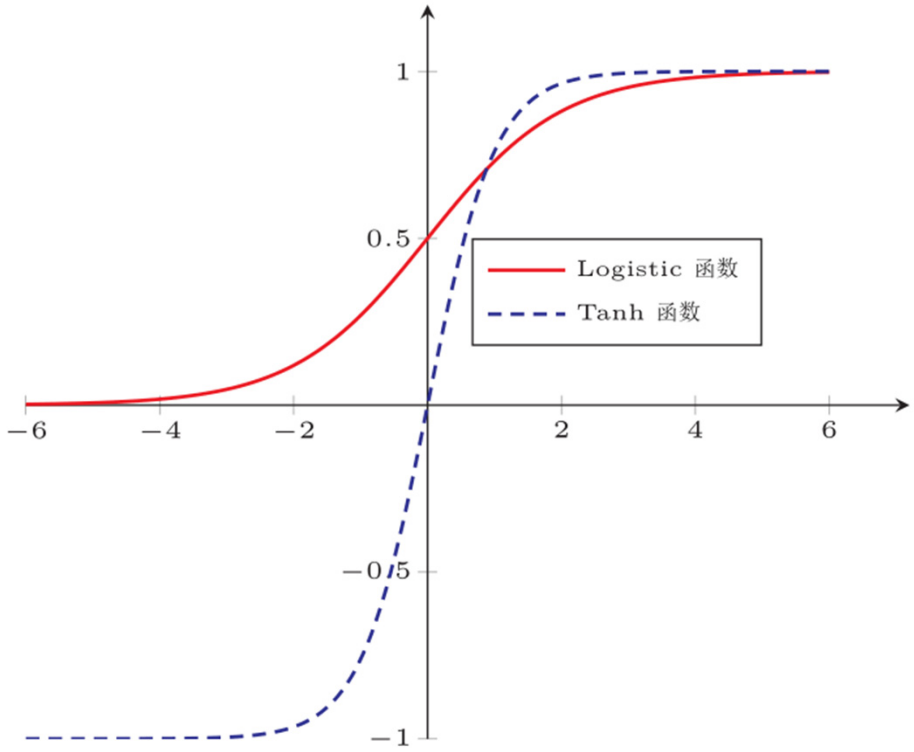
激活函数的性质

- 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。

常见激活函数

Logistic函数 $\sigma(x) = \dfrac{1}{1+\exp(-x)}$

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



- 性质：
  - 饱和函数：左饱和、右饱和、两端饱和
  - Tanh函数是零中心化的，而logistic函数的输出恒大于0

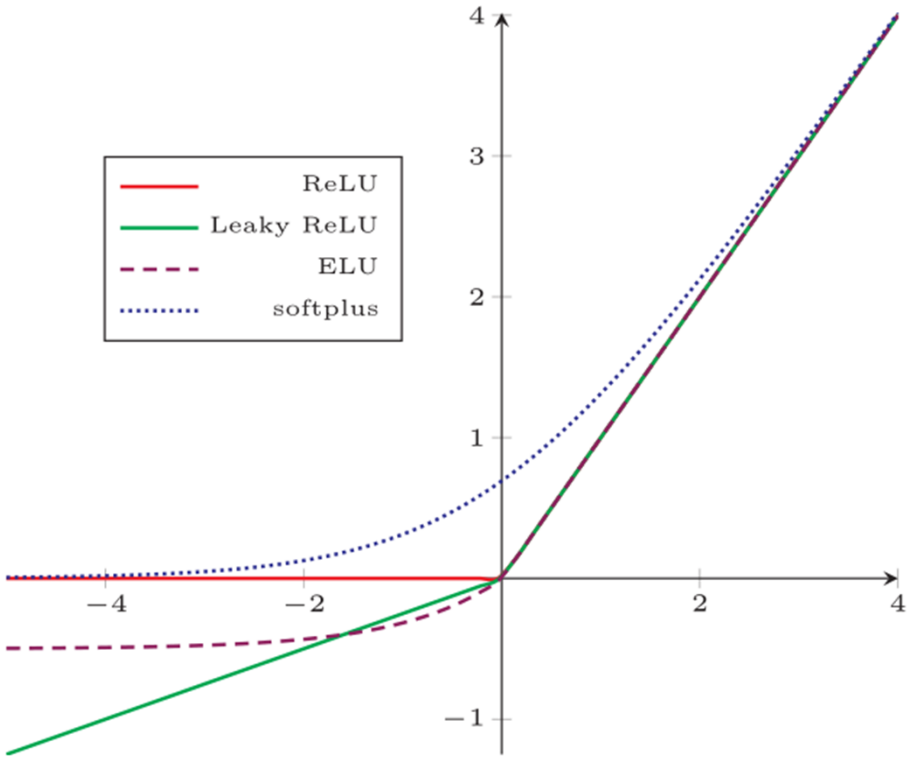
非零中心化的输出会使得其后一层的神经元的输入发生偏置偏移（bias shift），并进一步使得梯度下降的收敛速度变慢。

常见激活函数

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

$$\mathrm{LeakyReLU}(x) = \begin{cases} x & \text{if } x \\ \gamma x & \text{if } x \end{cases}$$
$$= \max(0, x)$$

$$\mathrm{PReLU}_i(x) = \begin{cases} x & \text{if } x > 0 \\ \gamma_i x & \text{if } x \leq 0 \end{cases}$$
$$= \max(0, x) + \gamma_i$$

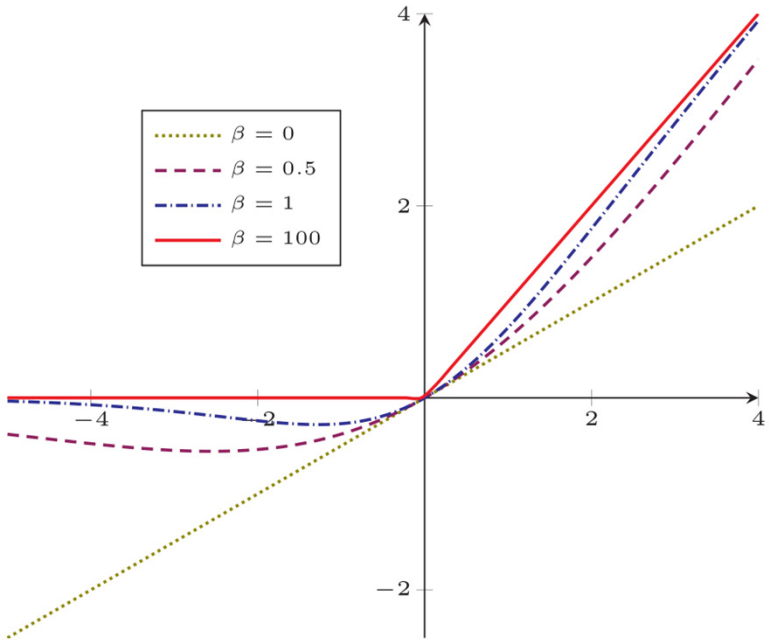


加高效。
解释性
制、宽兴奋边界
稀疏性
度上缓解梯度消失问题

化，偏置偏移
U问题（Dying ReLU

常见激活函数

Swish函数 $\mathrm{swish}(x) = x\sigma(\beta x)$



常见激活函数

- 高斯误差线性单元（Gaussian Error Linear Unit，GELU）

$$\mathrm{GELU}(x) = xP(X \leq x)$$

  - 其中P(X ≤ x)是高斯分布N(μ,σ 2)的累积分布函数，其中μ,σ为超参数，一般设μ = 0,σ = 1即可

- 由于高斯分布的累积分布函数为S型函数，因此GELU可以用Tanh函数或Logistic函数来近似

$$\mathrm{GELU}(x) \approx 0.5x\left(1 + \tanh\left(\sqrt{\frac{2}{\pi}}(x + 0.044715x^3)\right)\right)$$

或 $$\mathrm{GELU}(x) \approx x\sigma(1.702x).$$

常见激活函数

- Maxout单元　Maxout 单元的非线性函数定义为

$$\text{maxout}(\boldsymbol{x}) = \max_{k \in [1,K]} (z_k)$$

$$z_k = \boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x} + b_k$$

  - 每个单元的输入为上一层神经元的全部原始输出，是一个向量 $\mathbf{x} = [x_1; x_2; \ldots; x_D]$
  - 上式中，每个Maxout单元有K个权重向量$\mathbf{w}_k = [w_{k,1}, \ldots, w_{k,D}]^T$ 和偏置$b_k (1 \le k \le K)$

