深度学习1

主讲：郭春乐、刘夏雷

南开大学计算机学院

致谢：本课件主要内容来自浙江大学吴飞教授、

南开大学程明明教授

单选题 1分

设置

1.可以从最小化每个类簇的方差这一视角来解释K均值聚类的结果,下面对这一视角描述不正确的是(   )

A 最终聚类结果中每个聚类集合中所包含数据呈现出来差异性最小

B 每个样本数据分别归属于与其距离最近的聚类质心所在聚类集合

C 每个簇类的方差累加起来最小

D 每个簇类的质心累加起来最小

提交

单选题 1分

设置

2.下面对相关性()和独立性()描述不正确的是( )

A 如果两维变量线性不相关,则皮尔逊相关系数等于0

B 如果两维变量彼此独立,则皮尔逊相关系数等于0

C “不相关”是一个比“独立”要强的概念,即不相关一定相互独立

D 独立指两个变量彼此之间不相互影响

提交

单选题 1分

设置

3.下面对主成分分析的描述不正确的是(　)

A 主成分分析是一种特征降维方法

B 主成分分析可保证原始高维样本数据被投影映射后,其方差保持最大

C 在主成分分析中,将数据向方差最大方向进行投影,可使得数据所蕴含信息没有丢失,以便在后续处理过程中各个数据“彰显个性”

D 在主成分分析中,所得低维数据中每一维度之间具有极大相关度

提交

设置

4.下面对特征人脸算法描述不正确的是(　　)

A 特征人脸方法是一种应用主成分分析来实现人脸图像降维的方法

B 特征人脸方法是用一种称为“特征人脸(eigenface)”的特征向量按照线性组合形式来表达每一张原始人脸图像

C 每一个特征人脸的维数与原始人脸图像的维数一样大

D 特征人脸之间的相关度要尽可能大

提交

提纲

- 深度学习历史发展
- 前馈神经网络
- 卷积神经网络
- 循环神经网络
- 深度生成学习
- 深度学习应用

深度学习的历史发展

1943年，神经科学家Warren McCulloch和逻辑学家Walter Pitts合作提出了"McCulloch–Pitts (MCP) neuron"的思想。MCP可对输入信号线性加权组合，再用符号函数来输出线性加权组合结果，以模拟大脑复杂活动模式。MCP是最早的神经网络雏形。

赫布理论(Hebbian theory)：
神经元之间持续重复经验刺激可导致突触传递效能增加.

Neurons that fire together, wire together.

"神经元之间突触的强弱变化是学习与记忆的生理学基础"这一理论为联结主义人工智能研究提供了认知神经心理学基础。

MCP	Hebbian Theory	Orientation Selective Cell	MLP	Error Back-propagation	Deep Belief Network	○○○
1943	1949	1958	1969	1986	2006	

深度学习的历史发展

1958 年，David Hubel和Torsten Wiesel在实验中发现小猫后脑皮层中不同视觉神经元与瞳孔所受刺激之间存在某种对应关系，由此发现了一种被称为“方向选择性细胞(orientation selective cell)”的神经元细胞，从而揭示了 “视觉系统信息分层处理”这一机制。

神经网络研究的突破来自于Frank Rosenblatt在20世纪50年代所提出的“感知机（perceptron）”模型。

由于感知机中没有包含非线性变换操作的隐藏层，因此感知机表达能力较弱（如无法解决异或问题）。

MCP	Hebbian Theory	Orientation Selective Cell	MLP	Error Back-propagation	Deep Belief Network
1943	1949	1958	1969	1986	2006

深度学习的历史发展

最早由Werbos提出[Werbos 1974] [Werbos 1990]、并且由Rumelhar和Hinton等人[Rumelhart 1986]完善的误差后向传播（error backpropagation）算法解决了多层感知机中参数优化这一难题。

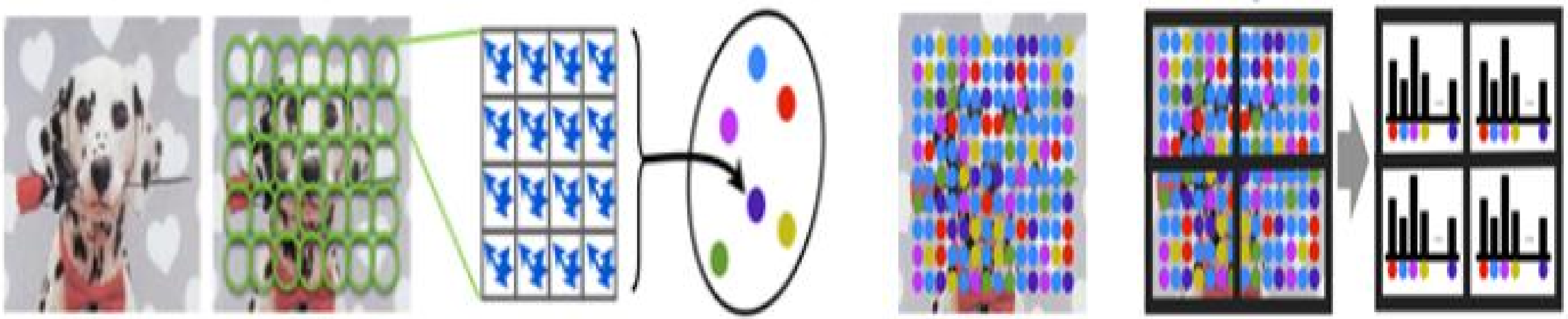
2006年，Hinton在《Science》等期刊上发表了论文，首次提出了“深度信念网络(deep belief network)”模型[Hinton 2006]，在相关分类任务上可取得性能超过了传统浅层学习模型（如支持向量机），使得深度架构引起了大家的关注。

MCP	Hebbian Theory	Orientation Selective Cell	MLP	Error Back-propagation	Deep Belief Network	○○○
1943	1949	1958	1969	1986	2006	

提纲

- 深度学习历史发展
- 前馈神经网络
- 卷积神经网络
- 循环神经网络
- 深度生成学习
- 深度学习应用

浅层学习 VS 深度学习：从分段学习到端到端学习



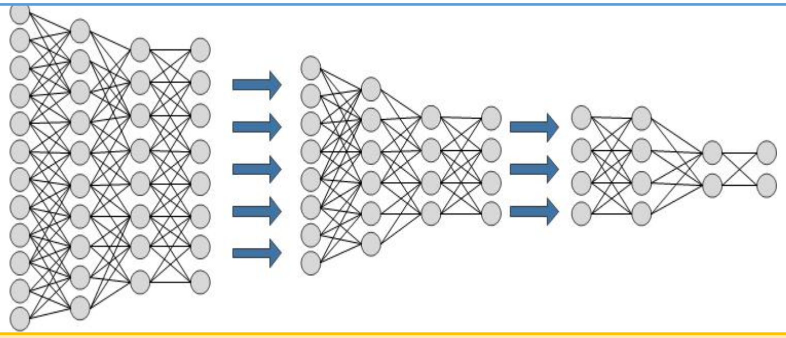
图像 | 稠密信息点 | 信息点表达 | 视觉词典 | 视觉单词 | 金字塔模型 | 直方图

识别

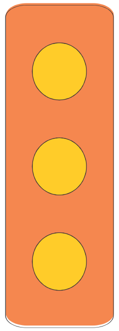
分类

聚类

输入端



端到端学习：
通过卷积、池化和误差反向传播等手段，进行特征学习



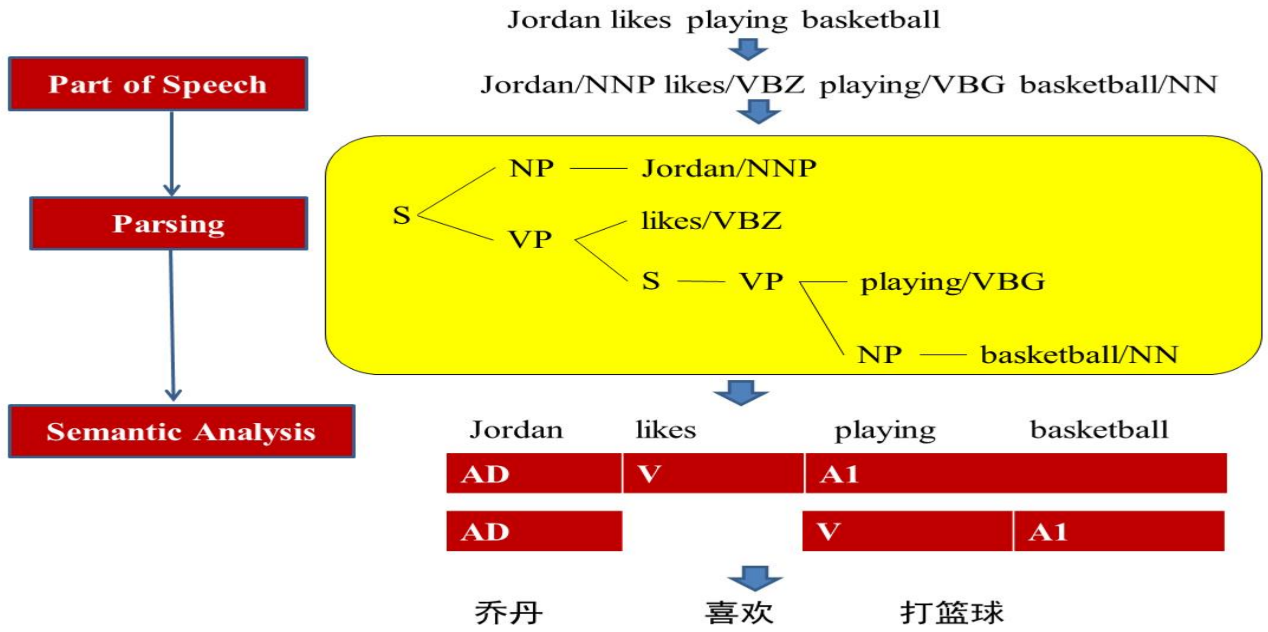
输出端：视觉对象

浅层学习 VS 深度学习：从分段学习到端到端学习

词性标注

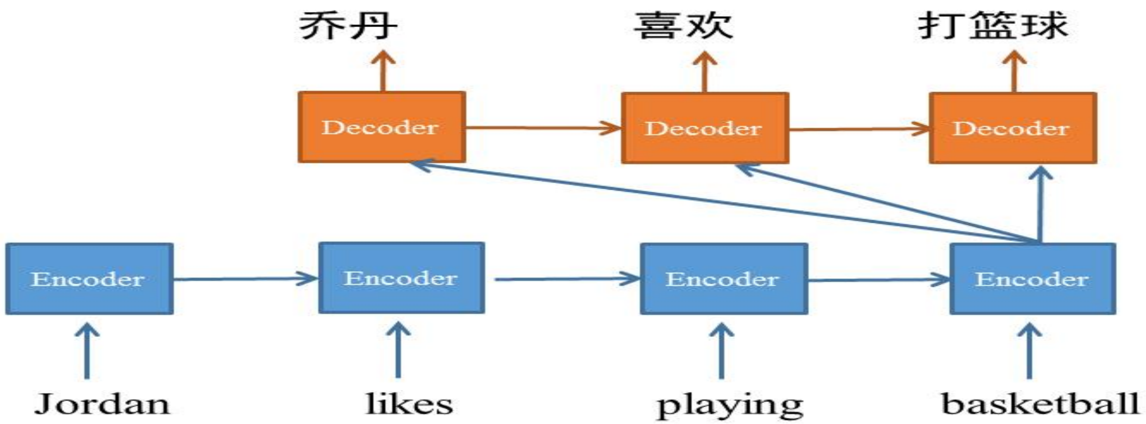
语法解析树

语义分析

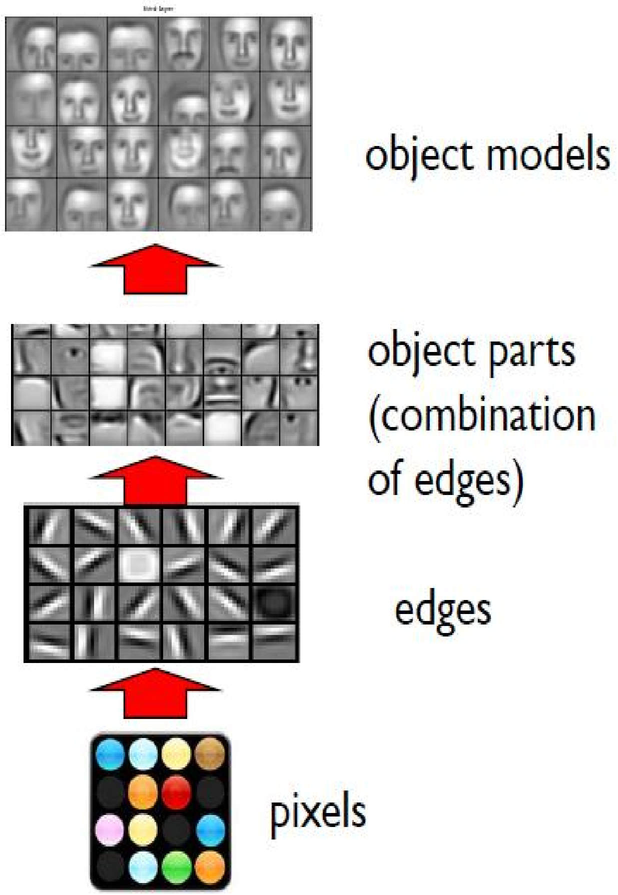


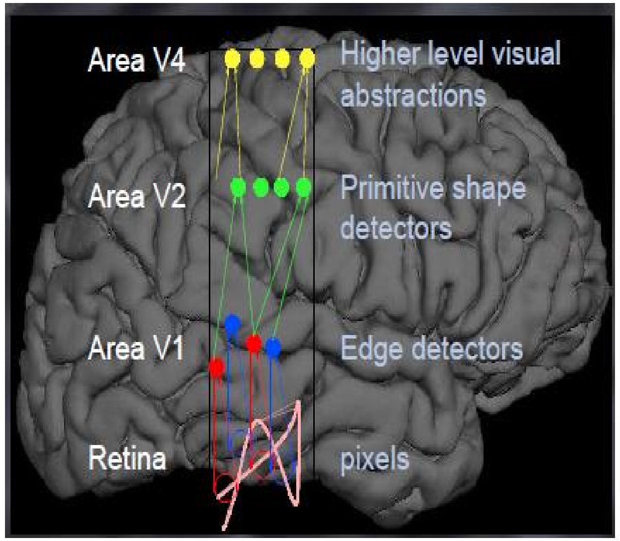
解码器

编码器

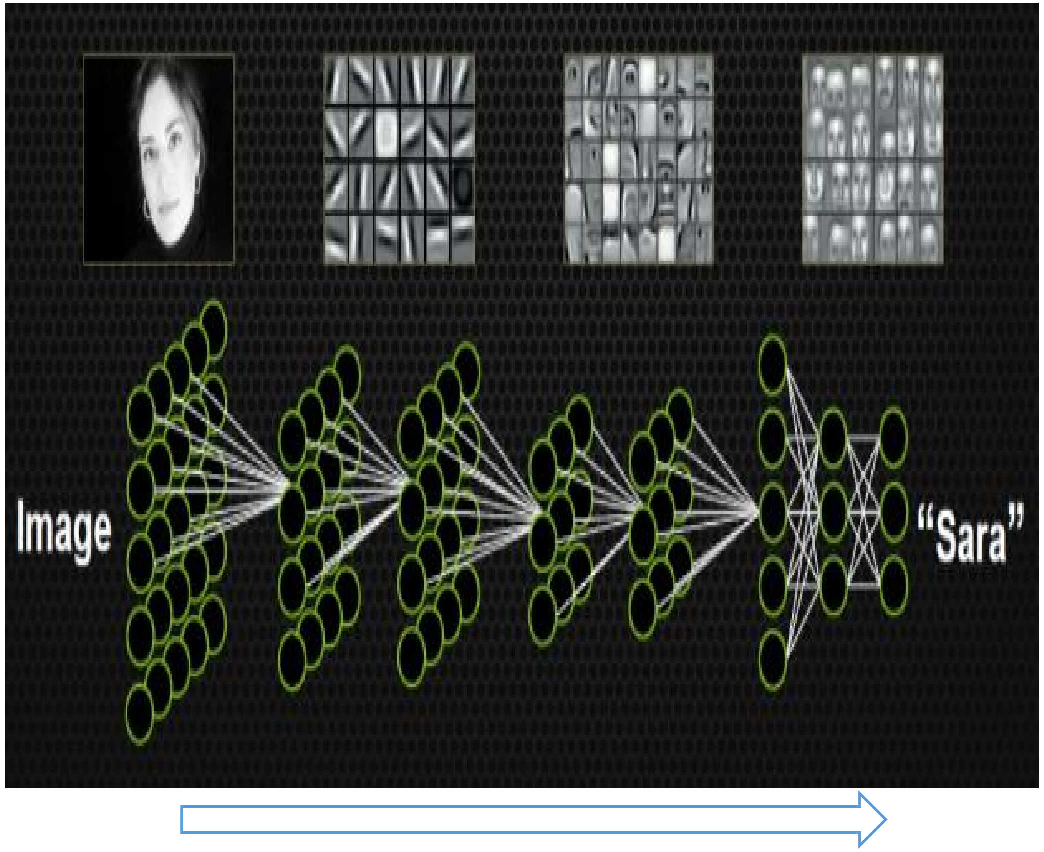


深度学习：以端到端的方式逐层抽象、逐层学习





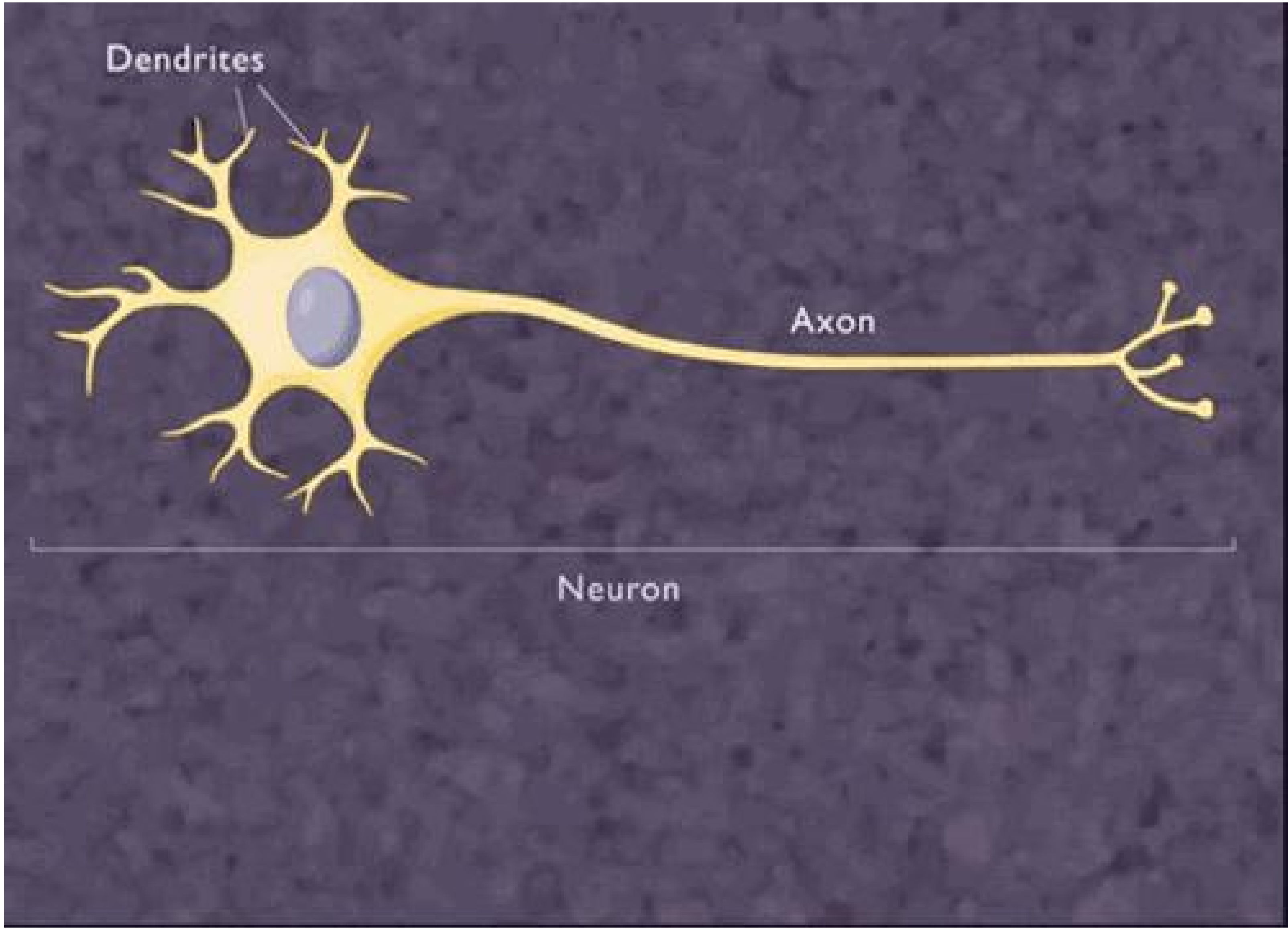
Slide credit: Andrew Ng



- 深度学习所得模型可视为一个复杂函数
- 非线性变换与映射的过程：像素点→语义

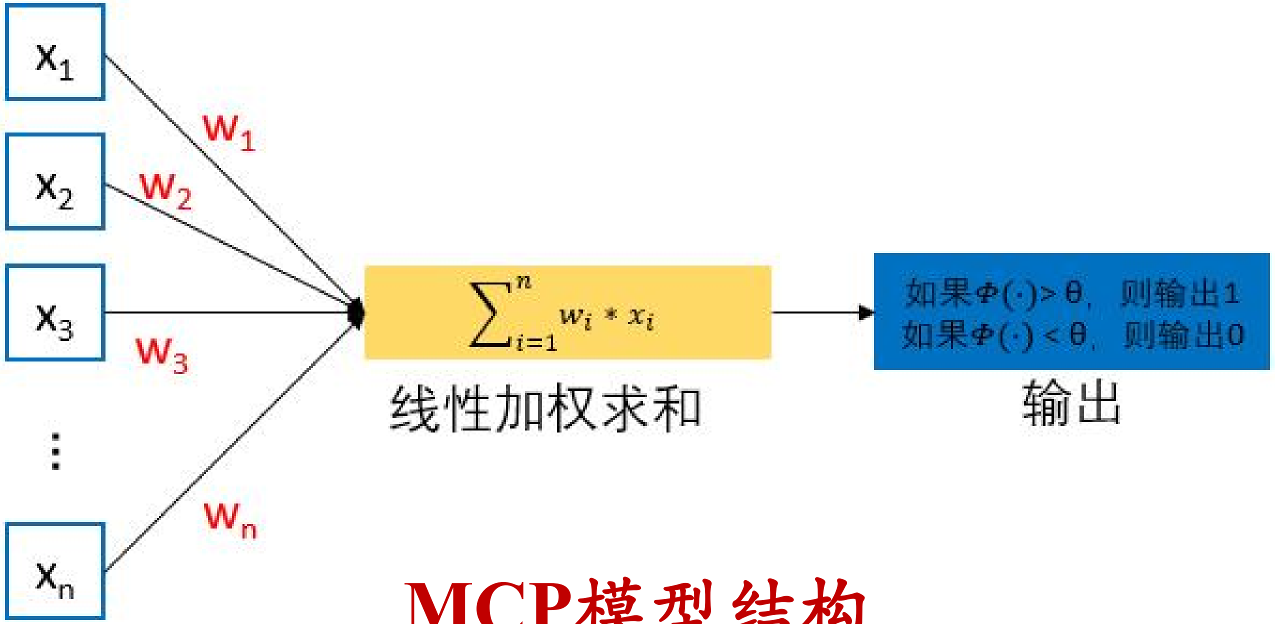
生物学中的神经元

- 神经元细胞有兴奋与抑制两种状态。多数神经元默认处于抑制状态，一旦某个神经元受到刺激并且电位超过一定的阈值后，这个神经元就被激活，处于兴奋状态，并向其他神经元传递信息。



