第 8 章　人工神经网络

人工神经网络及其应用

- 人工神经网络是对人脑或生物神经网络若干基本特性的抽象和模拟。为机器学习等许多问题的研究提供了一条新的思路，目前已经在模式识别、机器视觉、联想记忆、自动控制、信号处理、软测量、决策分析、智能计算、组合优化问题求解、数据挖掘等方面获得成功应用。

人工神经网络及其应用

- 神经网络（neural networks，NN）

  - 生物神经网络(natural neural network, NNN): 由中枢神经系统（脑和脊髓）及周围神经系统（感觉神经、运动神经等）所构成的错综复杂的神经网络，其中最重要的是脑神经系统。

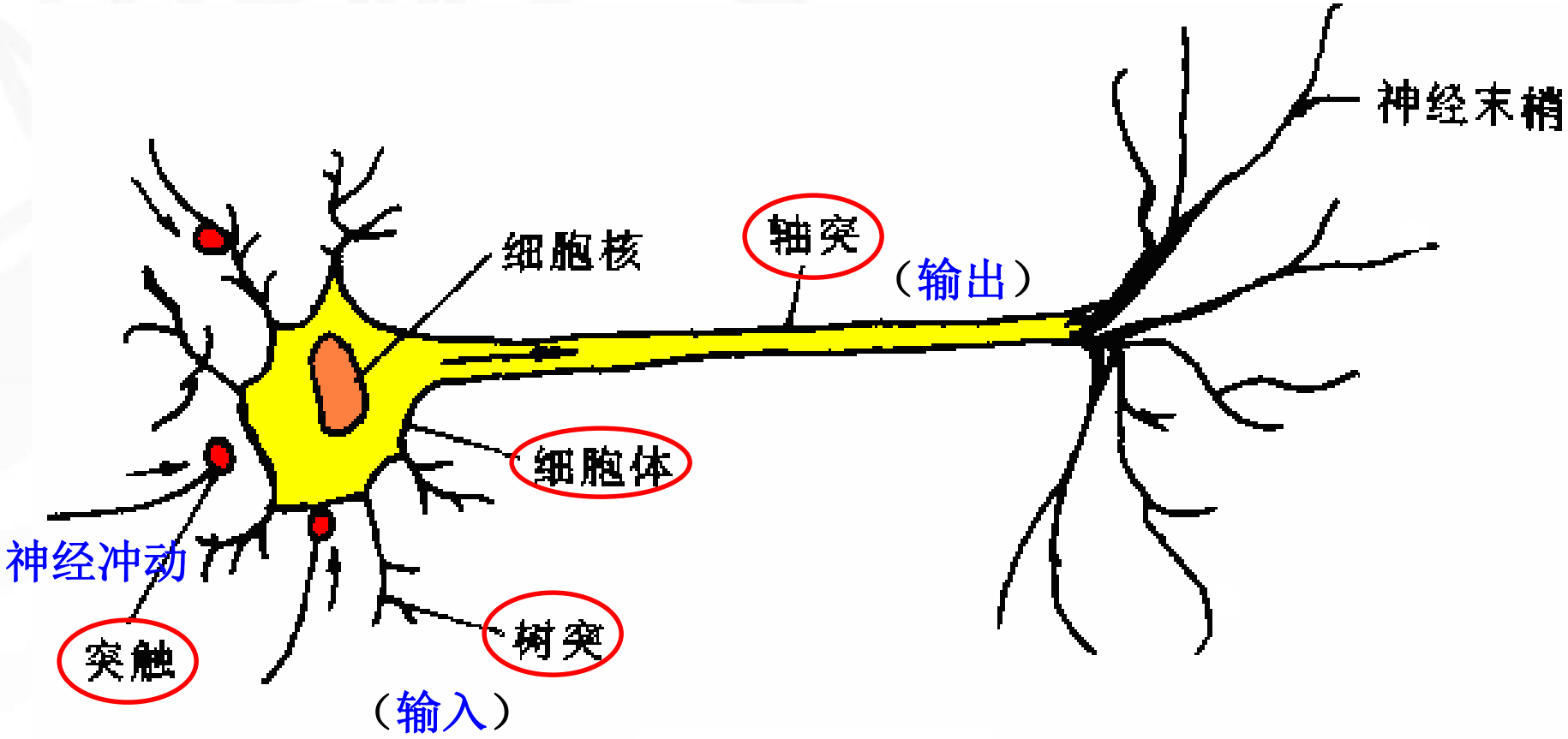
  - 人工神经网络(artificial neural networks, ANN): 模拟人脑神经系统的结构和功能，运用大量简单处理单元经广泛连接而组成的人工网络系统。

神经网络方法： 隐式的
知识表示方法

生物神经元的结构

- 人脑由一千多亿（1011亿－ 1014 亿）个神经细胞（神经元）交织在一起的网状结构组成，其中大脑皮层约140亿个神经元，小脑皮层约1000亿个神经元。
- 神经元约有1000种类型，每个神经元大约与10^3— 10^4个其他神经元相连接，形成极为错综复杂而又灵活多变的神经网络。
- 人的智能行为就是由如此高度复杂的组织产生的。浩瀚的宇宙中，也许只有包含数千忆颗星球的银河系的复杂性能够与大脑相比。

生物神经元的结构



生物神经元结构

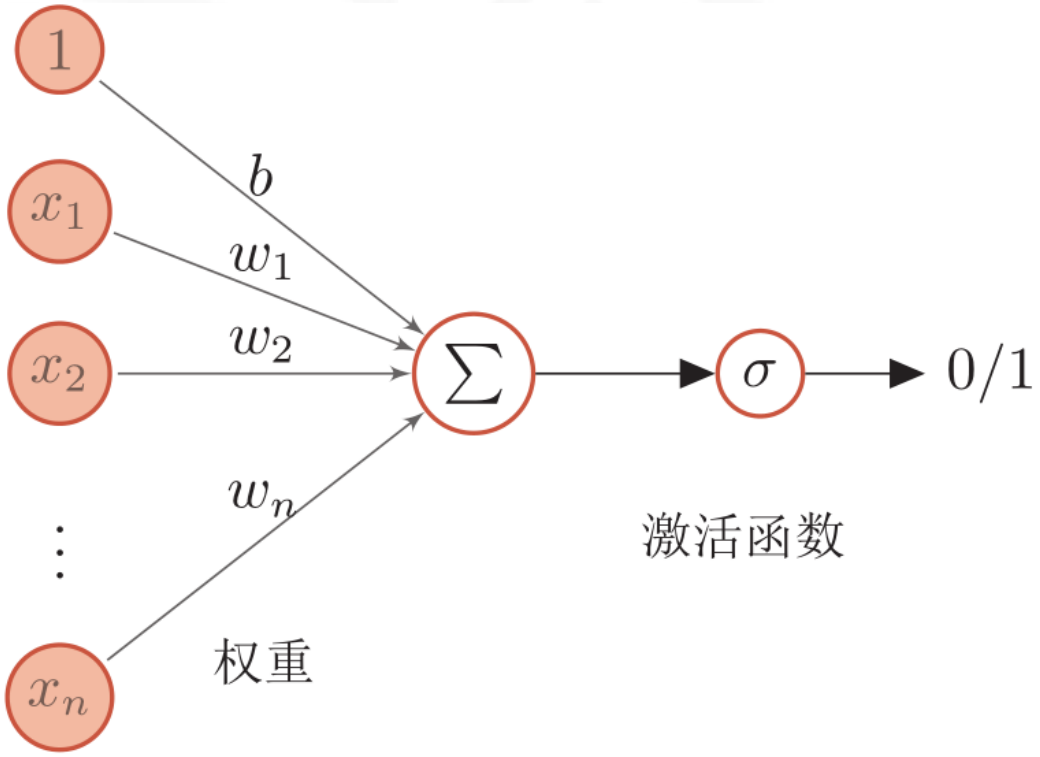
生物神经元的结构

- 工作状态：
- 兴奋状态：细胞膜电位 > 动作电位的阈值 → 神经冲动
- 抑制状态：细胞膜电位 < 动作电位的阈值
- 学习与遗忘：由于神经元结构的可塑性，突触的传递作用可增强和减弱 。

单个神经细胞只有两种状态：兴奋和抑制

神经元数学模型

1943年，麦克洛奇和皮兹提出M－P模型。一般模型：



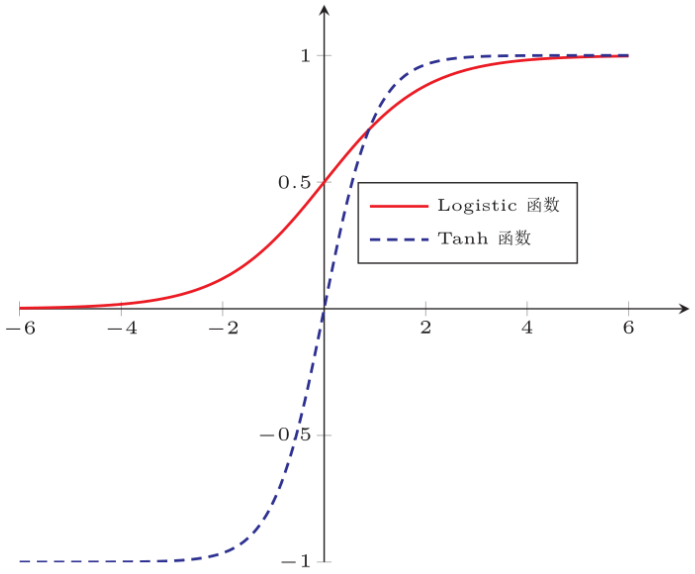
神经元数学模型

- 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

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

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



常见激活函数

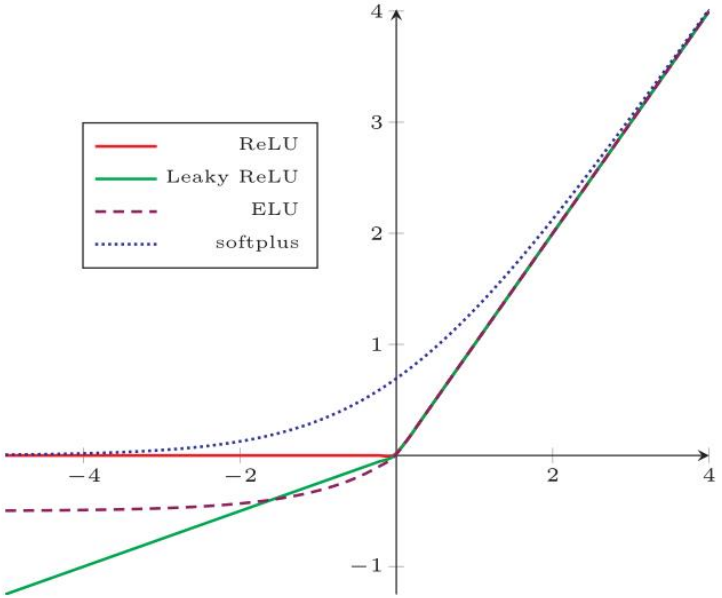
$$\begin{aligned}\mathrm{ReLU}(x) &= \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases} \\ &= \max(0, x).\end{aligned}$$

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

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

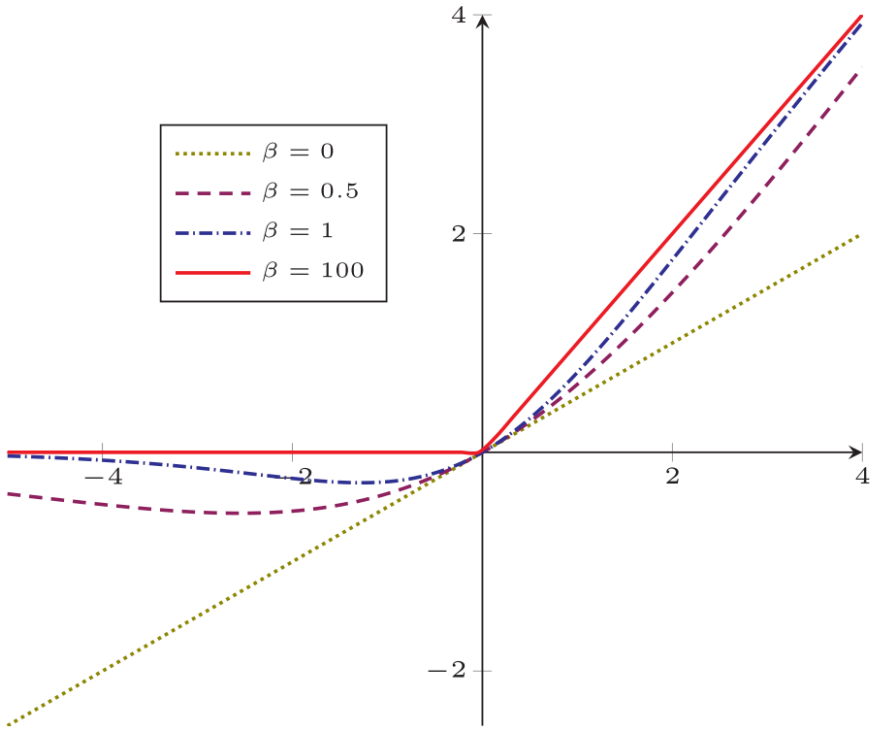
$$\begin{aligned}\mathrm{ELU}(x) &= \begin{cases} x & \text{if } x > 0 \\ \gamma(\exp(x) - 1) & \text{if } x \leq 0 \end{cases} \\ &= \max(0, x) + \min(0, \gamma(\exp(x) - 1))\end{aligned}$$