- Maxout单元不单是净输入到输出之间的非线性映射，而且还是整体学习输入到输出之间的非线性映射关系；可以视作分段线性函数

常见激活函数及其导数

激活函数	函数	导数
Logistic 函数	$f(x)=\frac{1}{1+\exp(-x)}$	$f'(x)=f(x)\big(1-f(x)\big)$
Tanh 函数	$f(x)=\frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$	$f'(x)=1-f(x)^2$
ReLU	$f(x)=\max(0,x)$	$f'(x)=I(x>0)$
ELU	$f(x)=\max(0,x)+\min\big(0,\gamma(\exp(x)-1)\big)$	$f'(x)=I(x>0)+I(x\leq 0)\cdot\gamma\exp(x)$
SoftPlus 函数	$f(x)=\log\big(1+\exp(x)\big)$	$f'(x)=\frac{1}{1+\exp(-x)}$

4.2 网络结构

网络结构

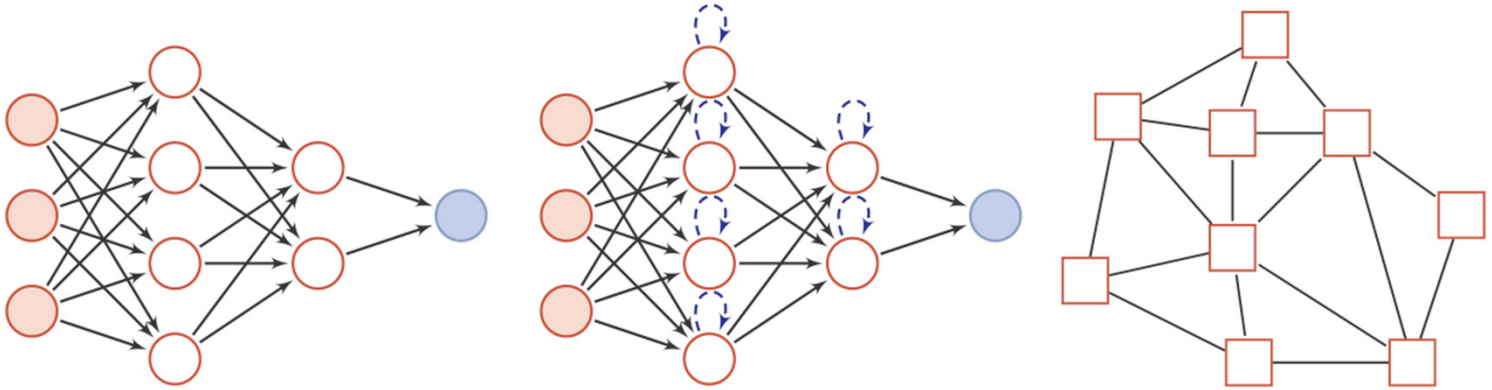
- 人工神经网络由神经元模型构成，这种由许多神经元组成的信息处理网络具有并行分布结构。

- 前馈网络：每组视作一个神经层，信息朝一个方向传递，包括全连接前馈网络和卷积神经网络；如图中（a）所示

- 记忆网络：不但可以接收其它神经元的信息，还可以接收自己的历史信息，包括循环神经网络，Hopefield网络、玻尔兹曼机等；如图中（b）所示

网络结构

- 图网络：定义在图结构数据上的神经网络，每个节点由一个或一组神经元构成，节点之间连接有向或者无向；它可以看作前馈网络或者记忆网络的泛化，包括GCN、GAT、MPNN；如图中（c）所示



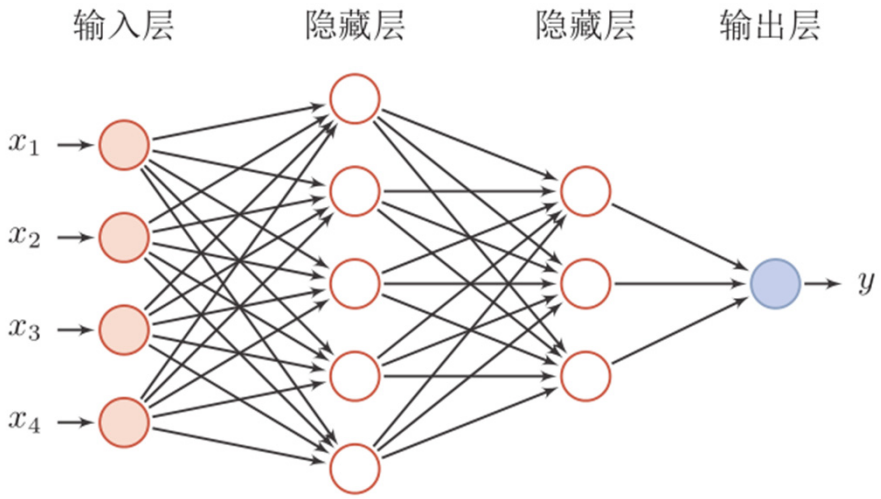
(a) 前馈网络　　(b) 记忆网络　　(c) 图网络

圆形节点表示一个神经元，方形节点表示一组神经元。

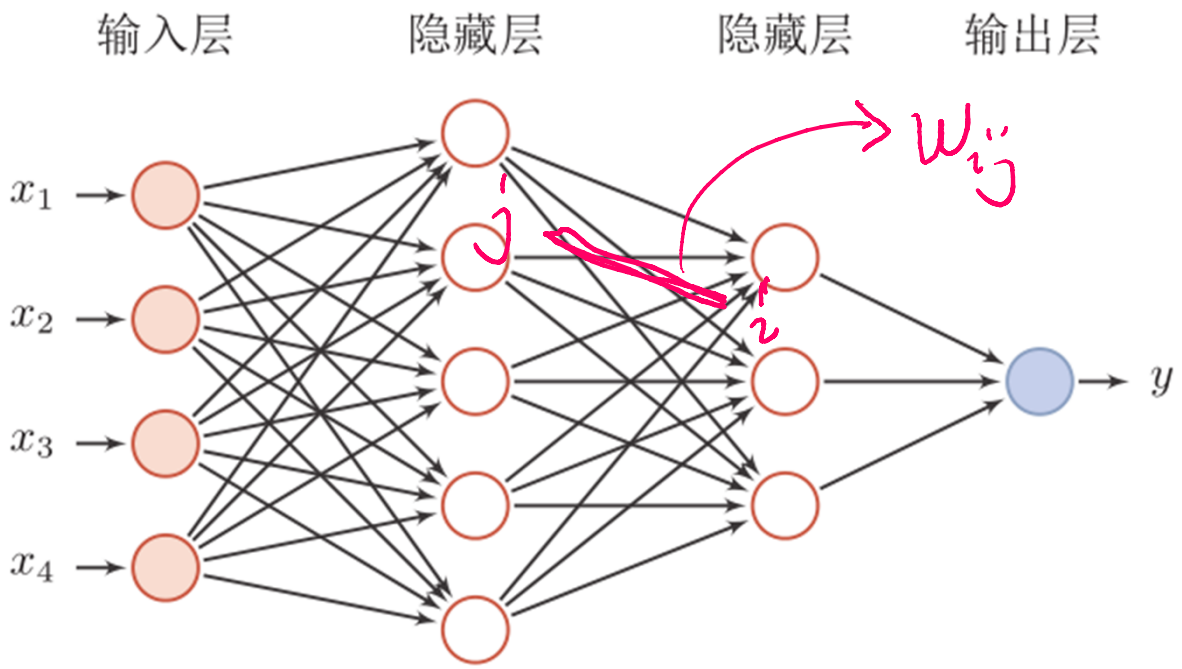
4.3 前馈神经网络

网络结构

- 前馈神经网络（全连接神经网络、多层感知器 MLP）
  - 各神经元分别属于不同的层，层内无连接。
  - 相邻两层之间的神经元全部两两连接。
  - 整个网络中无反馈，信号从输入层向输出层单向传播，可用一个有向无环图表示。



信息传递过程



前馈网络

- 给定一个前馈神经网络，用下面的记号来描述这样网络：

  - L：表示神经网络的层数；
  - n^l：表示第l层神经元的个数；
  - $f_l(\cdot)$：表示l层神经元的激活函数；
  - $W^{(l)} \in \mathbb{R}^{n^l \times n^{l-1}}$：表示$l-1$层到第$l$层的权重矩阵；
  - $\mathbf{b}^{(l)} \in \mathbb{R}^{n^l}$：表示$l-1$层到第$l$层的偏置；
  - $\mathbf{z}^{(l)} \in \mathbb{R}^{n^l}$：表示$l$层神经元的净输入（净活性值）；
  - $\mathbf{a}^{(l)} \in \mathbb{R}^{n^l}$：表示$l$层神经元的输出（活性值）。