神经元

- 基于神经元细胞的结构特性与传递信息方式，神经科学家Warren McCulloch和逻辑学家Walter Pitts合作提出了“McCulloch–Pitts (MCP) neuron”模型[McCulloch 1943]。在人工神经网络中，MCP模型成为人工神经网络中的最基本结构。

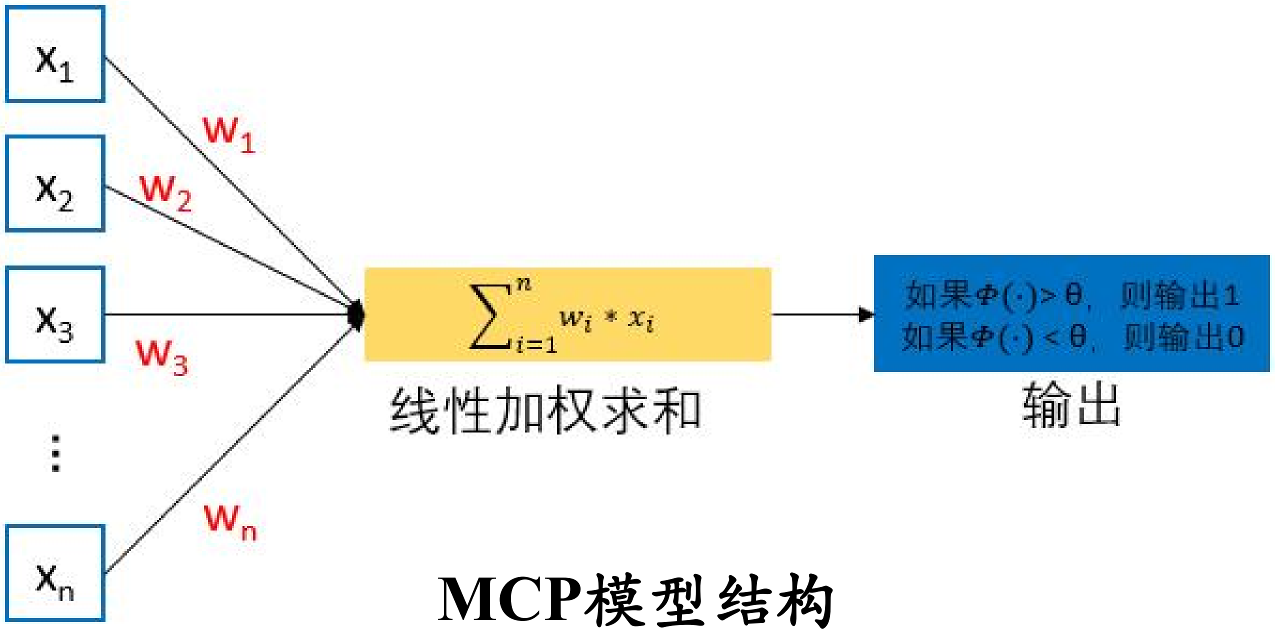


MCP模型结构

神经元

- 给定n个二值化（0或1）的输入数据 $x_i(1 \le i \le n)$与连接参数$w_i(1 \le i \le n)$，MCP神经元模型对输入数据线性加权求和，然后使用函数$\Phi(\cdot)$将加权累加结果映射为0或1，以完成两类分类的任务：

$$y = \Phi\left(\sum_{i=1}^{n} w_i x_i\right)$$

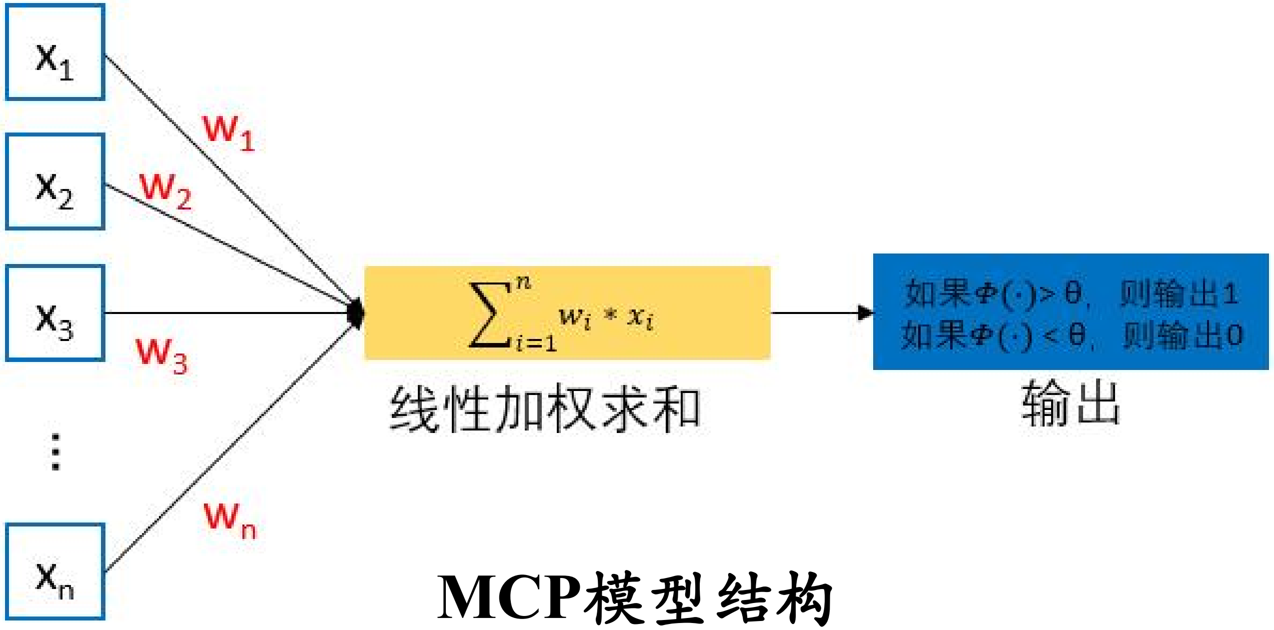


MCP模型结构

神经元

$$y = \boldsymbol{\Phi}\left(\sum_{i=1}^{n} w_i x_i\right)$$

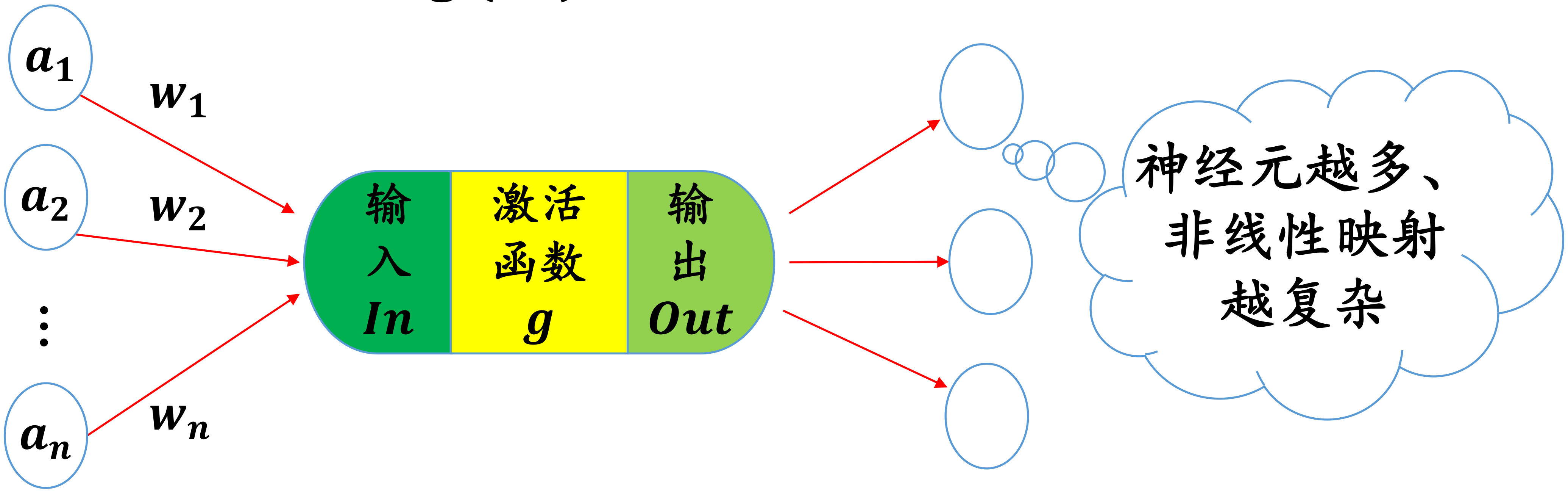
- 其中w_i为预先设定的连接权重值(取值范围$[0,1]$或$[-1,1]$)，表示对应输入数据对输出结果的影响(即权重)。$\Phi(\cdot)$将输入数据线性加权累加结果与预先设定阈值θ进行比较，输出比较结果1或0。



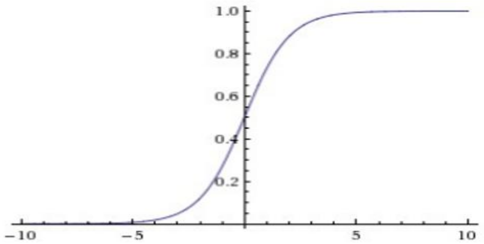
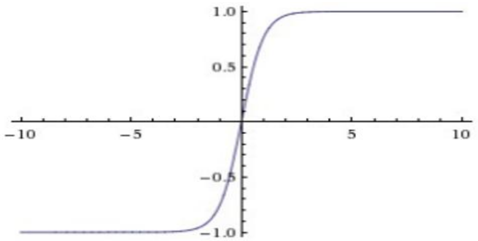
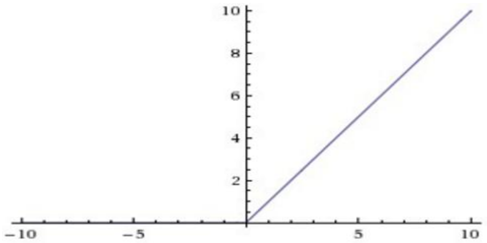
MCP模型结构

刻画神经元功能的数学模型

- **神经元是深度学习模型中基本单位**
  - 对相邻前向神经元输入信息进行加权累加：$In = \sum_{i=1}^{n} w_i * a_i$
  - 对累加结果进行非线性变换（通过激活函数）：$g(x)$
  - 神经元的输出：$Out = g(In)$



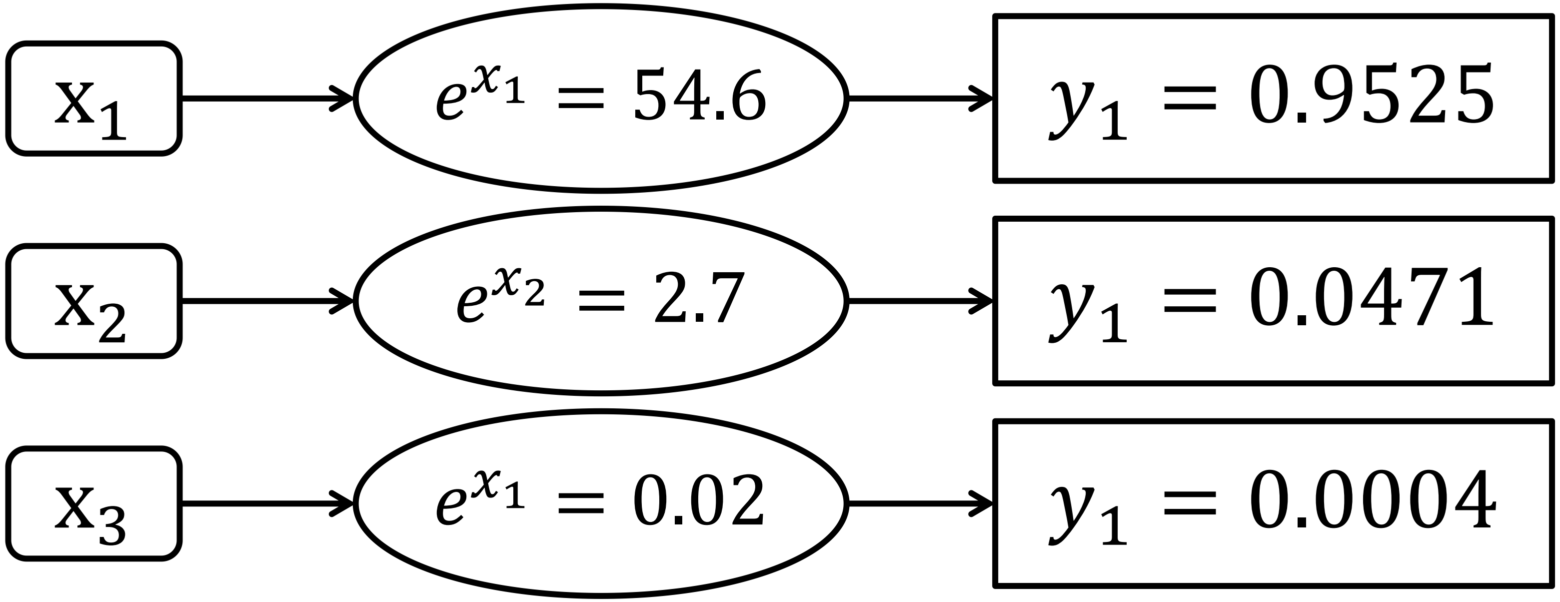
常用的激活函数：对输入信息进行非线性变换

激活函数名称	函数功能	函数图像	函数求导
Sigmoid	$f(x)=\dfrac{1}{1+e^{-x}}$		$f'(x)=f(x)(1-f(x))$
Tanh	$f(x)=\dfrac{2}{1+e^{-2x}}-1$		$f'(x)=1-f(x)^2$
Relu (Rectified Linear Unit)	$f(x)=\begin{cases}0, & for\ x<0\\ x, & for\ x\geq 0\end{cases}$		$f'(x)=\begin{cases}0, & for\ x<0\\ 1, & for\ x\geq 0\end{cases}$

神经网络使用非线性函数作为激活函数(activation function)，通过对多个非线性函数进行组合，来实现对输入信息的非线性变换

常用的激活函数：softmax函数

- Softmax函数一般用于多分类问题中
  - 将输入$\boldsymbol{x}_i$映射到第� 个类别的概率：$\boldsymbol{y}_i = \mathbf{softmax}(\boldsymbol{x}_i) = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}}$
  - $0 < y_i < 1$, $\sum_i y_i = 1$, 可将输出概率最大的作为分类目标



损失函数（Loss Function）

- **损失函数又称为代价函数（Cost Function）**
  - 用来计算模型预测值与真实值之间的误差。
  - 损失函数是神经网络设计中的一个重要组成部分。
  - 通过定义与任务相关的良好损失函数，在训练过程中可根据损失函数来计算神经网络的误差大小，进而优化神经网络参数。
- **两种最常用损失函数：**
  - 均方误差损失函数
  - 交叉熵损失函数

常用的损失函数

- **均方误差损失函数**
  - 通过计算预测值和实际值之间距离（即误差）的平方来衡量模型优劣。假设有n个训练数据x_i，每个训练数据x_i的真实输出为y_i，模型对x_i的预测值为$\hat{y}_i$。该模型在n个训练数据下所产生的均方误差损失可定义如下：

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

常用的损失函数

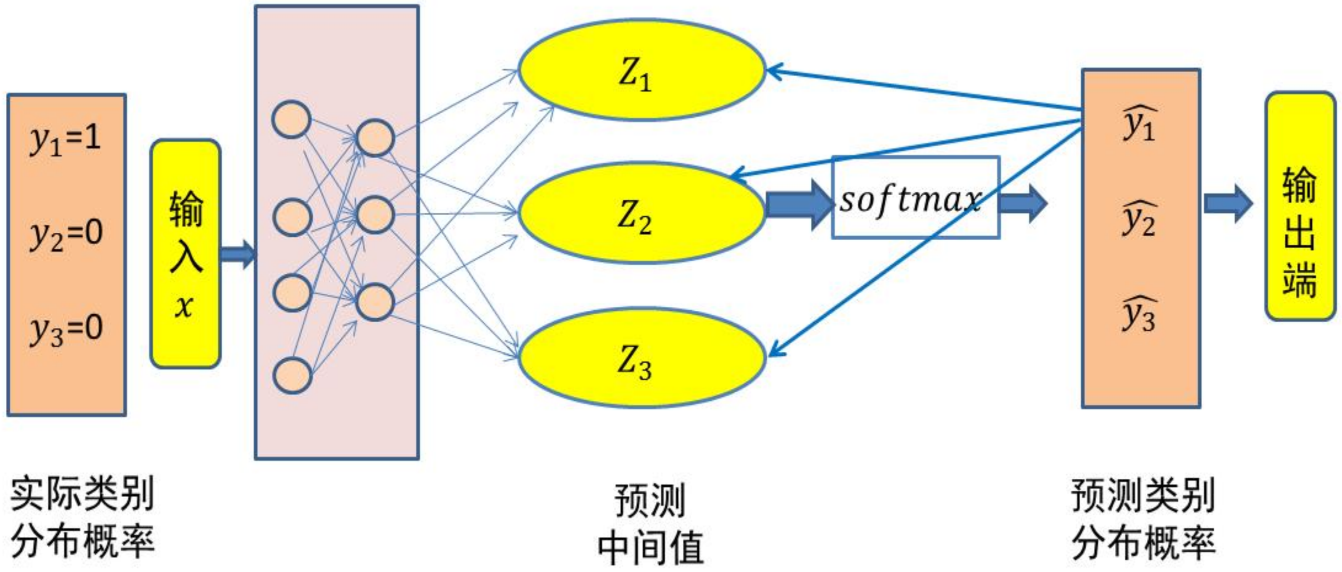
- **交叉熵（cross entropy）是信息论中的重要概念**
  - 用来度量两个概率分布间的差异。假定p和q是数据x的两个概率分布，通过q来表示p的交叉熵可如下计算：

$$\mathrm{H}(p,\ q) = -\sum_{x} p(x) * logq(x)$$

  - 刻画了两个概率分布之间的距离，旨在描绘通过概率分布q来表达概率分布p的困难程度。根据公式不难理解，交叉熵越小，两个概率分布p和q越接近。

常用的损失函数

- **交叉熵损失函数**
  - 假设数据x属于类别1。记x的实际类别分布概率为$y = (1,0,0)$
  - $\hat{y}$代表模型预测所得类别分布概率
  - 那么对x而言，y和$\hat{y}$的交叉熵损失函数定义为$-y \times \log(\hat{y})$

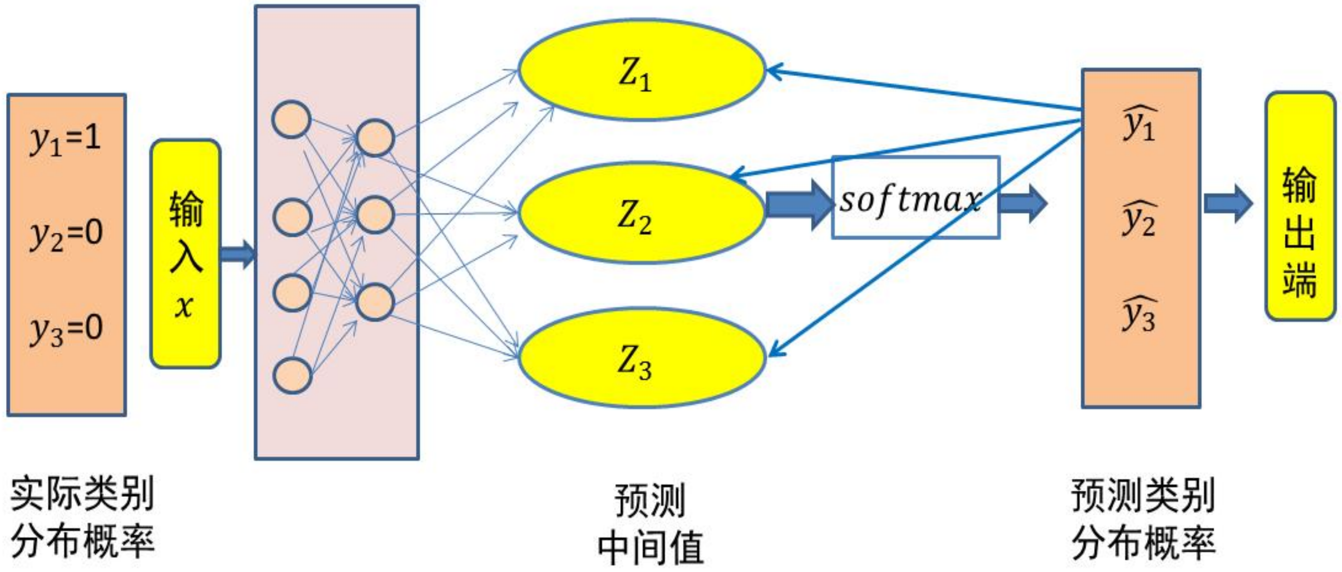


常用的损失函数

- **交叉熵损失函数**
  - 一个良好的神经网络要尽量保证对于每一个输入数据，类别分布概率的预测值与实际值之间的差距越小越好。
  - 可将交叉熵作为损失函数来训练神经网络。

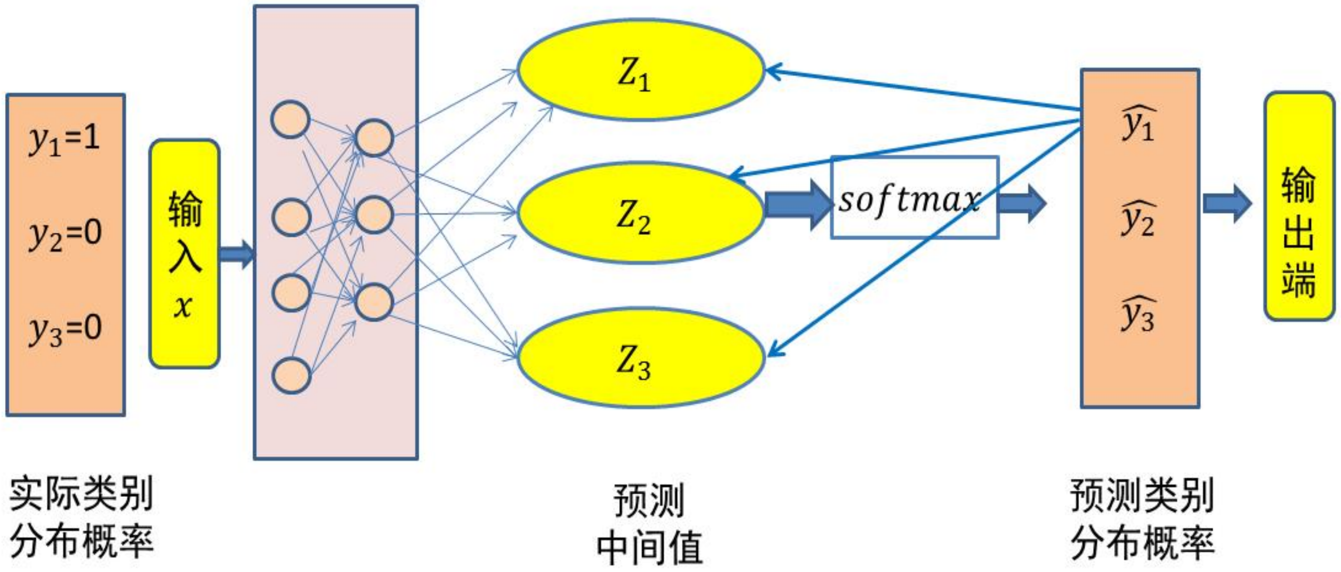
常用的损失函数

- **交叉熵损失函数：一个三个类别分类的例子**
  - 由于输入x属于类别1，因此实际类别概率分布为$y = (1,0,0)$
  - 预测中间值(z_1, z_2, z_3)，经过Softmax函数映射，得到预测的类别分布概率$\hat{y} = (\widehat{y_1}, \widehat{y_2}, \widehat{y_3})$。



