第4讲：前馈神经网络

师 佳

化学化工学院

化学工程与生物工程系

纲 要

- 神经网络概述
  - 概述
  - 神经元
  - 网络结构
- 前馈神经网络
  - 模型
  - 通用近似定理
  - 反向传播算法
  - 自动微分计算
  - 编程实现方法
  - 优化问题

神经网络概述

➢概述

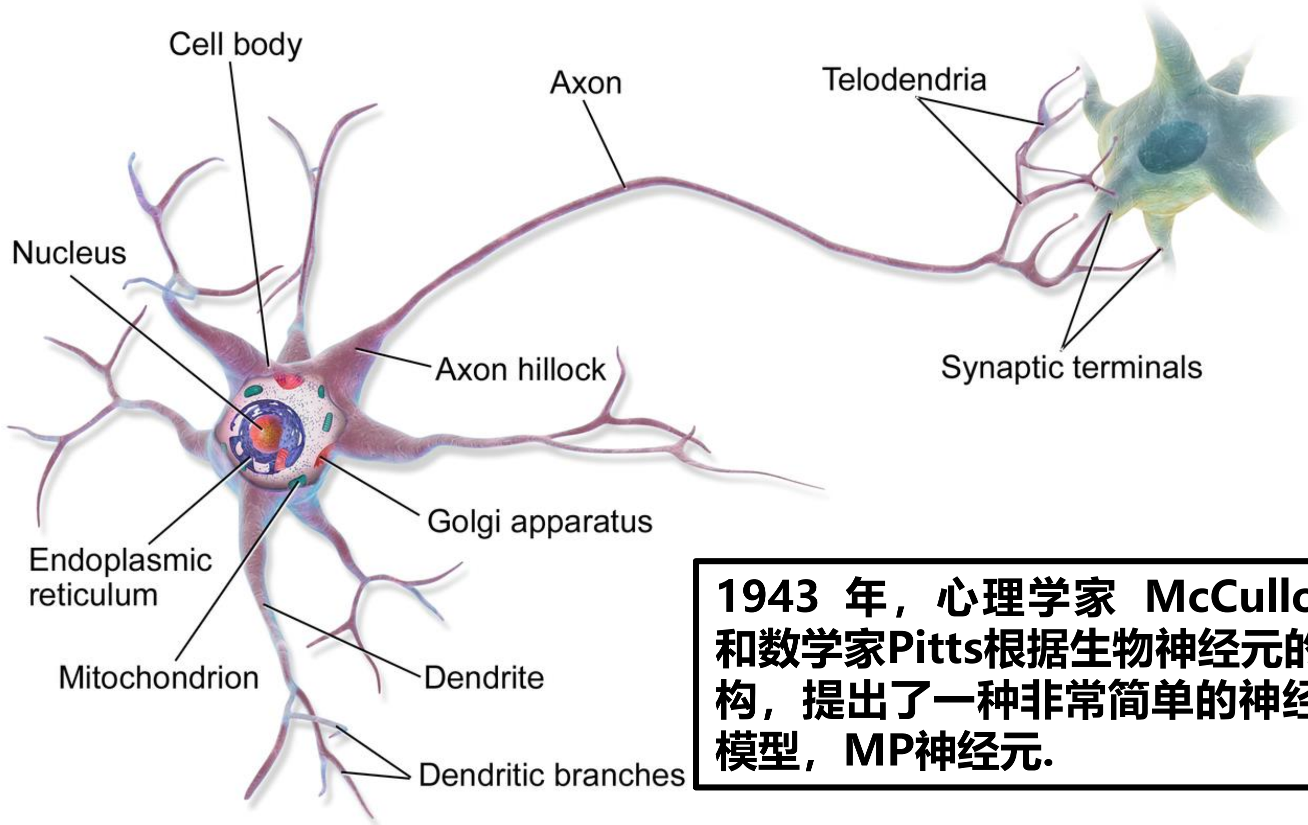
- 神经网络最早是作为一种主要的**连接主义模型**．诞生于20世纪80年代中后期，是最流行的一种**分布式并行处理(Parallel Distributed Processing，PDP)模型**，具有3个主要特征：
  - 信息表示是分布式的(非局部的)
  - 记忆和知识是存储在单元之间的连接上
  - 通过逐渐改变单元之间的连接强度来学习新的知识

- 从机器学习的角度来看，神经网络一般可以看作一个**非线性模型**，其基本组成单元为具有**非线性激活函数**的神经元，通过**大量神经元之间的连接**，使得神经网络成为一种**高度非线性的模型**.