前馈网络

- 前馈神经网络通过下面公式进行信息传播。

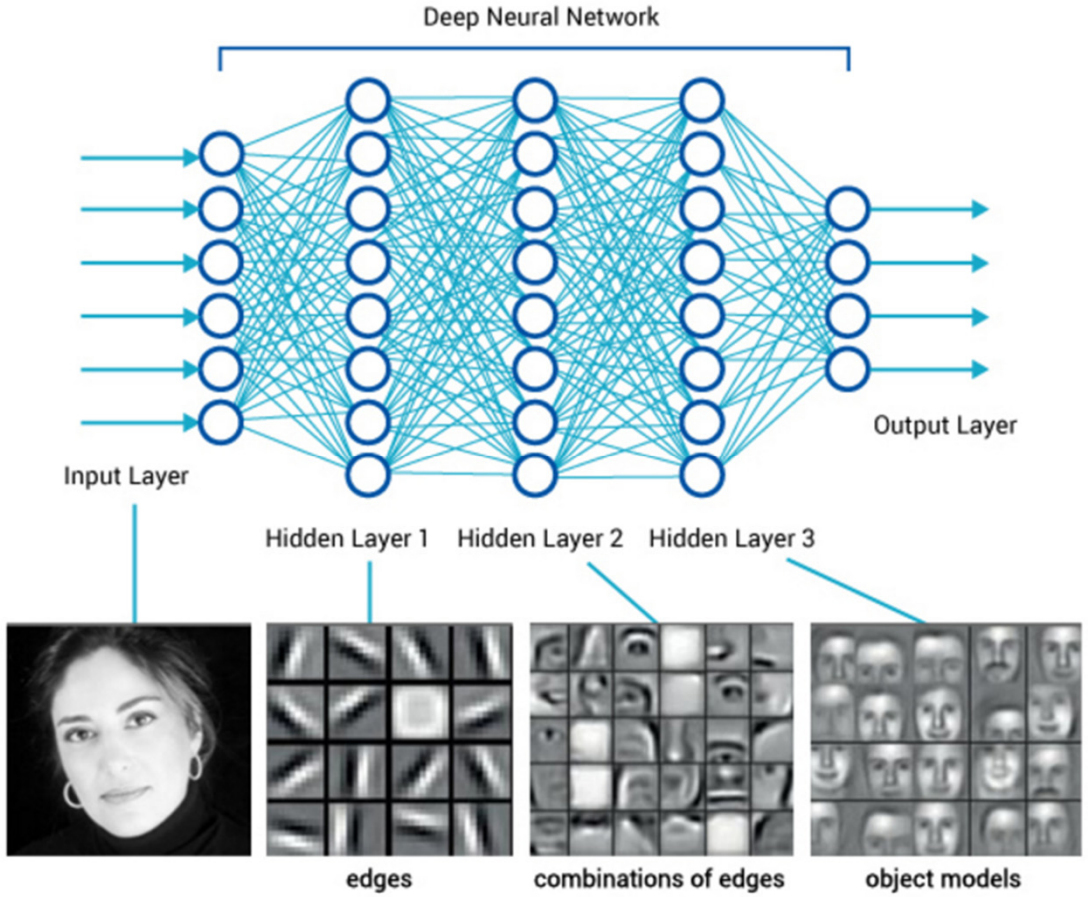
$$\mathbf{z}^{(l)} = W^{(l)} \cdot \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$$

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

- 前馈计算$\emptyset(x; W, b)$:

$$\mathbf{x} = \mathbf{a}^{(0)} \rightarrow \mathbf{z}^{(1)} \rightarrow \mathbf{a}^{(1)} \rightarrow \mathbf{z}^{(2)} \rightarrow \cdots \rightarrow \mathbf{a}^{(L-1)} \rightarrow \mathbf{z}^{(L)} \rightarrow \mathbf{a}^{(L)} = f(\mathbf{x}; W, \mathbf{b})$$

深层前馈神经网络


Deep Neural Network
Input Layer
Hidden Layer 1
Hidden Layer 2
Hidden Layer 3
Output Layer
edges
combinations of edges
object models

通用近似定理

定理 4.1 – 通用近似定理（Universal Approximation Theorem）[Cybenko, 1989, Hornik et al., 1989]： 令 $\varphi(\cdot)$ 是一个非常数、有界、单调递增的连续函数，$\mathcal{I}_d$ 是一个 d 维的单位超立方体 $[0,1]^d$，$C(\mathcal{I}_d)$ 是定义在 $\mathcal{I}_d$ 上的连续函数集合。对于任何一个函数 $f \in C(\mathcal{I}_d)$，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 $\mathbf{w}_i \in \mathbb{R}^d$，$i = 1, \cdots, m$，以至于我们可以定义函数

$$F(\mathbf{x}) = \sum_{i=1}^{m} v_i \varphi(\mathbf{w}_i^{\mathrm{T}} \mathbf{x} + b_i), \tag{4.33}$$

作为函数 f 的近似实现，即

$$|F(\mathbf{x}) - f(\mathbf{x})| < \epsilon, \forall \mathbf{x} \in \mathcal{I}_d. \tag{4.34}$$

其中 $\epsilon > 0$ 是一个很小的正数。

根据通用近似定理，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前馈神经网络，只要其隐藏层神经元的数量足够，它可以以任意的精度来近似任何从一个定义在实数空间中的有界闭集函数。

应用到机器学习

- 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。

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

分类器　　神经网络

- 如果$g(\cdot)$为logistic回归，那么logistic回归分类器可以看成神经网络的最后一层，神经网络直接输出类别y=1的条件概率 p(y=1|x)=a^L

应用到机器学习

- 对于多类分类问题
  - 如果使用softmax回归分类器，相当于网络最后一层设置C 个神经元，其输出经过softmax函数进行归一化后可以作为每个类的条件概率。

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{z}^{(L)})$$

  - 采用交叉熵损失函数，对于样本(x,y)，其损失函数为

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

参数学习

- 给定训练集为$D = \{(\boldsymbol{x}^{(n)}, y^{(n)})\}_{n=1}^{N}$，将每个样本$\boldsymbol{x}^{(n)}$输入给前馈神经网络，得到网络输出为$\hat{y}^{(n)}$，其在数据集D上的结构化风险函数为：

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

- 梯度下降（每次迭代时第l层W和b的更新）

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

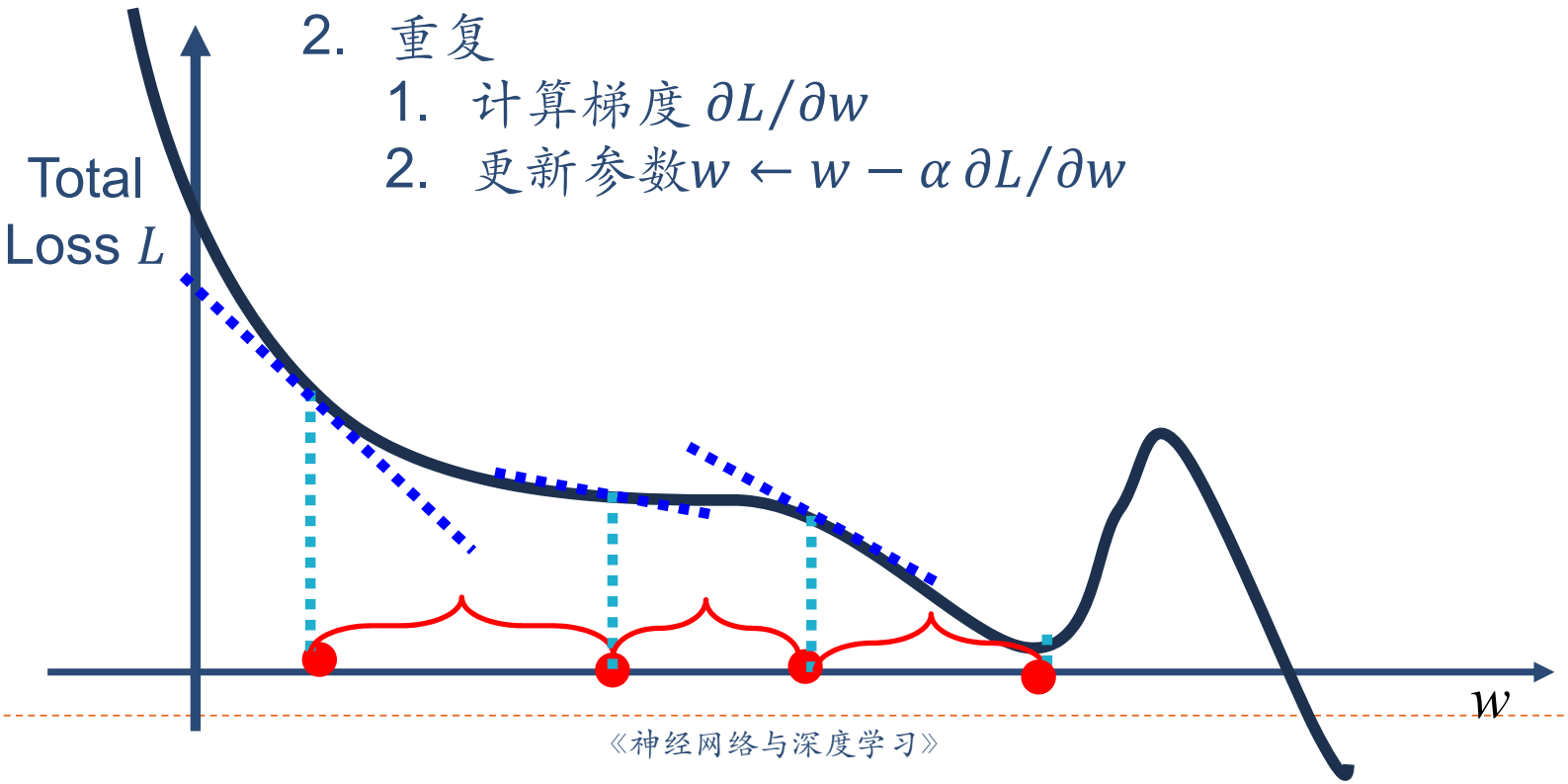
$$\mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} - \alpha\frac{\partial\mathcal{R}(W, \mathbf{b})}{\partial \mathbf{b}^{(l)}}$$