常用的损失函数

- **交叉熵损失函数：一个三个类别分类的例子**
  - 根据前面的介绍，$\widehat{y_1}$、$\widehat{y_2}$和$\widehat{y_3}$为(0,1)范围之间的一个概率值。由于样本x属于第一个类别，因此希望神经网络所预测得到的$\widehat{y_1}$取值要远远大于$\widehat{y_2}$和$\widehat{y_3}$的取值。



常用的损失函数

- **交叉熵损失函数**
  - 训练中可利用如下交叉熵损失函数来对模型参数进行优化：

$$\text{CrossEntropy} = -\left(y_1 \times \log(\widehat{y_1}) + y_2 \times \log(\widehat{y_2}) + y_3 \times \log(\widehat{y_3})\right)$$

  - 在上式中，y_2和y_3均为0、y_1为1，因此损失函数简化为：

$$-y_1 \times \log(\widehat{y_1}) = -\log(\widehat{y_1})$$

  - 在神经网络训练中，要将输入数据实际的类别概率分布与模型预测的类别概率分布之间的误差（即损失）从输出端向输入端传递，以便优化模型参数

常用的损失函数--交叉熵损失函数

- **据交叉熵计算的误差从$\widehat{y_1}$传递给z_1**

$$\frac{\partial \widehat{y_1}}{\partial z_1}=\frac{\partial}{\partial z_1}\left(\frac{e^{z_1}}{\sum_k e^{z_k}}\right)=\frac{(e^{z_1})' \times \sum_k e^{z_k}-e^{z_1} \times e^{z_1}}{(\sum_k e^{z_k})^2}$$

$$=\frac{e^{z_1}}{\sum_k e^{z_k}}-\frac{e^{z_1}}{\sum_k e^{z_k}} \times \frac{e^{z_1}}{\sum_k e^{z_k}}=\widehat{y_1}(1-\widehat{y_1})$$

- 注意：在推导中应用了$(e^x)'=e^x$、$(\frac{v}{u})'=\frac{v'u-vu'}{u^2}$等公式
- 由于交叉熵损失函数$-\log(\widehat{y_1})$对$\widehat{y_1}$求导的结果为$-\frac{1}{\widehat{y_1}}$，$\widehat{y_1}(1-\widehat{y_1})$与$-\frac{1}{\widehat{y_1}}$相乘为$\widehat{y_1}-1$。这说明一旦得到模型预测输出$\widehat{y_1}$，将该输出减去1就是交叉损失函数相对于$z_1$的偏导结果。

常用的损失函数--交叉熵损失函数

- **据交叉熵计算的误差从$\widehat{y_1}$传递给z_2（z_3的推导同z_2）**

$$\frac{\partial \widehat{y_1}}{\partial z_2} = \frac{\partial}{\partial z_2}\left(\frac{e^{z_1}}{\sum_k e^{z_k}}\right) = \frac{0 \times \sum_k e^{z_k} - e^{z_1} \times e^{z_2}}{(\sum_k e^{z_k})^2} = -\frac{e^{z_1}}{\sum_k e^{z_k}} \times \frac{e^{z_2}}{\sum_k e^{z_k}} = -\widehat{y_1}\widehat{y_2}$$

- 注意：在推导中应用了$(e^x)' = e^x$、$(\frac{v}{u})' = \frac{v'u - vu'}{u^2}$等公式

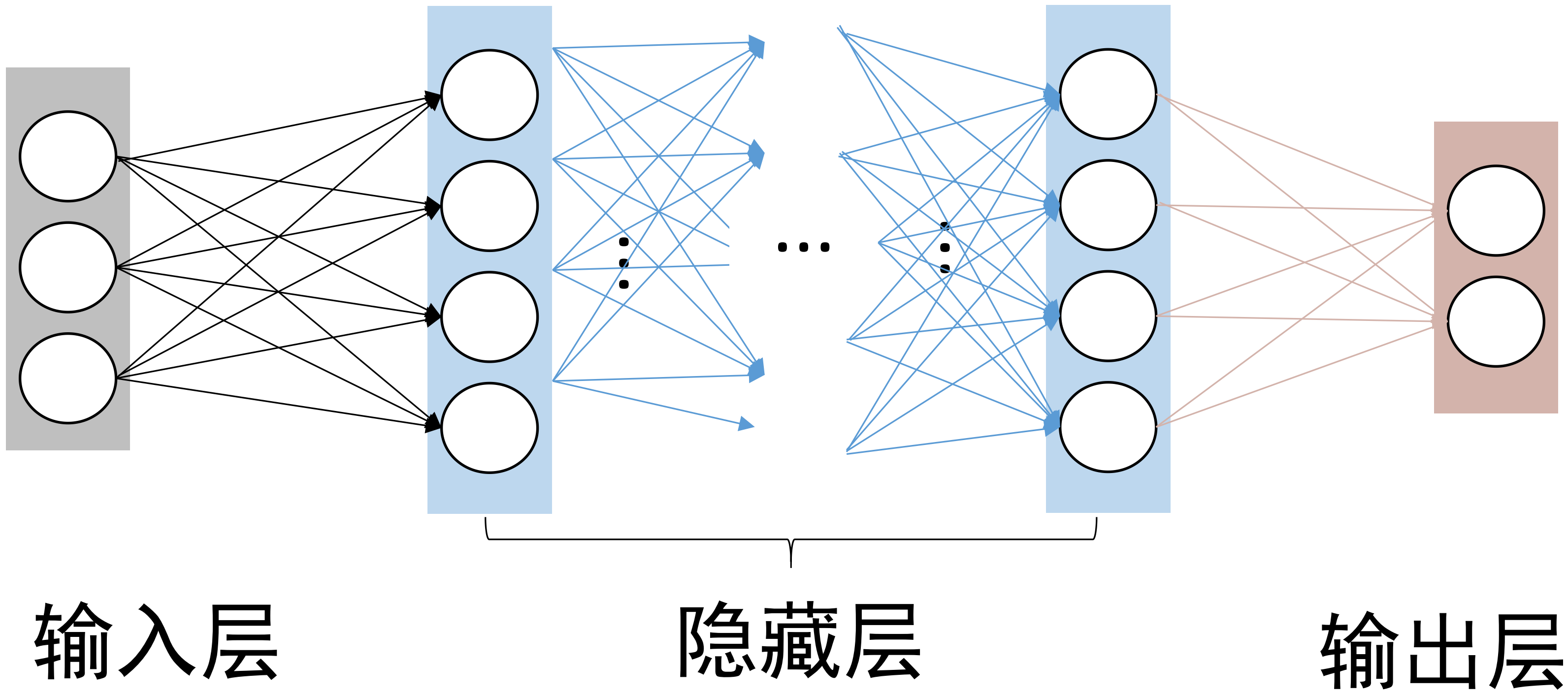
- 交叉熵损失函数导数为$-\frac{1}{\widehat{y_1}}$，相乘结果为$\widehat{y_2}$。这意味对于除第一个输出节点以外的节点进行偏导，模型预测输出就是交叉损失函数相对于其他节点的偏导。在z_1、z_2和z_3得到偏导结果后，再通过链式法则（后续介绍）将损失误差继续往输入端传递

常用的损失函数

- **交叉熵损失函数**
  - 上面的例子中，假设预测中间值(z_1,z_2,z_3)经过Softmax映射后为(0.34,0.46,0.20)。由于输入数据x属于第一类，显然这个输出不理想而需要对模型参数进行优化。如果选择交叉熵损失函数来优化模型，则(z_1,z_2,z_3)这一层的偏导为$(0.34-1,0.46,0.20)$。
  - Softmax和交叉熵损失函数相互结合，为偏导计算带来了极大便利。偏导计算使得损失误差从输出端向输入端传递，来对模型进行优化。交叉熵与Softmax函数结合在一起，因此也叫Softmax with cross-entropy loss

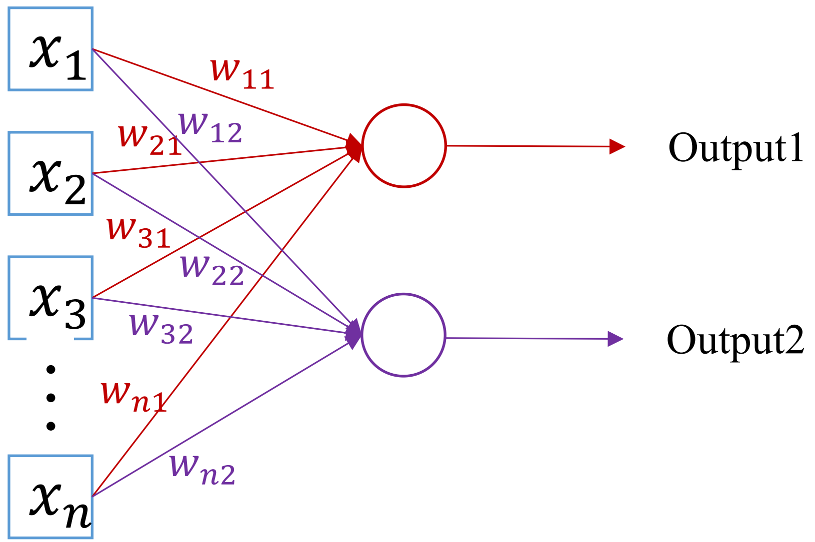
前馈神经网络（feedforward neural network）

- 各神经元接受前一级输入并输出到下一级，模型中没有反馈
- 层间“全连接”，即两个相邻层之间的神经元完全成对连接，但层内的神经元不相互连接。



前馈神经网络（feedforward neural network）

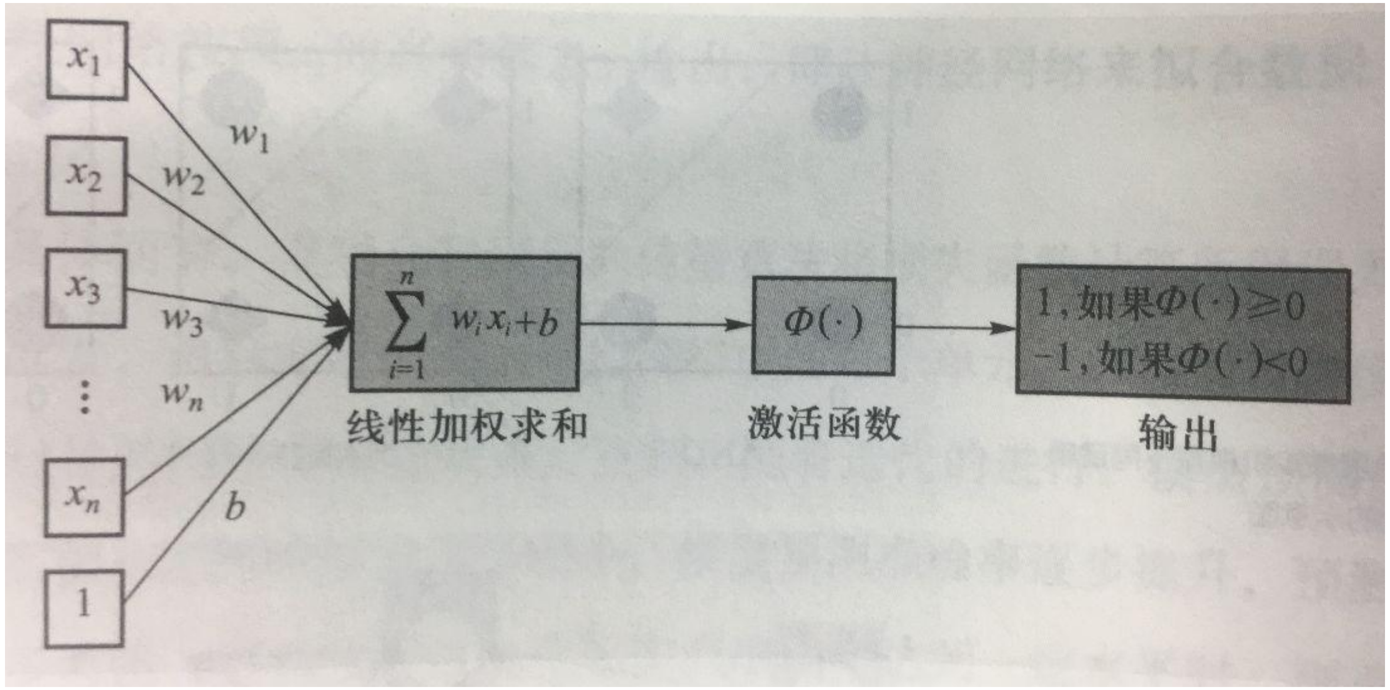
- **感知机网络(Perceptron Networks)是一种特殊的前馈神经网络**
  - 无隐藏层，只有输入层/输出层
  - 无法拟合复杂的数据



$w_{ij}(1 \le i \le n, 1 \le j \le 2)$ **构成了感知机模型参数**

感知机模型：单层感知机

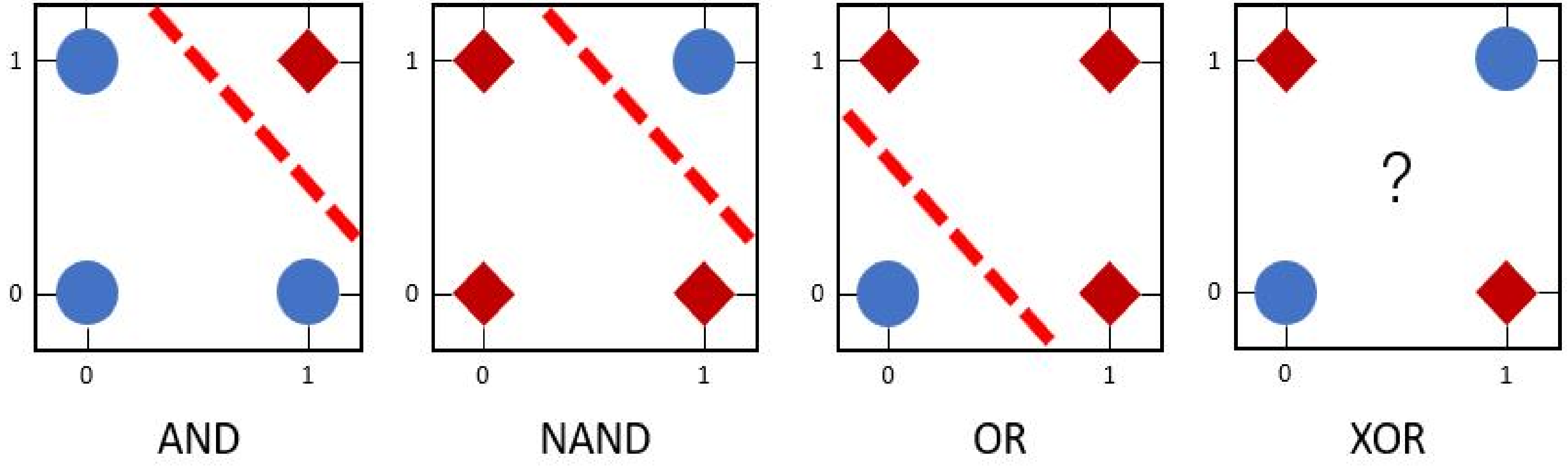
- 早期的感知机结构和MCP模型相似，由一个输入层和一个输出层构成，因此也被称为“单层感知机”。感知机的输入层负责接收实数值的输入向量，输出层则能输出1或-1两个值。



感知机模型

感知机模型：单层感知机

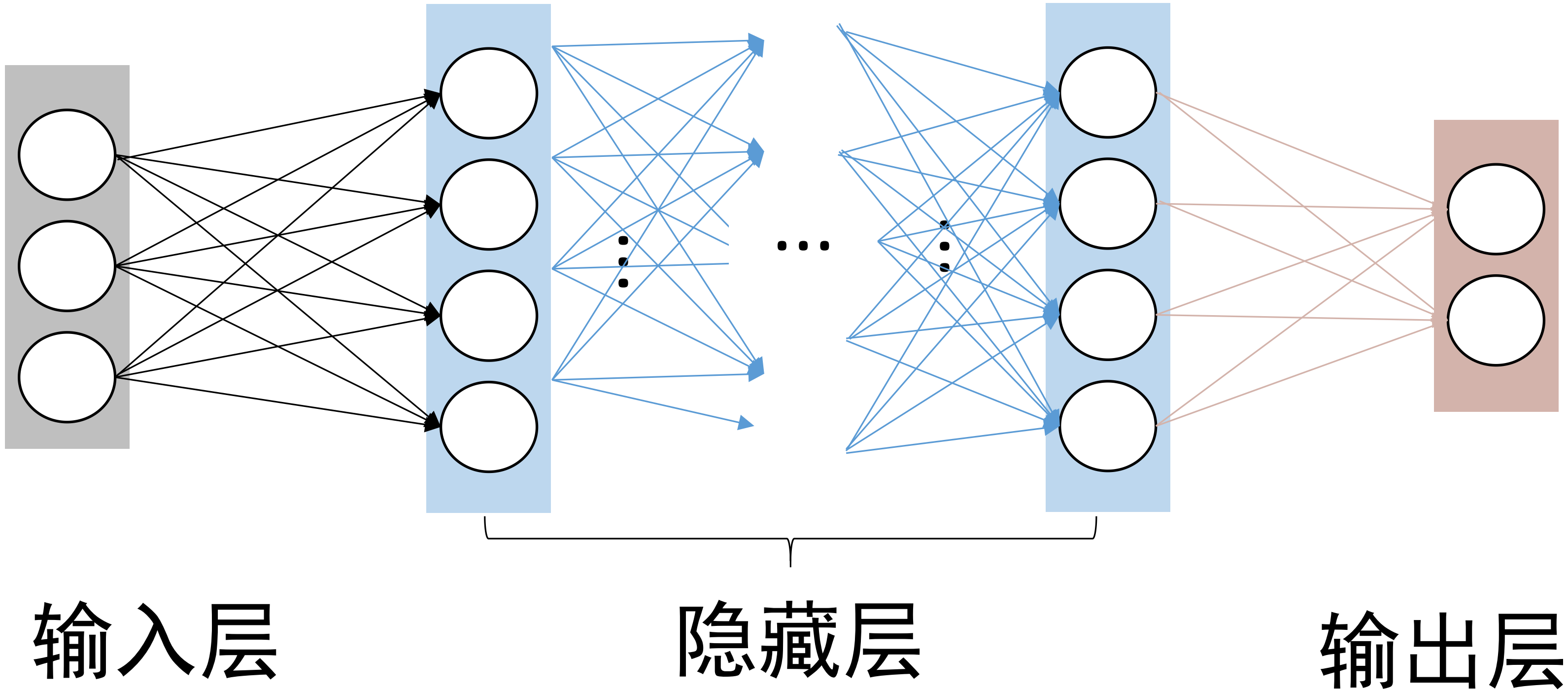
- **单层感知机可被用来区分线性可分数据**
  - 图中，逻辑与(AND)、逻辑与非(NAND)和逻辑或(OR)为线性可分函数，所以可利用单层感知机来模拟这些逻辑函数。
  - 逻辑异或（XOR）是非线性可分的，因此单层感知机无法模拟



单层感知机模拟不同逻辑函数功能的示意图

感知机模型：多层感知机

- 多层感知机由输入层、输出层和至少一层的隐藏层构成



多层感知机模型
前馈神经网络

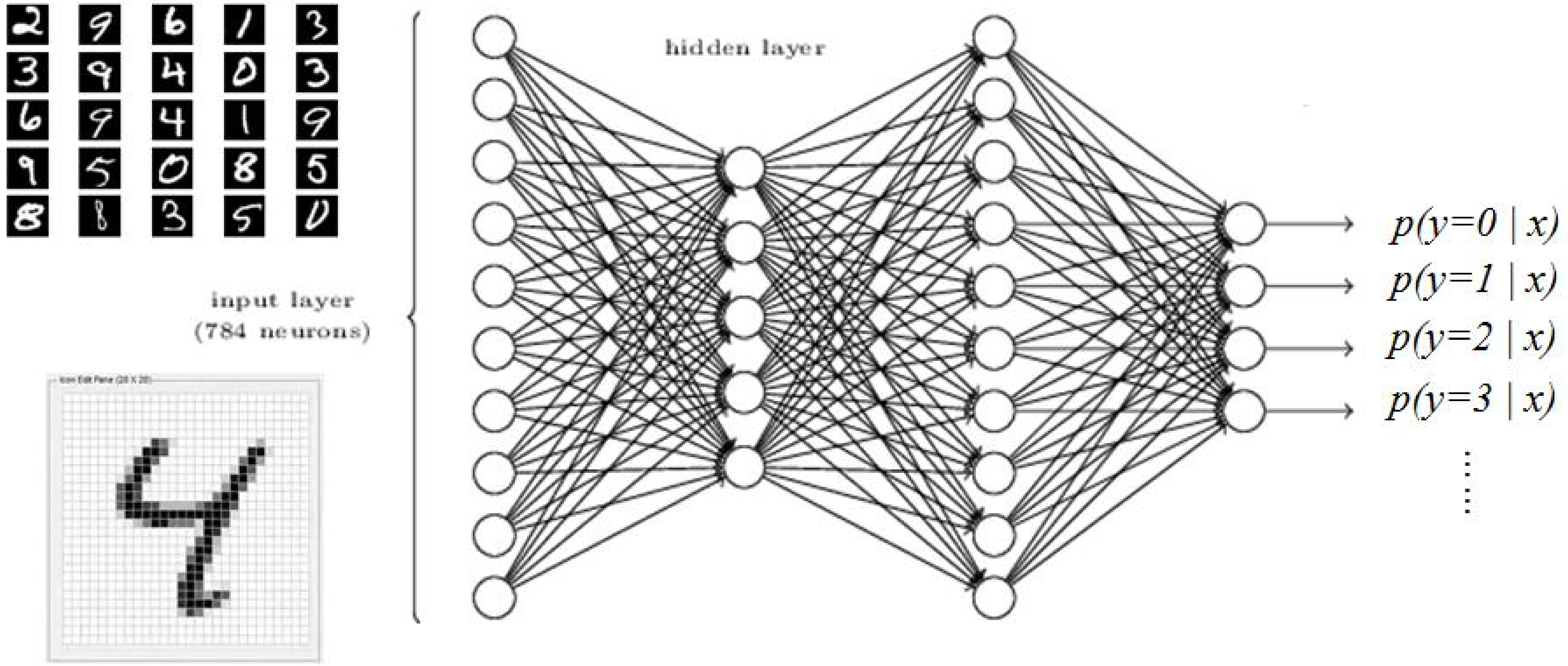
感知机模型：多层感知机

- **多层感知机由输入层、输出层和至少一层的隐藏层构成**
  - 各个隐藏层中神经元可接收相邻前序隐藏层中所有神经元传递的信息，加工处理后输出给相邻后续隐藏层中所有神经元
  - 各个神经元接受前一级的输入，并输出到下一级，模型中没有反馈
  - 层与层之间通过“全连接”进行链接，即两个相邻层之间的神经元完全成对连接，但层内的神经元不相互连接。