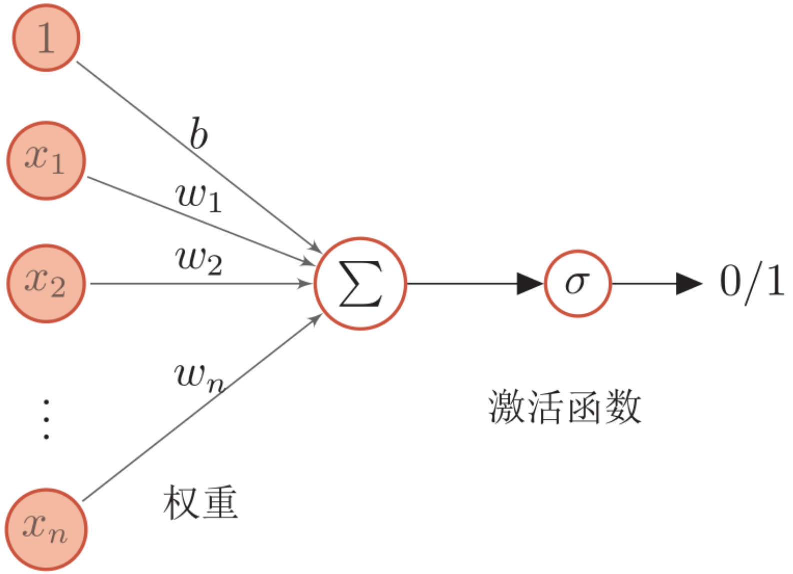
- 在引入**误差反向传播**来改进其学习能力之后，神经网络也越来越多地应用在各种机器学习任务上.

神经网络概述

➢神经元



1943 年，心理学家 McCulloch 和数学家Pitts根据生物神经元的结构，提出了一种非常简单的神经元模型，MP神经元.



➢ **净输入（Net Input）：**

$$z = \sum_{d=1}^{D} w_d x_d + b = \boldsymbol{w}^{\top}\boldsymbol{x} + b,$$

➢ **激活函数（Activation Function）：**

$$a = f(z)$$

神经网络概述

➢神经元

- ➢激活函数必须具备的性质：
  - ➢ **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.
- ➢常用的激活函数
  - ➢ **Sigmoid型函数**

1）Logistic函数

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

2）Tanh函数

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

➢ 性质：

1）饱和函数

2）连续可导

3）Tanh是零中心化

4）Logistic具有偏置偏移

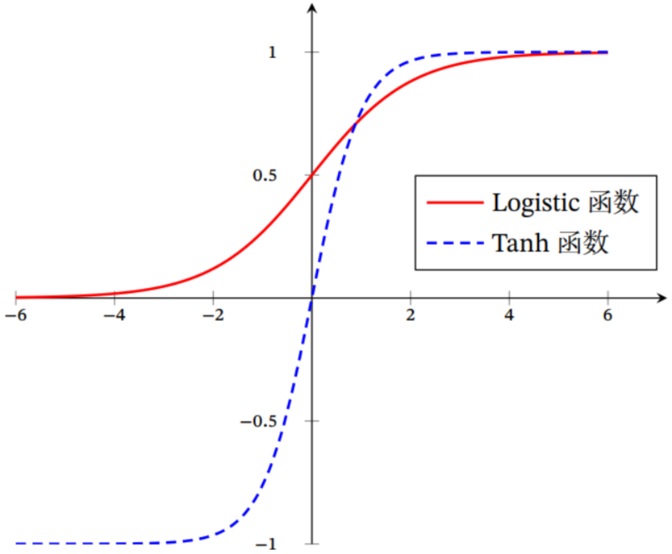


图 4.2 Logistic 函数和 Tanh 函数

神经网络概述

➢神经元

➢ 常用的激活函数

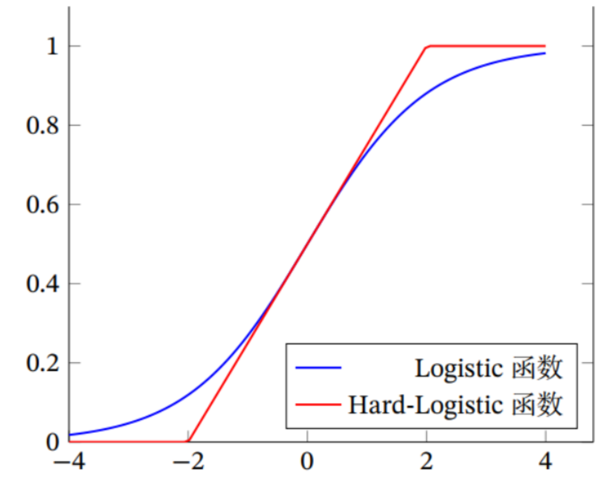
➢ Sigmoid型函数

1）Hard-Logistic函数

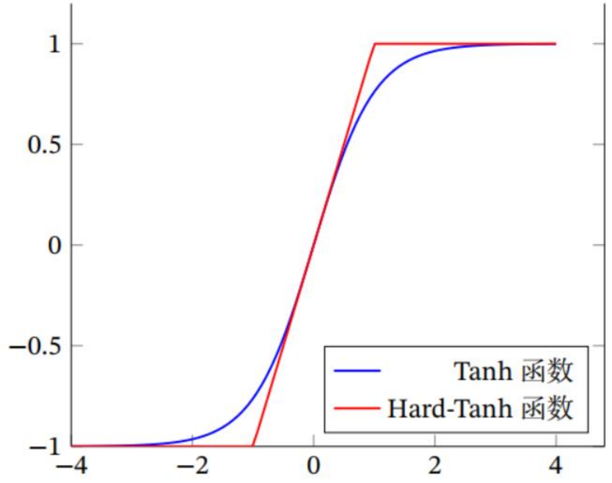
$$\text{hard-logistic}(x) = \max\left(\min(0.25x + 0.5, 1), 0\right)$$