梯度下降

网络参数w

梯度：$\frac{\partial \mathrm{L}(w)}{\partial w}=\lim_{\Delta w \to 0} \frac{\mathrm{L}(w+\Delta w)}{\Delta w}$

1. 初始化w
2. 重复
   1. 计算梯度 $\partial L/\partial w$
   2. 更新参数 $w \leftarrow w-\alpha\, \partial L/\partial w$



4.4 反向传播算法

如何计算梯度?

- 神经网络为一个复杂的复合函数
  - 链式法则

$$y = f^5(f^4(f^3(f^2(f^1(x))))) \rightarrow \frac{\partial y}{\partial x} = \frac{\partial f^1}{\partial \mathrm{x}} \frac{\partial f^2}{\partial f^1} \frac{\partial f^3}{\partial f^2} \frac{\partial f^4}{\partial f^3} \frac{\partial f^5}{\partial f^4}$$

- 反向传播算法
  - 根据前馈网络的特点而设计的高效方法

- 一个更加通用的计算方法
  - 自动微分（Automatic Differentiation，AD）

矩阵微积分

- 矩阵微积分（Matrix Calculus）是多元微积分的一种表达方式，即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 $$\frac{\partial \mathbf{y}}{\partial \mathrm{x}}=\left[\frac{\partial \mathrm{y}_1}{\partial \mathrm{x}}, \cdots, \frac{\partial \mathrm{y}_\mathrm{q}}{\partial \mathrm{x}}\right]$$
  - 标量关于向量的偏导数 $$\frac{\partial y}{\partial \boldsymbol{x}}=\left[\frac{\partial y}{\partial x_1}, \cdots, \frac{\partial y}{\partial x_p}\right]^{\mathrm{T}}$$
  - 向量关于向量的偏导数 $$\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}}=\begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_q}{\partial x_1} \\ \vdots & \vdots & \vdots \\ \frac{\partial y_1}{\partial x_p} & \cdots & \frac{\partial y_q}{\partial x_p} \end{bmatrix} \in \mathbb{R}^{p \times q}$$

链式法则

- 链式法则（Chain Rule）是在微积分中求复合函数导数的一种常用方法。（向量的微分链式法则）

（1）若 $x \in \mathbb{R}, \boldsymbol{u} = u(x) \in \mathbb{R}^s, \boldsymbol{g} = g(\boldsymbol{u}) \in \mathbb{R}^t$，则

$$\frac{\partial \boldsymbol{g}}{\partial x} = \frac{\partial \boldsymbol{u}}{\partial x} \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{u}} \quad \in \mathbb{R}^{1 \times t}.$$

（2）若 $\boldsymbol{x} \in \mathbb{R}^p, \boldsymbol{y} = g(\boldsymbol{x}) \in \mathbb{R}^s, \boldsymbol{z} = f(\boldsymbol{y}) \in \mathbb{R}^t$，则

$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{y}} \quad \in \mathbb{R}^{p \times t}.$$

（3）若 $X \in \mathbb{R}^{p \times q}$ 为矩阵，$\boldsymbol{y} = g(X) \in \mathbb{R}^s, z = f(\boldsymbol{y}) \in \mathbb{R}$，则

$$\frac{\partial z}{\partial X_{ij}} = \frac{\partial \boldsymbol{y}}{\partial X_{ij}} \frac{\partial z}{\partial \boldsymbol{y}} \quad \in \mathbb{R}.$$

反向传播算法（BP）

给定一个样本(x,y)，假设神经网络输出为 $\hat{\mathbf{y}}$，损失函数为 $\mathrm{L}\left(\mathbf{y},\hat{\mathbf{y}}\right)$，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

计算第l层中参数 $\mathbf{W}^{(l)}$ 和 $\mathbf{b}^{(l)}$

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

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

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

$$\begin{aligned}\frac{\partial \mathbf{z}^{(l)}}{\partial w_{ij}^{(l)}} &= \left[\frac{\partial z_1^{(l)}}{\partial w_{ij}^{(l)}}, \cdots, \frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l)}}, \cdots, \frac{\partial z_{m^{(l)}}^{(l)}}{\partial w_{ij}^{(l)}}\right] \\ &= \left[0, \cdots, \frac{\partial(\mathbf{w}_{i:}^{(l)}\mathbf{a}^{(l-1)} + b_i^{(l)})}{\partial w_{ij}^{(l)}}, \cdots, 0\right] \\ &= \left[0, \cdots, a_j^{(l-1)}, \cdots, 0\right] \\ &\triangleq \mathbb{I}_i(a_j^{(l-1)}) \quad \in \mathbb{R}^{m^{(l)}},\end{aligned}$$

$$\delta^{(l)} = \frac{\partial \mathcal{L}(\mathbf{y},\hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}} \quad \in \mathbb{R}^{m^{(l)}}$$ 误差项

$$\frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}} = \mathbf{I}_{m^{(l)}} \quad \in \mathbb{R}^{m^{(l)} \times m^{(l)}}$$

反向传播算法（BP）

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

$$\begin{aligned}
\frac{\partial \mathbf{z}^{(l)}}{\partial w_{ij}^{(l)}} &= \left[\frac{\partial z_1^{(l)}}{\partial w_{ij}^{(l)}}, \cdots, \frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l)}}, \cdots, \frac{\partial z_{m^{(l)}}^{(l)}}{\partial w_{ij}^{(l)}}\right] \\
&= \left[0, \cdots, \frac{\partial(\mathbf{w}_{i:}^{(l)}\mathbf{a}^{(l-1)} + b_i^{(l)})}{\partial w_{ij}^{(l)}}, \cdots, 0\right] \\
&= \left[0, \cdots, a_j^{(l-1)}, \cdots, 0\right] \\
&\triangleq \mathbb{I}_i(a_j^{(l-1)}) \quad \in \mathbb{R}^{m^{(l)}},
\end{aligned}$$

$$\frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}} = \mathbf{I}_{m^{(l)}} \quad \in \mathbb{R}^{m^{(l)} \times m^{(l)}}$$

计算误差项

$$\delta^{(l)} = \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}} \in \mathbb{R}^{m^{(l)}}$$

- 该偏导数表示第l层神经元对最终损失的影响，也反映了最终损失对第l层神经元的敏感程度，因此一般称为第l层神经元的**误差项**

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

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

- 误差项也反映了不同神经元对网络能力的贡献程度，从而较好的解决了**贡献度分配问题** CAP

按位计算

$$\frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} = \frac{\partial f_l(\mathbf{z}^{(l)})}{\partial \mathbf{z}^{(l)}} = \mathrm{diag}(f_l'(\mathbf{z}^{(l)}))$$

$$\frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{a}^{(l)}} = (W^{(l+1)})^{\mathrm{T}}$$

$$\frac{\partial \mathbf{A}\mathbf{x}}{\partial \mathbf{x}} = \mathbf{A}^{\mathrm{T}}$$

$$\begin{aligned}
\delta^{(l)} &\triangleq \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}} \\
&= \frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} \cdot \frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{a}^{(l)}} \cdot \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l+1)}} \\
&= \mathrm{diag}(f_l'(\mathbf{z}^{(l)})) \cdot (W^{(l+1)})^{\mathrm{T}} \cdot \delta^{(l+1)} \\
&= f_l'(\mathbf{z}^{(l)}) \odot \left((W^{(l+1)})^{\mathrm{T}} \delta^{(l+1)}\right),
\end{aligned}$$

- 可以看出第l层的误差项可以通过第l+1层的误差项计算得到，此即是误差的反向传播
- 亦即第l层的一个神经元的误差项是所有与该神经元相连的第l+1层的神经元的误差项的权重和，然后再乘上该神经元激活函数的梯度

反向传播算法

在计算出上面三个偏导数之后，公式 (4.49) 可以写为

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial w_{ij}^{(l)}} = \mathbb{I}_i(a_j^{(l-1)})\delta^{(l)} = \delta_i^{(l)} a_j^{(l-1)}.$$