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



常见激活函数

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



常见激活函数及其导数

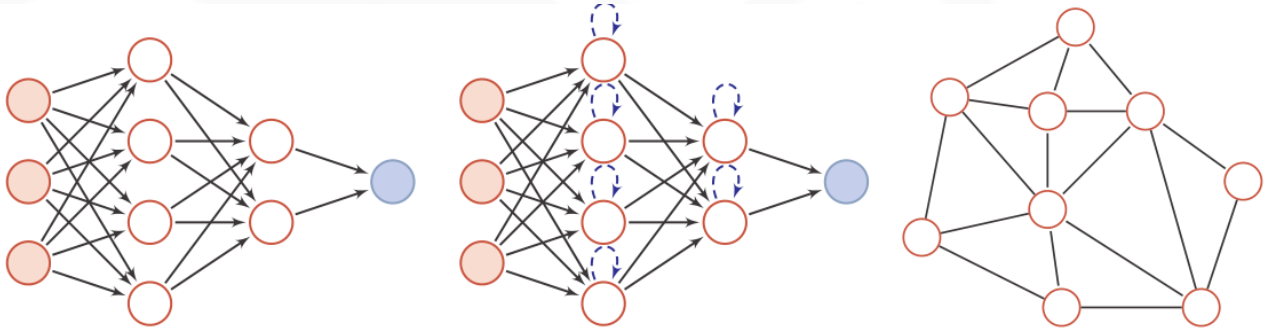
激活函数	函数	导数
Logistic 函数	$f(x)=\frac{1}{1+\exp(-x)}$	$f'(x)=f(x)\big(1-f(x)\big)$
Tanh 函数	$f(x)=\frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$	$f'(x)=1-f(x)^2$
ReLU	$f(x)=\max(0,x)$	$f'(x)=I(x>0)$
ELU	$f(x)=\max(0,x)+\min\big(0,\gamma(\exp(x)-1)\big)$	$f'(x)=I(x>0)+I(x\leq 0)\cdot\gamma\exp(x)$
SoftPlus 函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

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
  - 通过训练数据来学习神经网络的参数。

人工神经网络

- 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) 反馈网络　(c) 图网络

人工神经网络

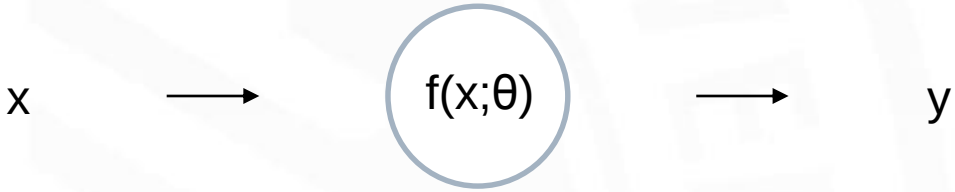

神经网络
Neural Network
人工神经网络
ANN
生物神经网络
前馈神经网络
反馈神经网络
单层神经网络
Perceptron
两层神经网络
MLP
多层神经网络
DL
普通深度网络
DNN
深度信念网络
DBN
卷积神经网络
CNN

前馈神经网络

前馈网络

目标

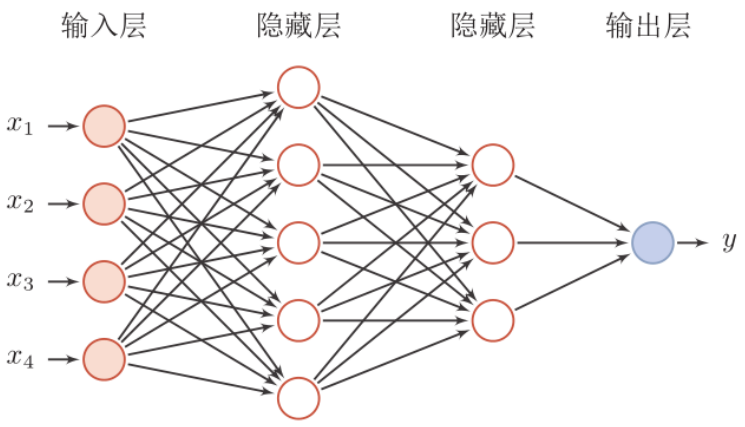
- 前馈网络的目标是近似某个函数 f^*.
- 例如，对于分类器，$y = f^*(x)$ 将输入 x 映射到一个类别y.



- 前馈网络即是为了学习这个参数θ的值，使它能够得到最佳的函数近似.

网络结构

- 前馈神经网络（全连接神经网络、多层感知器）
  - 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。



前馈网络

- 给定一个前馈神经网络，我们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。

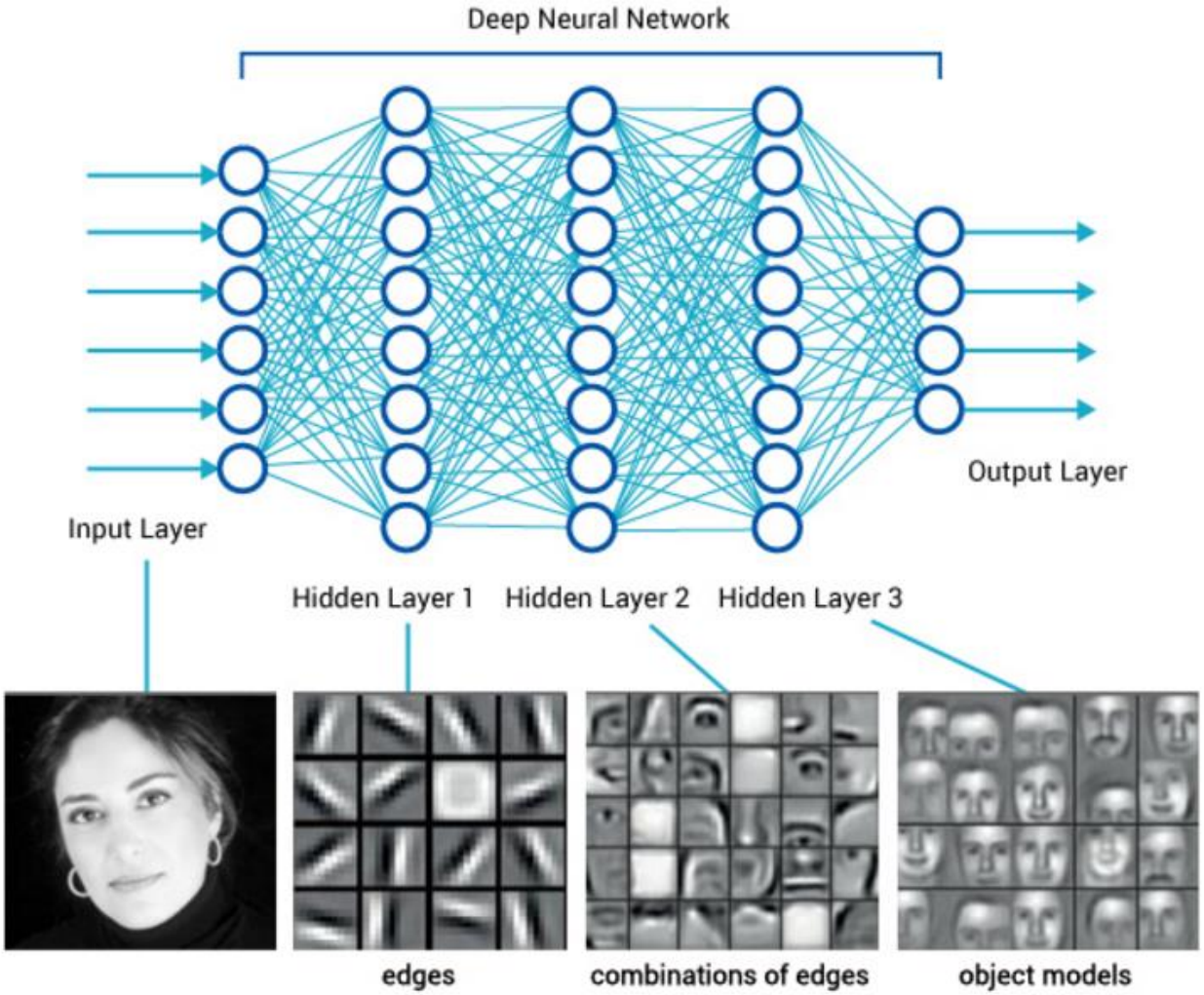
$$\mathbf{z}^{(l)} = W^{(l)} \cdot \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$$

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

- 前馈计算：

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

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。

通用近似定理

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。

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后验概率。

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

  - 采用交叉熵损失函数，对于样本(x,y)，其损失函数为

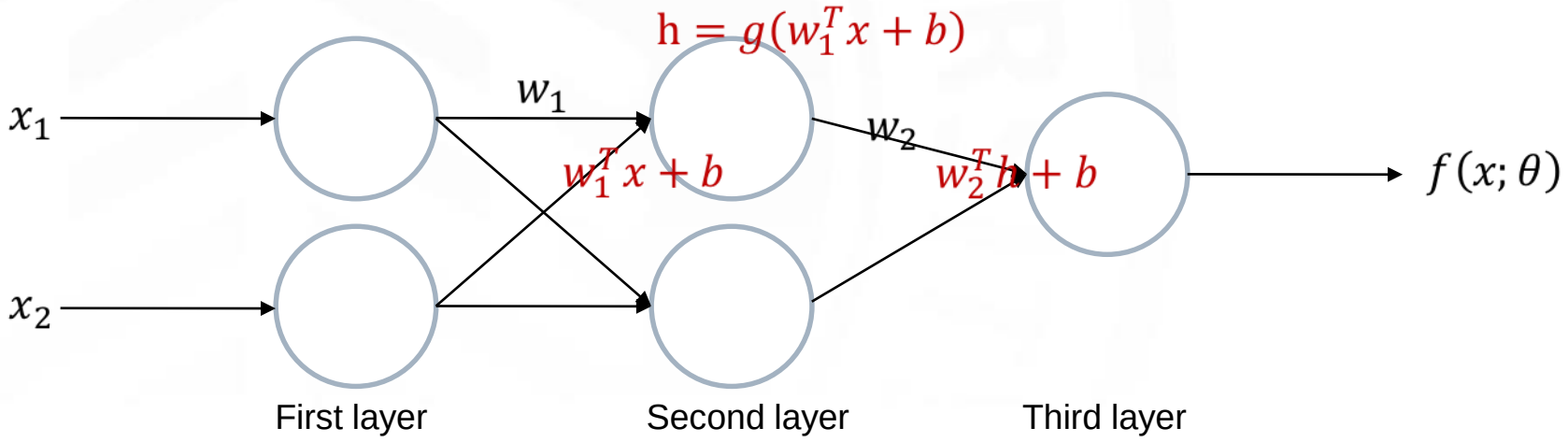
$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = -\mathbf{y}^{\mathrm{T}} \log \hat{\mathbf{y}}$$