2）Hard-Tanh函数

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



(a) Hard Logistic 函数



(b) Hard Tanh 函数

神经网络概述

➢神经元

➢常用的激活函数

➢ **ReLU(Rectified Linear Unit，修正线性单元)函数**

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

➢ **带泄露的ReLU(Leak ReLU)函数**

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

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

➢ **带参数的ReLU(Parametric ReLU)函数**

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

➢ **Softplus函数**

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

➢ **ELU(Exponential Linear Unit，指数线性单元)函数**

$$\mathrm{ELU}(x)=\begin{cases}x & \text{if } x>0\\ \gamma(\exp(x)-1) & \text{if } x\leq 0\end{cases}$$

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

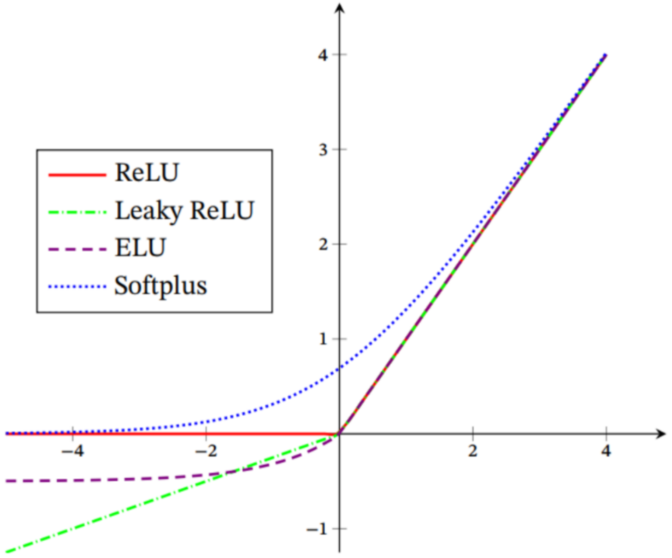


图 4.4　ReLU、Leaky ReLU、ELU 以及 Softplus 函数

神经网络概述

➢神经元

➢常用的激活函数

➢ **Swish 函数**

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

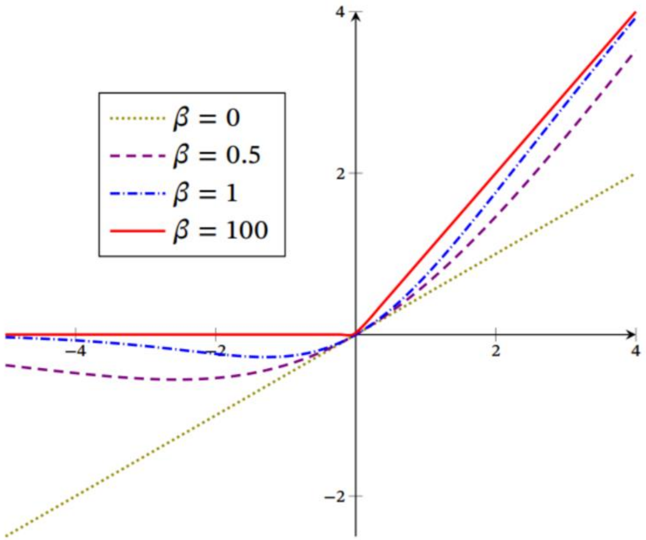


图 4.5 Swish 函数

线性函数和ReLU函数之间的非线性插值函数，其非线性程度由参数β 控制

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

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

其中$P(X \le x)$是高斯分布$\mathcal{N}(\mu,\sigma^2)$的累积分布函数

➢ **Maxout单元**

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

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

Maxout激活函数可以看作任意凸函数的分段线性近似，并且在有限的点上是不可微的.

