

深度学习

主讲：王亚星、刘夏雷、郭春乐
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

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（了解）**
- **深度生成学习（了解）**
- **深度学习应用（了解）**

习题回顾

第1题 | 单选题 1分

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

第2题 | 单选题 1分

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

习题回顾

第4题 | 单选题 1分

3.下面对主成分分析的描述不正确的是()

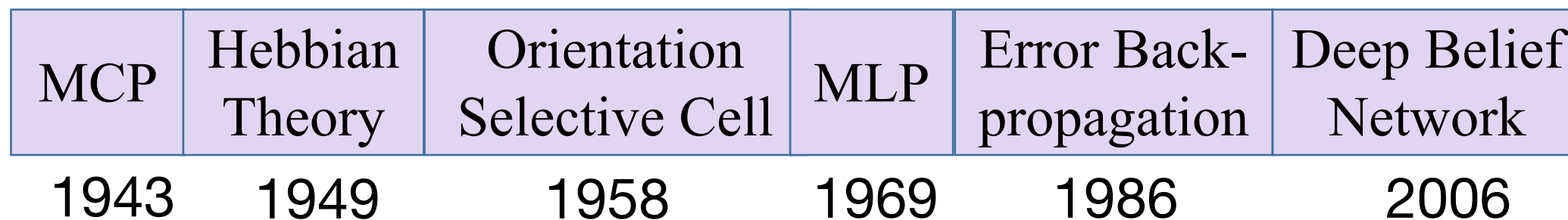
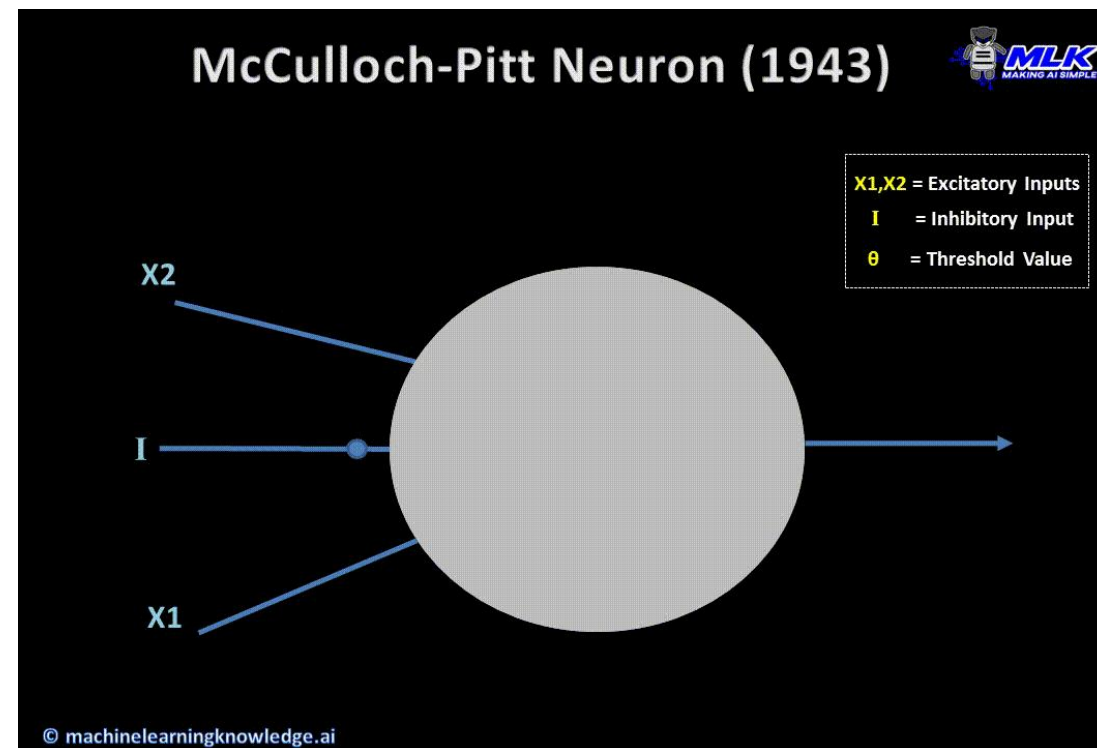
- A 主成分分析是一种特征降维方法
- B 主成分分析可保证原始高维样本数据被投影映射后,其方差保持最大
- C 在主成分分析中,将数据向方差最大方向进行投影,可使得数据所蕴含信息没有丢失,以便在后续处理过程中各个数据“彰显个性”
- D 在主成分分析中,所得低维数据中每一维度之间具有极大相关度

4.下面对特征人脸算法描述不正确的是()

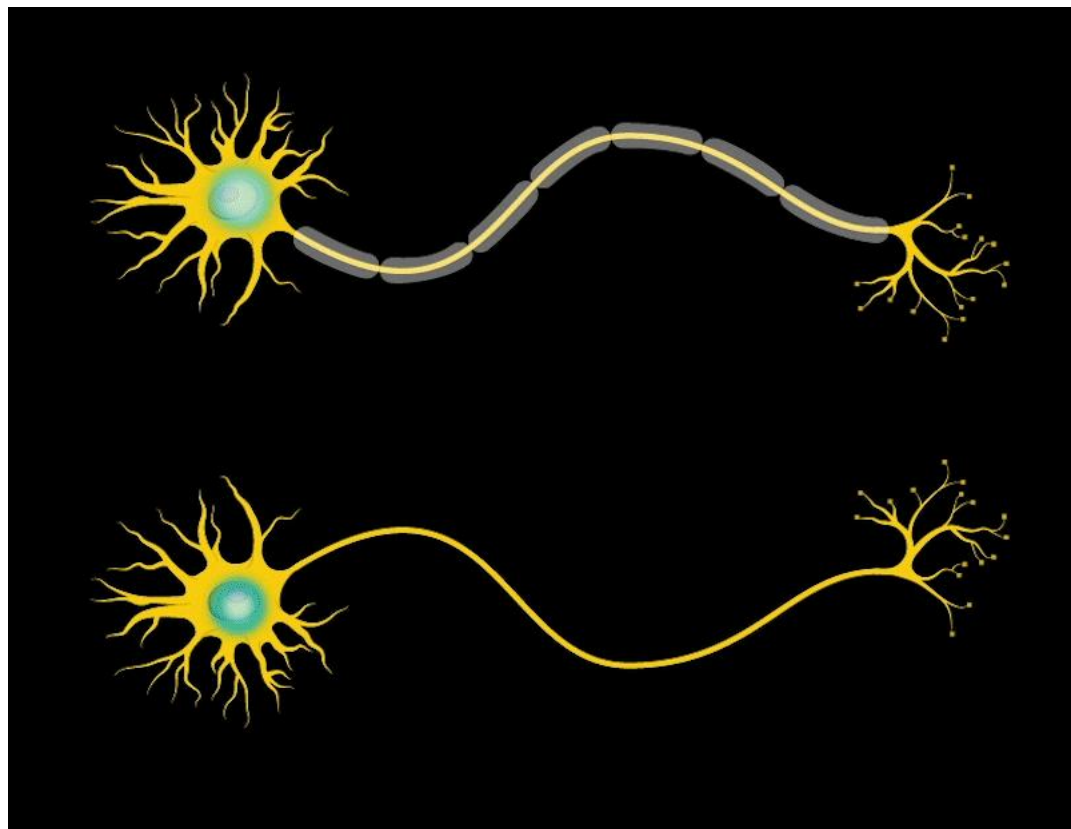
- A 特征人脸方法是一种应用主成分分析来实现人脸图像降维的方法
- B 特征人脸方法是用一种称为“特征人脸(eigenface)”的特征向量按照线性组合形式来表达每一张原始人脸图像
- C 每一个特征人脸的维数与原始人脸图像的维数一样大
- D 特征人脸之间的相关度要尽可能大

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。



深度学习的历史发展



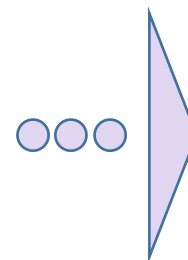
赫布理论(Hebbian theory):

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.