架构设计

架构（architecture）是指网络的整体结构，它应该具有多少单元，以及这些单元应该如何连接。

大多数神经网络被组织成称为层的单元组。大多数神经网络架构将这些层布置成链式结构，其中每一层都是前一层的函数。

在这些链式架构中，主要的架构考虑是选择网络的深度和每一层的宽度



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

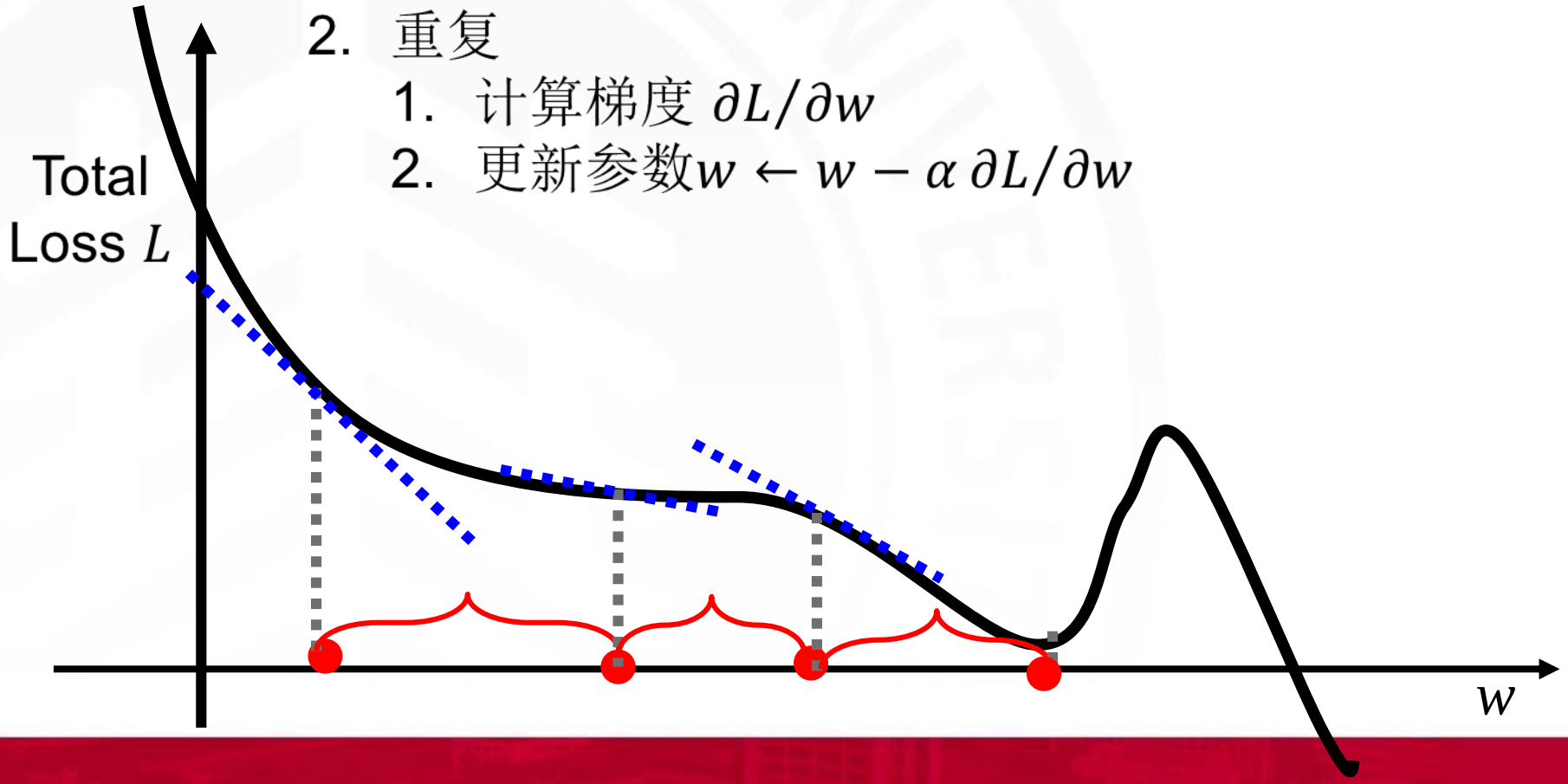
$$W^{(l)} \leftarrow W^{(l)} - \alpha\frac{\partial\mathcal{R}(W, \mathbf{b})}{\partial W^{(l)}}$$

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

梯度下降

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

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



如何计算梯度？

- 神经网络为一个复杂的复合函数
  - 链式法则

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

- 前向传播（forward propagation）

输入x提供初始信息，然后传播到每一层的隐藏单元，最终产生输出y。

- 反向传播算法
  - 根据前馈网络的特点而设计的高效方法
  - 允许来自代价函数的信息通过网络向后流动，以便计算梯度。

反向传播算法（BP算法）

- BP算法的基本思想：

（1）先计算每一层的状态和激活值，直到最后一层（即信号是前向传播的）

（2）计算每一层的误差，误差的计算过程是从最后一层向前推进的（反向传播算法名字的由来）

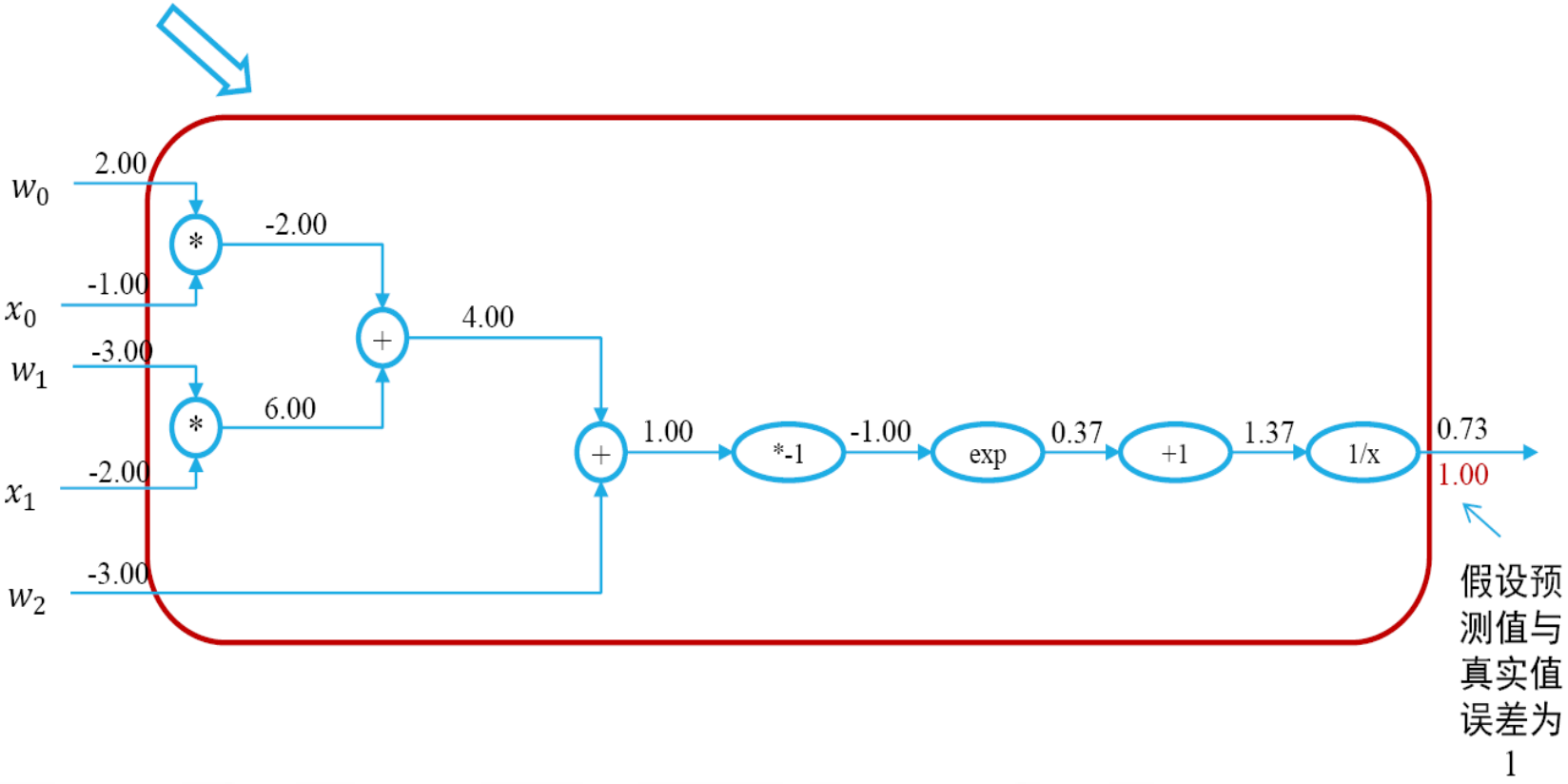
（3）更新参数（目标是误差变小）

（4）迭代前面两个步骤，直到满足停止准则。

- 将误差从后向前传递，将误差分摊给各层所有单元，从而获得各层单元所产生的误差，进而依据这个误差来让各层单元负起各自责任、修正各单元参数

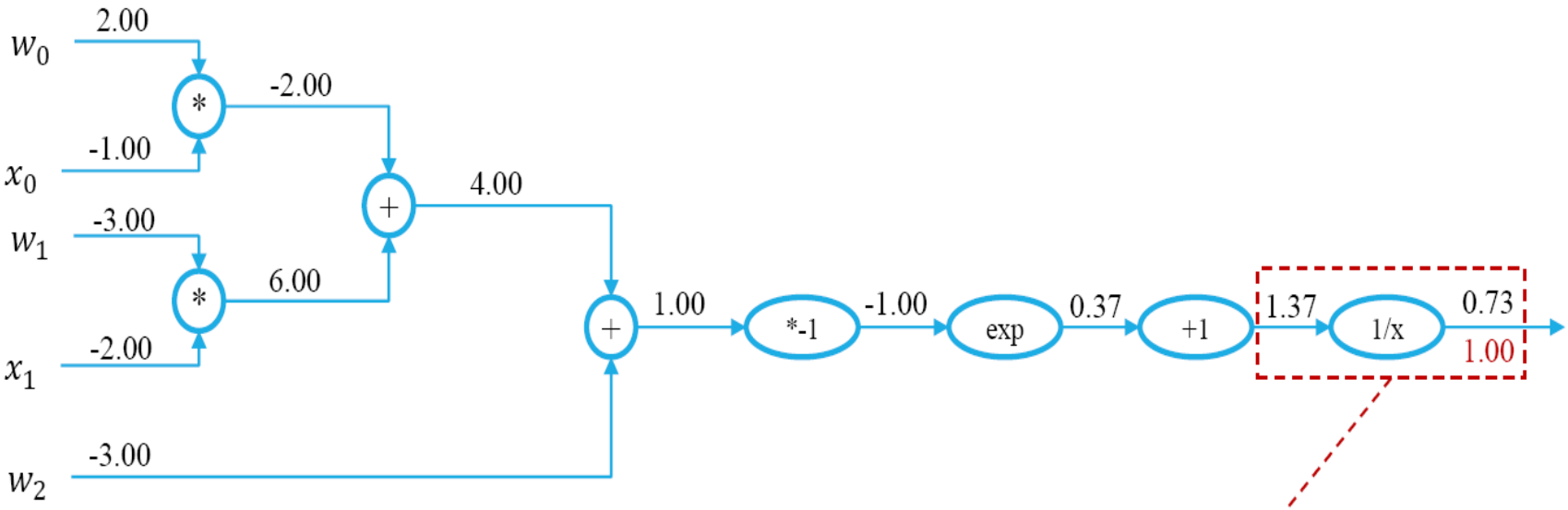
反向传播算法例子

$$f(w,x)=\frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



反向传播算法例子

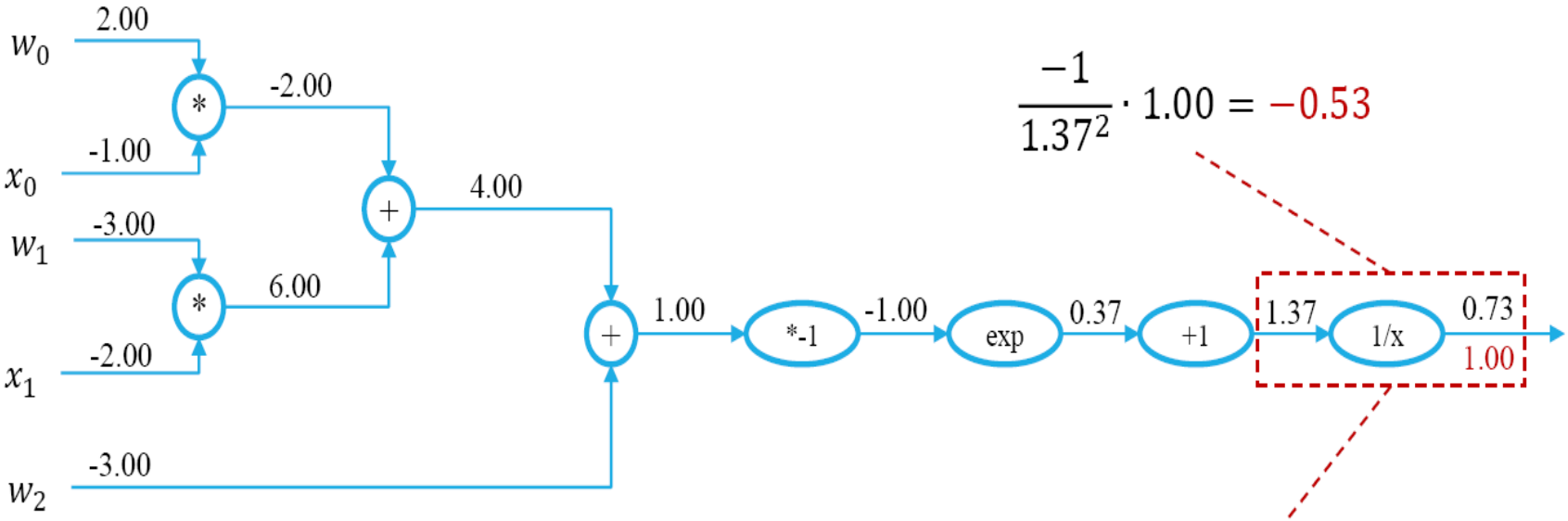
$$f(w,x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$