进一步，$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$ 关于第 l 层权重 $W^{(l)}$ 的梯度为

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

同理，$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$ 关于第 l 层偏置 $\mathbf{b}^{(l)}$ 的梯度为

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{b}^{(l)}} = \delta^{(l)}.$$

反向传播算法

计算最后一层（输出层）的误差项 $\delta^{(L)}=\dfrac{\partial L\left(\mathbf{y},\hat{\mathbf{y}}\right)}{\partial \mathbf{z}^{(L)}}$

$$\hat{\mathbf{y}}=\mathbf{a}^{(L)}=f_L\left(\mathbf{z}^{(L)}\right)$$

$$\delta^{(L)}=\frac{\partial L\left(\mathbf{y},\hat{\mathbf{y}}\right)}{\partial \mathbf{z}^{(L)}}$$

$$=\frac{\partial f_L\left(\mathbf{z}^{(L)}\right)}{\partial \mathbf{z}^{(L)}}\frac{\partial L\left(\mathbf{y},\hat{\mathbf{y}}\right)}{\partial \hat{\mathbf{y}}}$$

$$=\operatorname{diag}\left(f_L'\left(\mathbf{z}^{(L)}\right)\right)\frac{\partial L\left(\mathbf{y},\hat{\mathbf{y}}\right)}{\partial \hat{\mathbf{y}}}$$

$$=f_L'\left(\mathbf{z}^{(L)}\right)\odot\frac{\partial L\left(\mathbf{y},\hat{\mathbf{y}}\right)}{\partial \hat{\mathbf{y}}}$$

使用反向传播算法的随机梯度下降训练过程

算法 4.1: 使用反向传播算法的随机梯度下降训练过程

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

随机初始化 $W, \boldsymbol{b}$;

repeat

  对训练集 $\mathcal{D}$ 中的样本随机重排序;

  for $n = 1 \cdots N$ **do**

    从训练集 $\mathcal{D}$ 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 $\delta^{(l)}$; // 公式 (4.63)

    // 计算每一层参数的导数

    $\forall l, \quad \frac{\partial \mathcal{L}(\boldsymbol{y}^{(n)}, \hat{\boldsymbol{y}}^{(n)})}{\partial W^{(l)}} = \delta^{(l)} (\boldsymbol{a}^{(l-1)})^{\mathrm{T}}$; // 公式 (4.65)

    $\forall l, \quad \frac{\partial \mathcal{L}(\boldsymbol{y}^{(n)}, \hat{\boldsymbol{y}}^{(n)})}{\partial \boldsymbol{b}^{(l)}} = \delta^{(l)}$; // 公式 (4.66)

    // 更新参数

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

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

  end

until 神经网络模型在验证集 $\mathcal{V}$ 上的错误率不再下降;

输出: $W, \boldsymbol{b}$

4.5 自动梯度计算

自动梯度计算

手动求导转换为计算机程序的过程非常繁琐且容易出错。目前，可以使用计算机实现自动梯度计算，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，以减少截断误差

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

- ✓Δx 难以确定
- ✓实用性较差
- ✓计算复杂度高

自动梯度计算

- 符号微分（Symbolic Differentiation）

✓基于符号计算的自动求导方式（代数计算）

✓变量被视作符号，无需代入具体的值，输入输出都是数学表达式

✓包含基于规则的化简、因式分解、微分、积分等运算

例如：　输入　$3x-x+2x+1$

　　输出　$4x+1$

在实际应用中，经常使用下面的方式来计算梯度，以减少截断误差

■不足之处：

✓编译时间长

✓需要专门的数学计算语言

✓很难调试

自动梯度计算

- **自动微分（Automatic Differentiation）**

✓处理对象是一个函数或一段程序

✓将数值计算分解为基本操作，包含加减乘除和一些初等函数如exp/log/sin/cos等，然后利用链式法则来自动计算复合函数的梯度

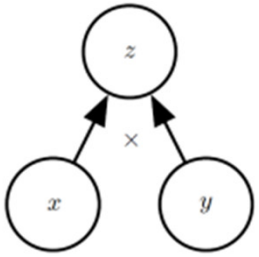
✓介于数值微分和符号微分：数值微分一开始就代数数值近似求解，而符号微分则直接对表达式进行求解

✓灵活性高：对用户透明；无需专门的数学语言和编程；采用图的方式进行计算

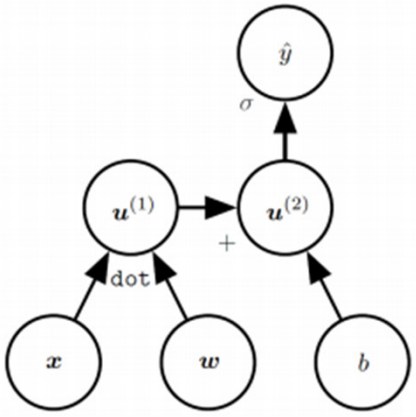
自动梯度计算

自动微分（Automatic Differentiation）

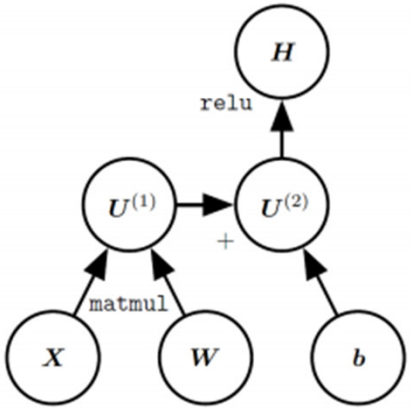
- 计算图
  - ✓将复合函数分解为一系列基本操作，并以图的形式连接起来
  - ✓数学运算的图结构表示，每个边代表一个基本操作，每个叶子节点代表一个输入变量或常量