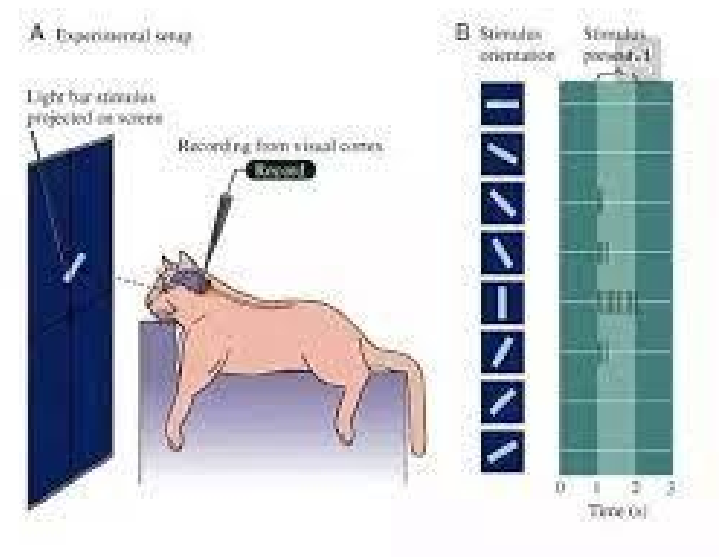
“神经元之间突触的强弱变化是学习与记忆的生理学基础”这一理论为联结主义人工智能研究提供了认知神经心理学基础。

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

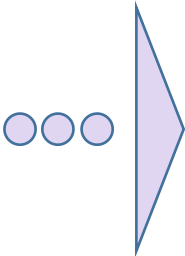


深度学习的历史发展

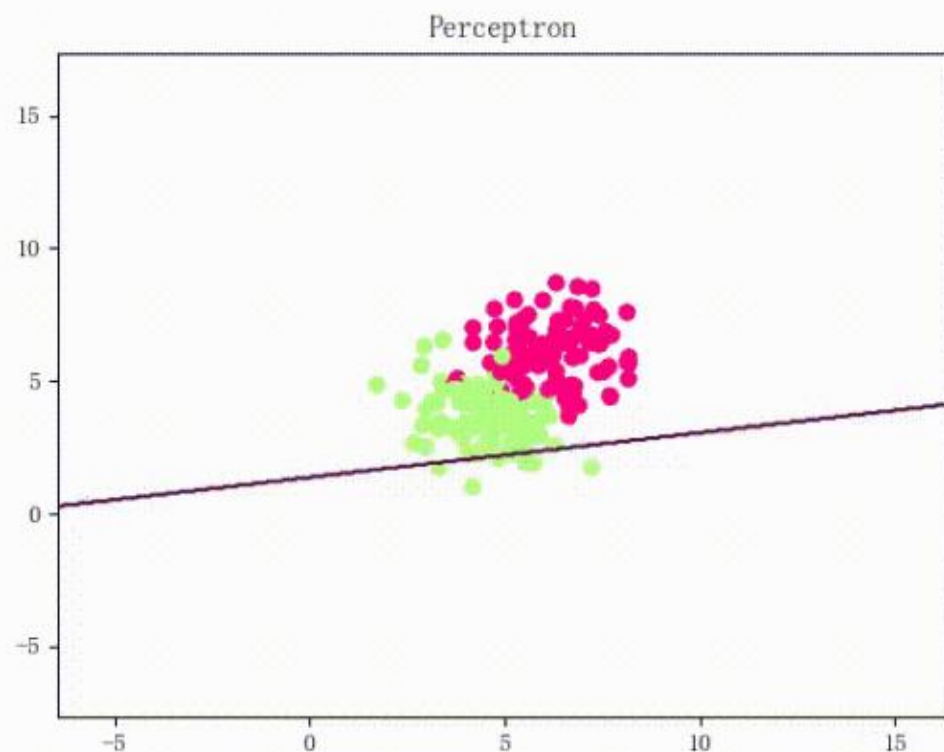
1958 年，David Hubel和Torsten Wiesel在实验中发现小猫后脑皮层中不同视觉神经元与瞳孔所受刺激之间存在某种对应关系，由此发现了一种被称为“方向选择性细胞(orientation selective cell)”的神经元细胞，从而揭示了“视觉系统信息分层处理”这一机制。



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

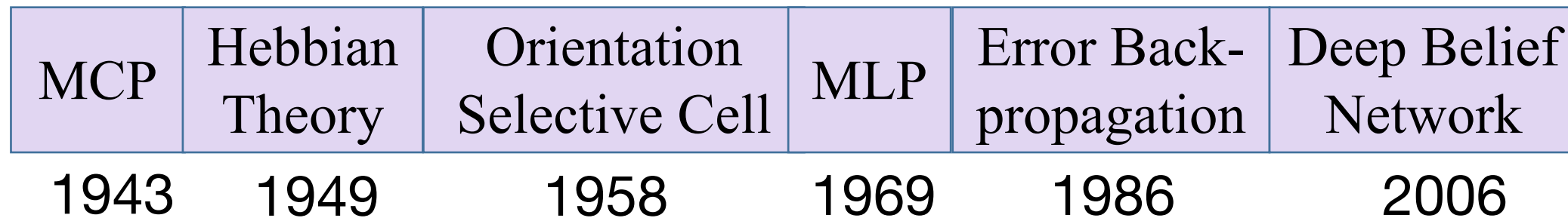


深度学习的历史发展



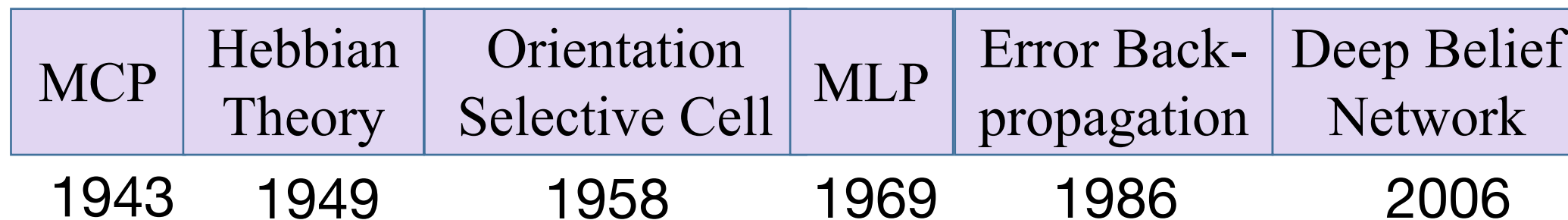
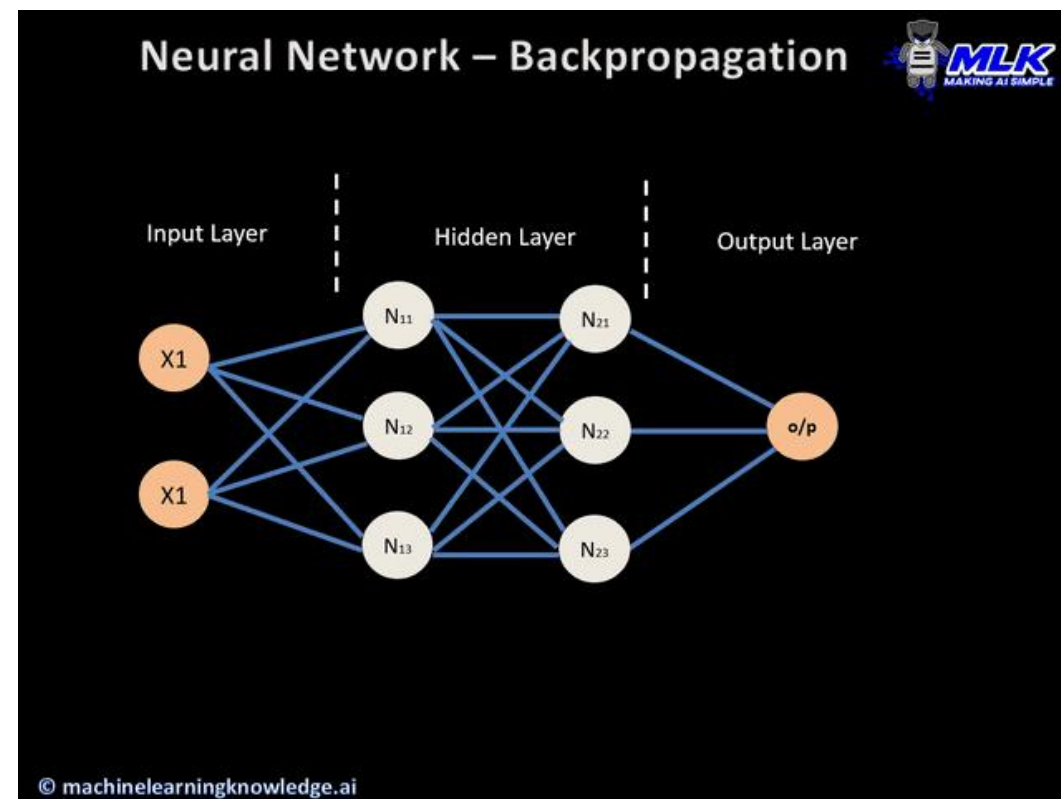
神经网络研究的突破来自于Frank Rosenblatt在20世纪50年代所提出的“**感知机 (perceptron)**”模型。