$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -\frac{1}{x^2}$$

反向传播算法例子

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

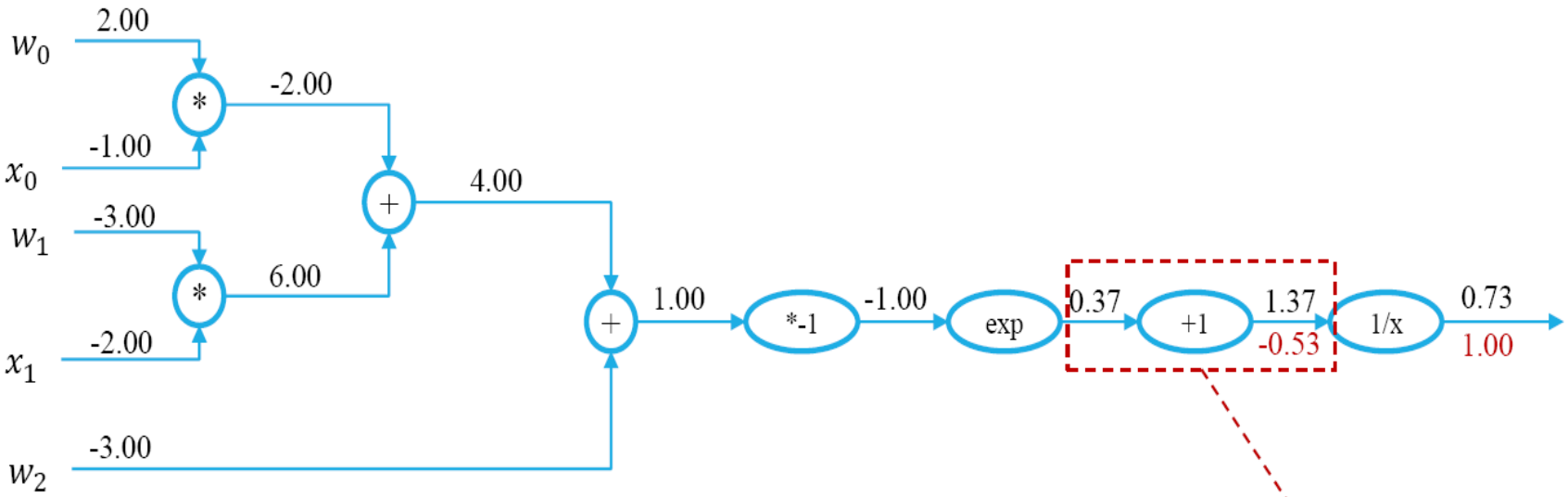


$$\frac{-1}{1.37^2} \cdot 1.00 = -0.53$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -\frac{1}{x^2}$$

反向传播算法例子

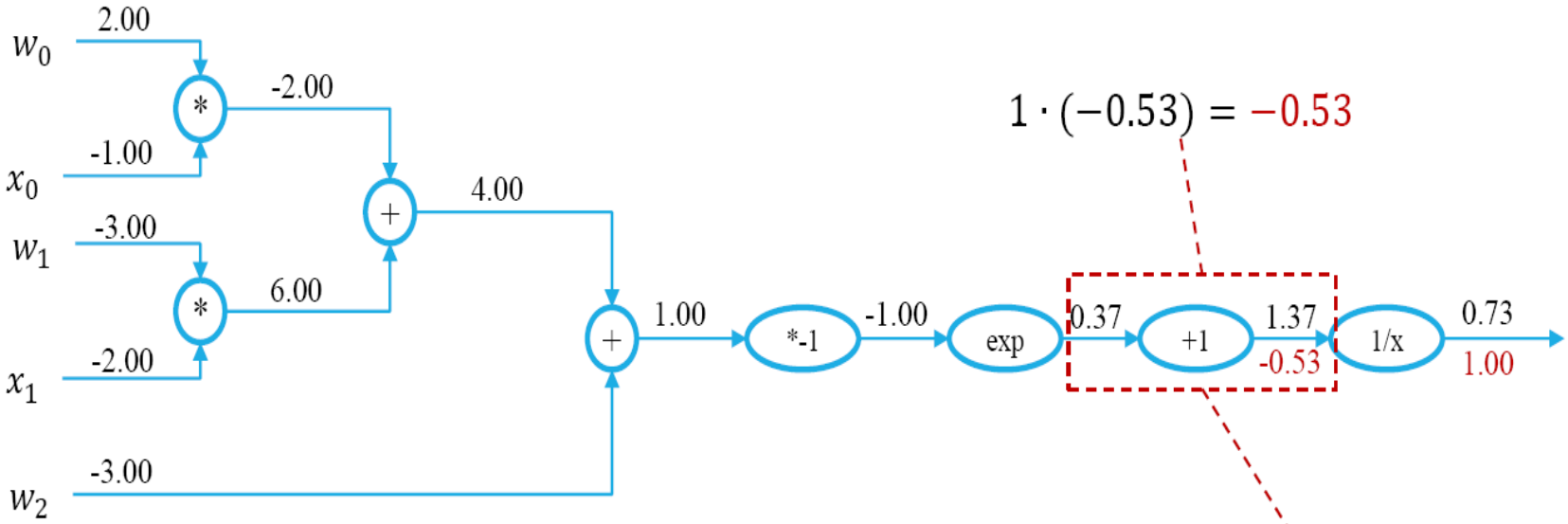
$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

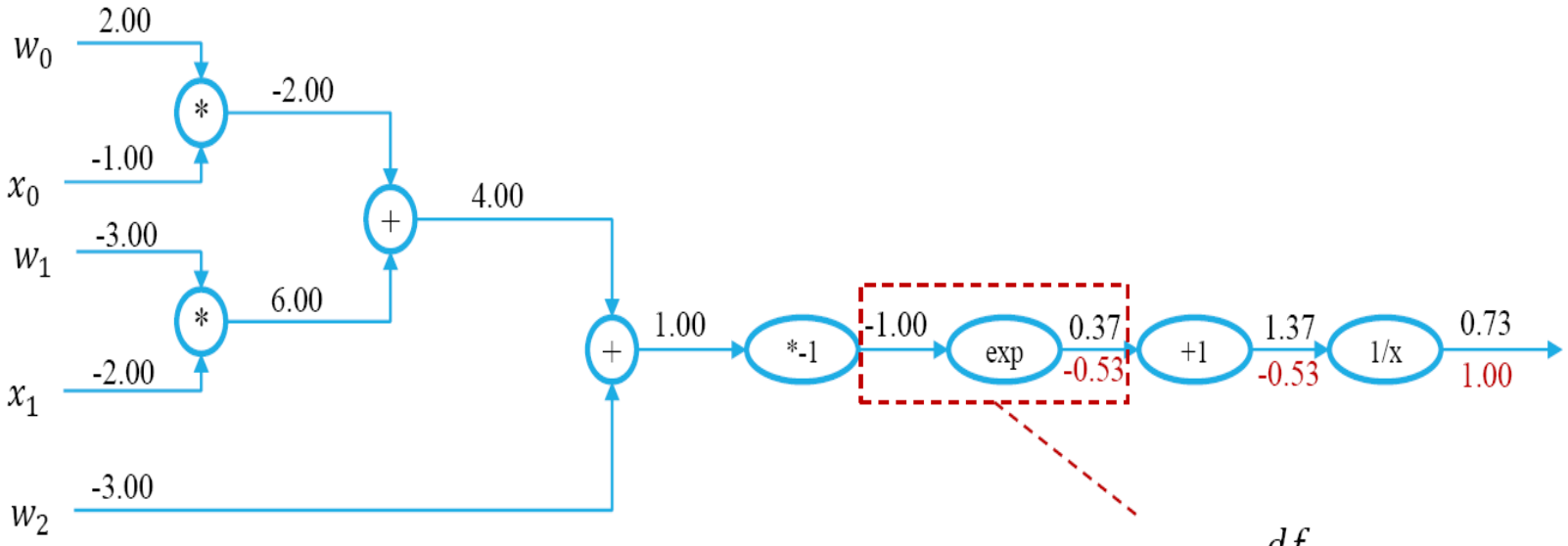
反向传播算法例子

$$f(w,x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$



反向传播算法例子

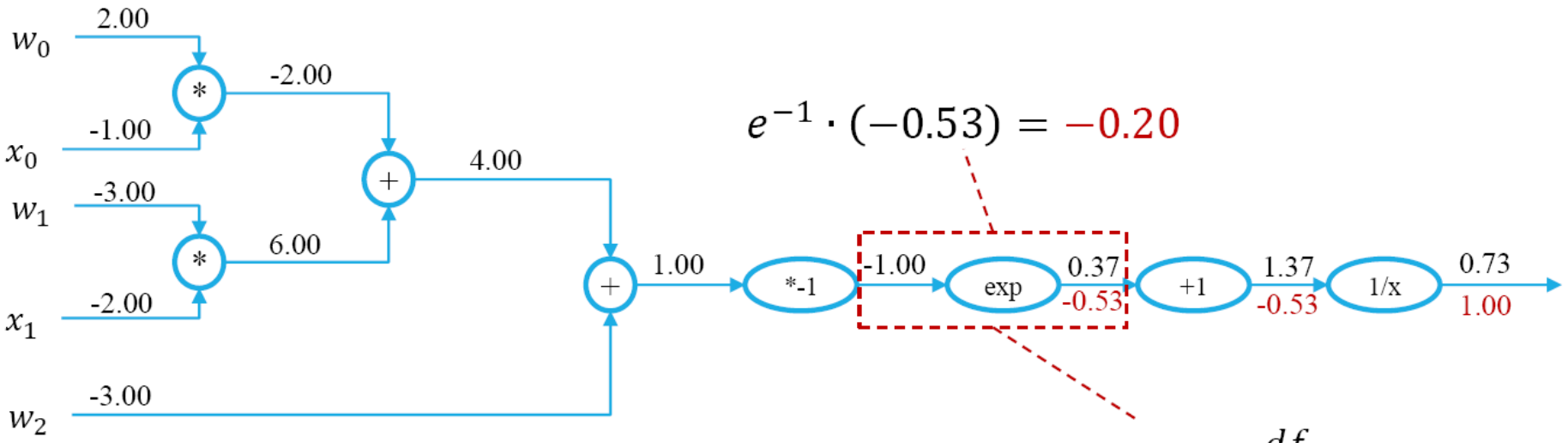
$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