神经网络概述

➢神经元

➢常用激活函数的导数

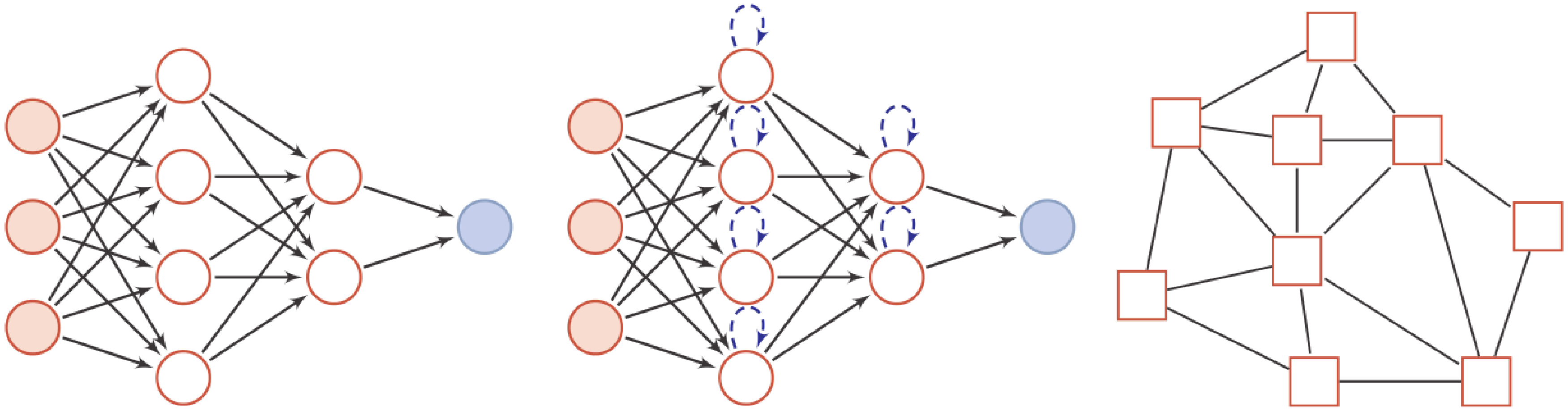
课后练习：证明

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 函数	$f(x)=\max(0,x)$	$f'(x)=I(x>0)$
ELU 函数	$f(x)=\max(0,x)+\min\big(0,\gamma(\exp(x)-1)\big)$	$f'(x)=I(x>0)+I(x\leq 0)\cdot\gamma\exp(x)$
SoftPlus 函数	$f(x)=\log\big(1+\exp(x)\big)$	$f'(x)=\frac{1}{1+\exp(-x)}$

神经网络概述

➢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**。



(a) 前馈网络　(b) 记忆网络　(c) 图网络

前馈网络可以看作一个**函数**，通过简单非线性函数的多次复合，实现输入空间到输出空间的复杂映射.	具有更强的计算和**记忆能力**.包括循环神经网络、Hopfield网络、玻尔兹曼机、受限玻尔兹曼机等.	是**前馈网络和记忆网络的泛化**，包含很多不同的实现方式，比如图卷积网络、图注意力网络、消息传递神经网络等.

前馈神经网络

➢模型

➢ **前馈神经网络中，各神经元分别属于不同的层．每一层的神经元可以接收前一层神经元的信号，并产生信号输出到下一层。**

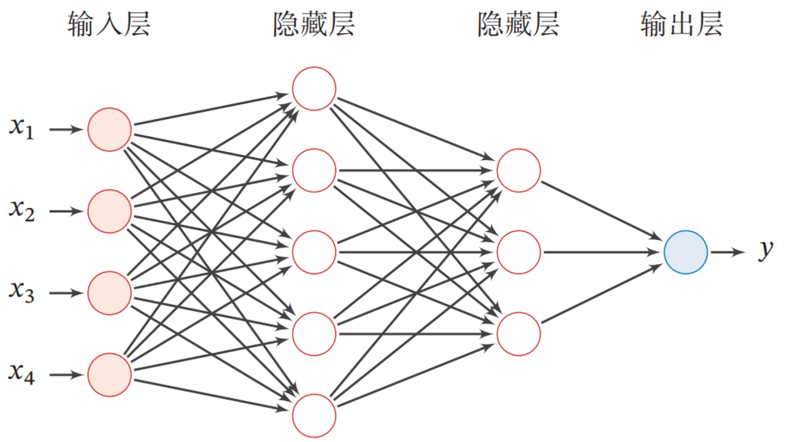


图 4.7 多层前馈神经网络

表 4.1 前馈神经网络的记号

记号	含义
L	神经网络的层数
M_l	第 l 层神经元的个数
$f_l(\cdot)$	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）

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

$$\boldsymbol{a}^{(l)} = f_l(\boldsymbol{z}^{(l)}). \qquad \boldsymbol{a}^{(0)} = \boldsymbol{x}$$

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

前馈神经网络

➢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 $\phi(\cdot)$ 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。对于任意给定的一个函数 $f \in C(\mathcal{I}_D)$，存在一个整数 M，和一组实数 $v_m, b_m \in \mathbb{R}$ 以及实数向量 $\boldsymbol{w}_m \in \mathbb{R}^D$，$m = 1, \cdots, M$，以至于我们可以定义函数

$$F(\boldsymbol{x}) = \sum_{m=1}^{M} v_m \phi(\boldsymbol{w}_m^{\mathsf{T}} \boldsymbol{x} + b_m), \tag{4.37}$$

作为函数 f 的近似实现，即

$$|F(\boldsymbol{x}) - f(\boldsymbol{x})| < \epsilon, \forall \boldsymbol{x} \in \mathcal{I}_D, \tag{4.38}$$

其中 $\epsilon > 0$ 是一个很小的正数。

- ➢ **只要隐藏层神经元的数量足够，可以以任意的精度来近似任何一个定义在实数空间 $\mathbb{R}^D$ 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.**