问题：如何优化网络参数？

$$w_{ij}(1 \le i \le n, 1 \le j \le m)$$

n为神经网络层数、 m为每层中神经元个数



问题：如何优化网络参数？

- **从标注数据出发，优化模型参数**
  - 标注数据：$(x_i, y_i)(1 \leq i \leq N)$
  - 评分函数(scoring function)将输入数据映射为类别置信度大小：$s = f(x) = W\varphi(x)$
  - 损失函数来估量模型预测值与真实值之间的差距。损失函数给出的差距越小，则模型鲁棒性就越好。常用的损失函数有softmax或者SVM。

问题：如何优化网络参数？

- **从标注数据出发，优化模型参数**
  - 标注数据：$(x_i, y_i)(1 \le i \le N)$
  - 评分函数(scoring function)将输入数据映射为类别置信度大小
  - 损失函数来估量模型预测值与真实值之间的差距

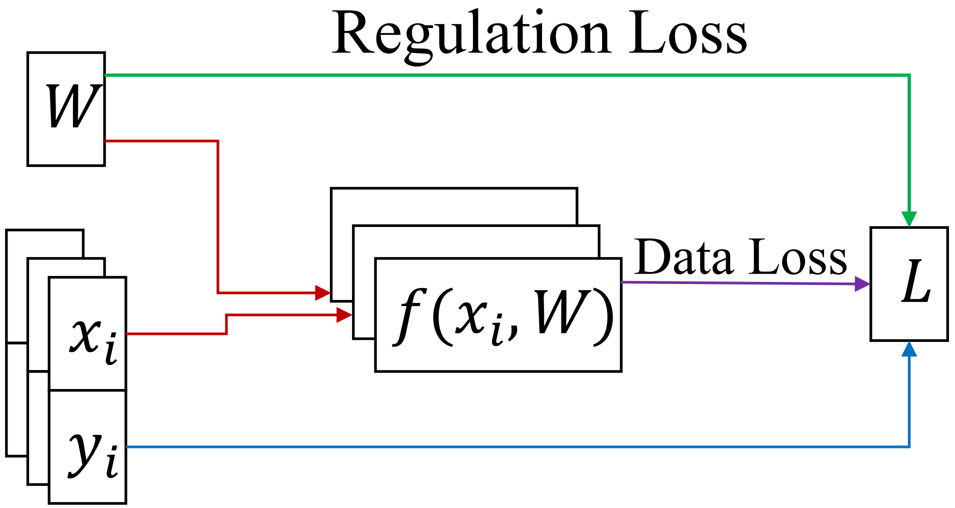
$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

分类正确样本的Softmax值所占比重越大，其Loss越小

$$L_i = \sum_{j \ne y_i} \max(0,\ s_j - s_{y_i} + 1)$$

分类正确样本置信度比分类错误样本置信度至少大1

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + R(W)$$



优化目标：损失值小、参数不复杂

参数优化：梯度下降（Gradient Descent)

- 梯度下降算法是一种使得**损失函数**最小化的方法。一元变量所构成函数� 在� 处梯度为：

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- 在多元函数中，梯度是对每一变量所求导数组成的向量
- 梯度的反方向是函数值下降最快的方向，因此是求解的方向

问题：如何优化网络参数？

- 假设损失函数$f(x)$是连续可微的多元变量函数，其泰勒展开如下(Δx是微小的增量)：

$$f(x+\Delta x)=f(x)+f'(x)\Delta x+\ldots+\frac{1}{n!}f^{(n)}(x)\,(\Delta x)^n$$

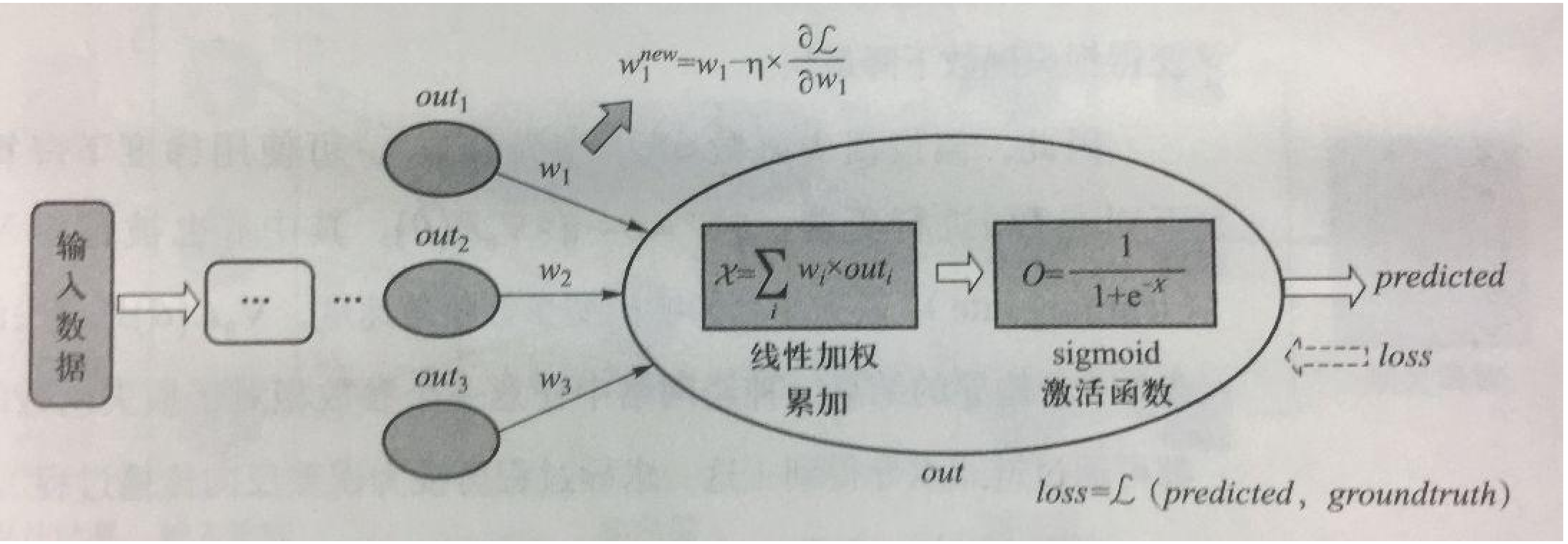
$$f(x+\Delta x)-f(x)\approx(\nabla f(x))^T\Delta x$$

- 因为 $(\nabla f(x))^T\Delta x=\|\nabla f(x)\|\|\Delta x\|\cos\theta$, 为了最小化损失函数，可以沿着梯度的反方向 (i.e., $\theta=\pi$)来寻找参数取值

$$f(x+\Delta x)-f(x)=\|\nabla f(x)\|\|\Delta x\|\cos\theta=-\underset{\substack{\uparrow\\ \text{步长}}}{\alpha}\|\nabla \underset{\substack{\uparrow\\ \text{函数偏导}}}{f}(x)\|$$

参数优化：误差反向传播（error Back Propagation)

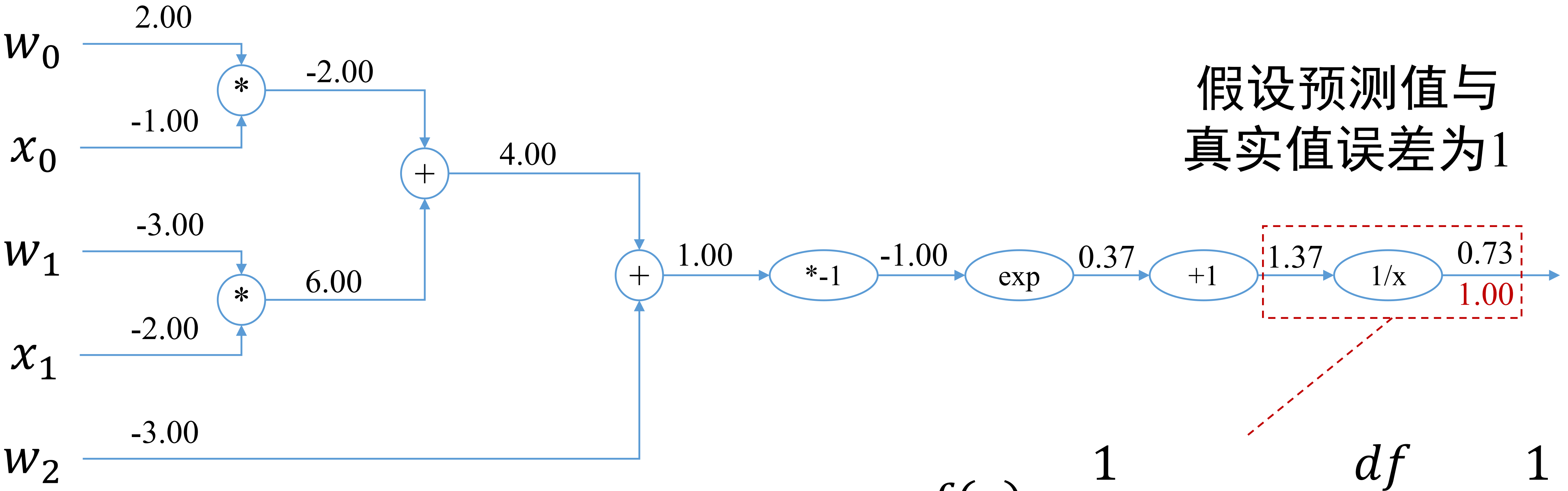
- **BP算法将输出层误差反向传播给隐藏层更新参数**
  - 将误差从后向前传递，将误差分摊给各层所有单元，从而获得各层单元所产生的误差，进而依据这个误差来让各层单元负起各自责任、修正各单元参数。