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）。



深度学习的历史发展

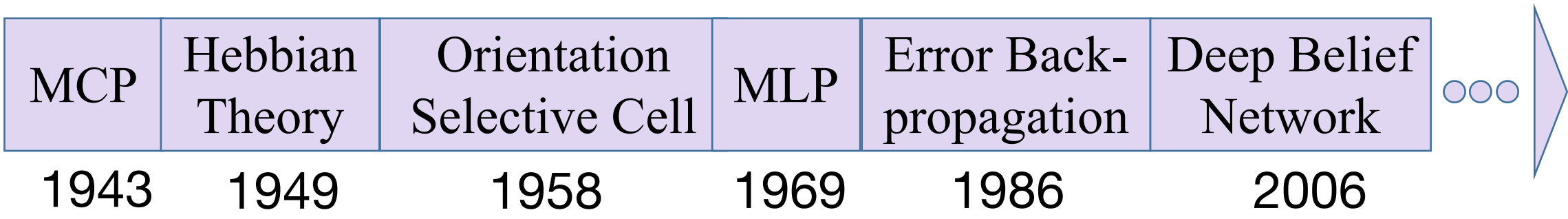
最早由 Werbos 提出 [Werbos 1974] [Werbos 1990]、并且由 Rumelhar 和 Hinton 等人 [Rumelhart 1986] 完善的**误差后向传播** (error backpropagation) 算法解决了多层感知机中参数优化这一难题。



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。

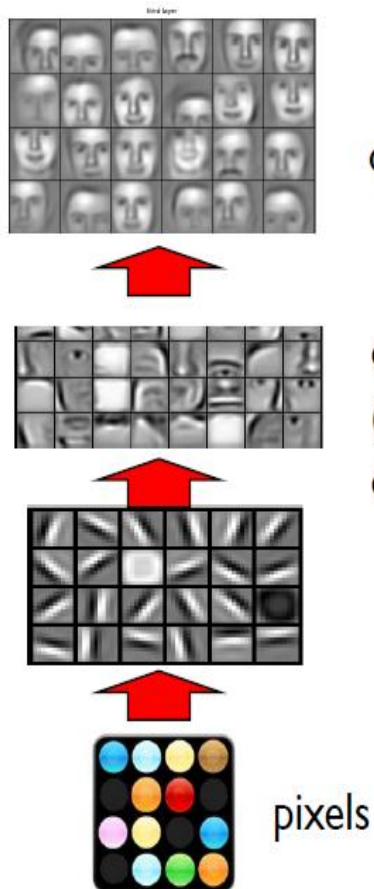
2006年, Hinton 在《Science》等期刊上发表了论文, 首次提出了“**深度信念网络** (deep belief network)”模型 [Hinton 2006], 在相关分类任务上可取得性能超过了传统浅层学习模型 (如支持向量机), 使得深度架构引起了大家的关注。



深度学习：以端到端的方式逐层抽象、逐层学习

端到端？

深度学习：以端到端的方式逐层抽象、逐层学习

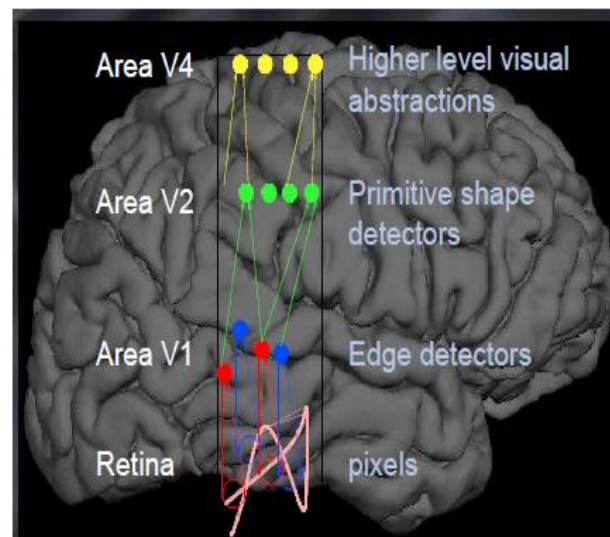


object models

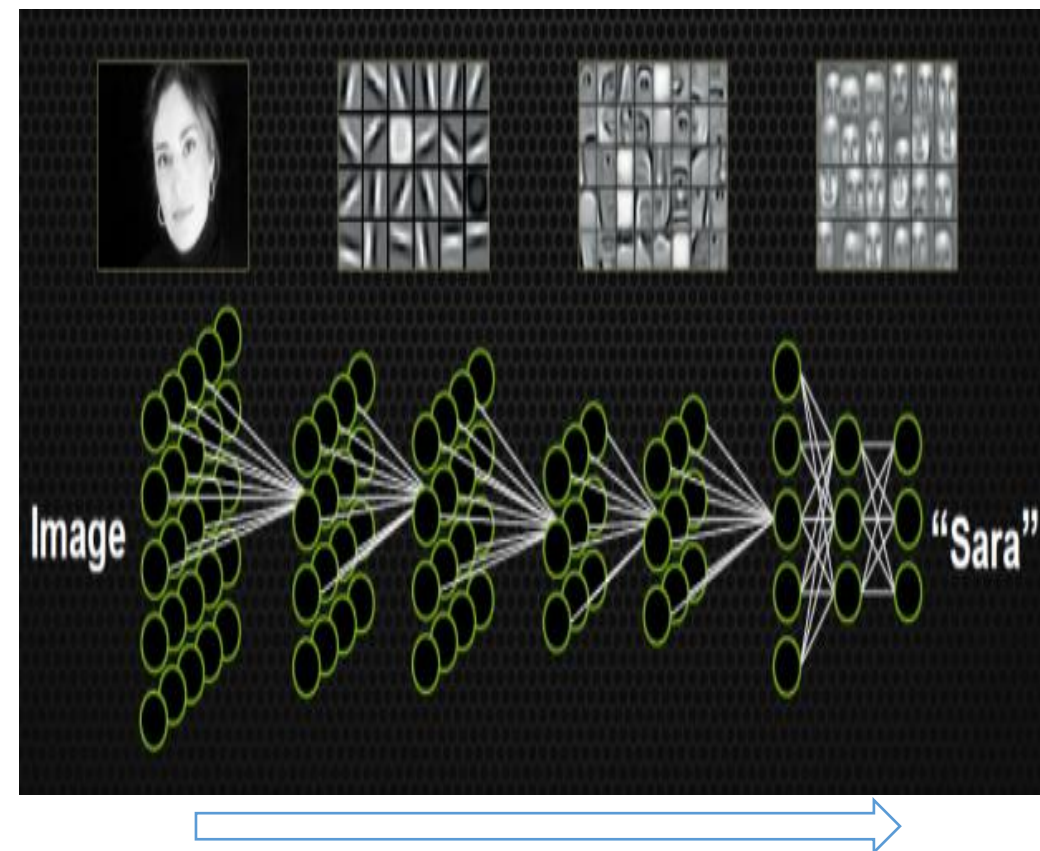
object parts
(combination
of edges)