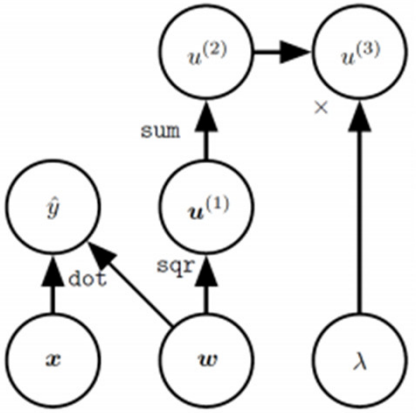
$z = xy$



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



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



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

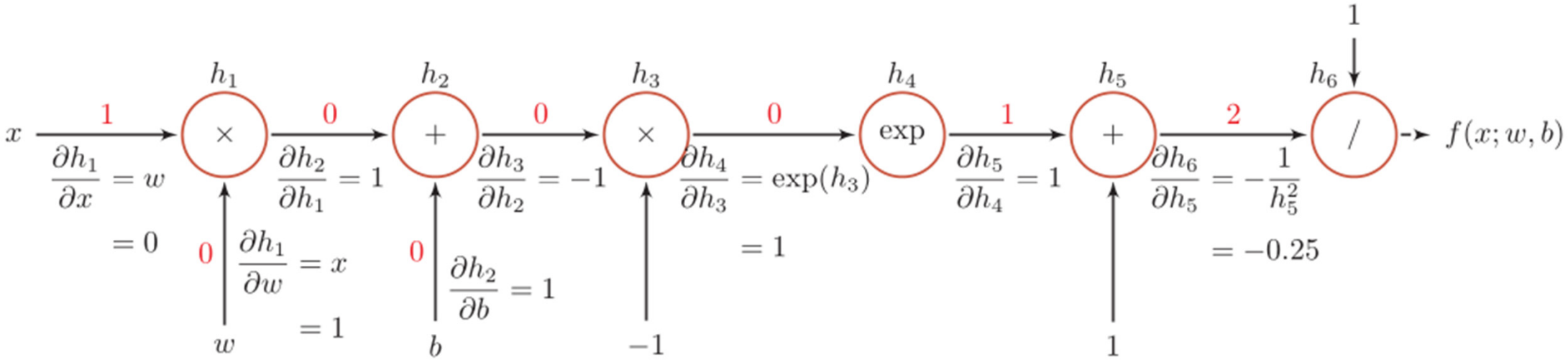
自动微分与计算图

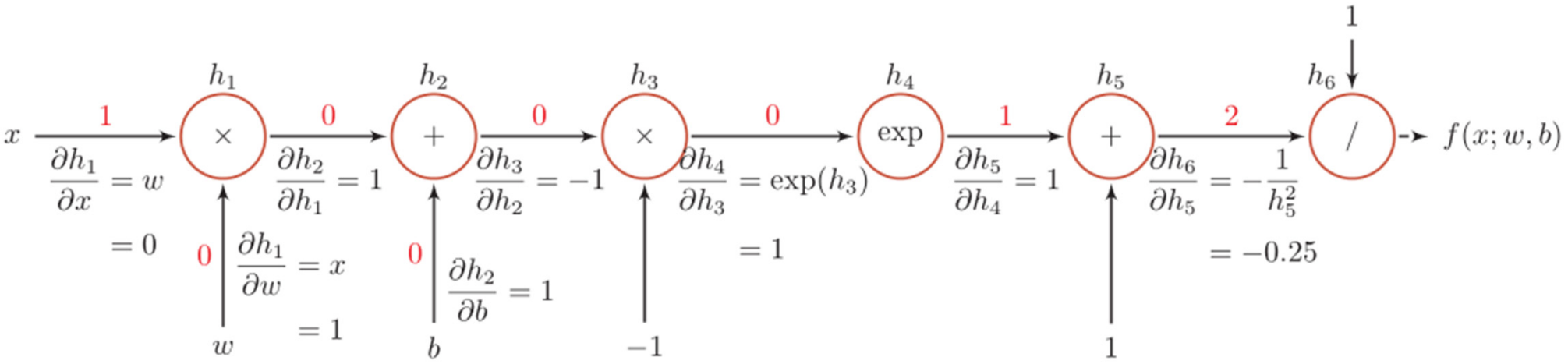
- 自动微分是利用链式法则来自动计算一个复合函数的梯度。

$$f(x;w,b)=\frac{1}{\exp(-(wx+b))+1}.$$

- 复合函数f(xlw,b)的计算图 computational graph

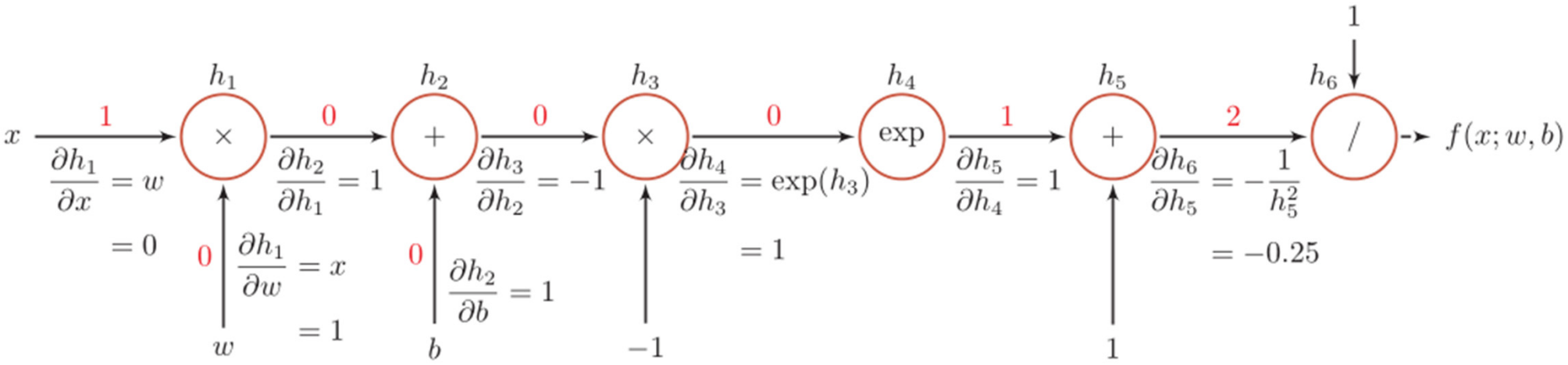
连边上的红色数字表示前向计算时复合函数中每个变量的实际取值 x=1,w=0,b=0





- 表中给出了f(x;w,b)的6个基本函数及其导数

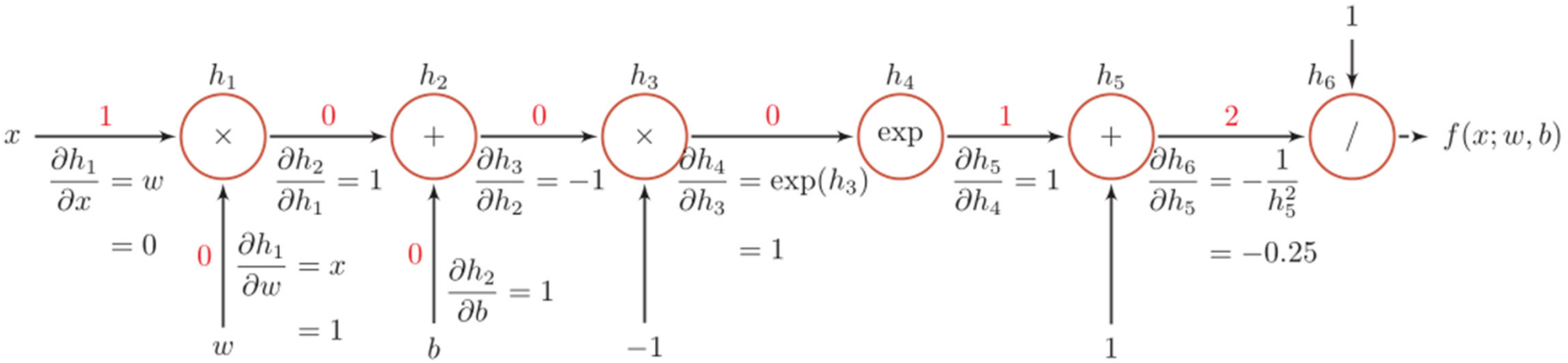
函数	导数	
$h_1 = x \times w$	$\frac{\partial h_1}{\partial w} = x$	$\frac{\partial h_1}{\partial x} = w$
$h_2 = h_1 + b$	$\frac{\partial h_2}{\partial h_1} = 1$	$\frac{\partial h_2}{\partial b} = 1$
$h_3 = h_2 \times -1$	$\frac{\partial h_3}{\partial h_2} = -1$	
$h_4 = \exp(h_3)$	$\frac{\partial h_4}{\partial h_3} = \exp(h_3)$	
$h_5 = h_4 + 1$	$\frac{\partial h_5}{\partial h_4} = 1$	
$h_6 = 1/h_5$	$\frac{\partial h_6}{\partial h_5} = -\frac{1}{h_5^2}$	



- 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；

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

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



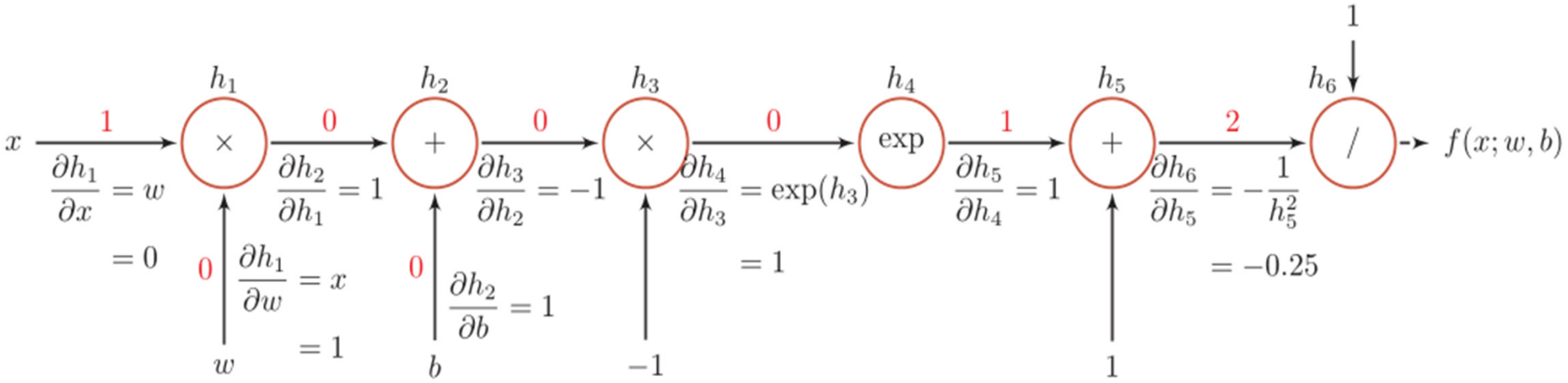
- f(x;w,b)关于参数w和b的导数可以通过计算图上的路径上的所有导数连乘得到；
  - ✓ 以对w求导为例，当x = 1,w = 0,b = 0时，可以得到