链式求导与模型参数更新示意图

误差反向传播例子

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

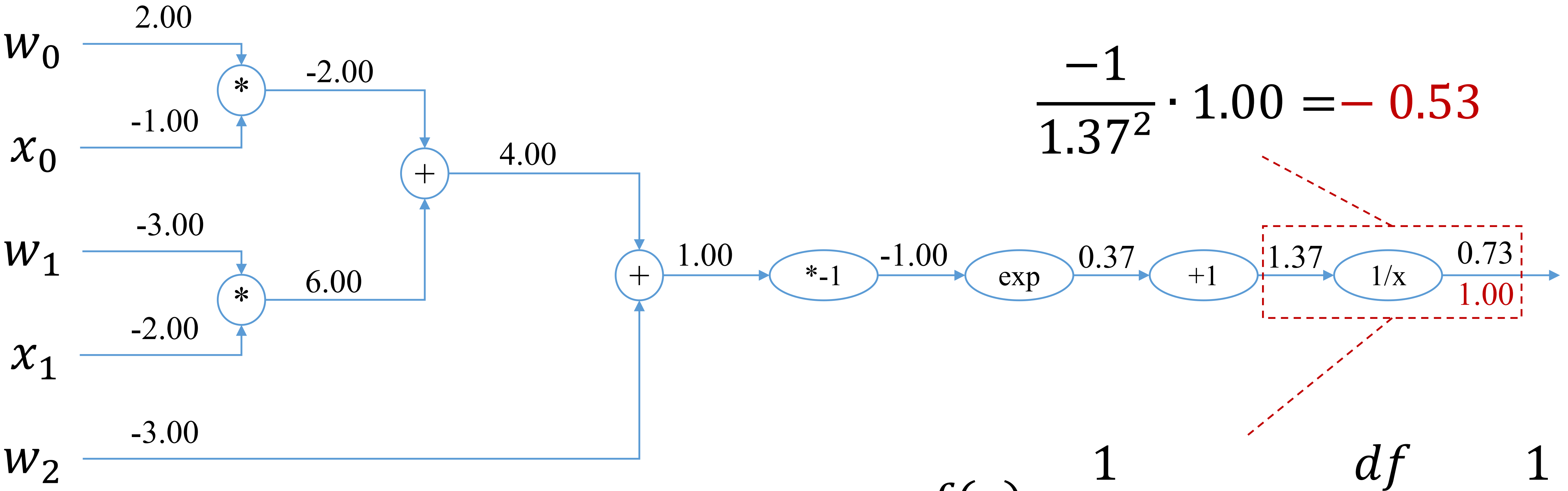


假设预测值与真实值误差为1

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

误差反向传播例子

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

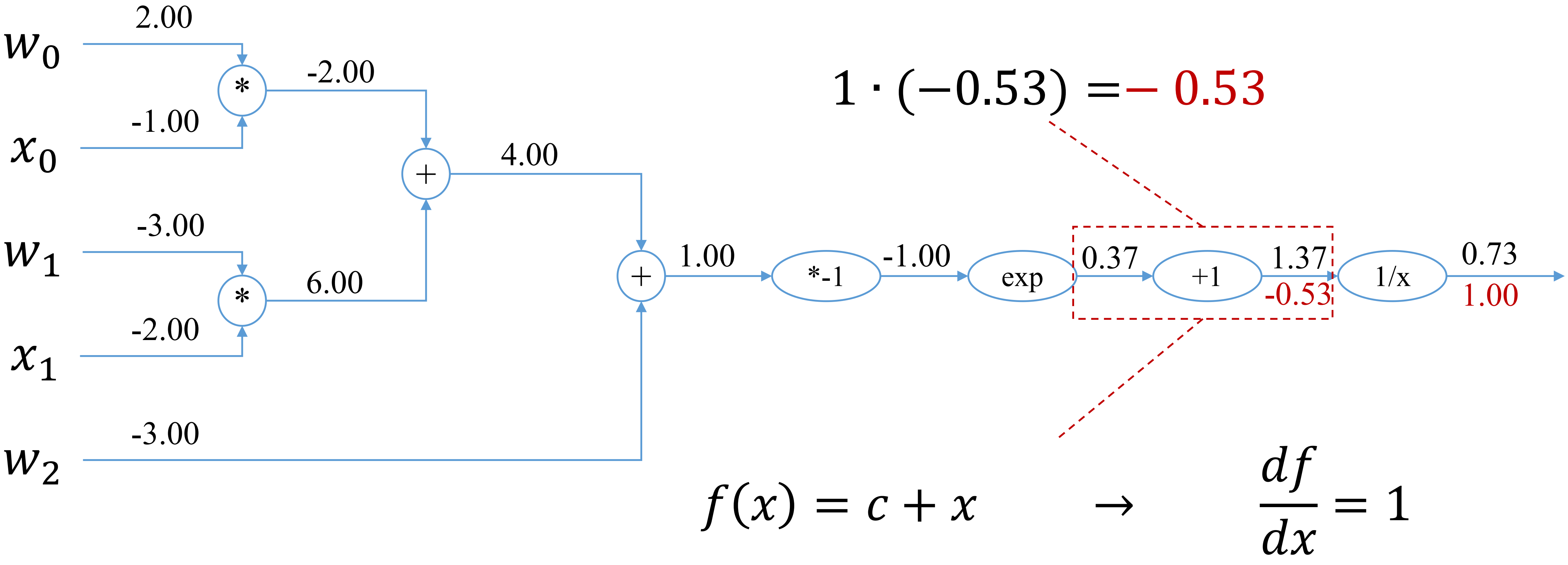


$$\frac{-1}{1.37^2} \cdot 1.00 = -0.53$$

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

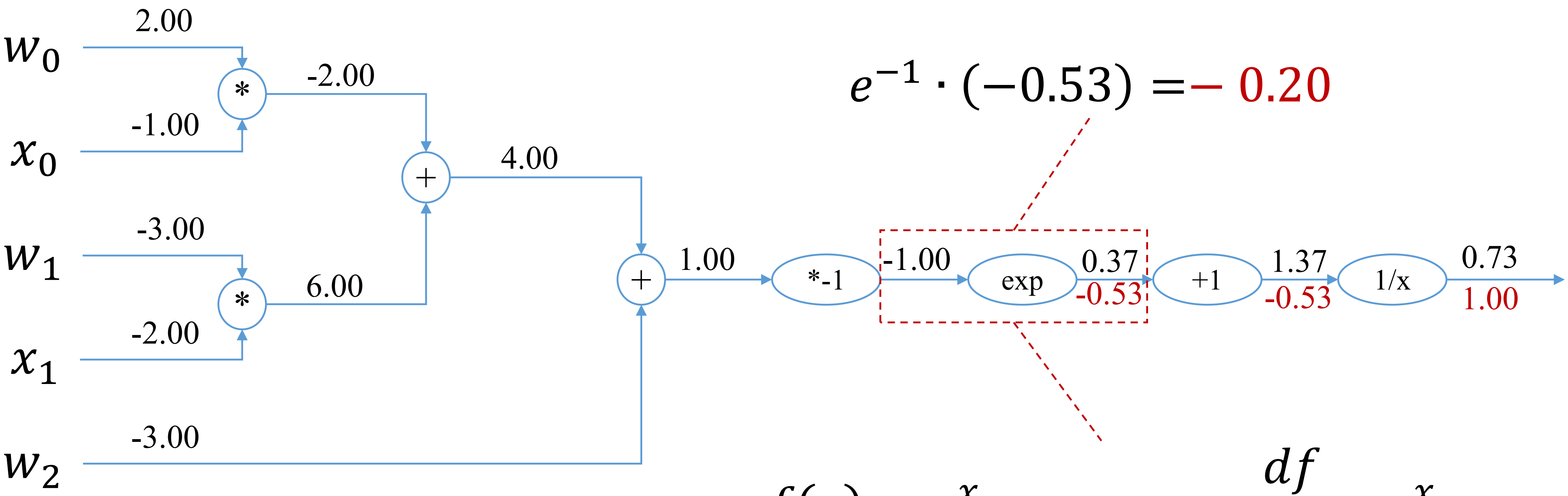
误差反向传播例子

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



误差反向传播例子

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

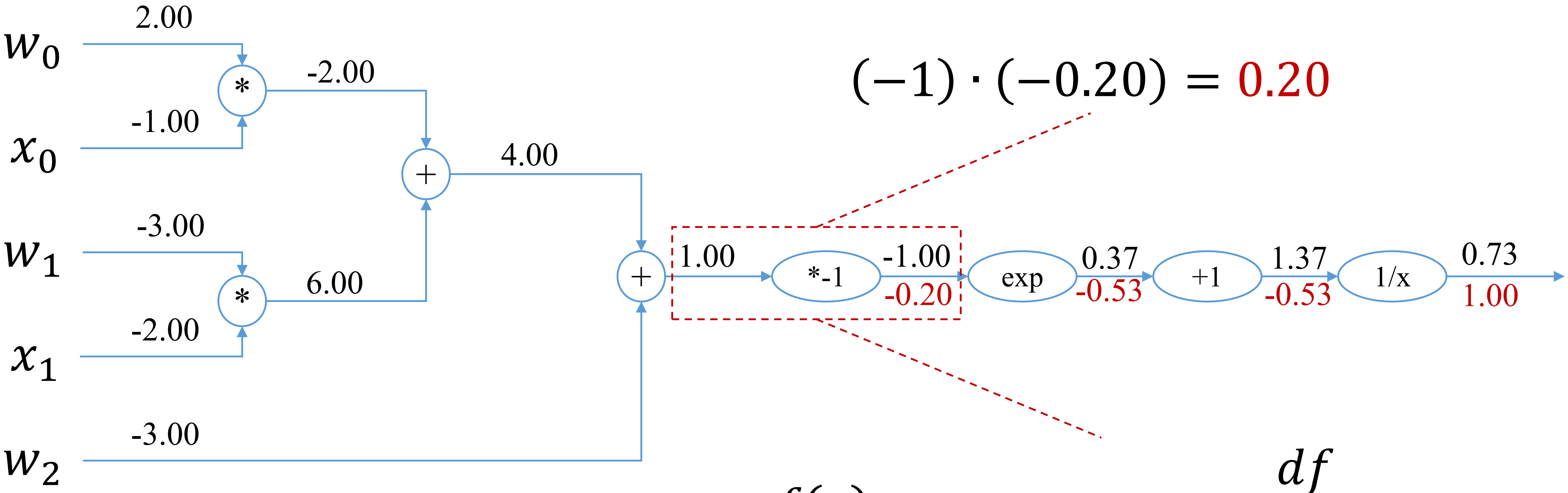


$$e^{-1}\cdot(-0.53)=-0.20$$

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

误差反向传播例子

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

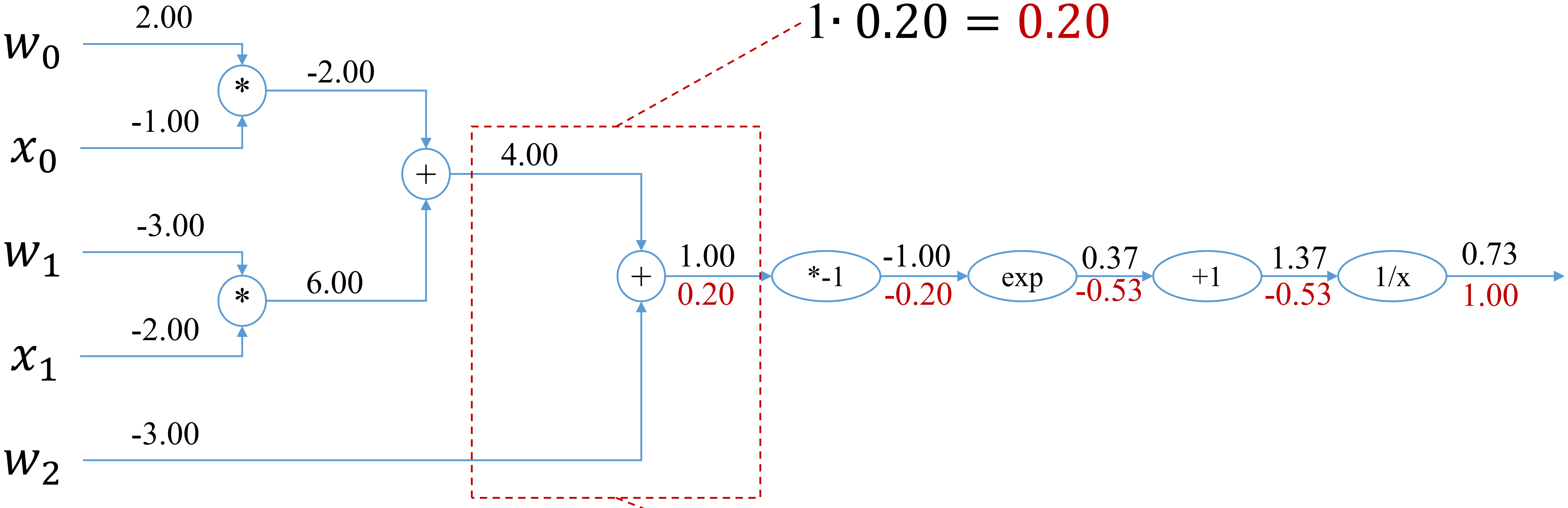


$$(-1)\cdot(-0.20)=0.20$$

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

误差反向传播例子

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

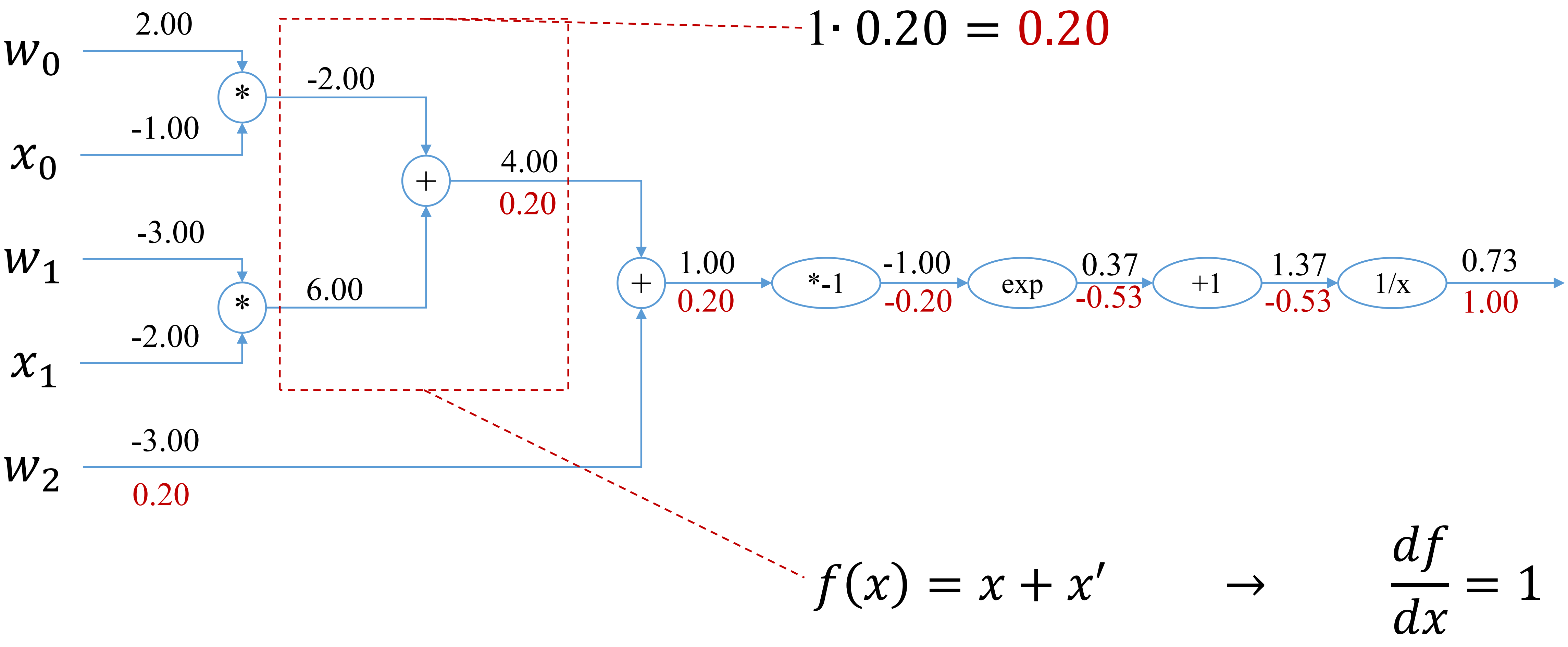


$1\cdot 0.20 = 0.20$

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

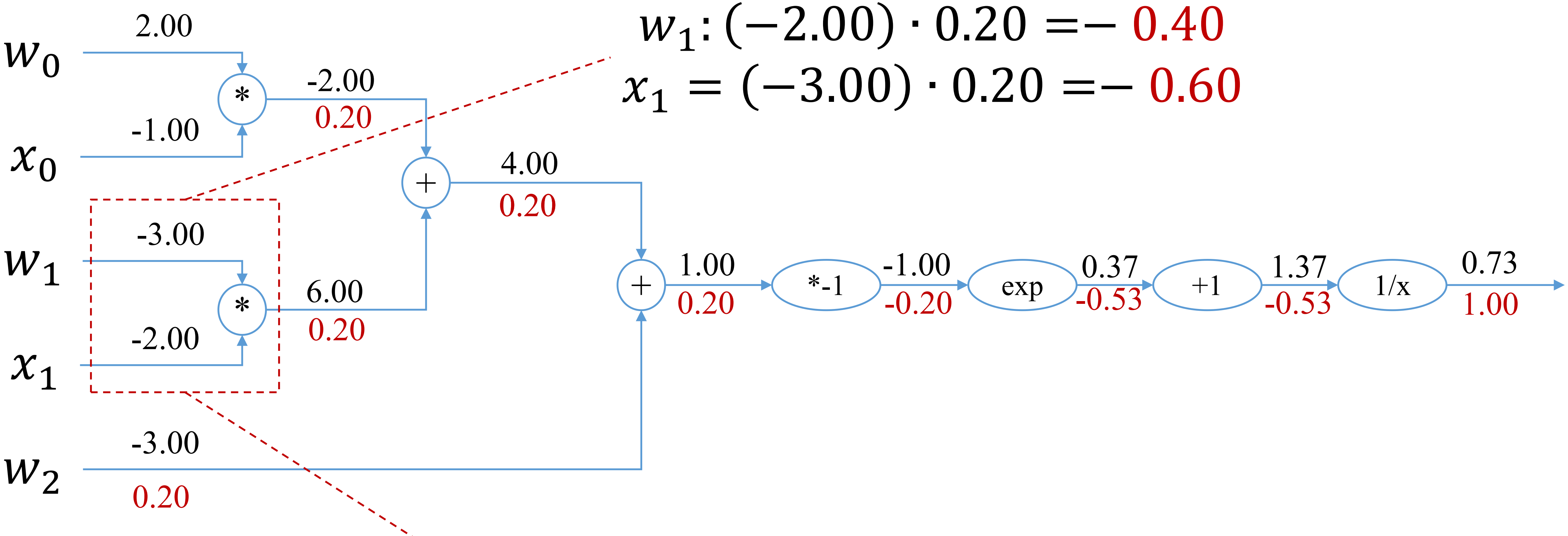
误差反向传播例子

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



误差反向传播例子

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



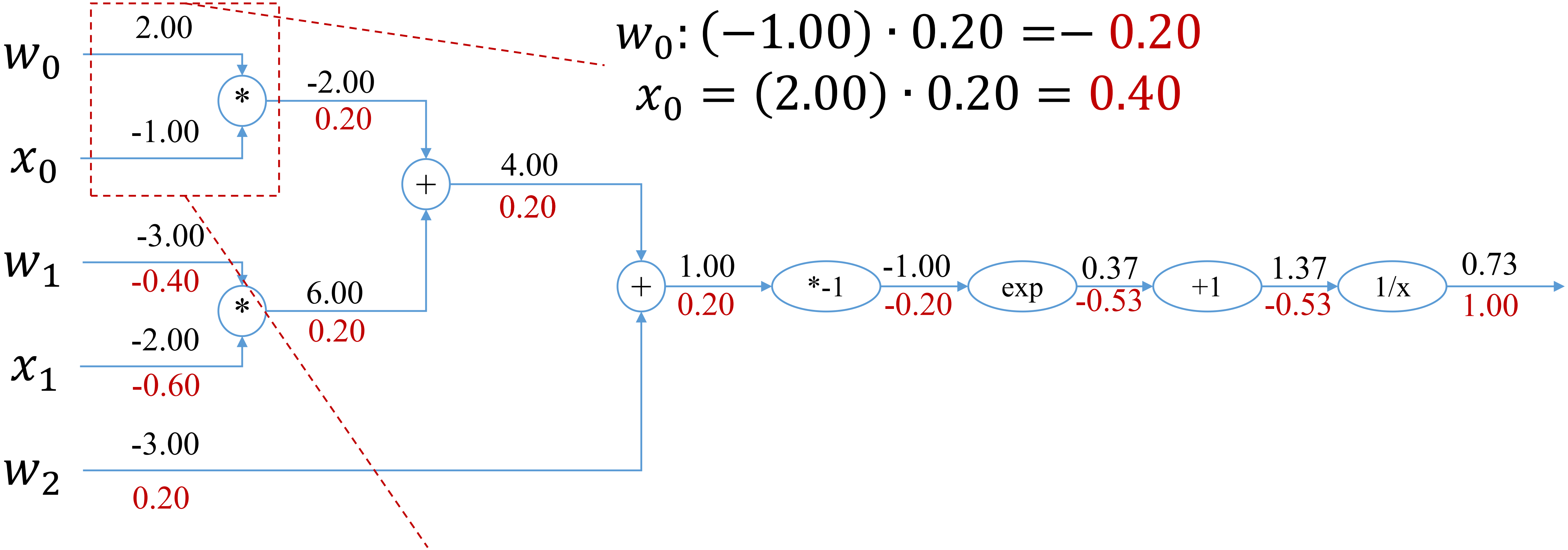
$w_1: (-2.00)\cdot 0.20 =- 0.40$

$x_1 = (-3.00)\cdot 0.20 =- 0.60$

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

误差反向传播例子

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



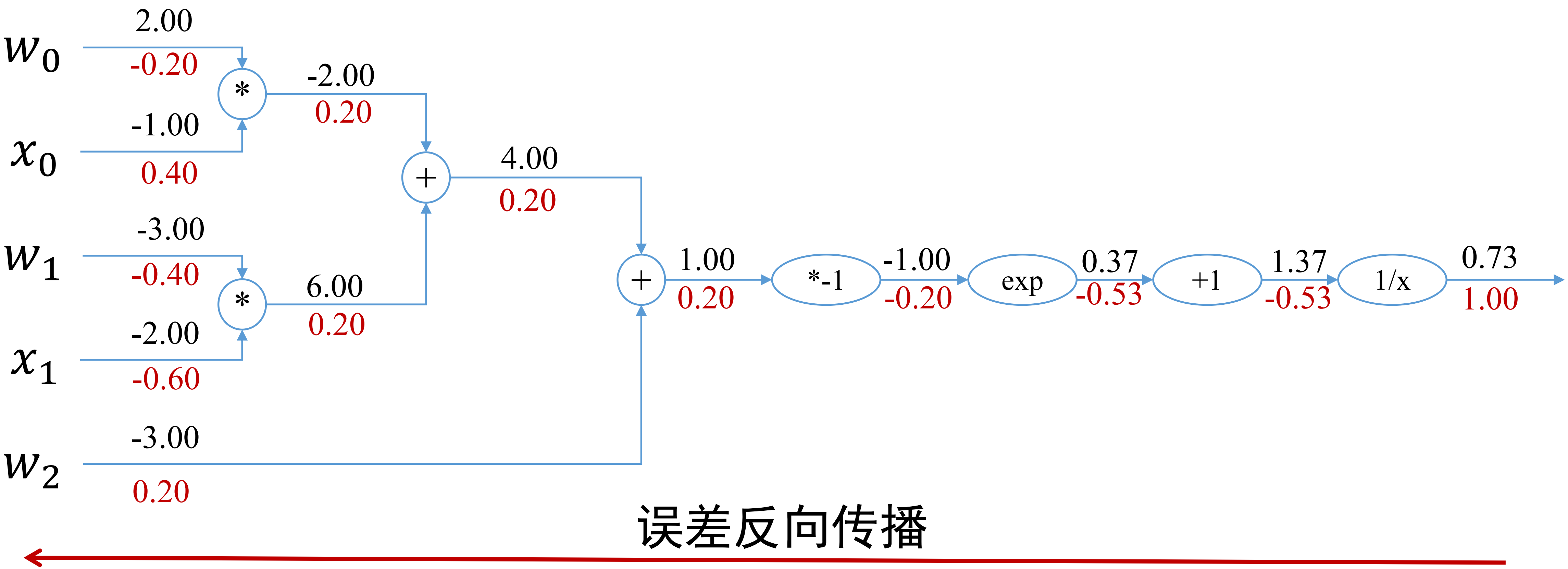
$$w_0 : (-1.00)\cdot 0.20 =- 0.20$$

$$x_0 = (2.00)\cdot 0.20 = 0.40$$

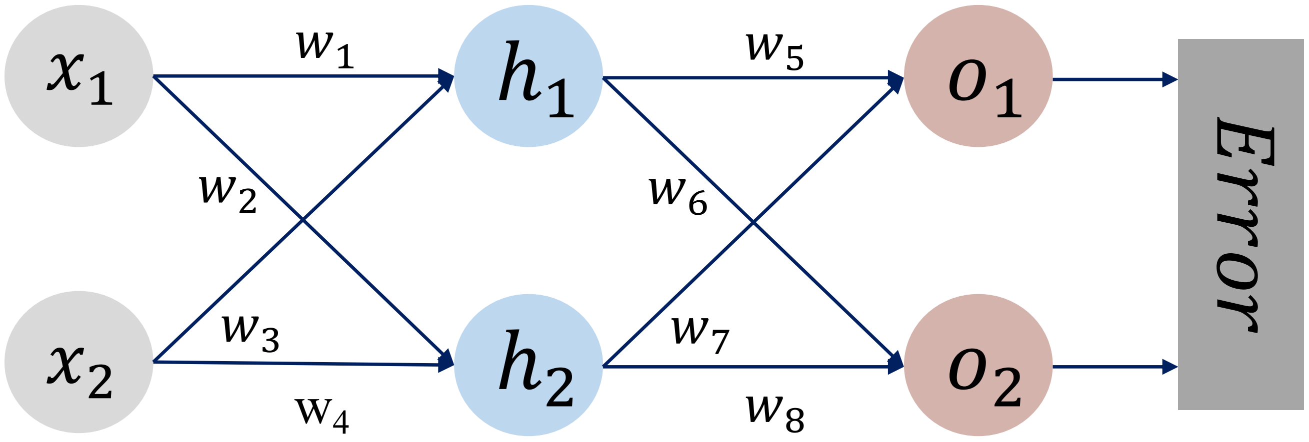
$$f(x) = x\cdot x' \quad \rightarrow \quad \frac{df}{dx} = x'$$

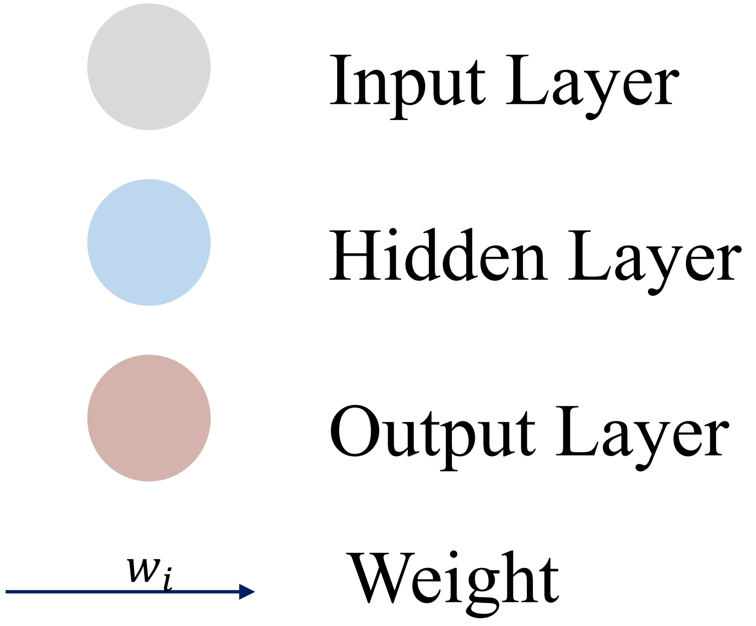
误差反向传播例子

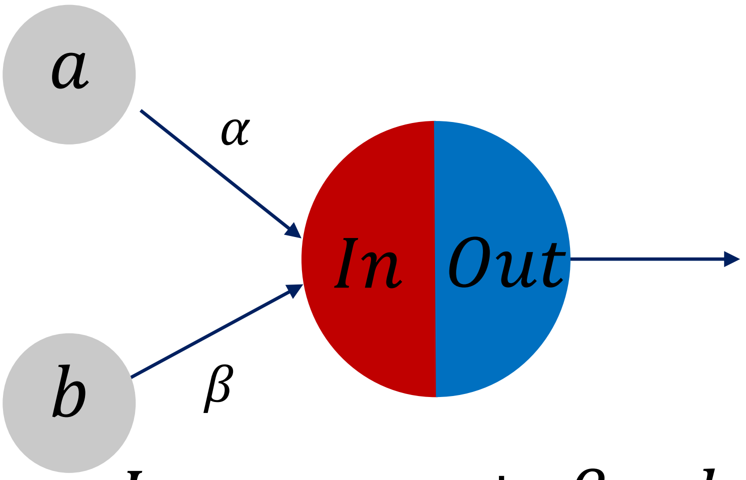
$$f(w,x)=\frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