edges

pixels

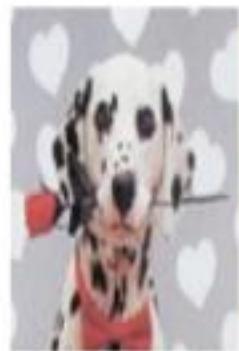


Slide credit: Andrew Ng



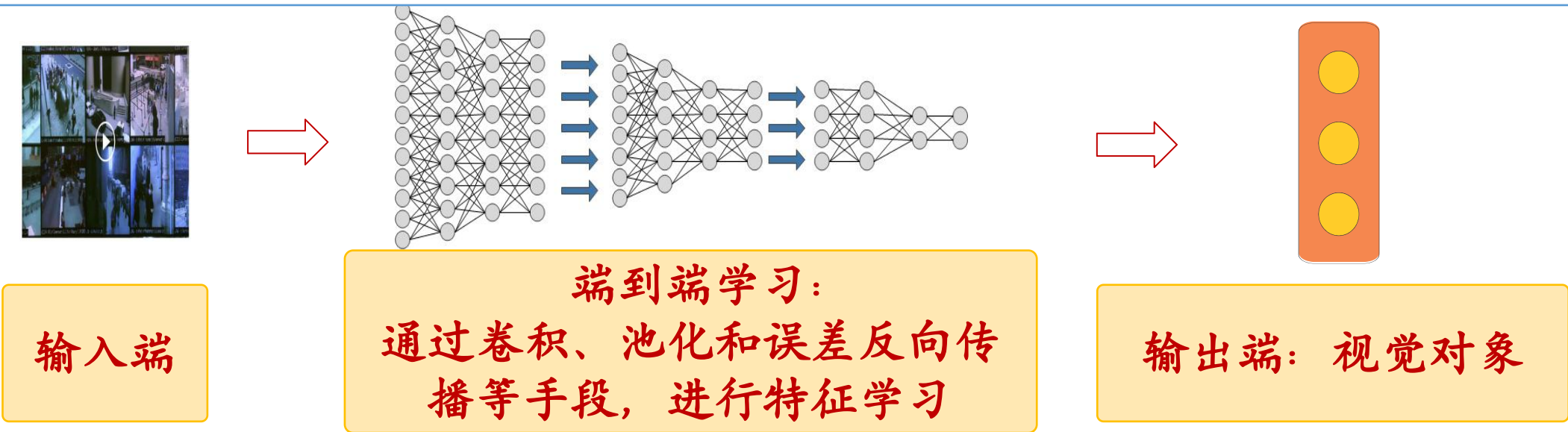
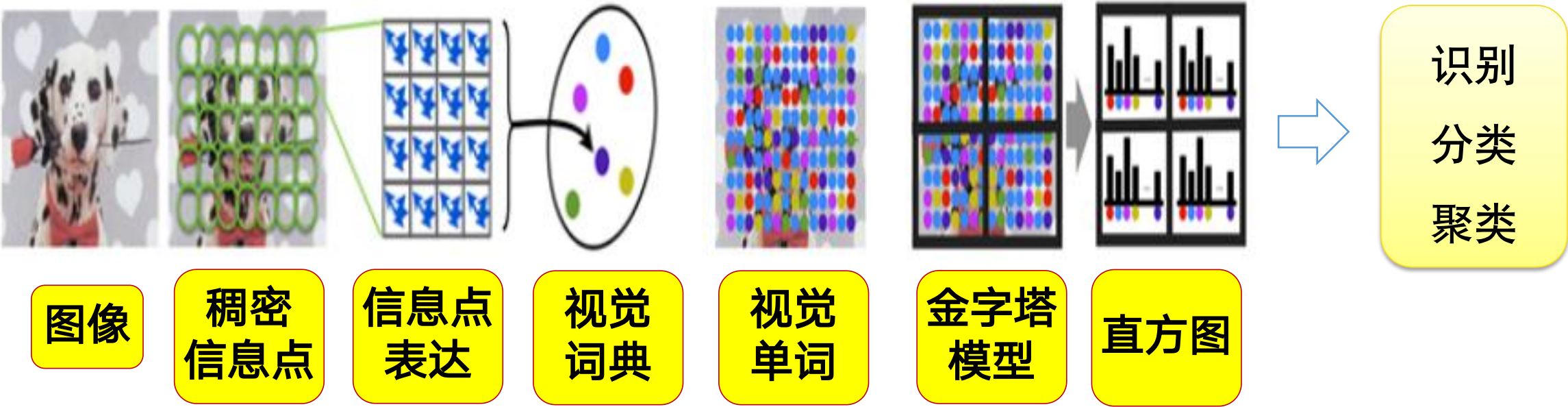
- 深度学习所得模型可视为一个复杂函数
- 非线性变换与映射的过程：像素点→语义