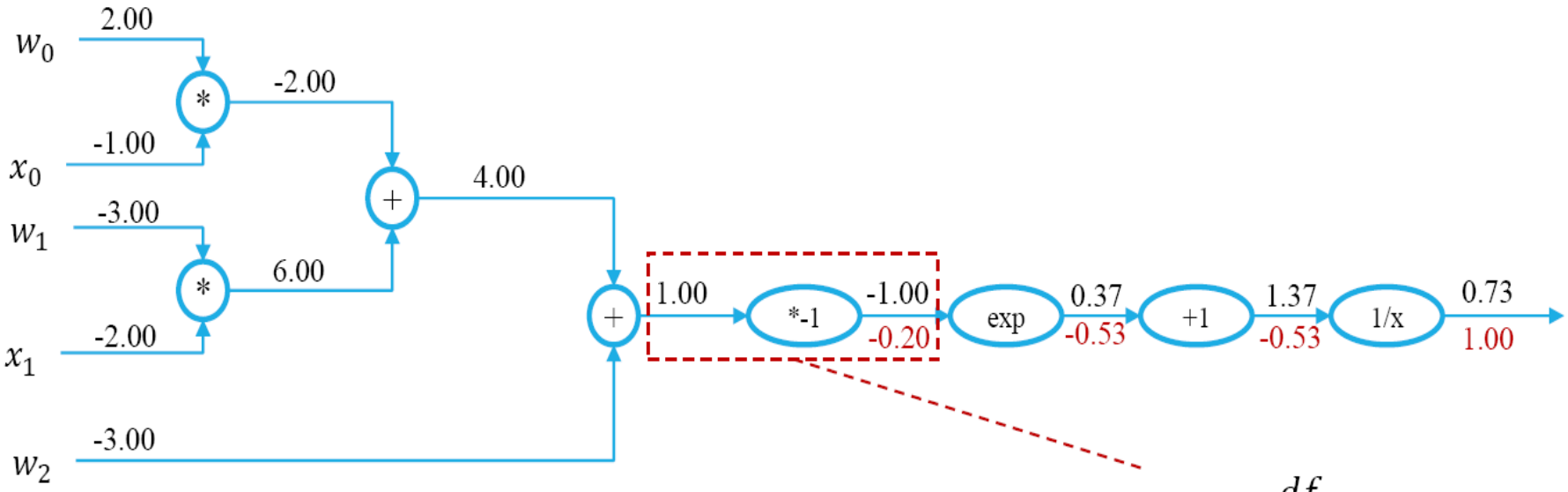
反向传播算法例子

$$f(w,x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$



反向传播算法例子

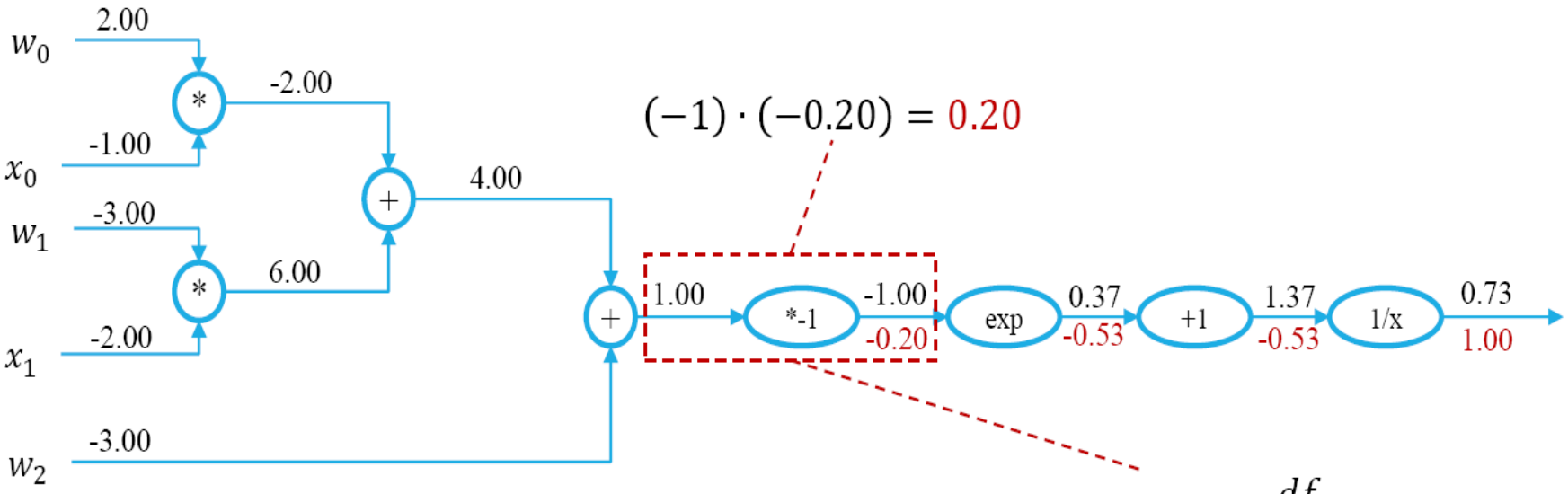
$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

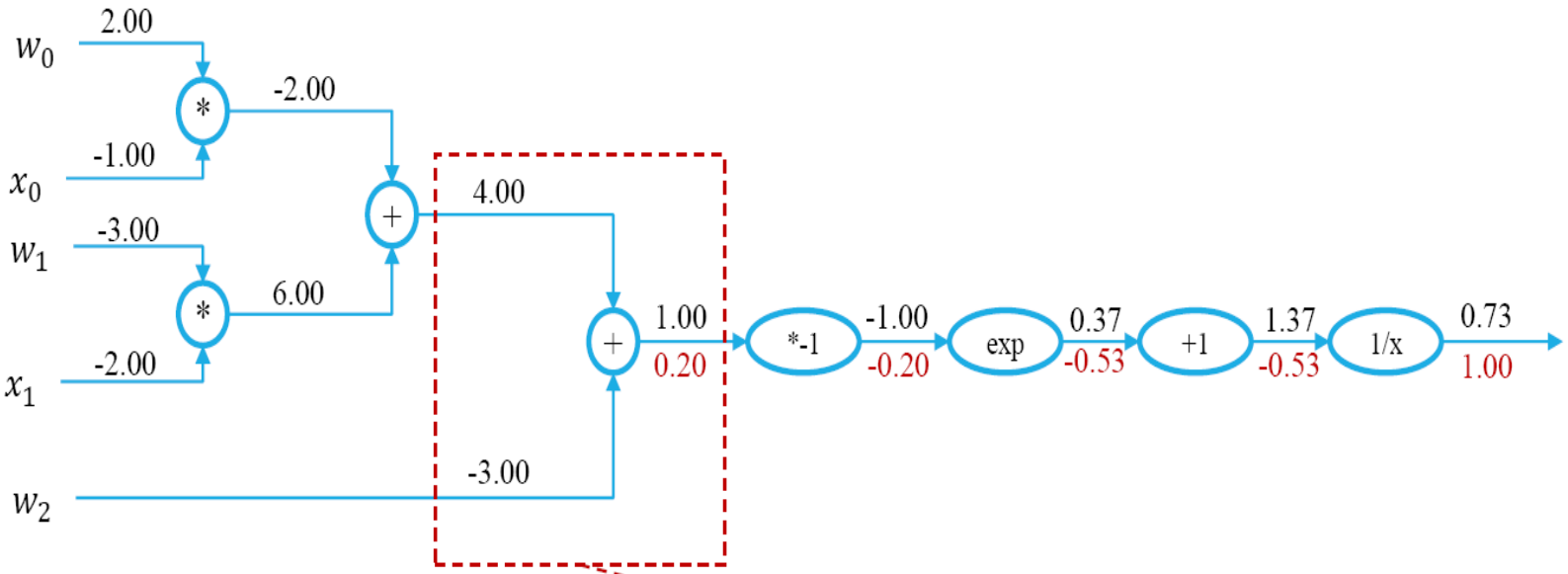
反向传播算法例子

$$f(w,x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$



反向传播算法例子

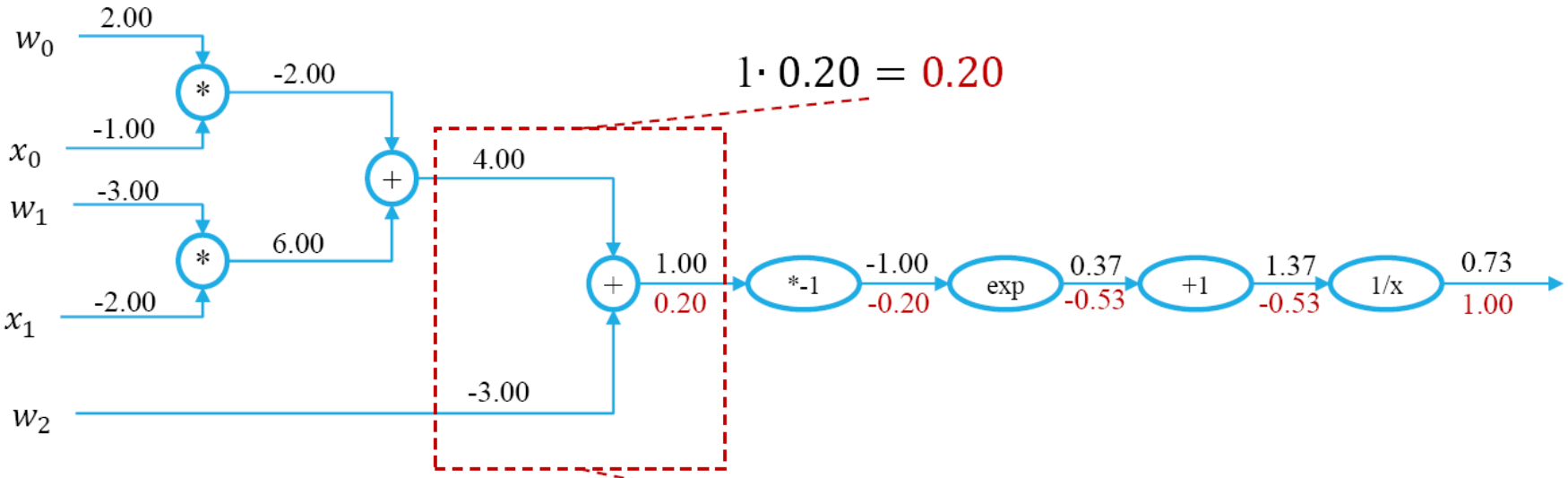
$$f(w,x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$



$$f(x) = x + x' \quad \rightarrow \quad \frac{df}{dx} = 1$$

反向传播算法例子

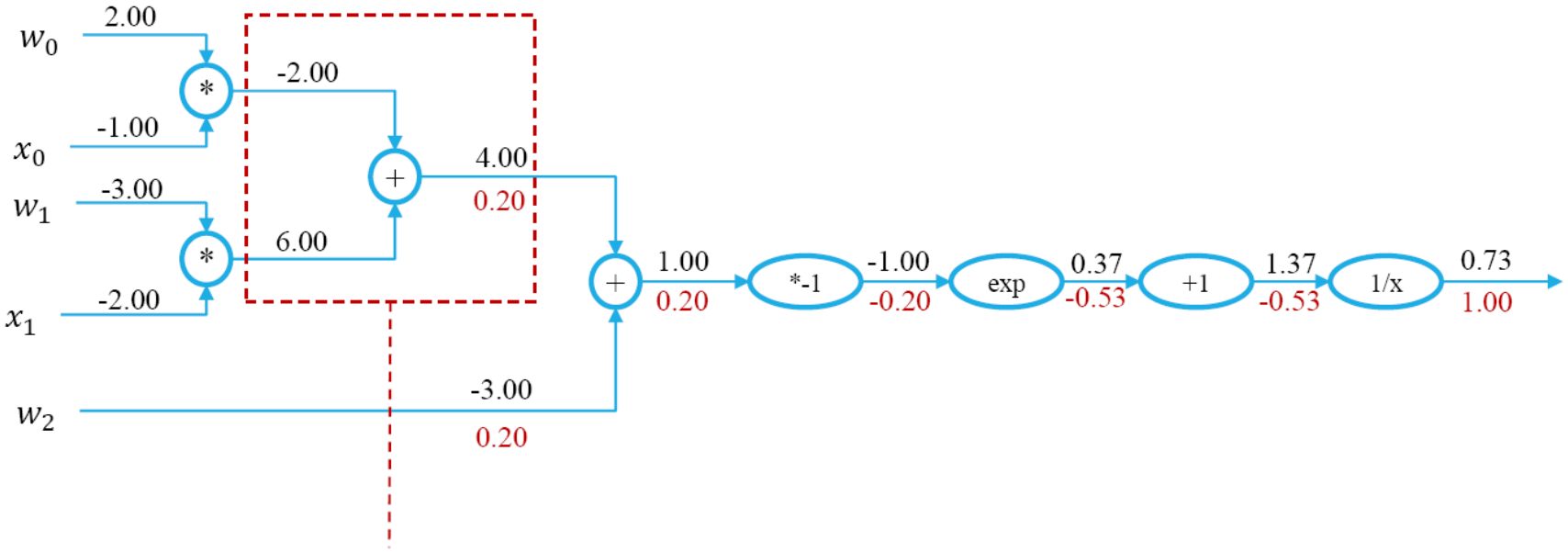
$$f(w,x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$