误差反向传播：前馈神经网络





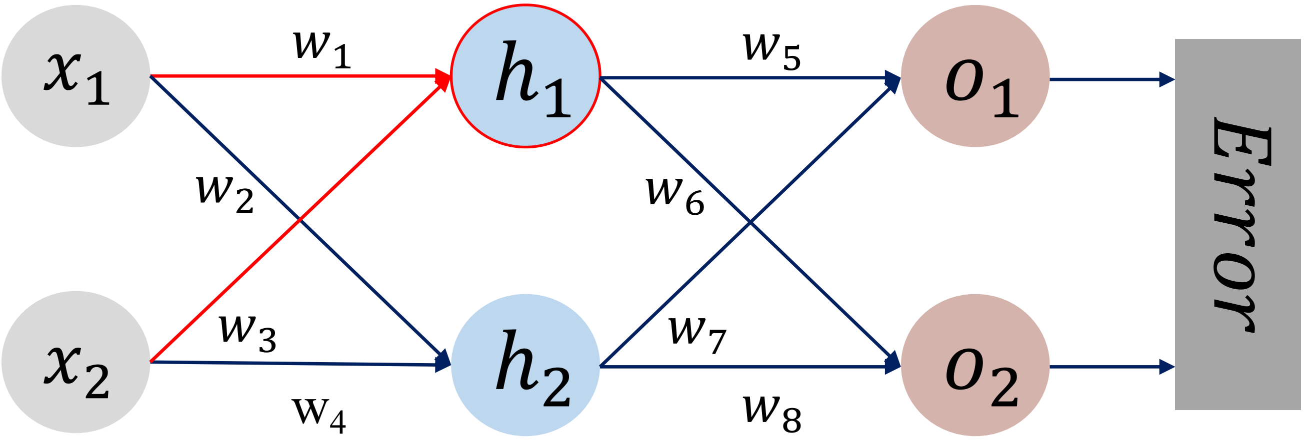


$$In = \alpha * a + \beta * b$$

$$Out = Sigmoid(In)$$

误差反向传播：前馈神经网络

- **前向传播**

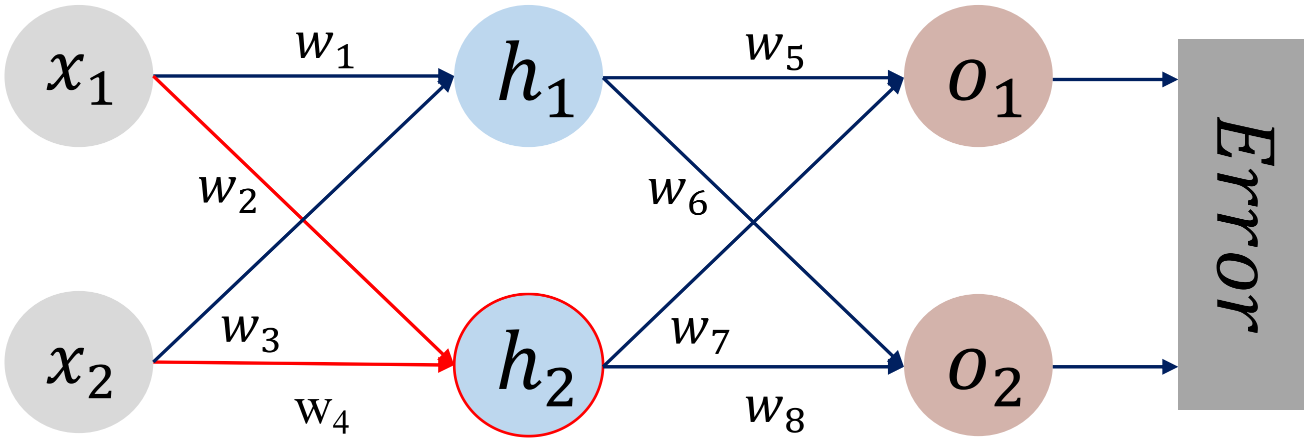


$$In_{h_1} = w_1 * x_1 + w_3 * x_2$$

$$h_1 = Out_{h_1} = Sigmoid(In_{h_1})$$

误差反向传播：前馈神经网络

- **前向传播**

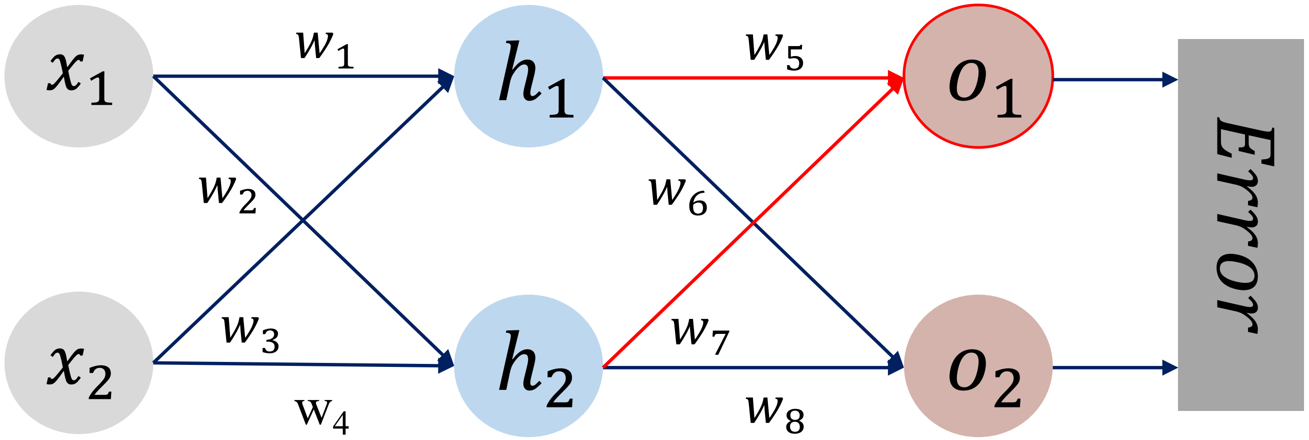


$$In_{h_2} = \ w_2 * x_1 + w_4 * x_2$$

$$h_2 = Out_{h_2} = Sigmoid(In_{h_2})$$

误差反向传播：前馈神经网络

- **前向传播**

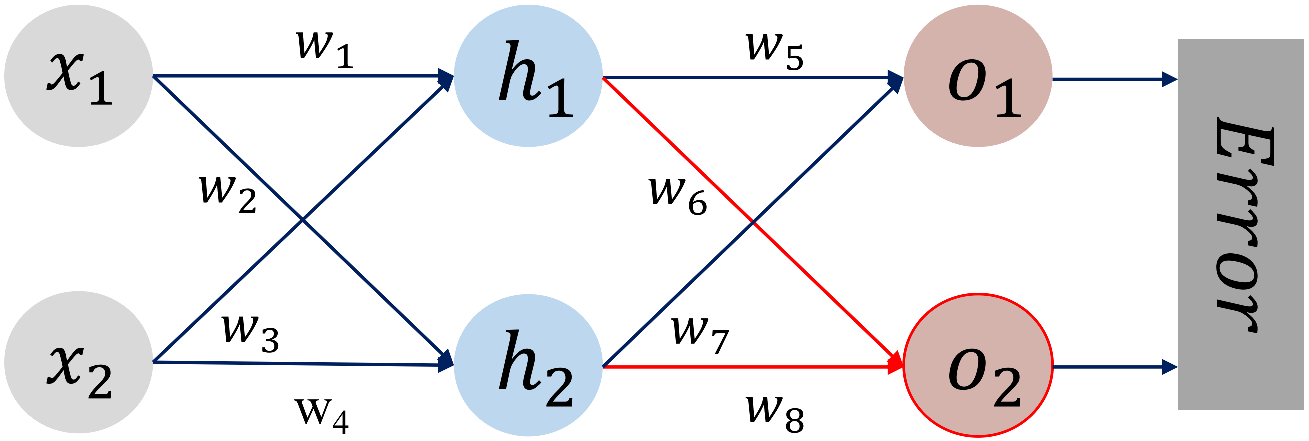


$$In_{o_1} = w_5 * h_1 + w_7 * h_2$$

$$o_1 = Out_{o_1} = Sigmoid(In_{o_1})$$

误差反向传播：前馈神经网络

- **前向传播**

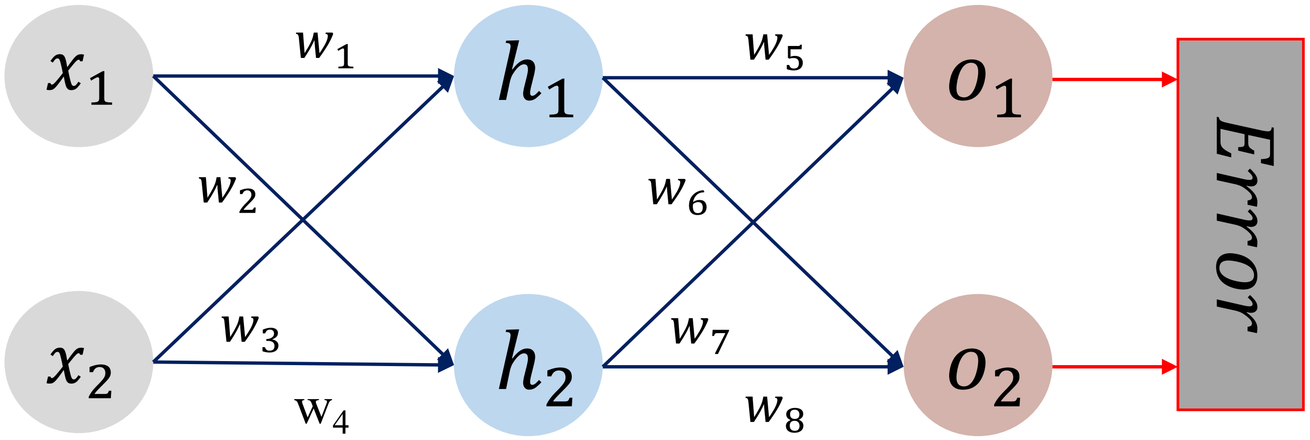


$$In_{o_2} = w_6 * h_1 + w_8 * h_2$$

$$o_2 = Out_{o_2} = Sigmoid(In_{o_2})$$

误差反向传播：前馈神经网络

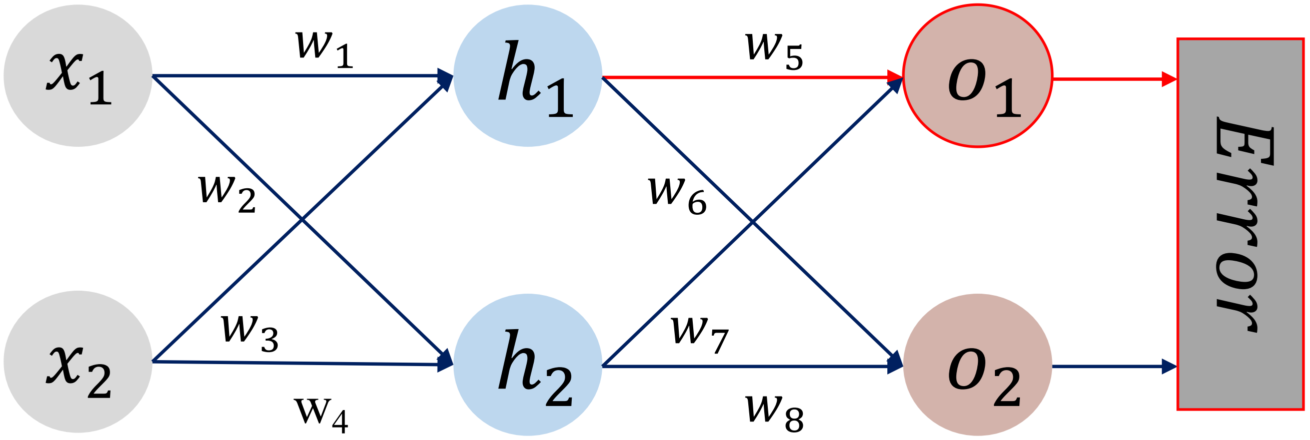
- **前向传播**



$$Error = \frac{1}{2}\sum_{i=1}^{2}(o_i - y_i)^2$$

误差反向传播：前馈神经网络

- **反向传播：梯度计算**
  - 计算$w_5 \sim w_8$的梯度（以w_5为例）



$$\delta_5 = \frac{\partial Error}{\partial w_5} = \frac{\partial Error}{\partial o_1} * \frac{\partial o_1}{\partial In_{o_1}} * \frac{\partial In_{o_1}}{\partial w_5}$$

where,

$$\frac{\partial Error}{\partial o_1} = o_1 - y_1 \quad \leftarrow \quad Error = \frac{1}{2}\sum_{i=1}^{2}(o_i - y_i)^2$$

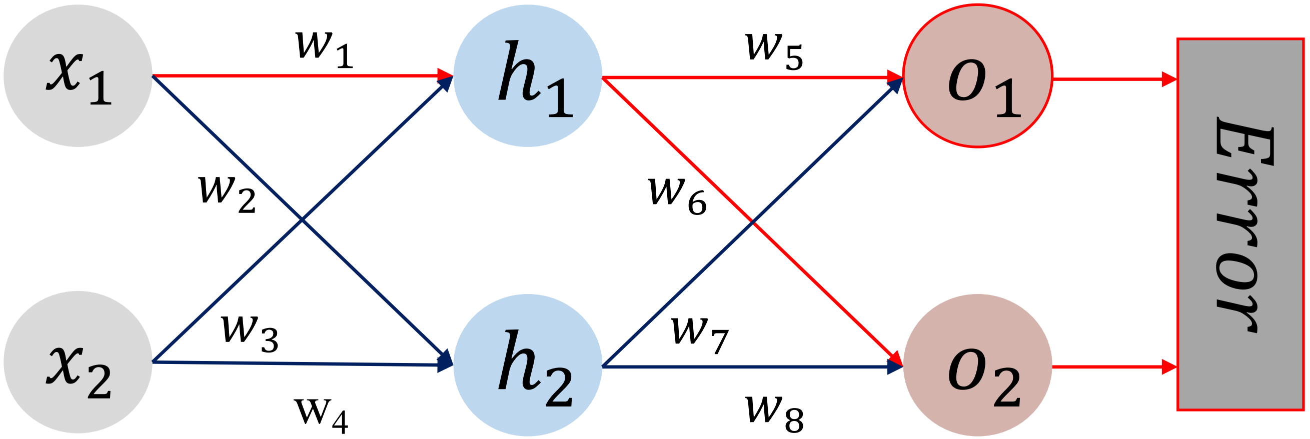
$$\frac{\partial o_1}{\partial In_{o_1}} = o_1 * (1 - o_1) \quad \leftarrow \quad o_1 = Out_{o_1} = Sigmoid(In_{o_1})$$

$$\frac{\partial In_{o_1}}{\partial w_5} = h_1 \quad \leftarrow \quad In_{o_1} = w_5 * h_1 + w_7 * h_2$$

误差反向传播：前馈神经网络

- 反向传播：梯度计算
  - 计算w_1~ w_4的梯度

    ($以w_1为例$)



$$\delta_1 = \frac{\partial Error}{\partial w_1} = \frac{\partial Error}{\partial o_1} * \frac{\partial o_1}{\partial w_1} + \frac{\partial Error}{\partial o_2} * \frac{\partial o_2}{\partial w_1}$$

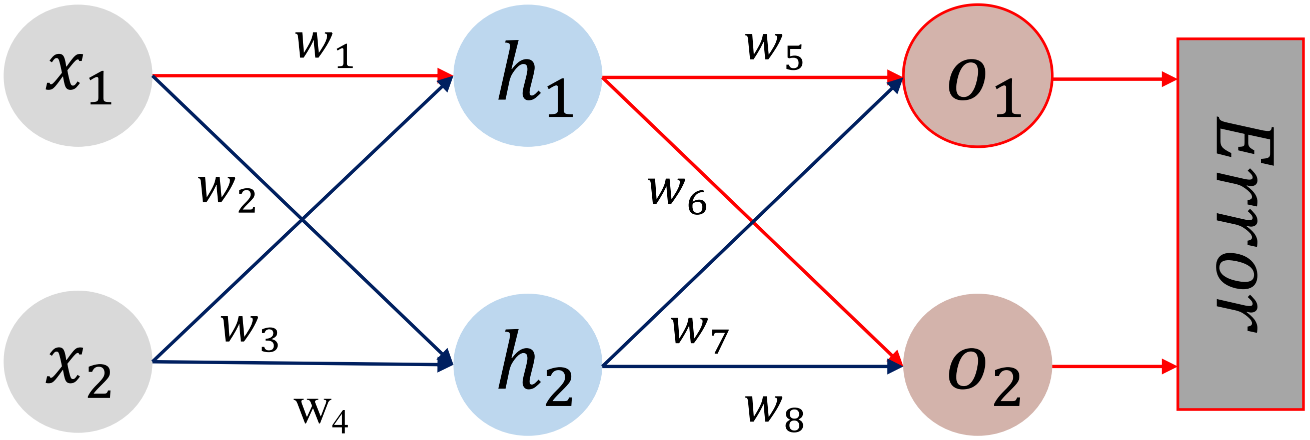
$$= \frac{\partial Error}{\partial o_1} * \frac{\partial o_1}{\partial In_{o_1}} * \frac{\partial In_{o_1}}{\partial h_1} * \frac{\partial h_1}{\partial In_{h_1}} * \frac{\partial In_{h_1}}{\partial w_1} + \frac{\partial Error}{\partial o_2} * \frac{\partial o_2}{\partial In_{o_2}} * \frac{\partial In_{o_2}}{\partial h_1} * \frac{\partial h_1}{\partial In_{h_1}} * \frac{\partial In_{h_1}}{\partial w_1}$$

$$= (\frac{\partial Error}{\partial o_1} * \frac{\partial o_1}{\partial In_{o_1}} * \frac{\partial In_{o_1}}{\partial h_1} + \frac{\partial Error}{\partial o_2} * \frac{\partial o_2}{\partial In_{o_2}} * \frac{\partial In_{o_2}}{\partial h_1}) * \frac{\partial h_1}{\partial In_{h_1}} * \frac{\partial In_{h_1}}{\partial w_1}$$

$$= (\frac{\partial Error}{\partial o_1} * \frac{\partial o_1}{\partial In_{o_1}} * w_5 + \frac{\partial Error}{\partial o_2} * \frac{\partial o_2}{\partial In_{o_2}} * w_6) * \frac{\partial h_1}{\partial In_{h_1}} * \frac{\partial In_{h_1}}{\partial w_1}$$

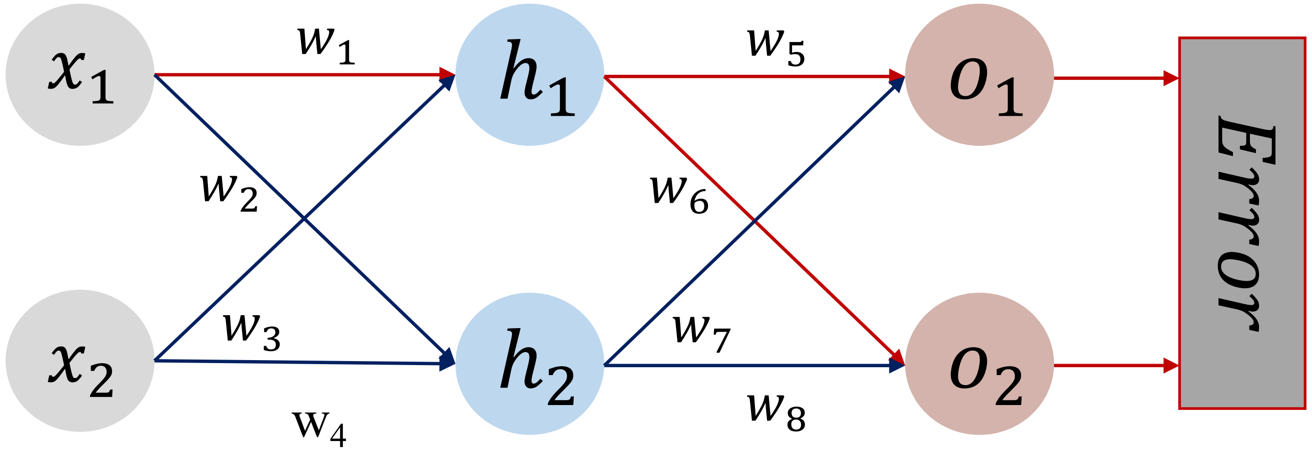
误差反向传播：前馈神经网络

- **反向传播：参数更新**
  - 其中� 被称为学习率



误差反向传播：前馈神经网络

反向传播：参数更新

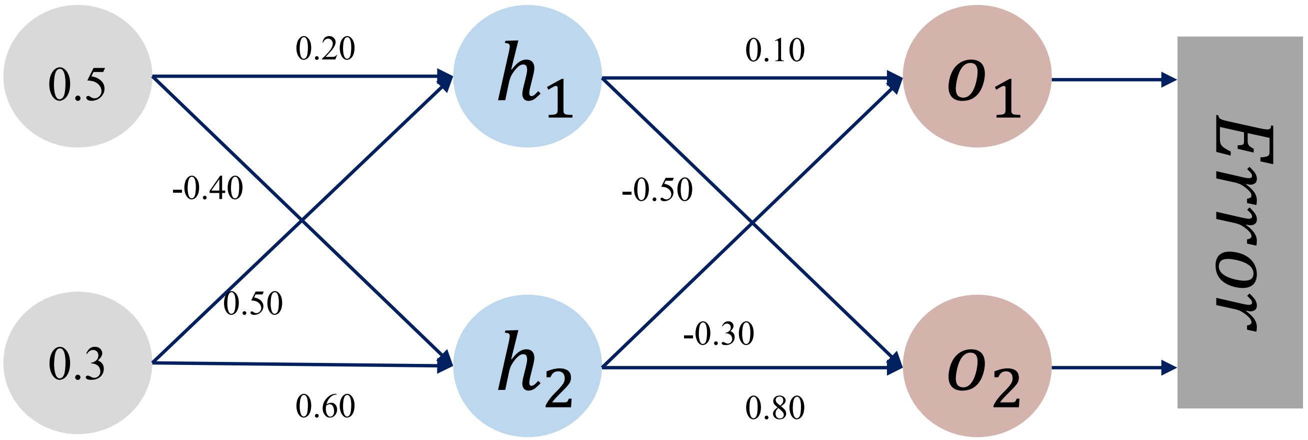


2. 更新参数，其中η 被称为学习率

$$w_i' = \underbrace{w_i}_{\text{原有参数}} - \underbrace{\eta}_{\text{步长}} * \underbrace{\delta_i}_{\text{传递误差}}$$

误差反向传播：前馈神经网络

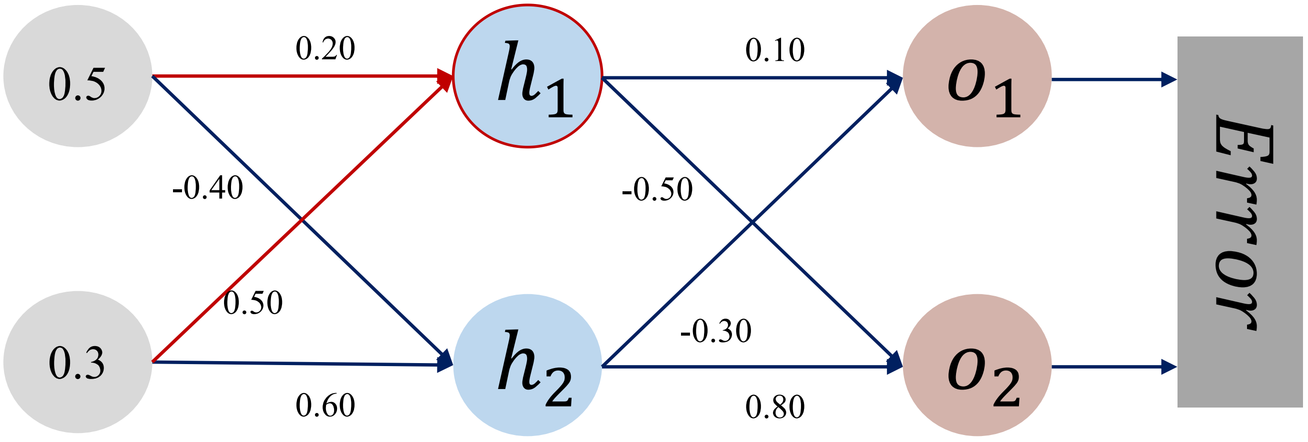
参数初始化



其中，$Error = \frac{1}{2}(o_1 - 0.23)^2 + \frac{1}{2}(o_2 - (-0.07))^2$，这里0.23和-0.07是对输入样本数据(0.5, 0.3)的标注信息