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

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

$$= 0.25.$$

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



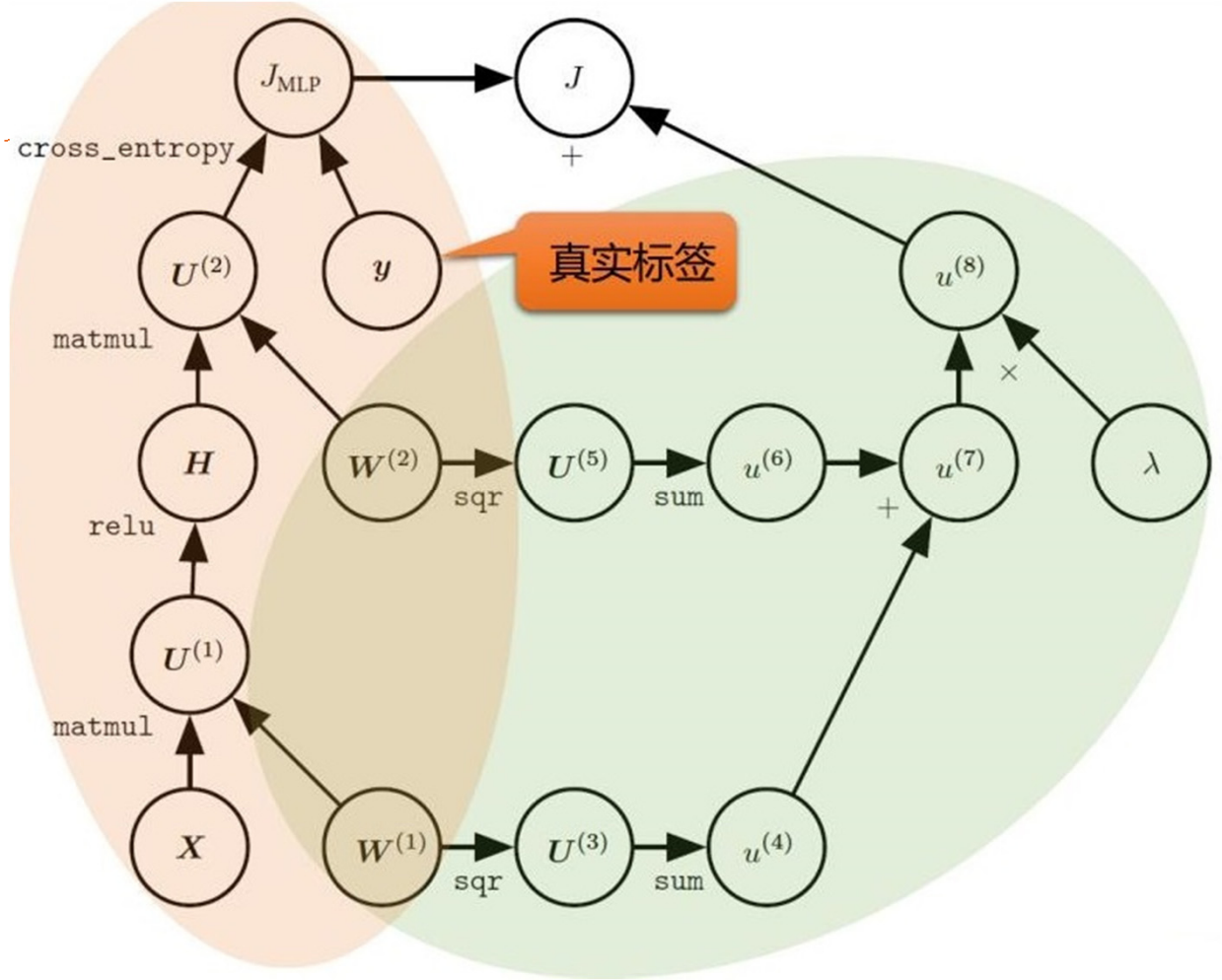
$$
\begin{aligned}
\frac{\partial h_1}{\partial w} &= x = 1 \\
\frac{\partial h_2}{\partial w} &= \frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial w} = 1 \times 1 = 1 \\
\frac{\partial h_3}{\partial w} &= \frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial w} = -1 \times 1 \\
&\vdots \qquad\qquad \vdots \\
\frac{\partial h_6}{\partial w} &= \frac{\partial h_6}{\partial h_5}\frac{\partial h_5}{\partial w} = -0.25 \times -1 = 0.25 \\
\frac{\partial f(x;w,b)}{\partial w} &= \frac{\partial f(x;w,b)}{\partial h_6}\frac{\partial h_6}{\partial w} = 1 \times 0.25 = 0.25
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial f(x;w,b)}{\partial h_6} &= 1 \\
\frac{\partial f(x;w,b)}{\partial h_5} &= \frac{\partial f(x;w,b)}{\partial h_6}\frac{\partial h_6}{\partial h_5} = 1 \times -0.25 \\
\frac{\partial f(x;w,b)}{\partial h_4} &= \frac{\partial f(x;w,b)}{\partial h_5}\frac{\partial h_5}{\partial h_4} = -0.25 \times 1 = -0.25 \\
&\vdots \qquad\qquad \vdots \\
\frac{\partial f(x;w,b)}{\partial w} &= \frac{\partial f(x;w,b)}{\partial h_1}\frac{\partial h_1}{\partial w} = 0.25 \times 1 = 0.25
\end{aligned}
$$

计算图实例

正则化单隐层MLP计算图

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



引用 https://rmcong.github.io/DLCourse/chapter4.pdf

静态计算图和动态计算图

■静态计算图

✓在编译时构建计算图，计算图构建好之后在程序运行时不能改变。

- Theano和Tensorflow

✓构建时可以进行优化、并行能力强

✓灵活性差

■动态计算图

✓在程序运行时动态构建

- DyNet，Chainer和PyTorch

✓不容易优化，当不同输入的网络结构不一致时，难以并行计算

✓灵活性比较高。

4.6 优化问题

优化问题

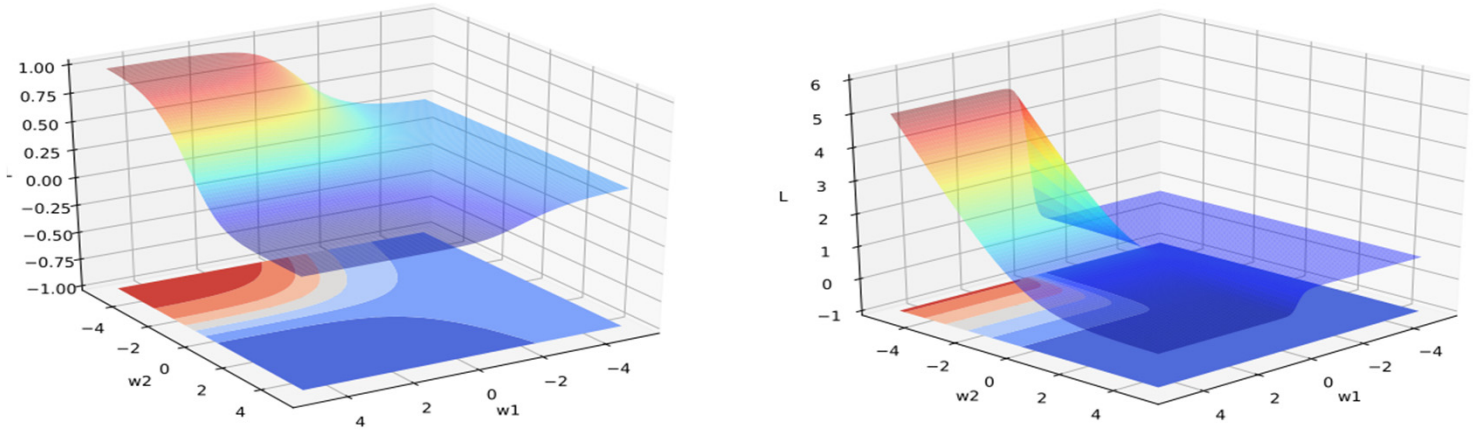
- 难点
  - 参数过多，影响训练
  - 非凸优化问题：即存在局部最优而非全局最优解，影响迭代
  - 梯度消失问题，下层参数比较难调
  - 参数解释起来比较困难
- 需求
  - 计算资源要大
  - 数据要多
  - 算法效率要好：即收敛快

优化问题

非凸优化问题

以最简单的1-1-1结构的两层神经网络为例 $y = \sigma(w_2\sigma(w_1 x))$,

w1和w2是参数，则输入样本(x,y)=(1,1)，对应的平方误差损失和交叉熵损失与参数的关系如下图



(a) 平方误差损失　　(b) 交叉熵损失

图 4.9 神经网络 $y = \sigma(w_2\sigma(w_1 x))$ 的损失函数

优化问题

梯度消失问题（Vanishing Gradient Problem）梯度弥散问题

- 误差反向传播算法

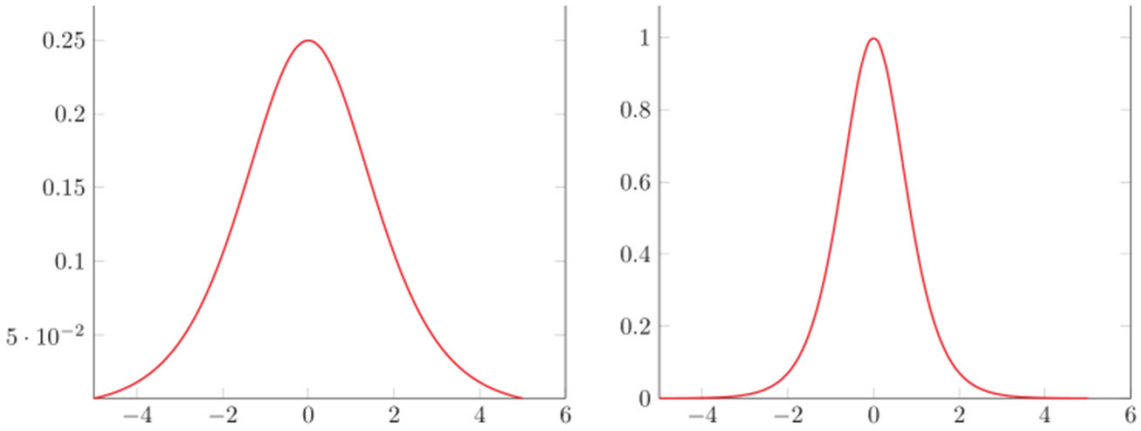
$$\delta^{(l)} = f_l'\left(z^{(l)}\right) \odot \left(\mathbf{W}^{(l+1)}\right)^{\mathrm{T}} \delta^{(l+1)}$$