浅层学习 VS 深度学习：从分段学习到端到端学习



图像

浅层学习 VS 深度学习：从分段学习到端到端学习

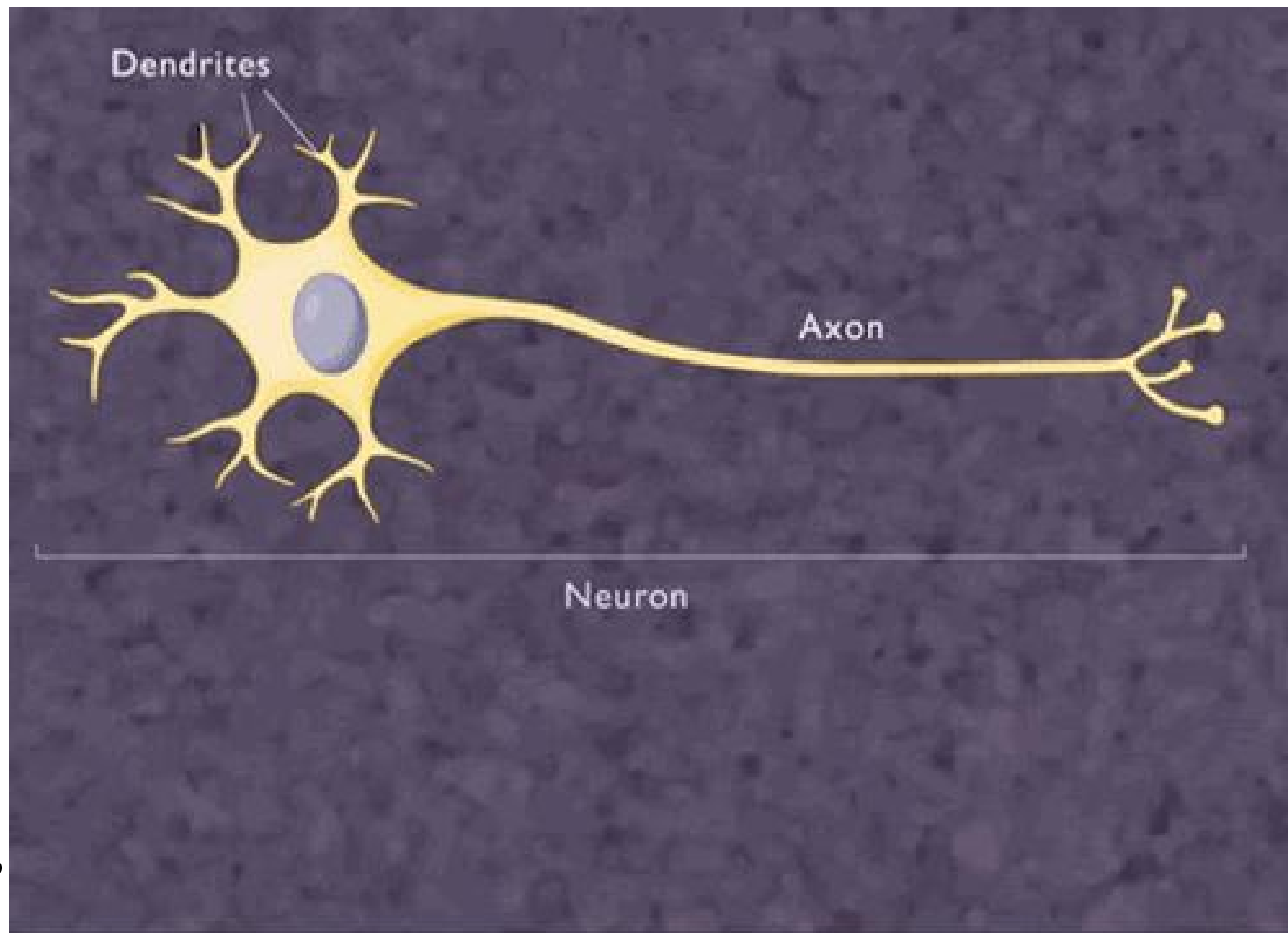


提纲

- 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

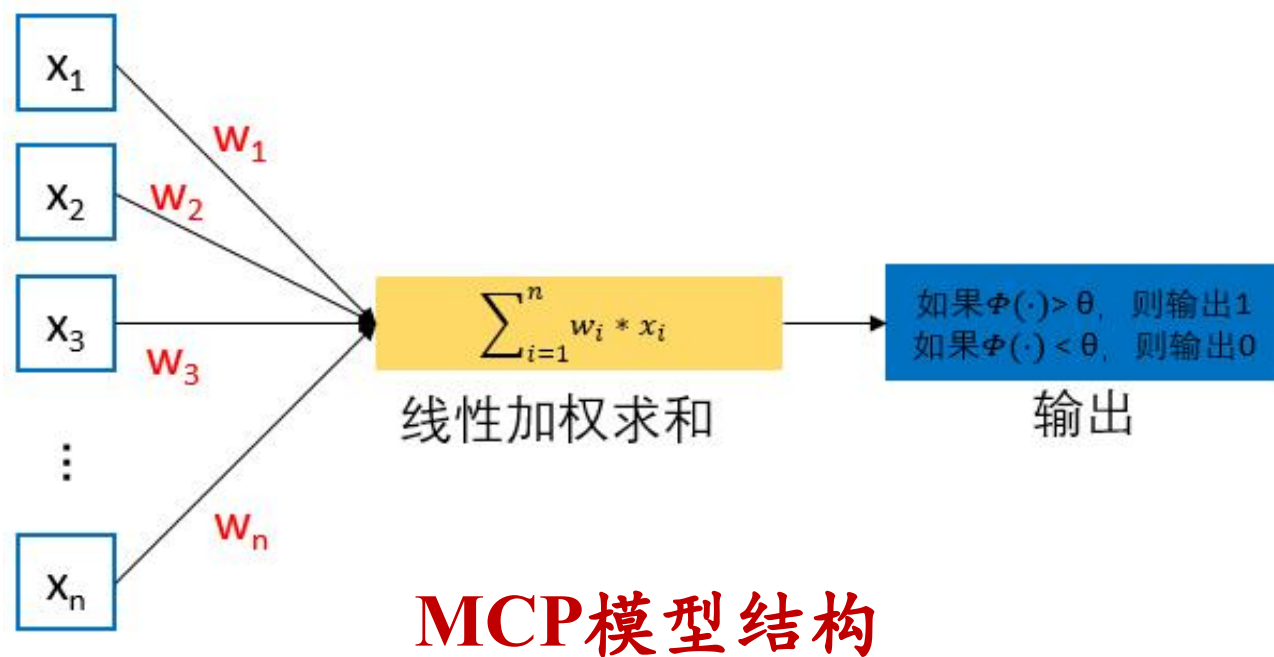
生物学中的神经元

- 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

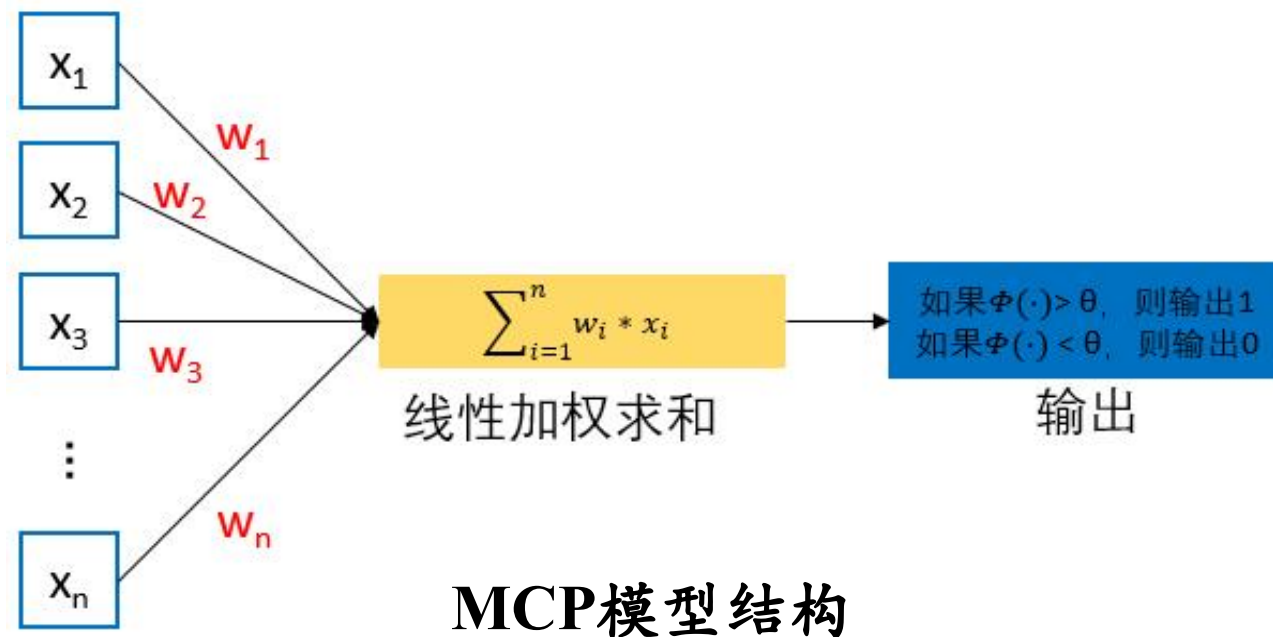
- 基于神经元细胞的结构特性与传递信息方式，神经科学家Warren McCulloch和逻辑学家Walter Pitts合作提出了“McCulloch–Pitts (MCP) neuron”模型[McCulloch 1943]。在人工神经网络中，MCP模型成为人工神经网络中的最基本结构。



神经元

- 给定 n 个**二值化（0或1）**的输入数据 $x_i (1 \leq i \leq n)$ 与连接参数 $w_i (1 \leq i \leq n)$ ，**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：

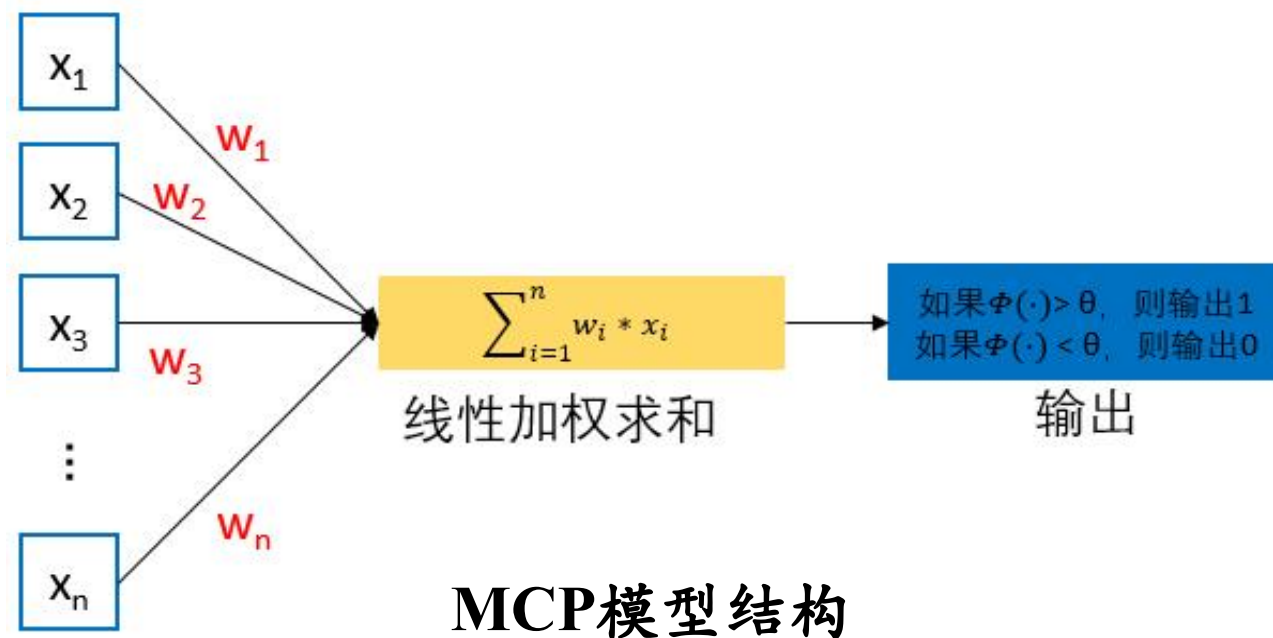
$$y = \Phi\left(\sum_{i=1}^n w_i x_i\right)$$