$$f(x) = x + x' \quad \rightarrow \quad \frac{df}{dx} = 1$$

反向传播算法例子

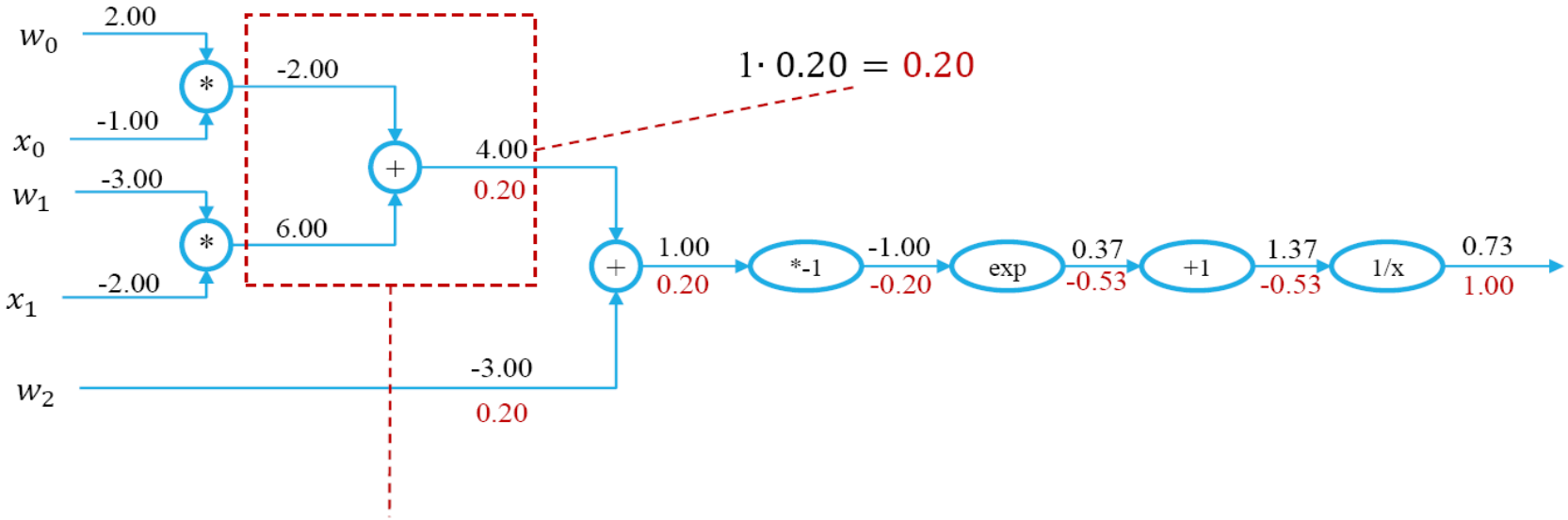
$$f(w,x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$



$$f(x) = x + x' \quad \rightarrow \quad \frac{df}{dx} = 1$$

反向传播算法例子

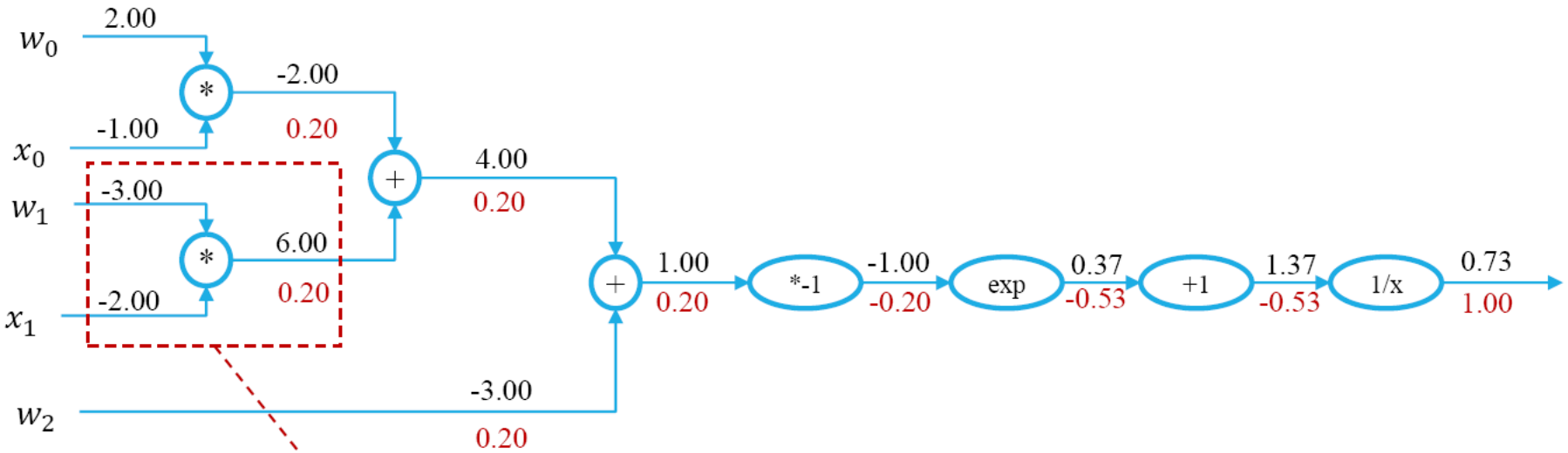
$$f(w,x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$



$$f(x) = x + x' \quad \rightarrow \quad \frac{df}{dx} = 1$$

反向传播算法例子

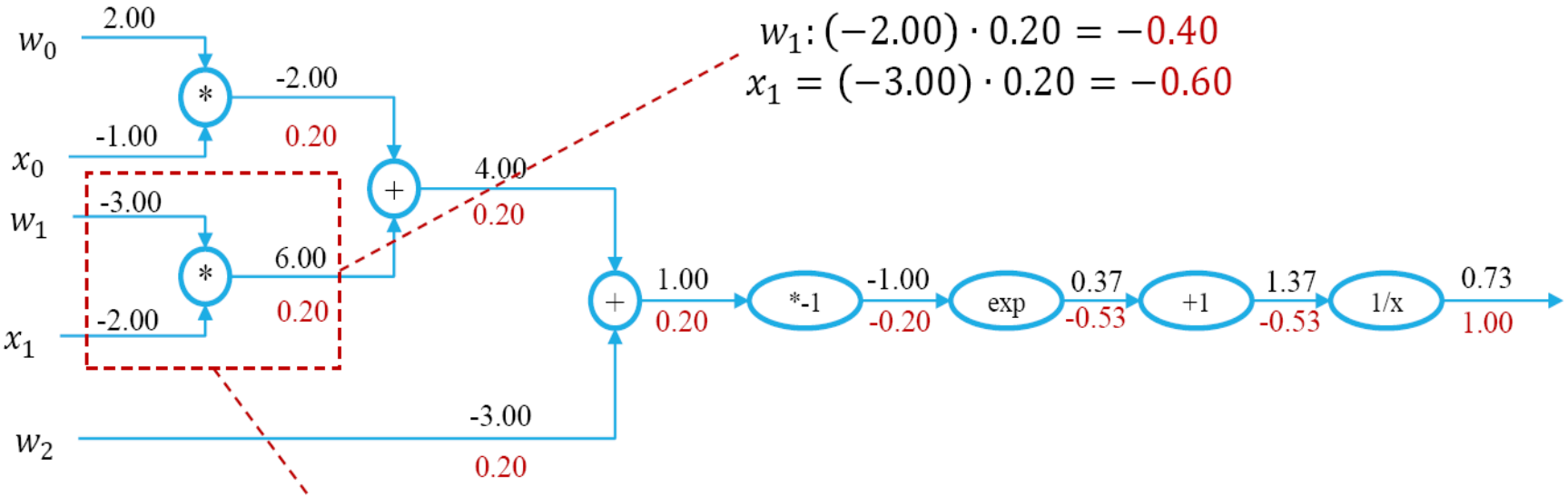
$$f(w,x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



$$f(x) = x \cdot x' \quad \rightarrow \quad \frac{df}{dx} = x'$$

反向传播算法例子

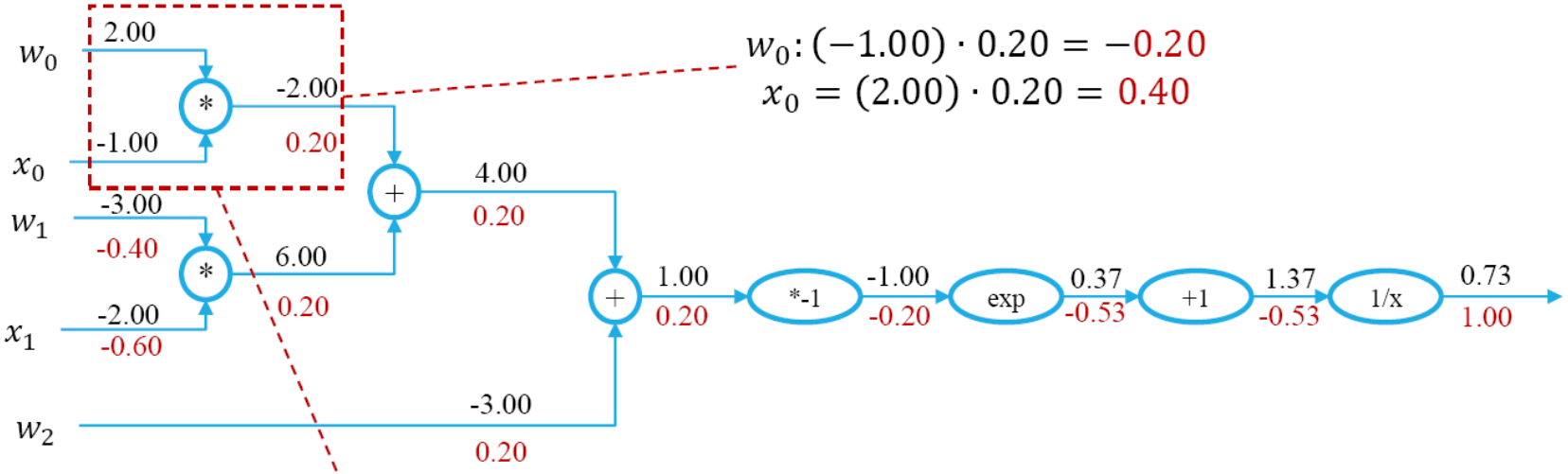
$$f(w,x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



$$f(x) = x \cdot x' \quad \rightarrow \quad \frac{df}{dx} = x'$$

反向传播算法例子

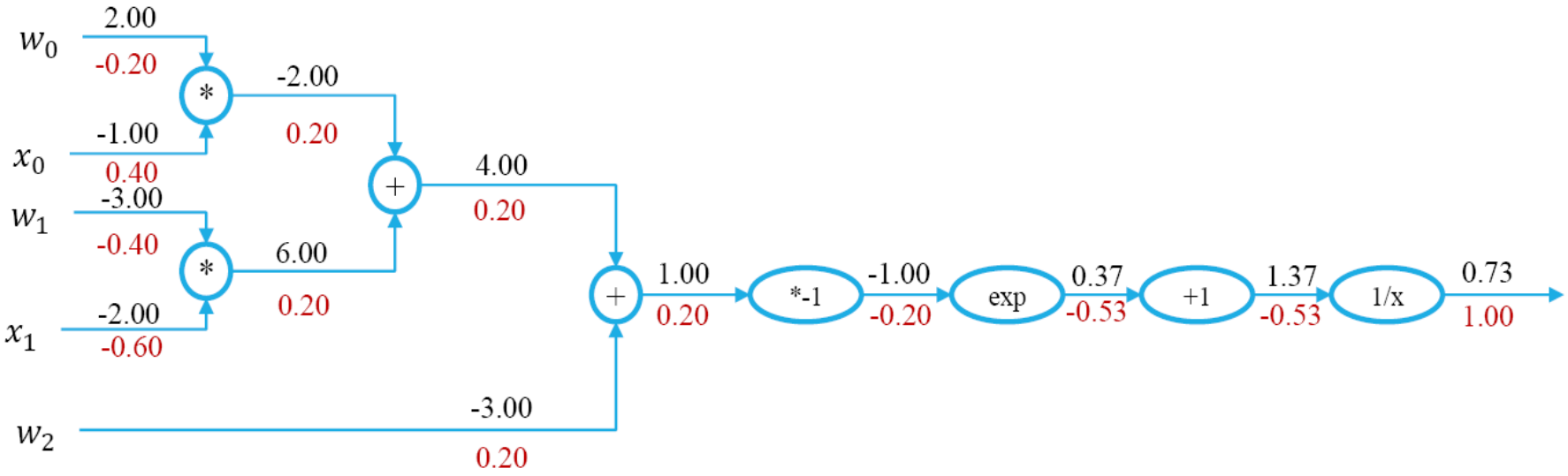
$$f(w,x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



$$f(x) = x \cdot x' \quad \rightarrow \quad \frac{df}{dx} = x'$$

反向传播算法例子

$$f(w,x)=\frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



误差反向传播

反向传播算法（BP算法）

三层感知器如图 1 所示。例子中，输入数据 $\boldsymbol{x} = (x_1, x_2, x_3)^{\mathsf{T}}$ 是 3 维的（对于第一层，可以认为 $a_i^{(1)} = x_i$），唯一的隐藏层有 3 个节点，输出数据是 2 维的。