前馈神经网络

➢通用近似定理

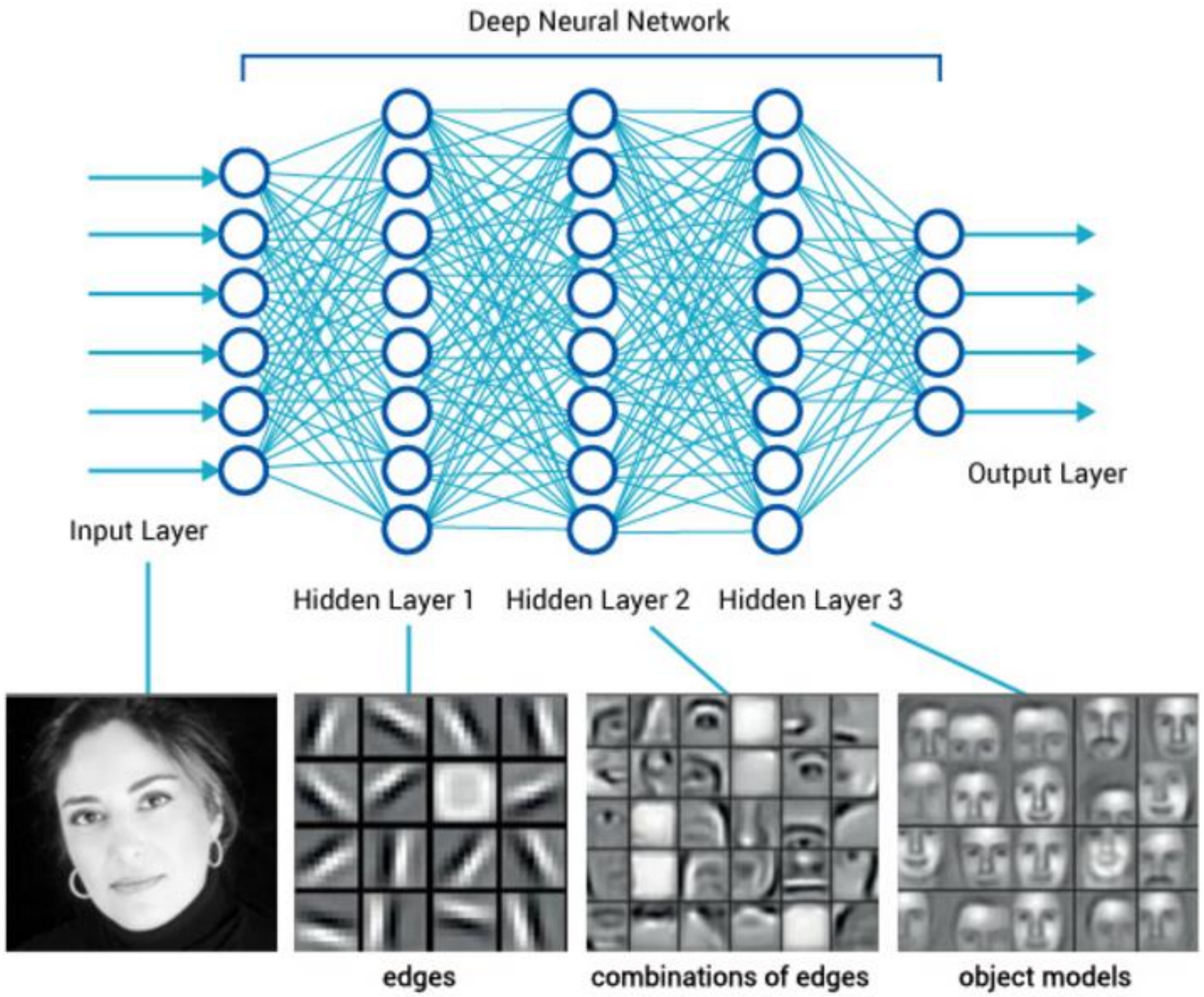
➢深层前馈神经网络

$$\boldsymbol{y} = f^{n}_{W_n}\left(\cdots f^{3}_{W_3}\left(f^{2}_{W_2}\left(f^{1}_{W_1}(\boldsymbol{x})\right)\right)\cdots\right) = \boldsymbol{f}_{\boldsymbol{W}}(\boldsymbol{x})$$

➢**多层前馈神经网络**可以看作一个非线性复合函数$\phi:\mathbb{R}^D\to\mathbb{R}^{D'}$，将输入$\boldsymbol{x}\in\mathbb{R}^D$映射到输出$\phi(\boldsymbol{x})\in\mathbb{R}^{D'}$．因此，多层前馈神经网络也可以看成是一种特征转换方法，其输出$\phi(\boldsymbol{x})$作为最后一层分类器的输入进行分类.