神经元

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

- 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。



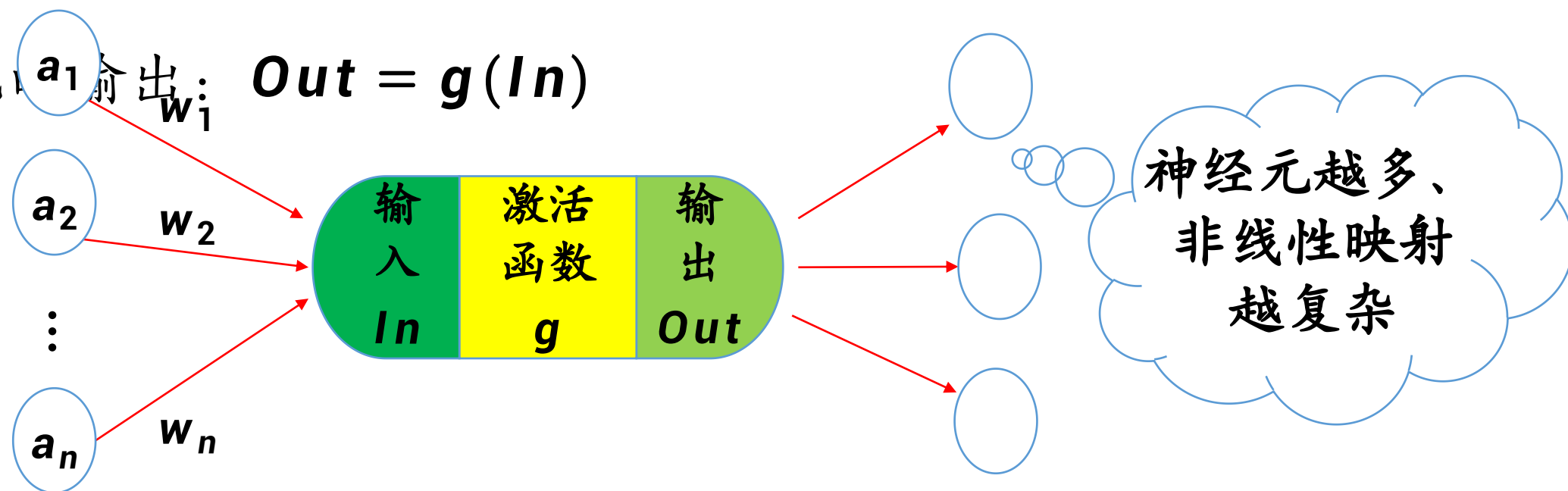
刻画神经元功能的数学模型

- 神经元是深度学习模型中基本单位

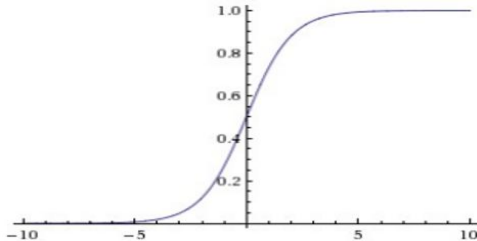
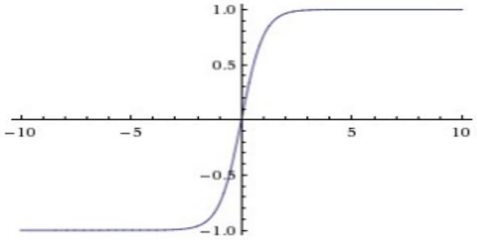
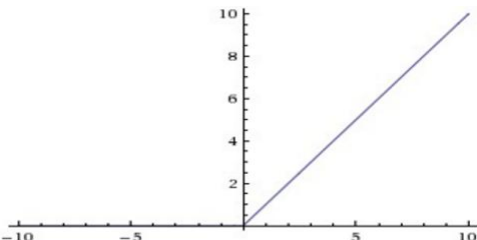
- 对相邻前向神经元输入信息进行加权累加： $ln = \sum_{i=1}^n w_i * a_i$

- 对累加结果进行非线性变换（通过激活函数）： $g(x)$ （类比神经元激活）

- 神经元输入输出： $Out = g(ln)$



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	$f(x) = \frac{1}{1 + e^{-x}}$		$f'(x) = f(x)(1 - f(x))$
Tanh	$f(x) = \frac{2}{1 + e^{-2x}} - 1$		$f'(x) = 1 - f(x)^2$
Relu (Rectified Linear Unit)	$f(x) = \begin{cases} 0, & \text{for } x < 0 \\ x, & \text{for } x \geq 0 \end{cases}$		$f'(x) = \begin{cases} 0, & \text{for } x < 0 \\ 1, & \text{for } x \geq 0 \end{cases}$

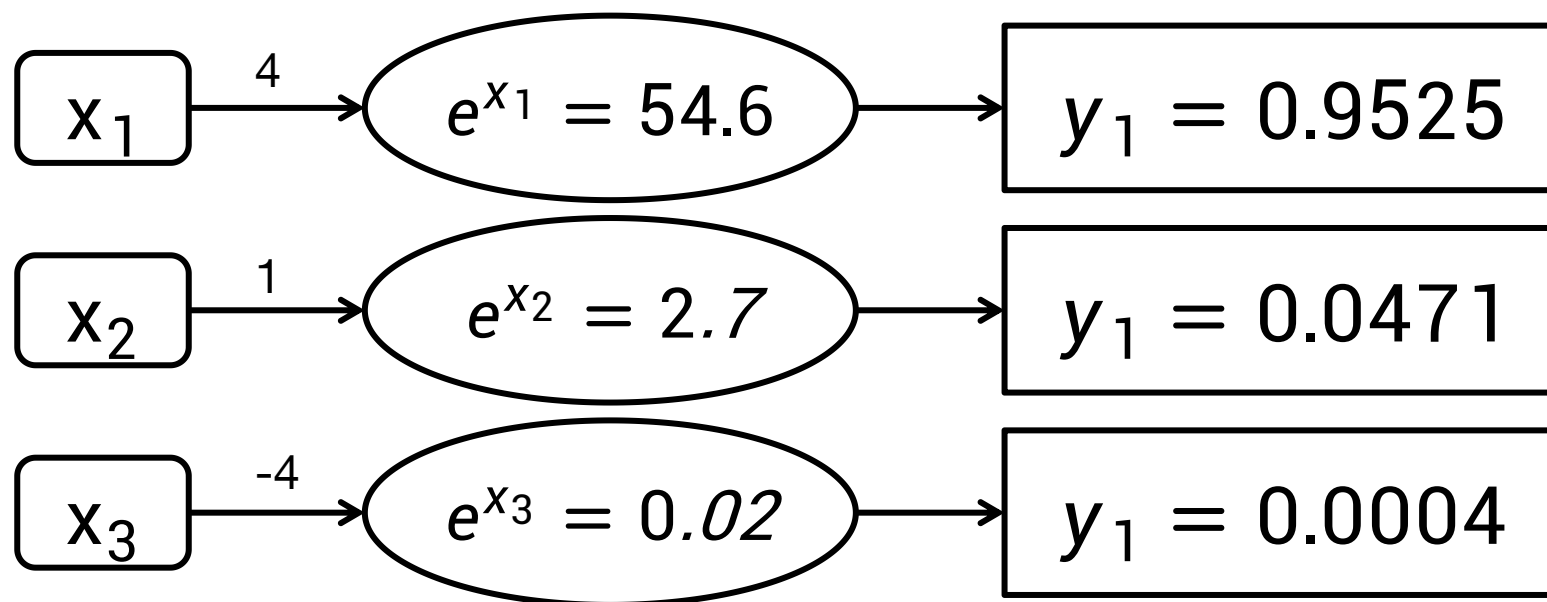
神经网络使用非线性函数作为激活函数(activation function)，通过对多个非线性函数进行组合，来实现对输入信息的非线性变换

常用的激活函数：softmax函数

- Softmax函数一般用于多分类问题中

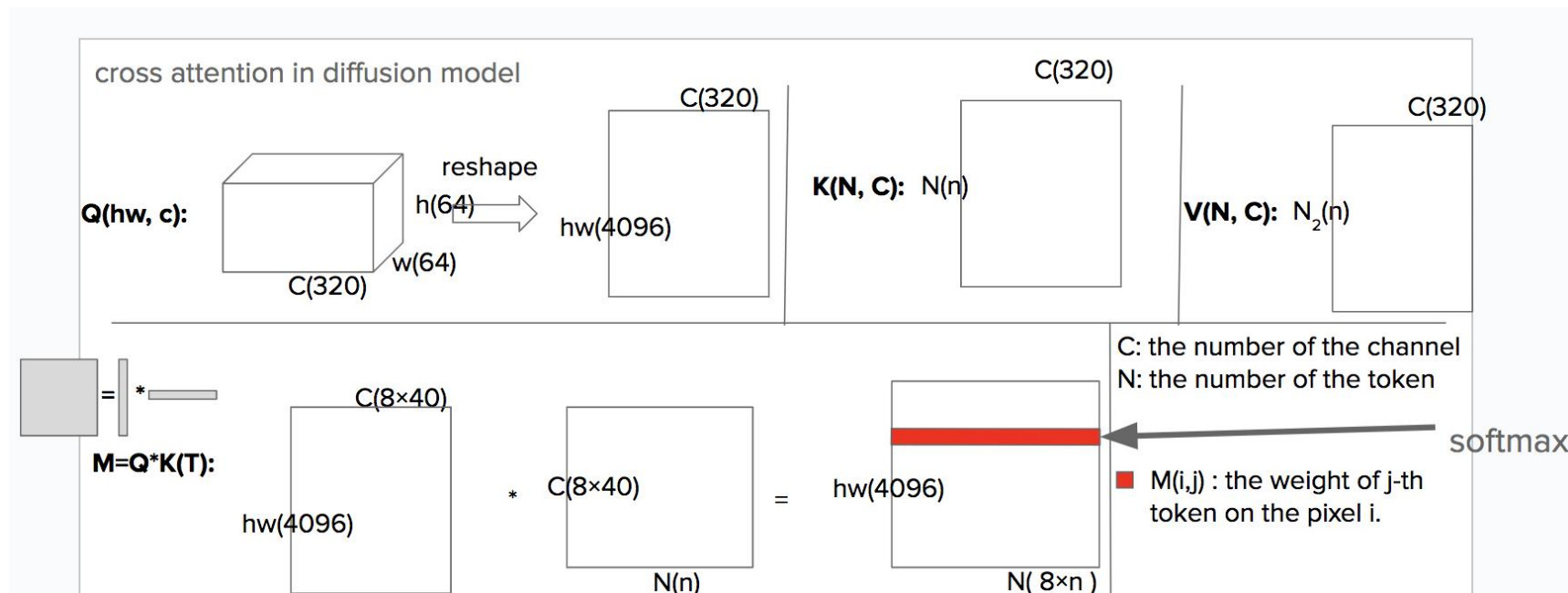
- 将输入 \mathbf{x}_i 映射到第 i 个类别的概率： $y_i = \text{softmax}(\mathbf{x}_i) = \frac{e^{x_i}}{\sum_{j=1}^k e^{x_j}}$