- 误差反向传播时，在每一层都要乘以该层激活函数的导数

- 当采用sigmoid型激活函数时，导数为

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

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



(a) logistic 函数的导数　　(b) tanh 函数的导数

优化问题

梯度消失问题（Vanishing Gradient Problem）梯度弥散问题

- 当采用sigmoid型激活函数时，导数为

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

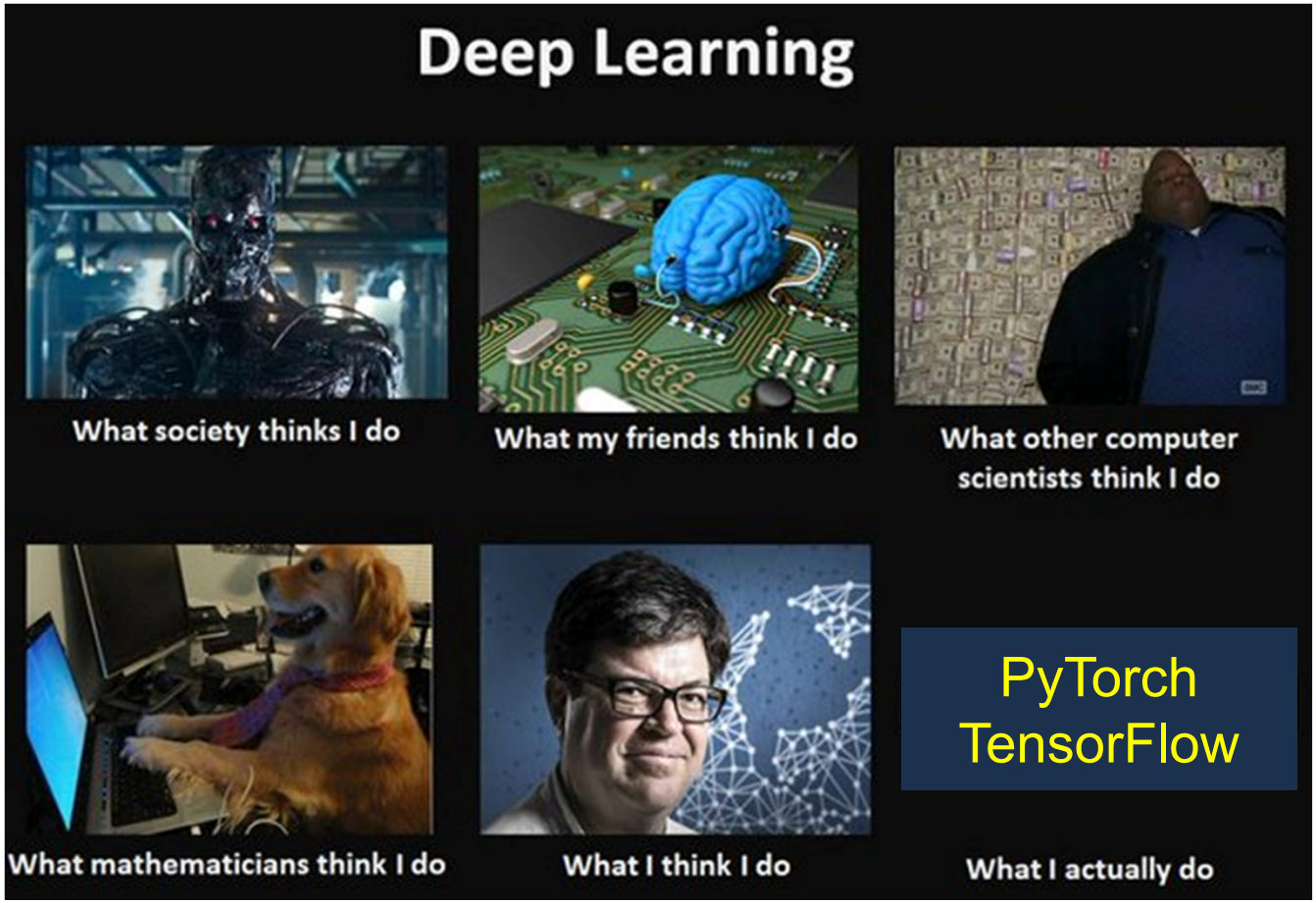
$$\tanh'(x)=1-(\tanh(x))^2\in[0,1]$$

$$y=f^5(f^4(f^3(f^2(f^1(x)))))$$

$$\frac{\partial y}{\partial x}=\frac{\partial f^1}{\partial x}\frac{\partial f^2}{\partial f^1}\frac{\partial f^3}{\partial f^2}\frac{\partial f^4}{\partial f^3}\frac{\partial f^5}{\partial f^4}$$

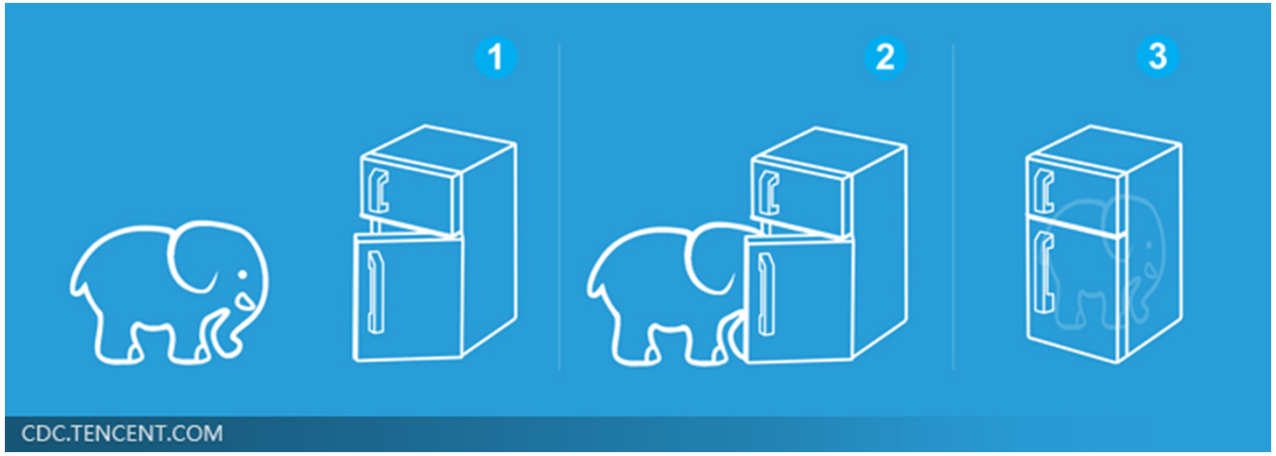
- 由于Sigmoid型函数的饱和性，饱和区导数趋近于0，如此，导数经过每一层都会衰减，随着层数的加深，造成了梯度消失，使得整个网络很难训练，这就是所谓的**梯度消失问题**，亦称**梯度弥散问题**

如何实现?


Deep Learning
What society thinks I do
What my friends think I do
What other computer scientists think I do
What mathematicians think I do
What I think I do
PyTorch
TensorFlow
What I actually do

深度学习的三个步骤

Deep Learning is so simple ……



总结

- 神经网络是一种典型的分布式并行处理模型，通过大量神经元之间的交互来处理信息，每一个神经元都发送兴奋或抑制信息到其它神经元。
- 神经网络中的激活函数一般为连续可导函数，选择合适的激活函数十分重要
- 前馈神经网络是一种类型简单的全连接神经网络（FCNN，Fully Connected Neural Network）或多层感知器。
- 前馈神经网络作为一种非线性模型，其能力可由通用近似定理来保证。通常，前馈神经网络由两层网络结构（隐藏层和输出层）构成，激活函数一般为sigmoid型函数，损失函数大多数为平方损失。

谢 谢