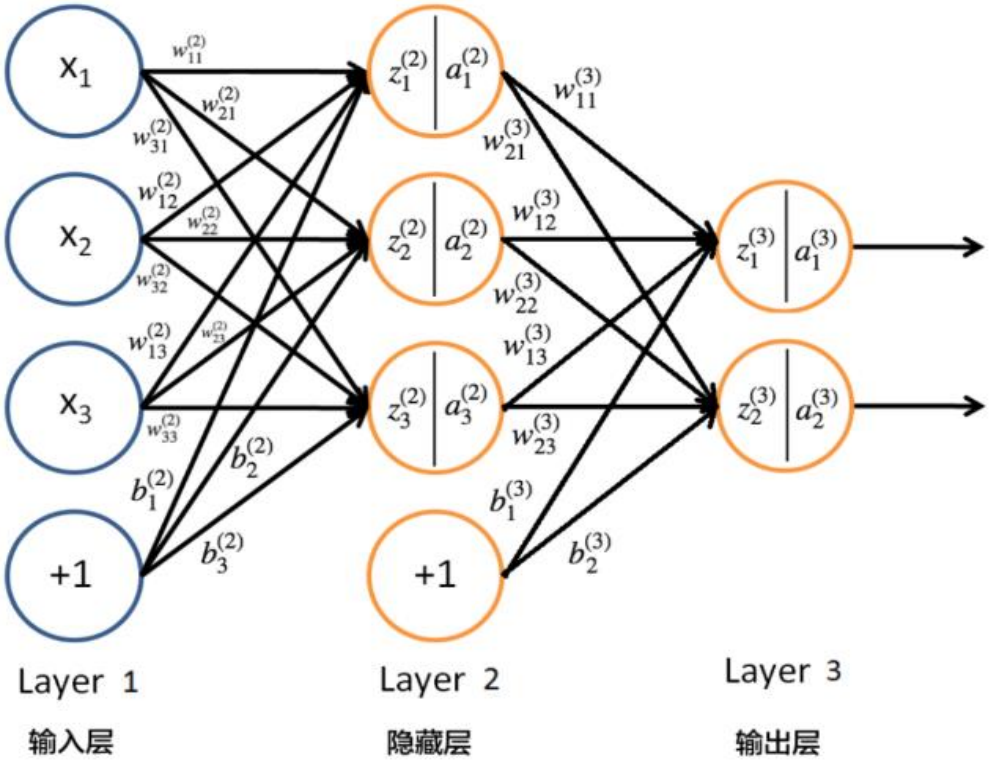


Figure 1: 三层感知器实例

信息前向传播

显然，图 1 所示神经网络的第 2 层神经元的状态及激活值可以通过下面的计算得到：

$$
\begin{aligned}
z_1^{(2)} &= w_{11}^{(2)} x_1 + w_{12}^{(2)} x_2 + w_{13}^{(2)} x_3 + b_1^{(2)} \\
z_2^{(2)} &= w_{21}^{(2)} x_1 + w_{22}^{(2)} x_2 + w_{23}^{(2)} x_3 + b_2^{(2)} \\
z_3^{(2)} &= w_{31}^{(2)} x_1 + w_{32}^{(2)} x_2 + w_{33}^{(2)} x_3 + b_3^{(2)} \\
a_1^{(2)} &= f(z_1^{(2)}) \\
a_2^{(2)} &= f(z_2^{(2)}) \\
a_3^{(2)} &= f(z_3^{(2)})
\end{aligned}
$$

类似地，第 3 层神经元的状态及激活值可以通过下面的计算得到：

$$
\begin{aligned}
z_1^{(3)} &= w_{11}^{(3)} a_1^{(2)} + w_{12}^{(3)} a_2^{(2)} + w_{13}^{(3)} a_3^{(2)} + b_1^{(3)} \\
z_2^{(3)} &= w_{21}^{(3)} a_1^{(2)} + w_{22}^{(3)} a_2^{(2)} + w_{23}^{(3)} a_3^{(2)} + b_2^{(3)} \\
a_1^{(3)} &= f(z_1^{(3)}) \\
a_2^{(3)} &= f(z_2^{(3)})
\end{aligned}
$$

可总结出，第 $l\,(2 \le l \le L)$ 层神经元的状态及激活值为（下面式子是向量表示形式）：

$$
\begin{aligned}
\boldsymbol{z}^{(l)} &= W^{(l)} \boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)} \\
\boldsymbol{a}^{(l)} &= f(\boldsymbol{z}^{(l)})
\end{aligned}
$$

对于 L 层感知器，网络的最终输出为 $\boldsymbol{a}^{(L)}$ 。前馈神经网络中信息的前向传递过程如下：

$$
\boldsymbol{x} = \boldsymbol{a}^{(1)} \to \boldsymbol{z}^{(2)} \to \cdots \to \boldsymbol{a}^{(L-1)} \to \boldsymbol{z}^{(L)} \to \boldsymbol{a}^{(L)} = \boldsymbol{y}
$$

误差反向传播

假设训练数据为 $\{(\boldsymbol{x}^{(1)},\boldsymbol{y}^{(1)}),(\boldsymbol{x}^{(2)},\boldsymbol{y}^{(2)}),\cdots,(\boldsymbol{x}^{(i)},\boldsymbol{y}^{(i)}),\cdots,(\boldsymbol{x}^{(N)},\boldsymbol{y}^{(N)})\}$，即共有 N 个。又假设输出数据为 n_L 维的，即 $\boldsymbol{y}^{(i)}=(y_1^{(i)},\cdots,y_{n_L}^{(i)})^{\mathsf{T}}$ 。

对某一个训练数据 $(\boldsymbol{x}^{(i)},\boldsymbol{y}^{(i)})$ 来说，其代价函数可写为：

$$\begin{aligned}E_{(i)}&=\frac{1}{2}\|\boldsymbol{y}^{(i)}-\boldsymbol{o}^{(i)}\|\\&=\frac{1}{2}\sum_{k=1}^{n_L}(y_k^{(i)}-o_k^{(i)})^2\end{aligned}$$

说明 1：$\boldsymbol{y}^{(i)}$ 为期望的输出（是训练数据给出的已知值），$\boldsymbol{o}^{(i)}$ 为神经网络对输入 $\boldsymbol{x}^{(i)}$ 产生的实际输出。
说明 2：代价函数中的系数 $\frac{1}{2}$ 显然不是必要的，它的存在仅仅是为了后续计算时更方便。
说明 3：以图 1 所示神经网络为例，$n_L=2,\boldsymbol{y}^{(i)}=(y_1^{(i)},y_2^{(i)})^{\mathsf{T}}$，从而有 $E_{(i)}=\frac{1}{2}(y_1^{(i)}-a_1^{(3)})^2+\frac{1}{2}(y_2^{(i)}-a_2^{(3)})^2$，如果展开到隐藏层，则有 $E_{(i)}=\frac{1}{2}(y_1^{(i)}-f(w_{11}^{(3)}a_1^{(2)}+w_{12}^{(3)}a_2^{(2)}+w_{13}^{(3)}a_3^{(2)}+b_1^{(3)}))^2+\frac{1}{2}(y_2^{(i)}-f(w_{21}^{(3)}a_1^{(2)}+w_{22}^{(3)}a_2^{(2)}+w_{23}^{(3)}a_3^{(2)}+b_2^{(3)}))^2$，还可以进一步展开到输入层（替换掉 $a_1^{(2)},a_2^{(2)},a_3^{(2)}$ 即可），最后可得：**代价函数 $E_{(i)}$ 仅和权重矩阵 $W^{(l)}$ 和偏置向量 $\boldsymbol{b}^{(l)}$ 相关，调整权重和偏置可以减少或增大代价（误差）。**

显然，所有训练数据的总体（平均）代价可写为：

$$E_{total}=\frac{1}{N}\sum_{i=1}^{N}E_{(i)}$$

误差反向传播

如果采用梯度下降法（在这里，又可称为“批量梯度下降法”），可以用下面公式更新参数 $w_{ij}^{(l)}, b_i^{(l)}, 2 \le l \le L$ ：

$$
\begin{aligned}
W^{(l)} &= W^{(l)} - \mu \frac{\partial E_{total}}{\partial W^{(l)}} \\
&= W^{(l)} - \frac{\mu}{N} \sum_{i=1}^{N} \frac{\partial E_{(i)}}{\partial W^{(l)}} \\
\boldsymbol{b}^{(l)} &= \boldsymbol{b}^{(l)} - \mu \frac{\partial E_{total}}{\partial \boldsymbol{b}^{(l)}} \\
&= \boldsymbol{b}^{(l)} - \frac{\mu}{N} \sum_{i=1}^{N} \frac{\partial E_{(i)}}{\boldsymbol{b}^{(l)}}
\end{aligned}
$$

误差反向传播

3.1 输出层的权重参数更新

把 E 展开到隐藏层，有：

$$
\begin{aligned}
E &= \frac{1}{2}\|\boldsymbol{y}-\boldsymbol{o}\| \\
&= \frac{1}{2}\|\boldsymbol{y}-\boldsymbol{a}^{(3)}\| \\
&= \frac{1}{2}\left((y_1-a_1^{(3)})^2+(y_2-a_2^{(3)})^2\right) \\
&= \frac{1}{2}\left((y_1-f(z_1^{(3)}))^2+(y_2-f(z_2^{(3)}))^2\right) \\
&= \frac{1}{2}\left((y_1-f(w_{11}^{(3)}a_1^{(2)}+w_{12}^{(3)}a_2^{(2)}+w_{13}^{(3)}a_3^{(2)}+b_1^{(3)}))^2+(y_2-f(w_{21}^{(3)}a_1^{(2)}+w_{22}^{(3)}a_2^{(2)}+w_{23}^{(3)}a_3^{(2)}+b_2^{(3)}))^2\right)
\end{aligned}
$$

由求导的链式法则，对“输出层神经元的权重参数”求偏导，有：

$$
\begin{aligned}
\frac{\partial E}{\partial w_{11}^{(3)}} &= \frac{1}{2}\cdot 2(y_1-a_1^{(3)})(-\frac{\partial a_1^{(3)}}{\partial w_{11}^{(3)}}) \\
&= -(y_1-a_1^{(3)})f'(z_1^{(3)})\frac{\partial z_1^{(3)}}{\partial w_{11}^{(3)}} \\
&= -(y_1-a_1^{(3)})f'(z_1^{(3)})a_1^{(2)}
\end{aligned}
$$

误差反向传播

如果我们把 $\frac{\partial E}{\partial z_i^{(l)}}$ 记为 $\delta_i^{(l)}$ ，即做下面的定义：

$$\delta_i^{(l)} \equiv \frac{\partial E}{\partial z_i^{(l)}}$$

则 $\frac{\partial E}{\partial w_{11}^{(3)}}$ 显然可以写为：

$$\begin{aligned}\frac{\partial E}{\partial w_{11}^{(3)}} &= \frac{\partial E}{\partial z_1^{(3)}} \frac{\partial z_1^{(3)}}{\partial w_{11}^{(3)}} \\ &= \delta_1^{(3)} a_1^{(2)}\end{aligned}$$