- $0 < y_i < 1$, $\sum_i y_i = 1$, 可将输出概率最大的作为分类目标



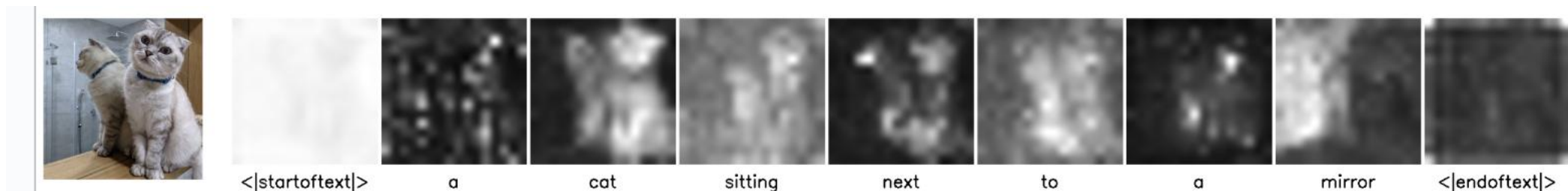
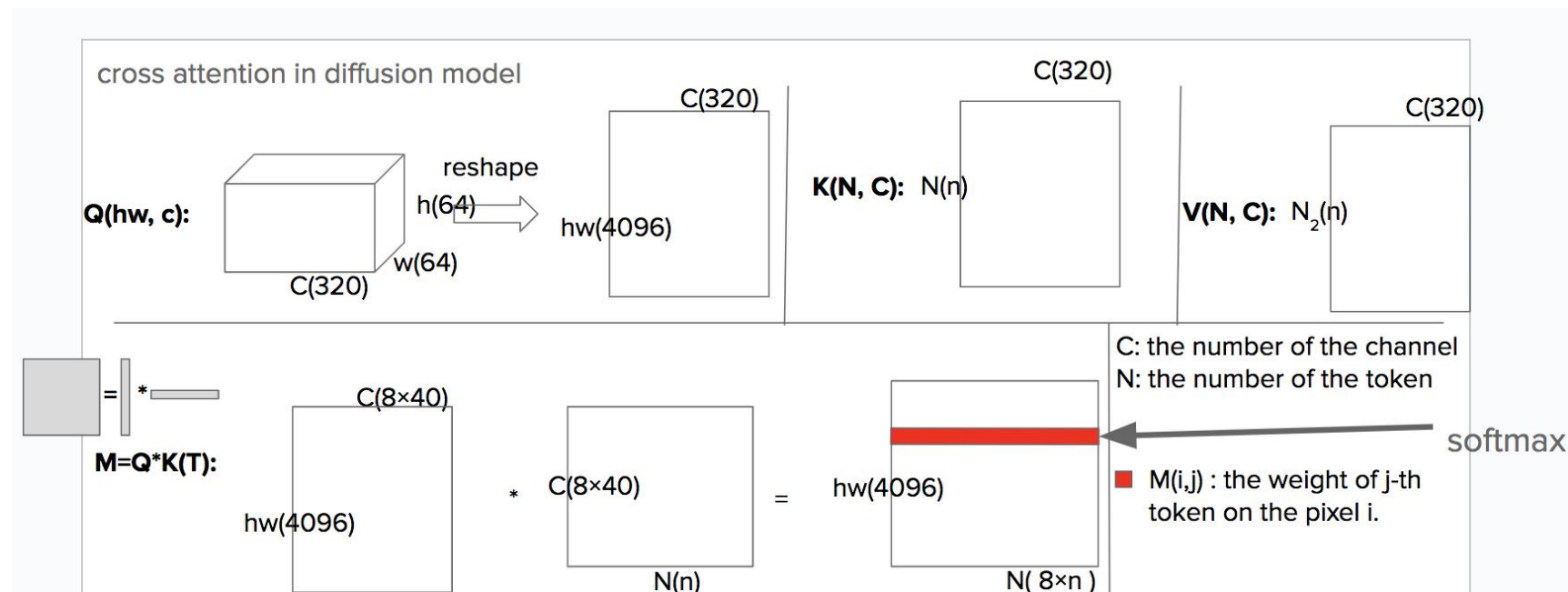
常用的激活函数：softmax函数

- Softmax函数一般用于多分类问题中

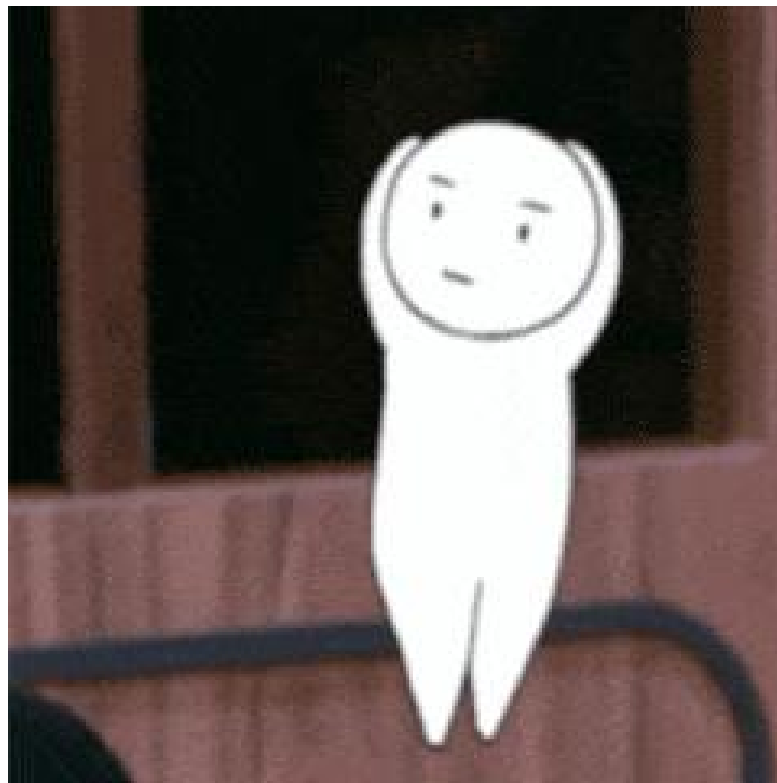


常用的激活函数：softmax函数

- Softmax函数一般用于多分类问题中



损失函数 (Loss Function)