误差反向传播：前馈神经网络

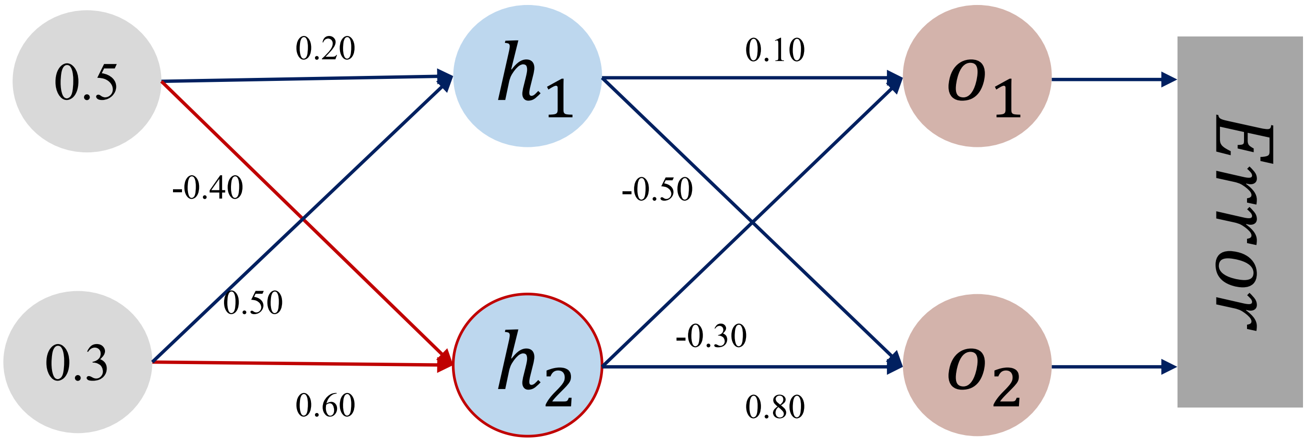
前向传播



$$h_1 = sigmoid(0.20 * 0.50 + 0.50 * 0.30) = 0.56$$

误差反向传播：前馈神经网络

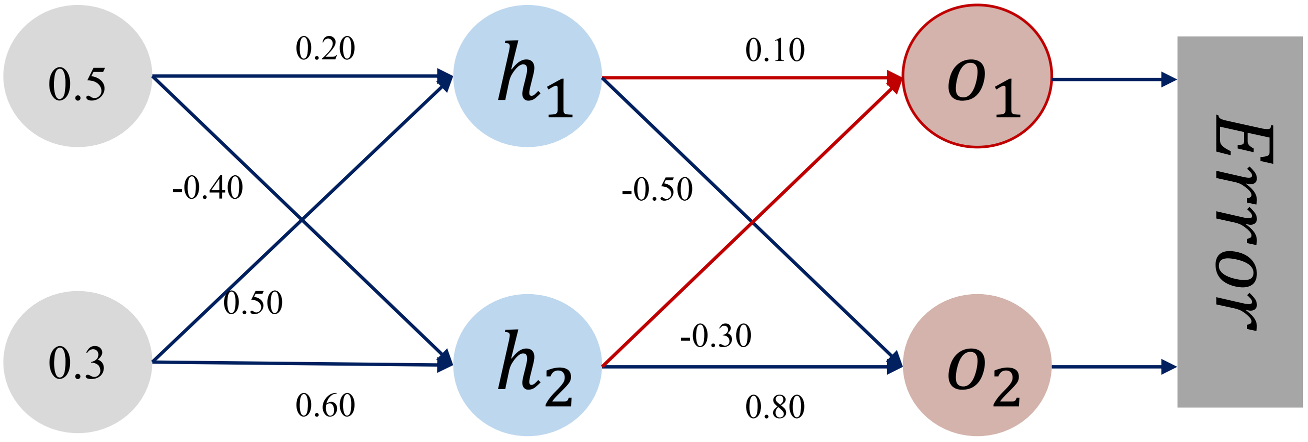
前向传播



$$h_2 = sigmoid(-0.40 * 0.50 + 0.60 * 0.30) = 0.50$$

误差反向传播：前馈神经网络

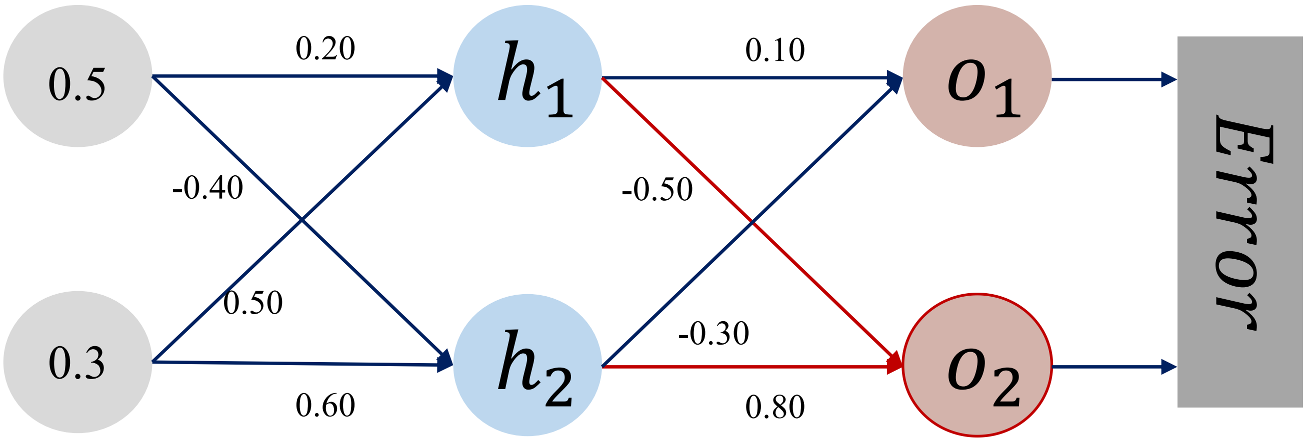
前向传播



$$o_1 = sigmoid(0.10 * 0.56 + (-0.30 * 0.50) = 0.48$$

误差反向传播：前馈神经网络

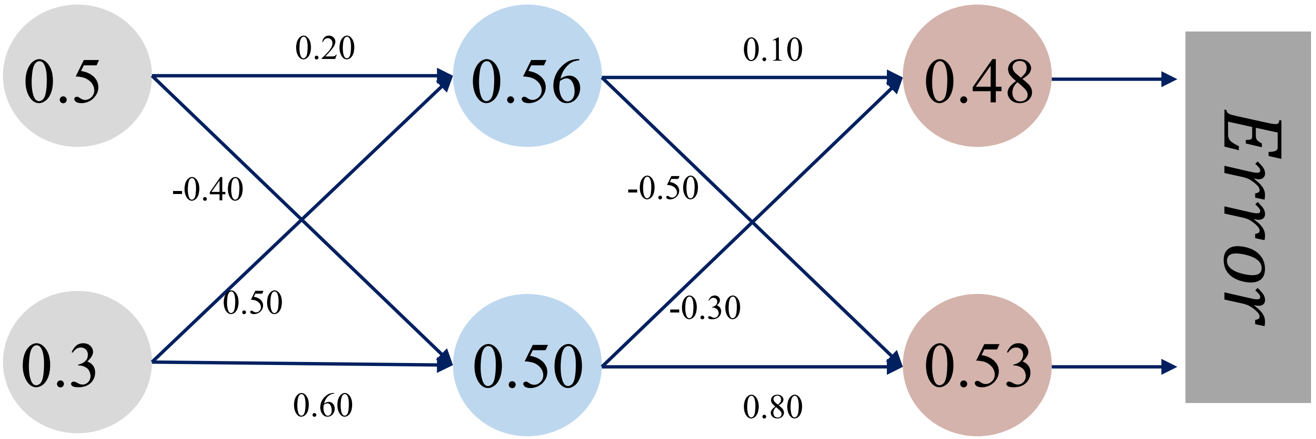
前向传播



$$o_2 = sigmoid(-0.50 * 0.56 + 0.80 * 0.50) = 0.53$$

误差反向传播：前馈神经网络

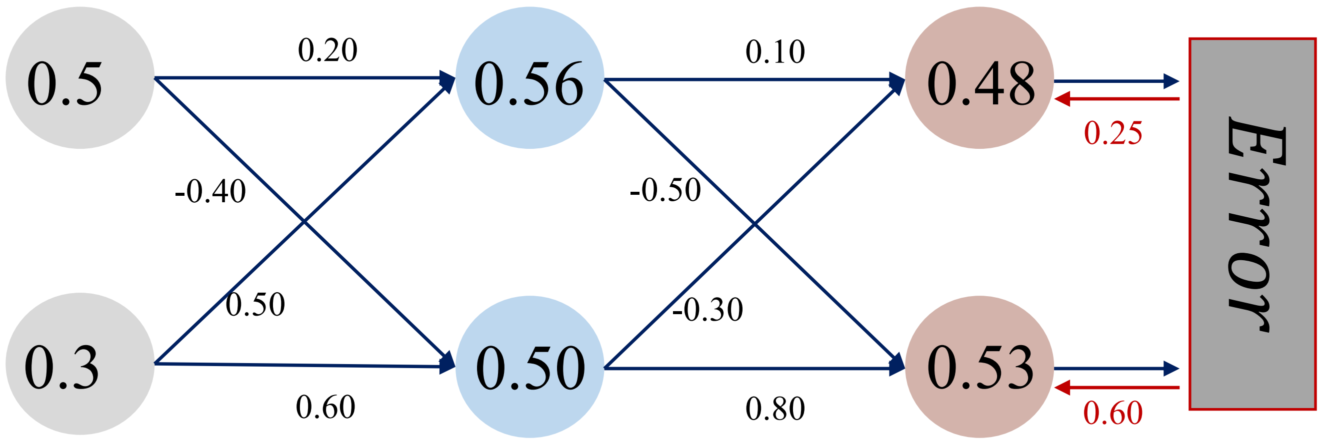
反向传播（假设学习速率Learning Rate $\eta = 1$）



$$Error = \frac{1}{2}(0.48 - 0.23)^2 + \frac{1}{2}(0.53 - (-0.07))^2 = 0.21$$

误差反向传播：前馈神经网络

梯度计算

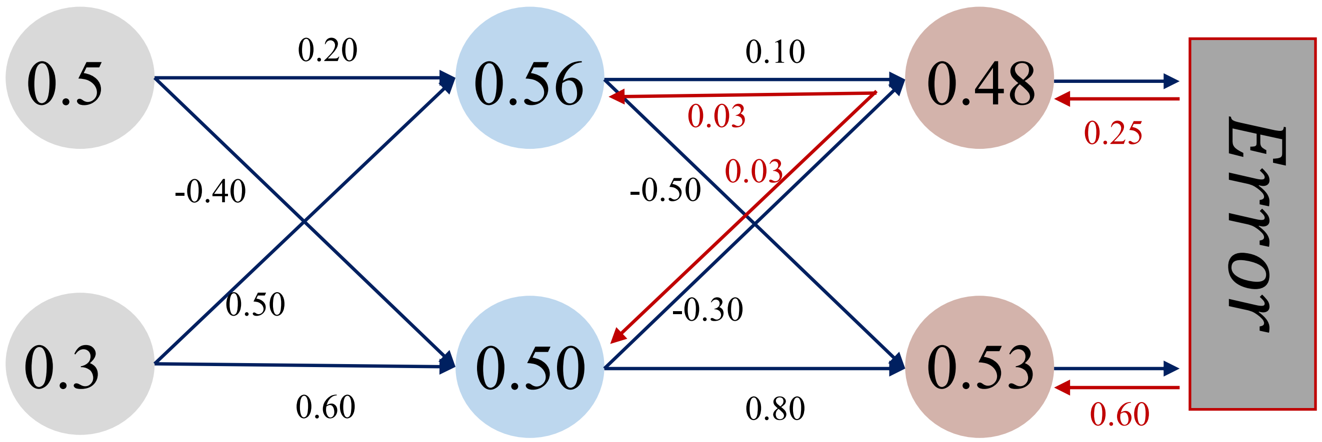


$$\delta_{o^1} = \frac{\partial Error}{\partial o_1} = o_1 - 0.23 = 0.25$$

$$\delta_{o^2} = \frac{\partial Error}{\partial o_2} = o_2 - (-0.07) = 0.60$$

误差反向传播：前馈神经网络

梯度计算

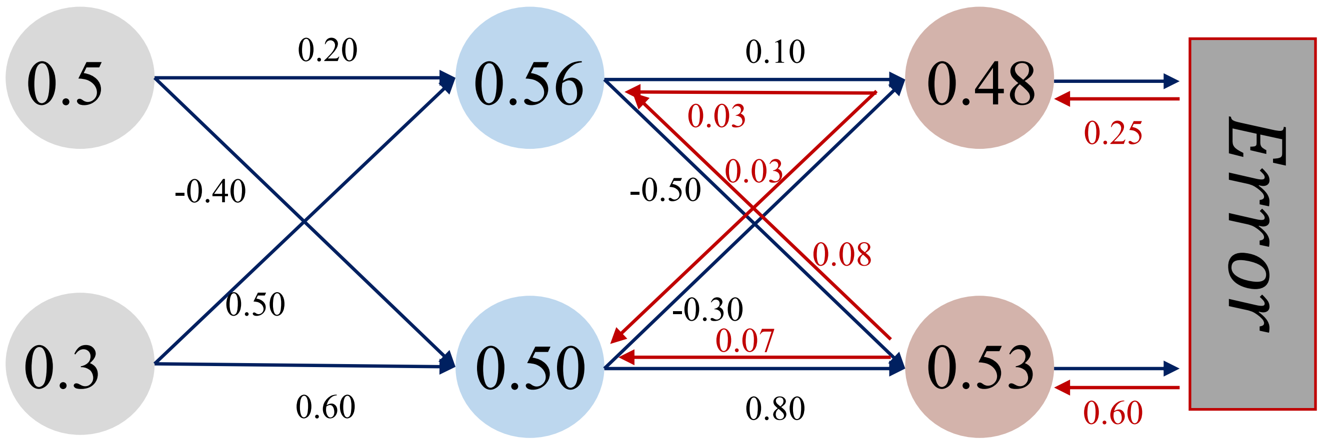


$$\delta_5 = \frac{\partial Error}{\partial w_5} = \frac{\partial Error}{\partial o_1} * \frac{\partial o_1}{\partial In_{o_1}} * \frac{\partial In_{o_1}}{\partial w_5} = 0.25 * 0.48 * (1 - 0.48) * 0.56 = 0.03$$

$$\delta_7 = \frac{\partial Error}{\partial w_7} = \frac{\partial Error}{\partial o_1} * \frac{\partial o_1}{\partial In_{o_1}} * \frac{\partial In_{o_1}}{\partial w_7} = 0.25 * 0.48 * (1 - 0.48) * 0.50 = 0.03$$

误差反向传播：前馈神经网络

梯度计算

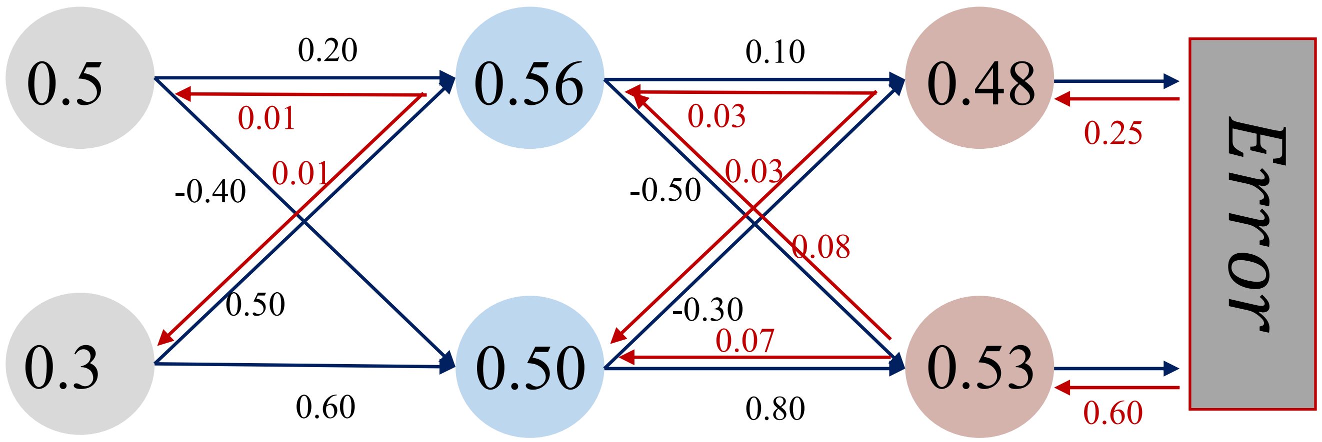


$$\delta_6 = \frac{\partial Error}{\partial w_6} = \frac{\partial Error}{\partial o_2} * \frac{\partial o_2}{\partial In_{o_2}} * \frac{\partial In_{o_2}}{\partial w_6} = 0.60 * 0.53 * (1 - 0.53) * 0.56 = 0.08$$

$$\delta_8 = \frac{\partial Error}{\partial w_8} = \frac{\partial Error}{\partial o_2} * \frac{\partial o_2}{\partial In_{o_2}} * \frac{\partial In_{o_2}}{\partial w_8} = 0.60 * 0.53 * (1 - 0.53) * 0.50 = 0.07$$

误差反向传播：前馈神经网络

梯度计算(本页计算有误，请同学根据正确的公式计算结果)

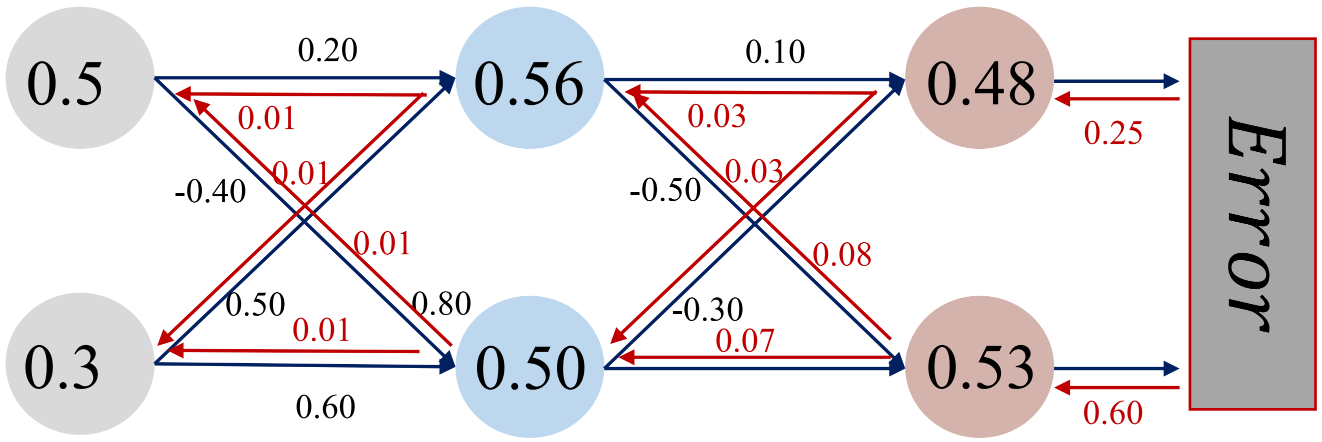


$$\delta_1 = \frac{\partial Error}{\partial w_1} = (\delta_5 + \delta_6) * \frac{\partial h_1}{\partial In_{h_1}} * \frac{\partial In_{h_1}}{\partial w_1} = (0.03 + 0.08) * 0.56 * (1 - 0.56) * 0.50$$
$$= 0.01$$

$$\delta_3 = \frac{\partial Error}{\partial w_3} = (\delta_5 + \delta_6) * \frac{\partial h_1}{\partial In_{h_1}} * \frac{\partial In_{h_1}}{\partial w_3} = (0.03 + 0.08) * 0.56 * (1 - 0.56) * 0.30 = 0.01$$

误差反向传播：前馈神经网络

梯度计算(本页计算有误，请同学根据正确的公式计算结果)



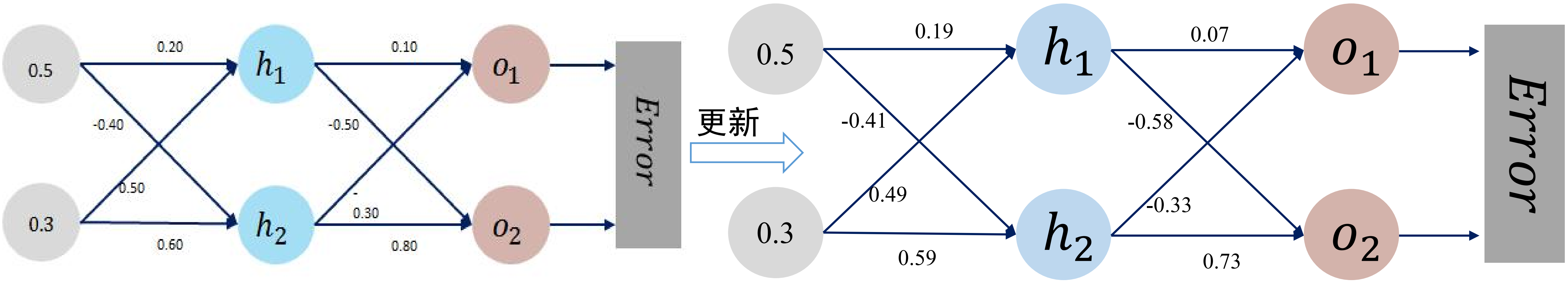
$$\delta_2 = \frac{\partial Error}{\partial w_2} = (\delta_7 + \delta_8) * \frac{\partial h_2}{\partial In_{h_2}} * \frac{\partial In_{h_2}}{\partial w_2} = (0.03 + 0.07) * 0.50 * (1 - 0.50) * 0.5 = 0.01$$

$$\delta_4 = \frac{\partial Error}{\partial w_4} = (\delta_7 + \delta_8) * \frac{\partial h_2}{\partial In_{h_2}} * \frac{\partial In_{h_2}}{\partial w_4} = (0.03 + 0.07) * 0.50 * (1 - 0.50) * 0.3 = 0.01$$

误差反向传播：前馈神经网络

参数更新

$$w_i' = \underbrace{w_i}_{\text{原有参数}} - \underbrace{\eta}_{\text{步长}} * \underbrace{\delta_i}_{\text{传递误差}}$$



参数优化：误差反向传播 (error back propagation, BP)

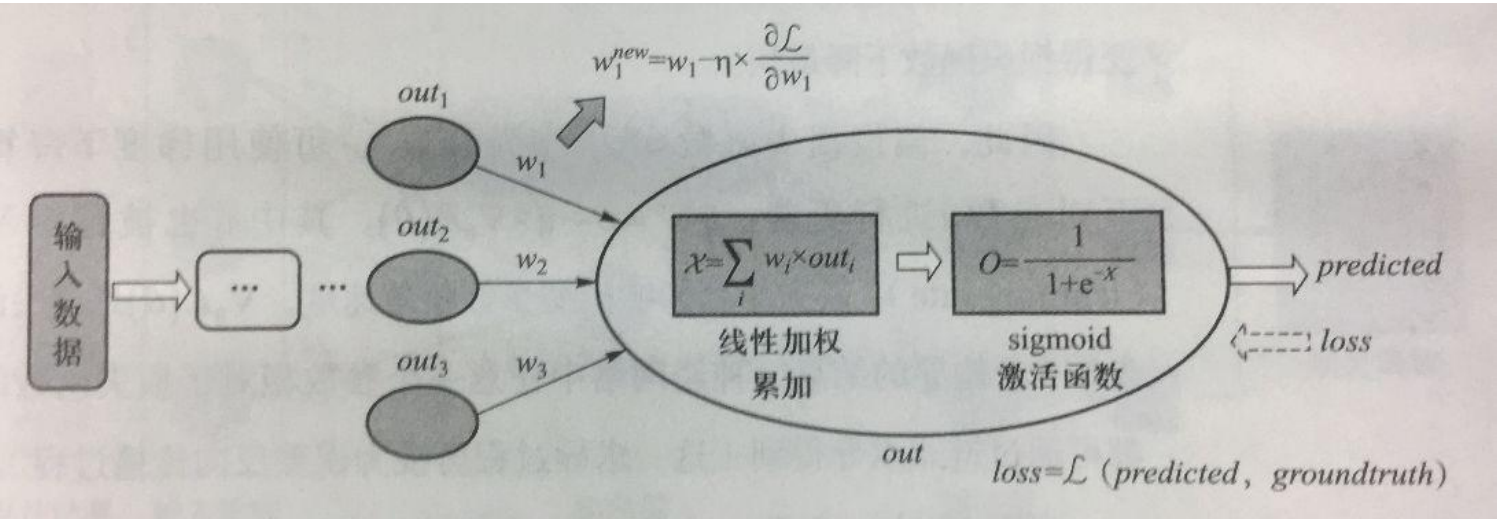


图6.7 链式求导与模型参数更新示意图

为了使损失函数$\mathcal{L}$取值减少（从而保证模型预测结果与实际结果之间的差距越来越小），需要求取损失函数$\mathcal{L}$相对于w_1的偏导，然后按照损失函数梯度的反方向选取一个微小的增量，来调整w_1的取值，就能够保证损失函数取值减少。即将w_1变为$w_1-\eta\frac{d\mathcal{L}}{dw_1}$后，能使得损失误差减少。这里$\frac{d\mathcal{L}}{dw_1}$为损失函数$\mathcal{L}$对$w_1$的偏导。于是，需要求取损失函数$\mathcal{L}$对变量参数$w_1$的偏导。

参数优化：误差反向传播 (error back propagation, BP)

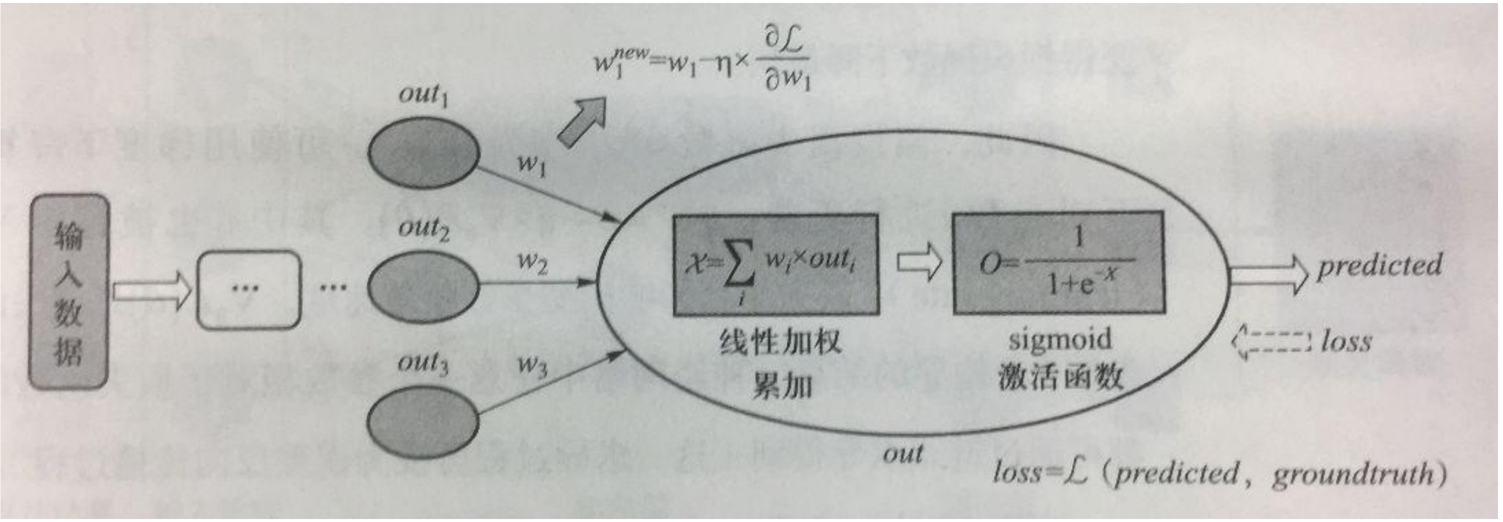


图6.7 链式求导与模型参数更新示意图

- 由于w_1与加权累加函数$\mathcal{X}$和sigmoid函数均有关，因此$\frac{d\mathcal{L}}{dw_1}=\frac{d\mathcal{L}}{dO}\frac{dO}{d\mathcal{X}}\frac{d\mathcal{X}}{dw_1}$。在这个链式求导中：$\frac{d\mathcal{L}}{dO}$与损失函数的定义有关；$\frac{dO}{d\mathcal{X}}$是对sigmoid函数求导，结果为$\frac{1}{1+e^{-\mathcal{X}}}\times(1-\frac{1}{1+e^{-\mathcal{X}}})$；$\frac{d\mathcal{X}}{dw_1}$是加权累加函数$w_1\times out_1+w_2\times out_2+w_3\times out_3$对$w_1$求导，结果为$out_1$。
- 链式求导实现了损失函数对某个自变量求偏导，好比将损失误差从输出端向输入端逐层传播，通过这个传播过程来更新该自变量取值。梯度下降法告诉我们，只要沿着损失函数梯度的反方向来更新参数，就可使得损失函数下降最快。

参数优化：误差反向传播 (error back propagation, BP)