其中：$\delta_1^{(3)} = \frac{\partial E}{\partial z_1^{(3)}} = \frac{\partial E}{\partial a_1^{(3)}} \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} = -(y_1 - a_1^{(3)}) f'(z_1^{(3)})$

误差反向传播

对于输出层神经元的其它权重参数，同样可求得：

$$\frac{\partial E}{\partial w_{12}^{(3)}} = \delta_1^{(3)} a_2^{(2)}$$
$$\frac{\partial E}{\partial w_{13}^{(3)}} = \delta_1^{(3)} a_3^{(2)}$$
$$\frac{\partial E}{\partial w_{21}^{(3)}} = \delta_2^{(3)} a_1^{(2)}$$
$$\frac{\partial E}{\partial w_{22}^{(3)}} = \delta_2^{(3)} a_2^{(2)}$$
$$\frac{\partial E}{\partial w_{23}^{(3)}} = \delta_2^{(3)} a_3^{(2)}$$

其中，$\delta_2^{(3)} = -(y_2 - a_2^{(3)}) f'(z_2^{(3)})$

说明：之所以要引入记号 $\delta_i^{(l)}$，除了它能简化 $\frac{\partial E}{\partial w_{ij}^{(l)}}$ 和 $\frac{\partial E}{\partial b_i^{(l)}}$ 的表达形式外；更重要的是我们可以通过 $\delta_i^{(l+1)}$ 来求解 $\delta_i^{(l)}$（后文将说明），这样可以充分利用之前计算过的结果来加快整个计算过程。

误差反向传播

推广到一般情况，假设神经网络共 L 层，则：

$$\delta_i^{(L)} = -(y_i - a_i^{(L)}) f'(z_i^{(L)}) \quad (1 \le i \le n_L)$$

$$\frac{\partial E}{\partial w_{ij}^{(L)}} = \delta_i^{(L)} a_j^{(L-1)} \quad (1 \le i \le n_L, 1 \le j \le n_{L-1})$$

如果把上面两式表达为矩阵（向量）形式，则为：

$$\boldsymbol{\delta}^{(L)} = -(\boldsymbol{y} - \boldsymbol{a}^{(L)}) \odot f'(\boldsymbol{z}^{(L)})$$

$$\nabla_{W^{(L)}} E = \boldsymbol{\delta}^{(L)} (\boldsymbol{a}^{(L-1)})^{\mathsf{T}}$$

注：符号 $\odot$ 表示 Element-wise Product Operator，又称作 Hadamard product 。规则简单，把对应位置的元素分别相乘即可。如：

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{pmatrix} \odot \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} & a_{12}b_{12} \\ a_{21}b_{21} & a_{22}b_{22} \\ a_{31}b_{31} & a_{32}b_{32} \end{pmatrix}$$

误差反向传播

3.2 隐藏层的权重参数更新

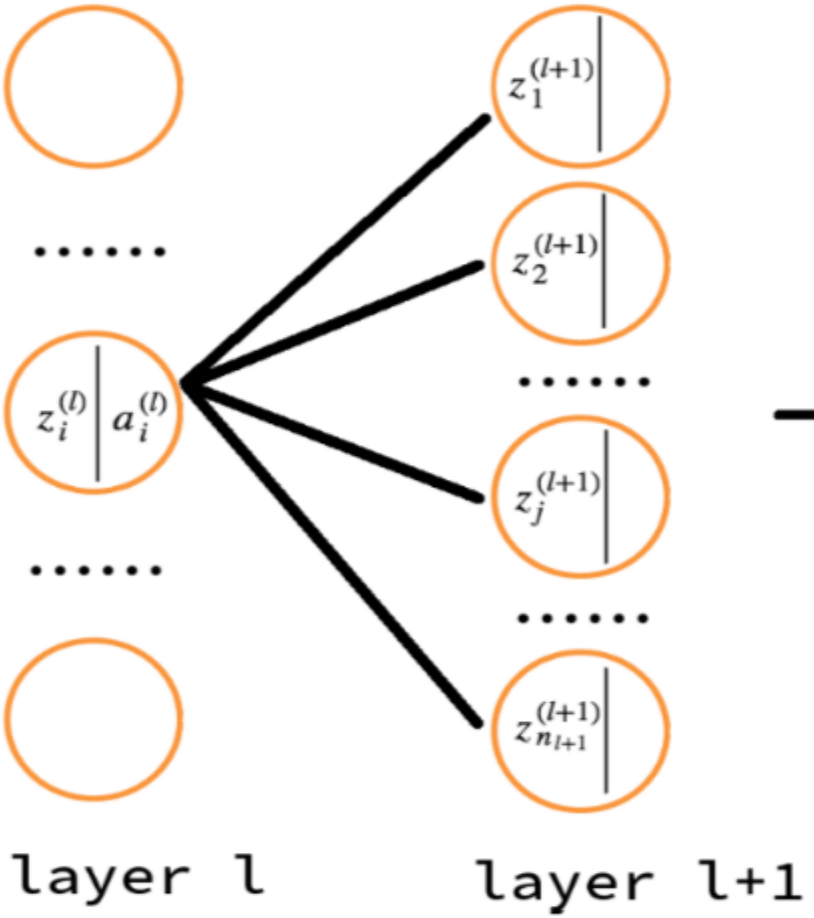
对"隐藏层神经元的权重参数"求偏导，利用 $\delta_i^{(l)}$ 的定义，有：

$$\begin{aligned}\frac{\partial E}{\partial w_{ij}^{(l)}} &= \frac{\partial E}{\partial z_i^{(l)}} \frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l)}} \\ &= \delta_i^{(l)} \frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l)}} \\ &= \delta_i^{(l)} a_j^{(l-1)}\end{aligned}$$

其中 $\delta_i^{(l)}, 2 \le l \le L-1$ 的推导如下：

$$\begin{aligned}\delta_i^{(l)} &\equiv \frac{\partial E}{\partial z_i^{(l)}} \\ &= \sum_{j=1}^{n_{l+1}} \frac{\partial E}{\partial z_j^{(l+1)}} \frac{\partial z_j^{(l+1)}}{\partial z_i^{(l)}} \\ &= \sum_{j=1}^{n_{l+1}} \delta_j^{(l+1)} \frac{\partial z_j^{(l+1)}}{\partial z_i^{(l)}}\end{aligned}$$

误差反向传播



$$\delta_i^{(l)} = \sum_{j=1}^{n_{l+1}} \delta_j^{(l+1)} w_{ji}^{(l+1)} f'(z_i^{(l)})$$

$$= \left(\sum_{j=1}^{n_{l+1}} \delta_j^{(l+1)} w_{ji}^{(l+1)} \right) f'(z_i^{(l)})$$

$\longrightarrow$ E

Figure 2: 由 $\boldsymbol{\delta}^{(l+1)}$ 求 $\delta_i^{(l)}$

误差反向传播

3.3输出层和隐藏层的偏置参数更新

$$\frac{\partial E}{\partial b_i^{(l)}} = \frac{\partial E}{\partial z_i^{(l)}} \frac{\partial z_i^{(l)}}{b_i^{(l)}}$$
$$= \delta_i^{(l)}$$

对应的矩阵（向量）形式为：

$$\nabla_{b^{(l)}} E = \boldsymbol{\delta}^l$$

误差反向传播

3.4 BP算法四个核心公式

前面已经完整地介绍了误差反向传播算法，可总结为下面四个公式：

$$\delta_i^{(L)} = -(y_i - a_i^{(L)}) f'(z_i^{(L)}) \tag{BP-1}$$

$$\delta_i^{(l)} = \left(\sum_{j=1}^{n_{l+1}} \delta_j^{(l+1)} w_{ji}^{(l+1)} \right) f'(z_i^{(l)}) \tag{BP-2}$$

$$\frac{\partial E}{\partial w_{ij}^{(l)}} = \delta_i^{(l)} a_j^{(l-1)} \tag{BP-3}$$

$$\frac{\partial E}{\partial b_i^{(l)}} = \delta_i^{(l)} \tag{BP-4}$$

误差反向传播

BP 算法四个核心公式就是求某个训练数据的代价函数对参数的偏导数，它的具体应用步骤总结如下：

第一步，初始化参数 $W, \boldsymbol{b}$ 。
一般地，把 $w_{ij}^{(l)}, b_i^{(l)}, 2 \leq l \leq L$ 初始化为一个很小的，接近于零的随机值。

注意：不要把 $w_{ij}^{(l)}, b_i^{(l)}, 2 \leq l \leq L$ 全部初始化为零或者相同的其它值，这会导致对于所有 i ， $w_{ij}^{(l)}$ 都会取相同的值。

第二步，利用下面的“前向传播”公式计算每层的状态和激活值：

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

误差反向传播

第三步，计算 $\boldsymbol{\delta}^{(l)}$

首先，利用下面公式计算输出层的 $\boldsymbol{\delta}^{(L)}$

$$\delta_i^{(L)} = -(y_i - a_i^{(L)}) f'(z_i^{(L)}), \quad (1 \le i \le n_L)$$

其中，y_i 是期望的输出（这是训练数据给出的已知值），$a_i^{(L)}$ 是神经网络对训练数据产生的实际输出。

然后，利用下面公式从第 $L-1$ 层到第 2 层依次计算隐藏层的 $\boldsymbol{\delta}^{(l)}, (l = L-1, L-2, L-3, \cdots, 2)$

$$\delta_i^{(l)} = \left(\sum_{j=1}^{n_{l+1}} \delta_j^{(l+1)} w_{ji}^{(l+1)} \right) f'(z_i^{(l)}), \quad (1 \le i \le n_l)$$

第四步，按下面公式求这个训练数据的代价函数对参数的偏导数：

$$\frac{\partial E}{\partial w_{ij}^{(l)}} = \delta_i^{(l)} a_j^{(l-1)}$$

$$\frac{\partial E}{\partial b_i^{(l)}} = \delta_i^{(l)}$$

误差反向传播

3.6 BP 算法总结:用“批量梯度下降”算法更新参数

“批量梯度下降”算法更新参数的总结如下：

(1) 用 BP 算法四个核心公式求得每一个训练数据的代价函数对参数的偏导数；

(2) 按下面公式更新参数：

$$W^{(l)} = W^{(l)} - \frac{\mu}{N}\sum_{i=1}^{N}\frac{\partial E_{(i)}}{\partial W^{(l)}}$$

$$\boldsymbol{b}^{(l)} = \boldsymbol{b}^{(l)} - \frac{\mu}{N}\sum_{i=1}^{N}\frac{\partial E_{(i)}}{\boldsymbol{b}^{(l)}}$$

(3) 迭代执行第 (1), (2) 步，直到满足停止准则（比如相邻两次迭代的误差的差别很小，或者直接限制迭代的次数）。

说明：每对参数进行一次更新都要遍历整个训练数据集，当训练数据集不大时这不是问题，当训练数据集非常巨大时，可以采用随机梯度下降法（每次仅使用一个训练数据来更新参数）。

误差反向传播

4 梯度消失问题及其解决办法

前面介绍过，误差反向传播有下面迭代公式：

$$\delta_i^{(l)} = \left(\sum_{j=1}^{n_{l+1}} \delta_j^{(l+1)} w_{ji}^{(l+1)} \right) f'(z_i^{(l)})$$

其中用到了激活函数 $f(x)$ 的导数。误差从输出层反向传播时，在每一层都要乘以激活函数 $f(x)$ 的导数。

如果我们使用 $\sigma(x)$ 或 $\tanh(x)$ 做为激活函数，则其导数为：

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)) \in [0, 0.25]$$
$$\tanh'(x) = 1 - (\tanh(x))^2 \in [0, 1]$$

可以看到，它们的导数的值域都会小于 1。这样，误差经过每一层传递都会不断地衰减。**当网络导数比较多时，梯度不断地衰减，甚至消失，这使得整个网络很难训练。这就是梯度消失问题** (Vanishing gradient problem)。

减轻梯度消失问题的一个方法是使用线性激活函数（比如 rectifier 函数）或近似线性函数（比如 softplus 函数）。这样，激活函数的导数为 1，误差可以很好地传播，训练速度会提高。

THE END