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

$$p(y=1|\boldsymbol{x}) = a^{(L)} \qquad \hat{\boldsymbol{y}} = \mathrm{softmax}(\boldsymbol{z}^{(L)})$$



前馈神经网络

➢反向传播算法

➢学习准则：结构化风险函数

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

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

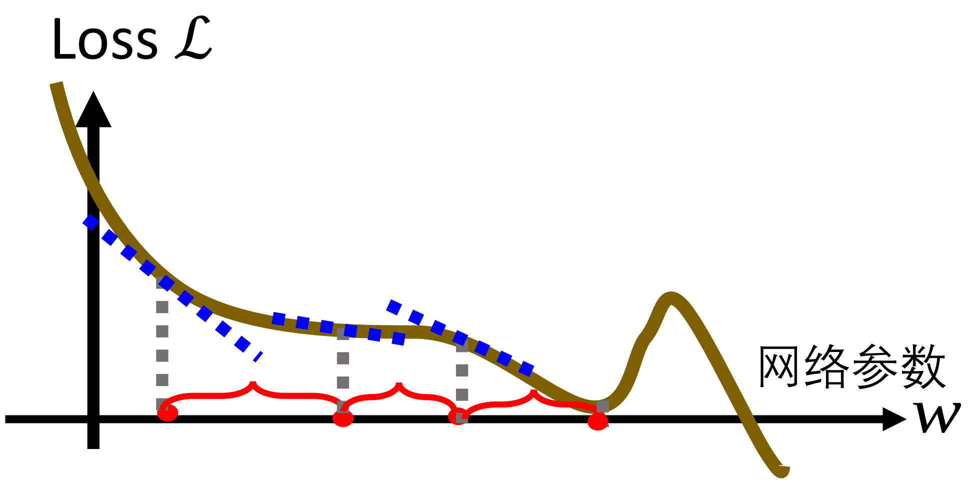
➢优化算法：梯度下降法

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

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

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

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



前馈神经网络

- ➢ **反向传播算法**
  - ➢ **梯度计算**

$$\frac{\partial \mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial w_{ij}^{(l)}} = \underbrace{\frac{\partial \boldsymbol{z}^{(l)}}{\partial w_{ij}^{(l)}} \frac{\partial \mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \boldsymbol{z}^{(l)}}}_{\text{分母布局}} = \underbrace{\frac{\partial \mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \boldsymbol{z}^{(l)}} \frac{\partial \boldsymbol{z}^{(l)}}{\partial w_{ij}^{(l)}}}_{\text{分子布局}}$$

$$= \left[\frac{\partial z_1^{(l)}}{\partial w_{ij}^{(l)}} \quad \frac{\partial z_2^{(l)}}{\partial w_{ij}^{(l)}} \quad \cdots \quad \frac{\partial z_{M_l}^{(l)}}{\partial w_{ij}^{(l)}} \right]$$

$$= \left[\frac{\partial L(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial z_1^{(l)}} \quad \frac{\partial L(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial z_2^{(l)}} \quad \cdots \quad \frac{\partial L(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial z_{M_l}^{(l)}} \right]^T$$

$$= \left[\frac{\partial L(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial z_1^{(l)}} \quad \frac{\partial L(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial z_2^{(l)}} \quad \cdots \quad \frac{\partial L(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial z_{M_l}^{(l)}} \right]$$

$$= \left[\frac{\partial z_1^{(l)}}{\partial w_{ij}^{(l)}} \quad \frac{\partial z_2^{(l)}}{\partial w_{ij}^{(l)}} \quad \cdots \quad \frac{\partial z_{M_l}^{(l)}}{\partial w_{ij}^{(l)}} \right]^T$$

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

$$= \begin{bmatrix} \frac{\partial z_1^{(l)}}{\partial b_1^{(l)}} & \frac{\partial z_2^{(l)}}{\partial b_1^{(l)}} & \cdots & \frac{\partial z_{M_l}^{(l)}}{\partial b_1^{(l)}} \\ \frac{\partial z_1^{(l)}}{\partial b_2^{(l)}} & \frac{\partial z_2^{(l)}}{\partial b_2^{(l)}} & \cdots & \frac{\partial z_{M_l}^{(l)}}{\partial b_2^{(l)}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial z_1^{(l)}}{\partial b_{M_l}^{(l)}} & \frac{\partial z_2^{(l)}}{\partial b_{M_l}^{(l)}} & \cdots & \frac{\partial z_{M_l}^{(l)}}{\partial b_{M_l}^{(l)}} \end{bmatrix}$$

前馈神经网络

➢反向传播算法

➢梯度计算

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

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

$$= \left[0, \cdots, \boxed{\frac{\partial(\boldsymbol{w}_{i:}^{(l)}\boldsymbol{a}^{(l-1)} + b_i^{(l)})}{\partial w_{ij}^{(l)}}}, \cdots, 0\right]$$

$$= \left[0, \cdots, a_j^{(l-1)}, \cdots, 0\right]$$

$$\triangleq \mathbb{I}_i(a_j^{(l-1)}) \quad \in \mathbb{R}^{1\times M_l},$$

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

$$\frac{\partial \boldsymbol{z}^{(l+1)}}{\partial \boldsymbol{a}^{(l)}} = (\boldsymbol{W}^{(l+1)})^{\intercal} \quad \in \mathbb{R}^{M_l\times M_{l+1}}$$