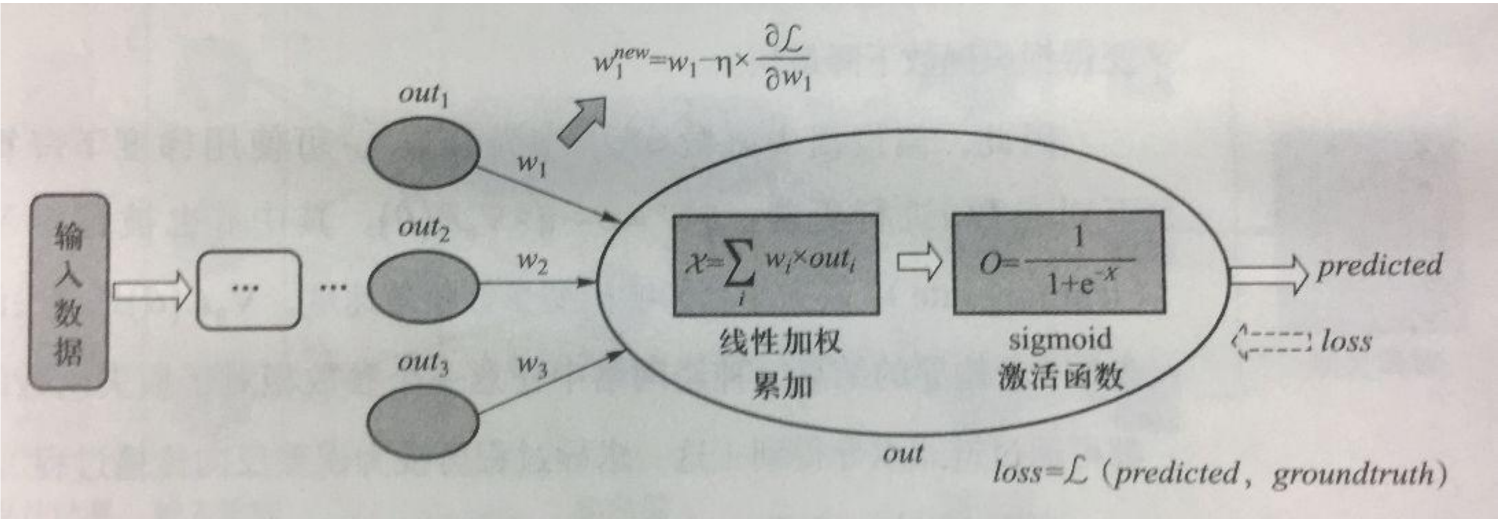


图6.7 链式求导与模型参数更新示意图

- 参数w_1在下一轮迭代中的取值被调整为：$w_1^{new}=w_1-\eta\times\frac{d\mathcal{L}}{dw_1}=w_1-\eta\times\frac{d\mathcal{L}}{dO}\frac{dO}{d\mathcal{X}}\frac{d\mathcal{X}}{dw_1}=w_1-\eta\times\frac{d\mathcal{L}}{dO}\times\frac{1}{1+e^{-\mathcal{X}}}\times(1-\frac{1}{1+e^{-\mathcal{X}}})\times out_1$。
- 按照同样的方法，可调整w_2、w_3以及其它图中未显示参数的取值。经过这样的调整，在模型参数新的取值作用下，损失函数$\mathcal{L}(\theta)$会以最快下降方式逐渐减少，直至减少到最小值（即全局最小值）。

参数优化：误差反向传播 (error back propagation, BP)

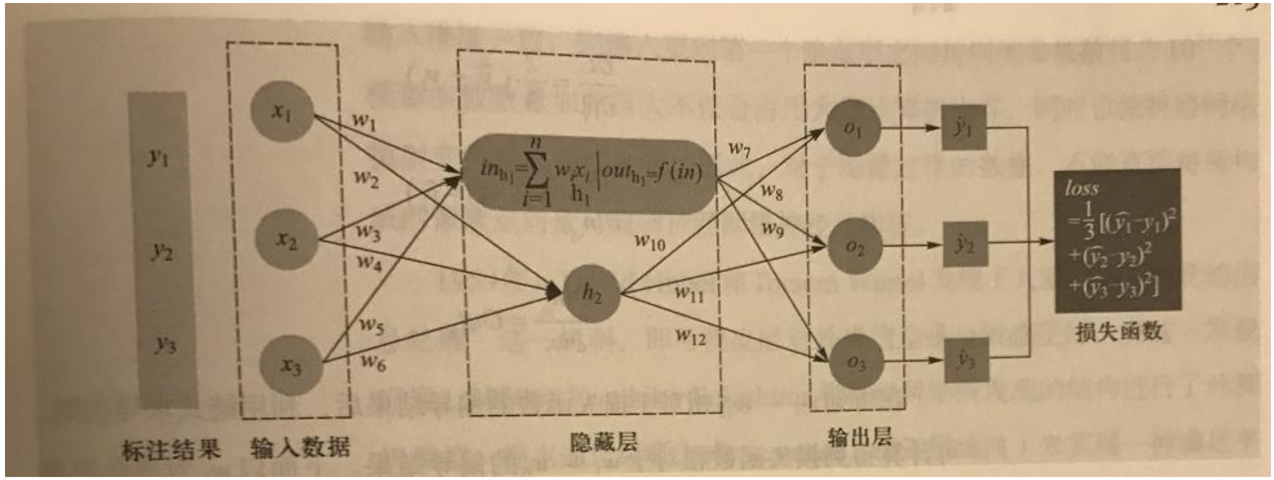


图6.8 包含一个隐藏层的多层感知机

- 通过一个三类分类的具体例子来介绍神经网络中参数更新过程。给定一个包含输入层、一层隐藏层和输出层的多层感知机，其中隐藏层由两个神经元构成。
- 网络使用Sigmoid函数作为神经元的激活函数，使用均方损失函数来计算网络输出值与实际值之间的误差。

参数优化：误差反向传播 (error back propagation, BP)

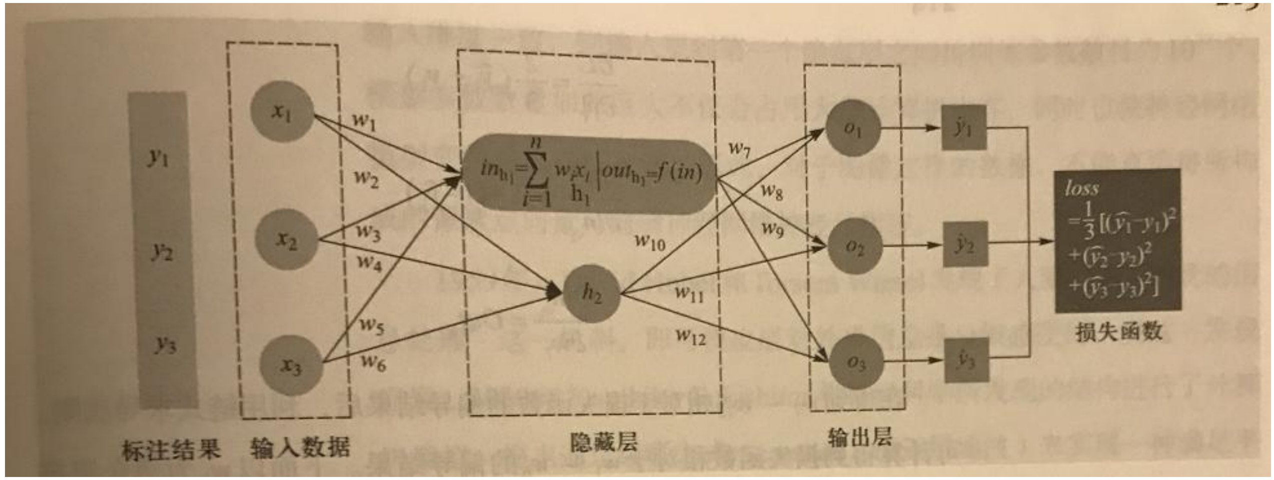


图6.8 包含一个隐藏层的多层感知机

每个神经元完成如下两项任务：

1）对相邻前序层所传递信息进行线性加权累加；

2）对加权累加结果进行非线性变换。假设样本数据输入为(x_1, x_2, x_3)，其标注信息为(y_1, y_2, y_3)。在三类分类问题中，对于输入数据(x_1, x_2, x_3)，y_1、y_2和y_3中只有一个取值为1、其余两个取值为0。

参数优化：误差反向传播 (error back propagation, BP)

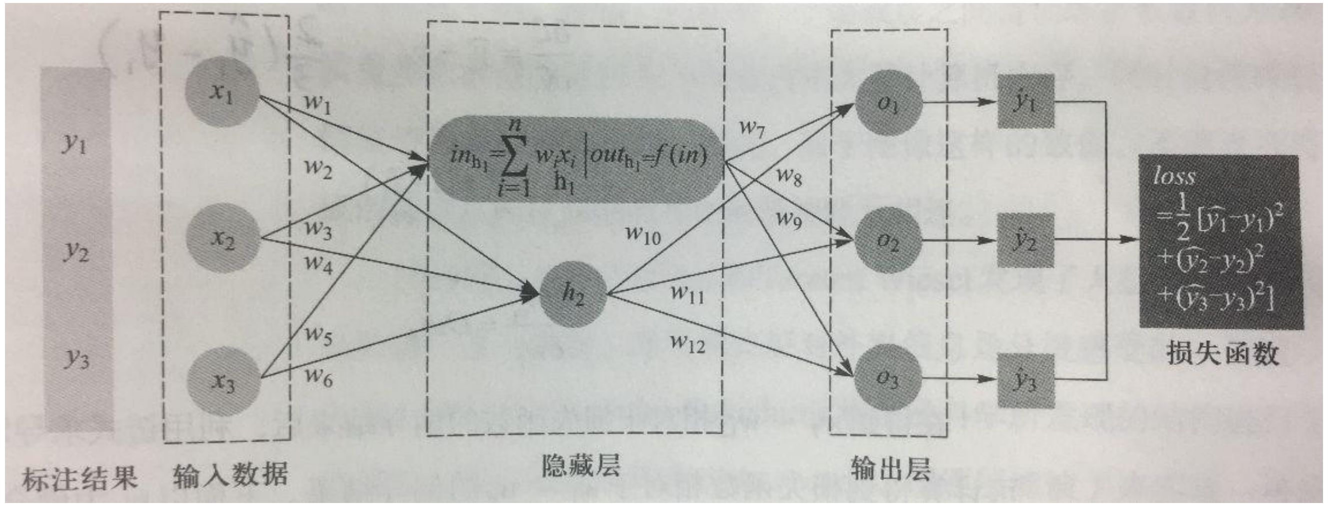


图6.8 包含一个隐藏层的多层感知机

单元	相邻前序神经元传递信息线性累加	非线性变换
h_1	$In_{h_1}=w_1*x_1+w_3*x_2+w_5*x_3$	$Out_{h_1}=sigmoid(In_{h_1})$
h_2	$In_{h_2}=w_2*x_1+w_4*x_2+w_6*x_3$	$Out_{h_2}=sigmoid(In_{h_2})$
o_1	$In_{o_1}=w_7*Out_{h_1}+w_{10}*Out_{h_2}$	$\hat{y}_1=Out_{o_1}=sigmoid(In_{o_1})$
o_2	$In_{o_2}=w_8*Out_{h_1}+w_{11}*Out_{h_2}$	$\hat{y}_2=Out_{o_2}=sigmoid(In_{o_2})$
o_3	$In_{o_3}=w_9*Out_{h_1}+w_{12}*Out_{h_2}$	$\hat{y}_3=Out_{o_3}=sigmoid(In_{o_3})$

表6.2 在图6.8中神经网络各单元计算结果

参数优化：误差反向传播（error back propagation, BP)

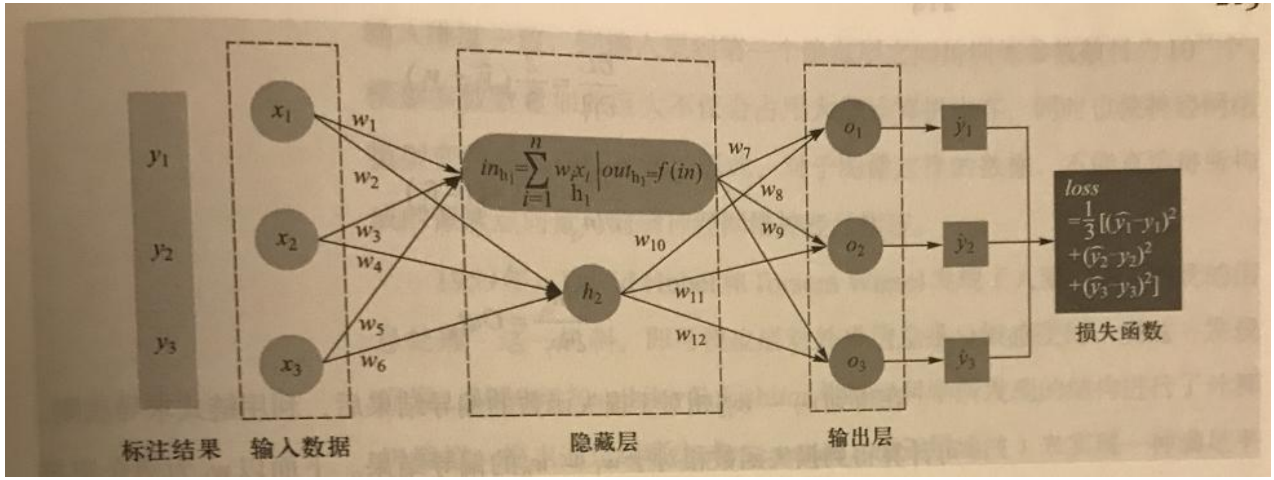


图6.8 包含一个隐藏层的多层感知机

一旦神经网络在当前参数下给出了预测结果$(\widehat{y_1},\widehat{y_2},\widehat{y_3})$后，通过均方误差损失函数来计算模型预测值与真实值$(y_1,y_2,y_3)$之间误差，记为$loss=\frac{1}{3}\sum_{i=1}^{3}(\hat{y}_i-y_i)^2$。

接下来通过梯度下降和误差反向传播方法，沿着损失函数梯度的反方向来更改参数变量取值，使得损失函数快速下降到其最小值。由于损失函数对w_7~w_{12}的偏导计算相似，在此以w_7为例来介绍如何更新w_7这一参数取值。记损失函数为$\mathcal{L}(w)$，$\mathcal{L}(w)=\frac{1}{3}\sum_{i=1}^{3}(\hat{y}_i-y_i)^2$。

参数优化：误差反向传播 (error back propagation, BP)

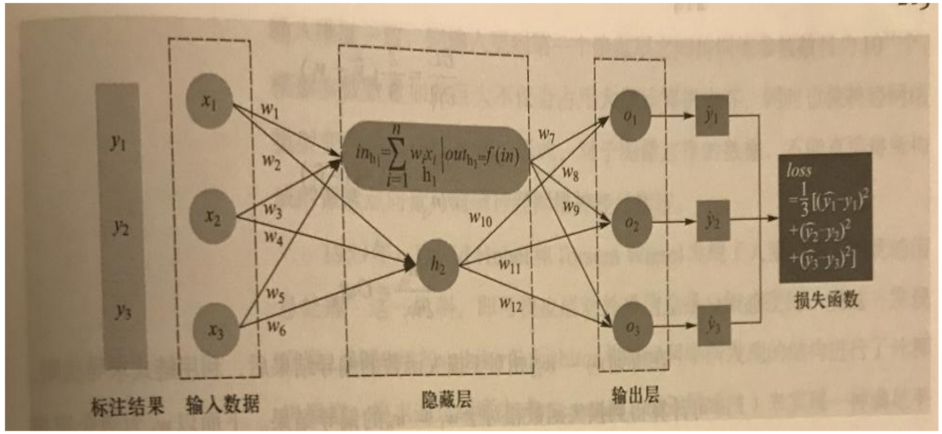


图6.8 包含一个隐藏层的多层感知机

损失函数$\mathcal{L}$对参数w_7的偏导可如下计算：

$$\delta_7 = \frac{\partial \mathcal{L}}{\partial w_7} = \frac{\partial \mathcal{L}}{\partial \widehat{y_1}} * \frac{\partial \widehat{y_1}}{\partial In_{o_1}} * \frac{\partial In_{o_1}}{\partial w_7}$$

上述公式中每一项结果如下：

$$\frac{\partial \mathcal{L}}{\partial \widehat{y_1}} = \widehat{y_1} - y_1$$

$$\frac{\partial \widehat{y_1}}{\partial In_{o_1}} = \widehat{y_1} * (1 - \widehat{y_1})$$

$$\frac{\partial In_{o_1}}{\partial w_7} = Out_{h_1}$$

参数优化：误差反向传播 (error back propagation, BP)

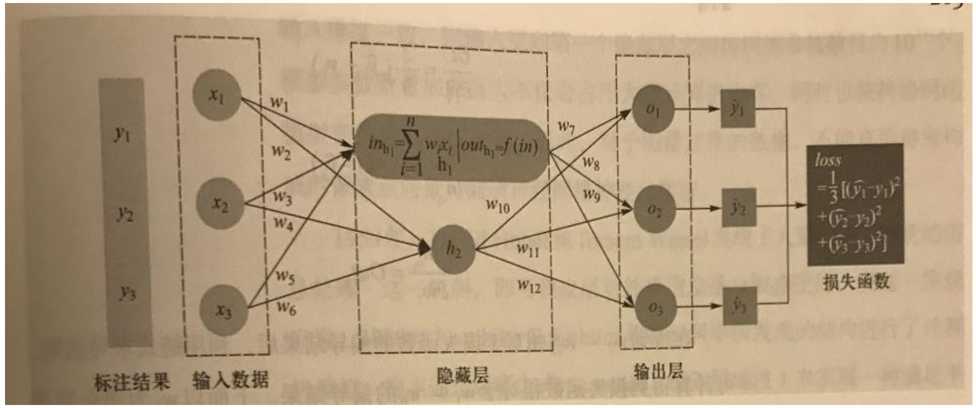


图6.8 包含一个隐藏层的多层感知机

在计算得到所有参数相对于损失函数$\mathcal{L}$的偏导结果后，利用梯度下降算法，通过$w_i^{new}=w_i-\eta*\delta_i$来更新参数取值。然后不断迭代，直至模型参数收敛，此时损失函数减少到其最小值。

下面以w_1为例介绍损失函数$\mathcal{L}$对w_1的偏导δ_1。

$$
\begin{aligned}
\delta_1 &= \frac{\partial\mathcal{L}}{\partial w_1} \\
&= \frac{\partial\mathcal{L}}{\partial\widehat{y_1}}*\frac{\partial\widehat{y_1}}{\partial w_1}+\frac{\partial\mathcal{L}}{\partial\widehat{y_2}}*\frac{\partial\widehat{y_2}}{\partial w_1}+\frac{\partial\mathcal{L}}{\partial\widehat{y_3}}*\frac{\partial\widehat{y_3}}{\partial w_1} \\
&= \frac{\partial\mathcal{L}}{\partial\widehat{y_1}}*\frac{\partial\widehat{y_1}}{\partial In_{o_1}}*\frac{\partial In_{o_1}}{\partial Out_{h_1}}*\frac{\partial Out_{h_1}}{\partial In_{h_1}}*\frac{\partial In_{h_1}}{\partial w_1}+\frac{\partial\mathcal{L}}{\partial\widehat{y_2}}*\frac{\partial\widehat{y_2}}{\partial In_{o_2}} \\
&\quad *\frac{\partial In_{o_2}}{\partial Out_{h_1}}*\frac{\partial Out_{h_1}}{\partial In_{h_1}}*\frac{\partial In_{h_1}}{\partial w_1}+\frac{\partial\mathcal{L}}{\partial\widehat{y_3}}*\frac{\partial\widehat{y_3}}{\partial In_{o_3}}*\frac{\partial In_{o_3}}{\partial Out_{h_1}} \\
&\quad *\frac{\partial Out_{h_1}}{\partial In_{h_1}}*\frac{\partial In_{h_1}}{\partial w_1} \\
&= \left(\frac{\partial\mathcal{L}}{\partial\widehat{y_1}}*\frac{\partial\widehat{y_1}}{\partial In_{o_1}}*\frac{\partial In_{o_1}}{\partial Out_{h_1}}+\frac{\partial\mathcal{L}}{\partial\widehat{y_2}}*\frac{\partial\widehat{y_2}}{\partial In_{o_2}}*\frac{\partial In_{o_2}}{\partial Out_{h_1}}+\frac{\partial\mathcal{L}}{\partial\widehat{y_3}}\right. \\
&\quad \left.*\frac{\partial\widehat{y_3}}{\partial In_{o_3}}*\frac{\partial In_{o_3}}{\partial Out_{h_1}}\right)*\frac{\partial Out_{h_1}}{\partial In_{h_1}}*\frac{\partial In_{h_1}}{\partial w_1} \\
&= (\delta_7+\delta_8+\delta_9)*\frac{\partial Out_{h_1}}{\partial In_{h_1}}*\frac{\partial In_{h_1}}{\partial w_1}
\end{aligned}
$$

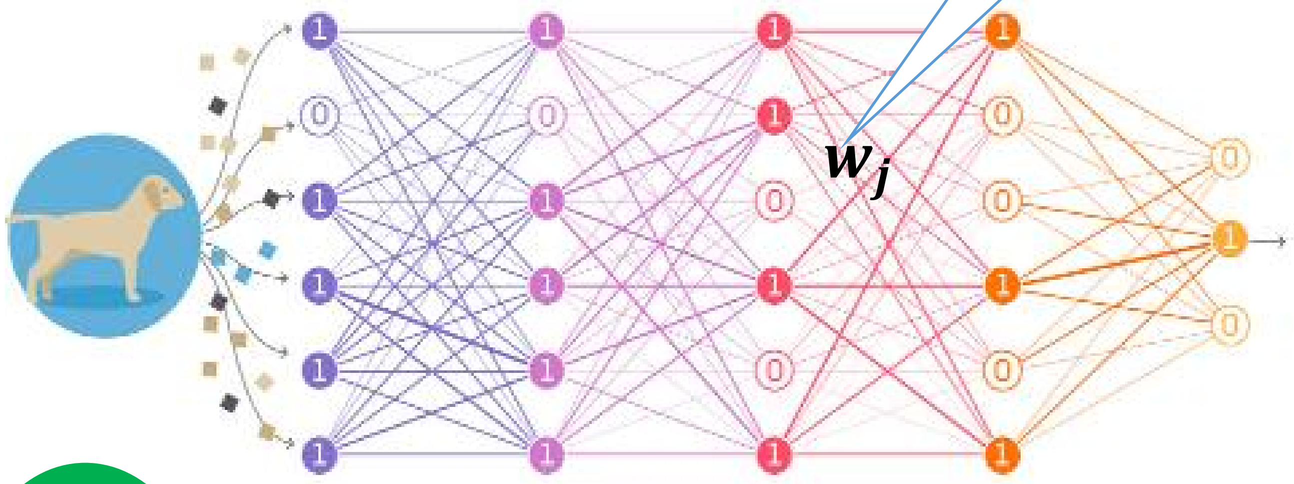
机器学习的能力在于拟合和优化

$$w_j^{new} = w_j^{old} - \eta \times \frac{\partial Model_{loss}}{\partial w_j}$$

调节参数

梯度下降

误差后向传播



dog

像素点构成

第一层 关键点

第二层 边界点

第三层 组件

第四层 特征

调节网络结构（炼金术）

谢谢！