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

$$\delta^{(l)} \triangleq \frac{\partial \mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial \boldsymbol{z}^{(l)}}$$

误差项

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

$$= \boxed{\mathrm{diag}(f_l'(\boldsymbol{z}^{(l)}))} \cdot \boxed{(\boldsymbol{W}^{(l+1)})^{\intercal}} \cdot \boxed{\delta^{(l+1)}}$$

$$= f_l'(\boldsymbol{z}^{(l)}) \odot \left((\boldsymbol{W}^{(l+1)})^{\intercal}\delta^{(l+1)}\right) \quad \in \mathbb{R}^{M_l}$$

反向传播（BP）

前馈神经网络

➢反向传播算法

➢梯度计算

$$
\begin{aligned}
\frac{\partial \mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial w_{ij}^{(l)}} &= \frac{\partial \boldsymbol{z}^{(l)}}{\partial w_{ij}^{(l)}} \frac{\partial \mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \boldsymbol{z}^{(l)}} \\
&= \mathbb{I}_i(a_j^{(l-1)})\delta^{(l)} \\
&= [0,\cdots,a_j^{(l-1)},\cdots,0][\delta_1^{(l)},\cdots,\delta_i^{(l)},\cdots,\delta_{M_l}^{(l)}]^\top \\
&= \delta_i^{(l)} a_j^{(l-1)},
\end{aligned}
$$

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

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

$$\delta^{(l)} = f_l'(\boldsymbol{z}^{(l)}) \odot \left((\boldsymbol{W}^{(l+1)})^\top \delta^{(l+1)}\right) \quad \in \mathbb{R}^{M_l}$$

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, $1 \le l \le L$.

```
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);                         // 公式 (4.63)
        // 计算每一层参数的导数
        ∀l,  ∂L(y^(n), ŷ^(n)) / ∂W^(l) = δ^(l) (a^(l−1))^T;      // 公式 (4.68)
        ∀l,  ∂L(y^(n), ŷ^(n)) / ∂b^(l) = δ^(l);                   // 公式 (4.69)
        // 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 神经网络模型在验证集 V 上的错误率不再下降;
```

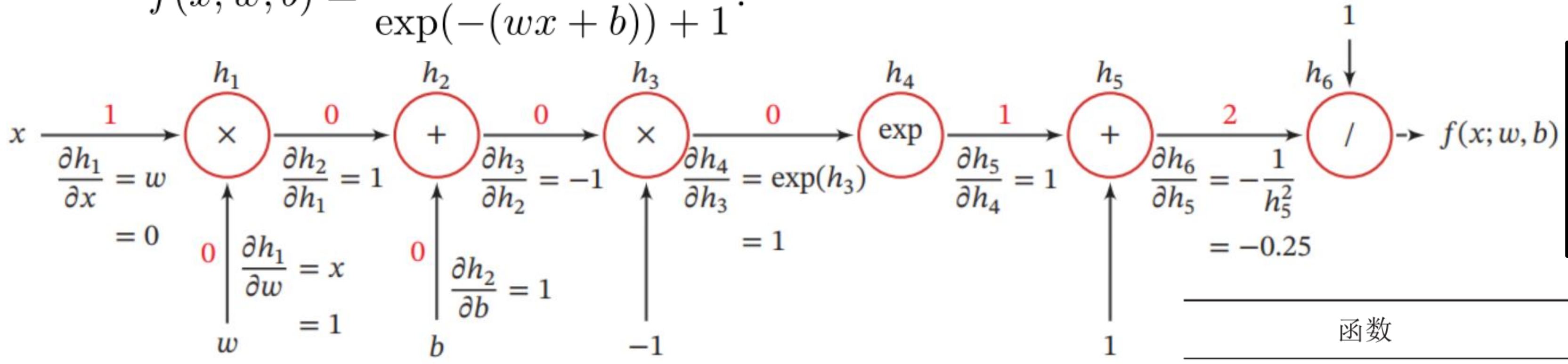
输出: $\boldsymbol{W}, \boldsymbol{b}$

注意：输出层的误差项要根据损失函数的定义来计算

前馈神经网络

➢自动微分计算

- ➢**自动微分**的基本原理是所有的数值计算可以**分解为一些基本操作**，然后利用**链式法则**来自动计算一个复合函数的梯度
- ➢**例：** $$f(x;w,b)=\frac{1}{\exp(-(wx+b))+1}.$$



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再进行相加，得到最终的梯度

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

- ➢**当x=1,w=0,b=0时**

$$\frac{\partial f(x;w,b)}{\partial w}\Big|_{x=1,w=0,b=0}=\frac{\partial f(x;w,b)}{\partial h_6}\frac{\partial h_6}{\partial h_5}\frac{\partial h_5}{\partial h_4}\frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial w}$$
$$=1\times -0.25\times 1\times 1\times -1\times 1\times 1$$
$$=0.25.$$

- ➢**反向模式**和反向传播的计算梯度的方式相同

前馈神经网络

➢自动微分计算

- ➢输出层的误差项要根据损失函数的定义来计算
- ➢常见激活函数的导数

表 4.3 常见激活函数及其导数

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

前馈神经网络

➢编程实现方法

- ➢ **前馈神经网络的训练过程分为以下三步：**
  - ➢ 前向计算每一层的状态和激活值，直到最后一层
  - ➢ 反向计算每一层的参数的偏导数
  - ➢ 按照梯度下降算法更新参数
- ➢ **静态计算图**（Tensorflow）：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，便于并行计算，灵活性差。
- ➢ **动态计算图**（PyTorch）：在程序运行时动态构建，灵活度高，不便于并行计算。
- ➢ **实现流程**

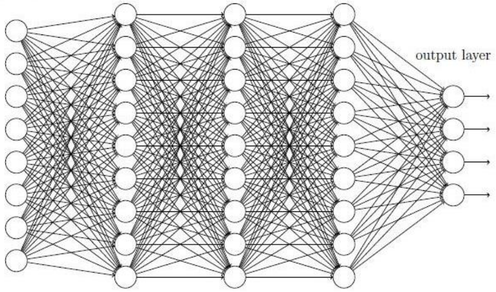
1 定义网络 → 2 损失函数 → 3 优化

```
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import SGD

model = Sequential()
model.add(Dense(output_dim=64, input_dim=100))
model.add(Activation("relu"))
model.add(Dense(output_dim=10))
model.add(Activation("softmax"))

model.compile(loss='categorical_crossentropy',
   optimizer='sgd', metrics=['accuracy'])

model.fit(X_train, Y_train, nb_epoch=5, batch_size=32)

loss = model.evaluate(X_test, Y_test, batch_size=32)
```

前馈神经网络

➢优化问题

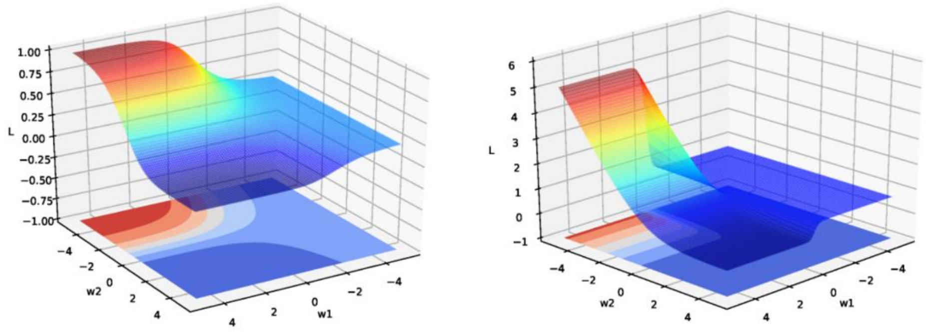
- ➢神经网络的参数学习比线性模型要更加困难：
  - ➢ 参数过多，影响训练



参数数量

$$P = \sum_{l=1}^{N-1} M_l + 1 \ \ M_{l+1}$$

  - ➢ 非凸优化问题：即存在局部最优而非全局最优解，影响迭代



(a) 平方误差损失　　(b) 交叉熵损失

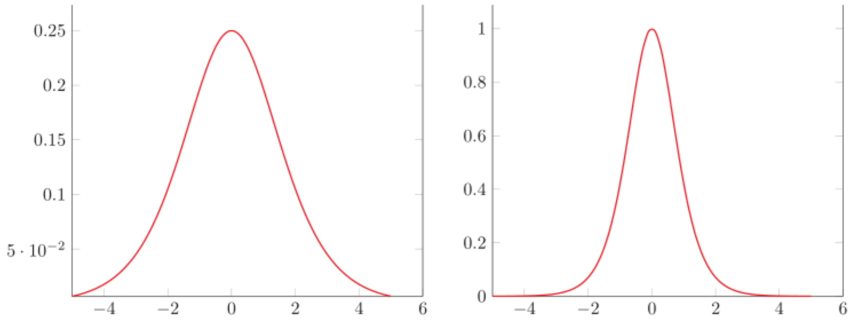
图 4.10　神经网络 $y = \sigma(w_2\sigma(w_1 x))$ 的损失函数

  - ➢ 梯度消失问题，造成下层参数比较难调

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

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

$$\delta^{(l)} = f_l'(\boldsymbol{z}^{(l)}) \odot \left((\boldsymbol{W}^{(l+1)})^{\intercal}\delta^{(l+1)}\right) \quad \in \mathbb{R}^{M_l}$$



(a) logistic 函数的导数　　(b) tanh 函数的导数

  - ➢ 可解释性差

课后作业

- 完成
  - 习题4-1
  - 习题4-2
  - 习题4-4
- 思考
  - 习题4-7
  - 习题4-8
  - 习题4-9

谢 谢