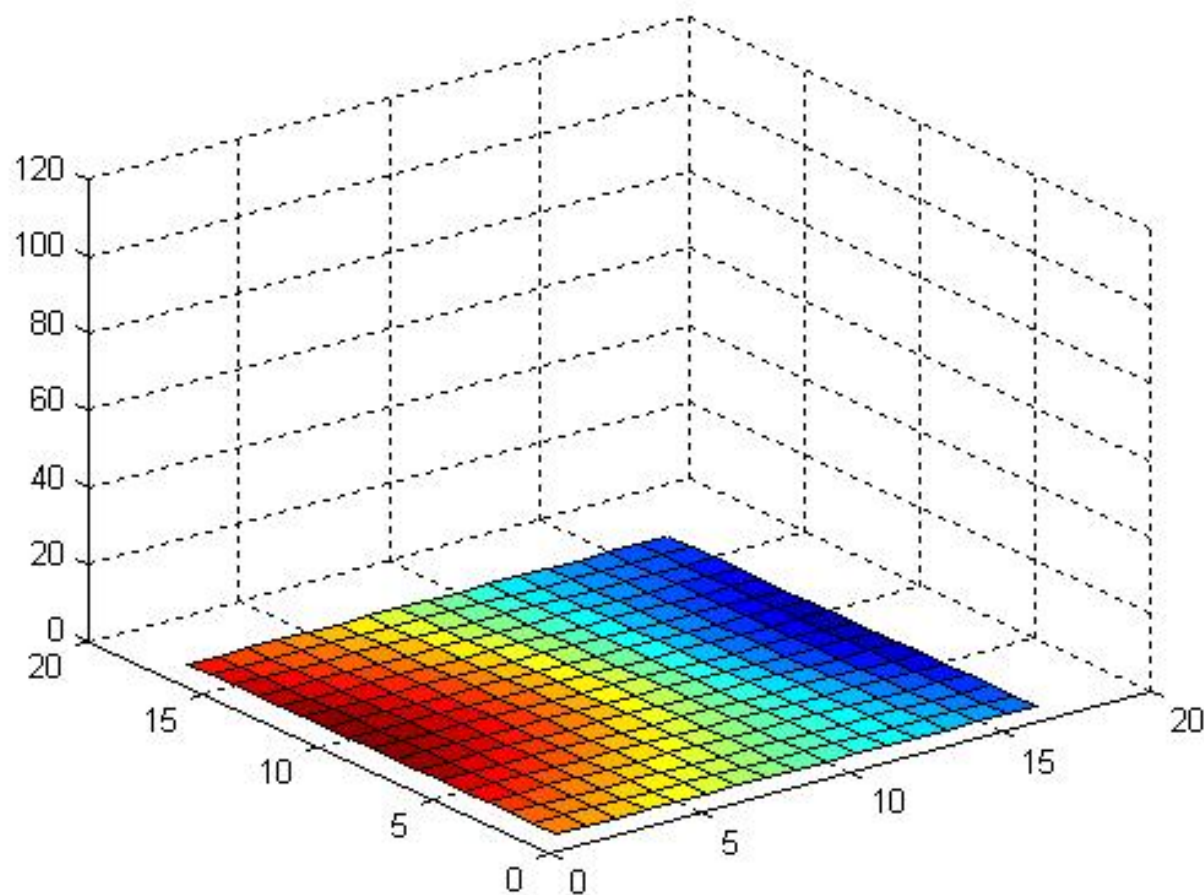


损失函数 (Loss Function)

- **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

损失函数 (Loss Function)

帮助参数更新，达到最优



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：

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

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：

$$H(p, q) = - \sum_x p(x) * \log q(x)$$

- 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 $y = (1,0,0)$

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

常用的损失函数

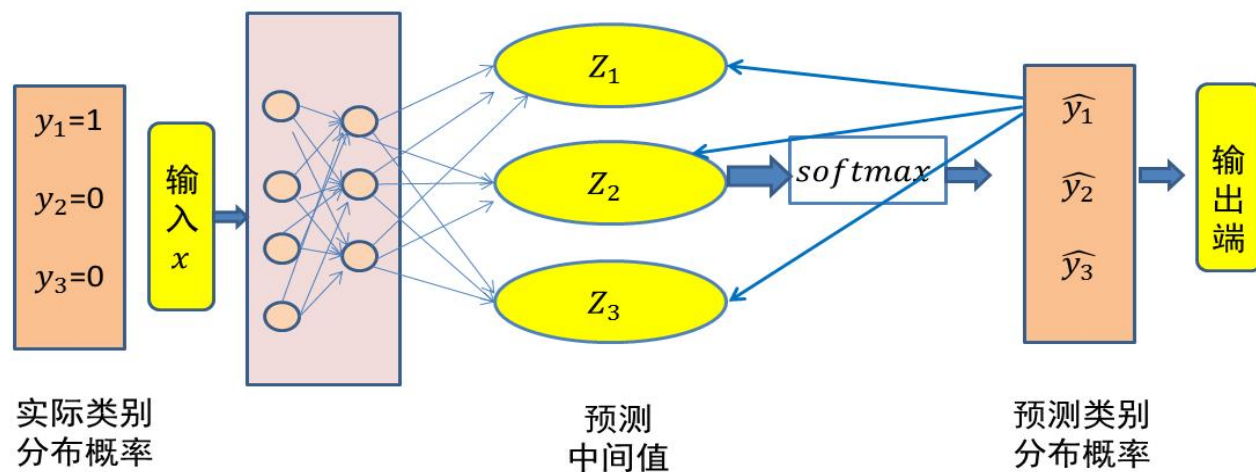
- 交叉熵损失函数

- 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

- 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 $-y \times \log(\hat{y})$



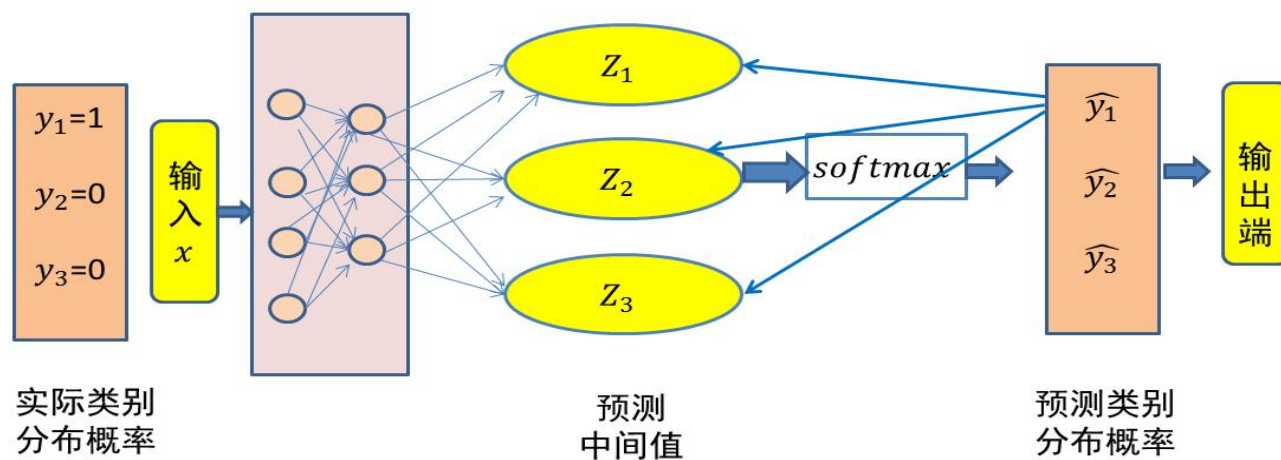
常用的损失函数

- 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
 - 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 $\hat{y} = (\hat{y}_1, \hat{y}_2, \hat{y}_3)$ 。

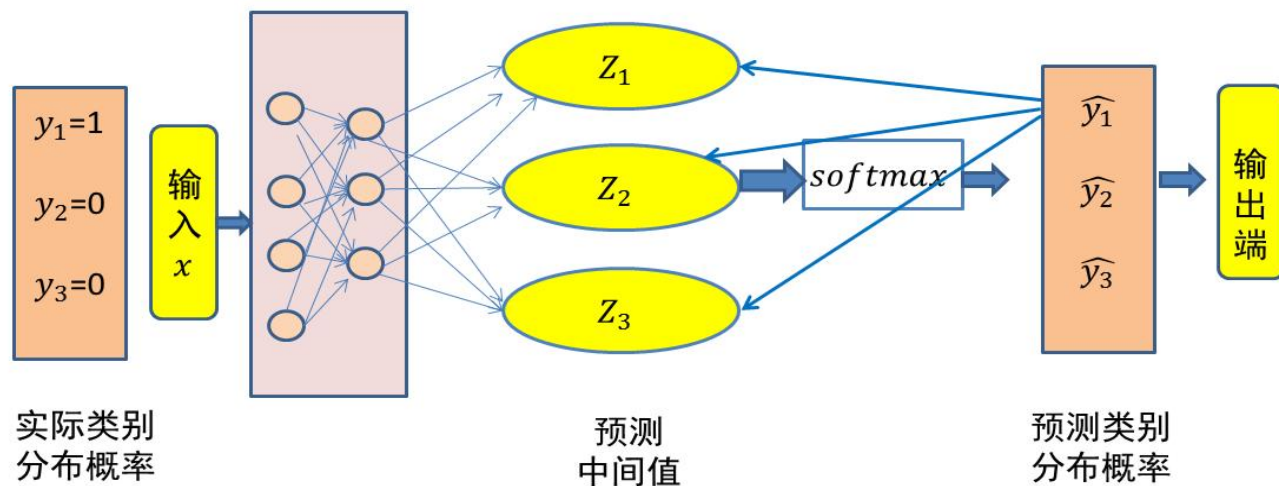


常用的损失函数

- 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， \hat{y}_1 、 \hat{y}_2 和 \hat{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 \hat{y}_1 取值要远远大于 \hat{y}_2 和 \hat{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：

$$\text{CrossEntropy} = - (y_1 \times \log(\widehat{y}_1) + y_2 \times \log(\widehat{y}_2) + y_3 \times \log(\widehat{y}_3))$$

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：

$$\text{CrossEntropy} = -(y_1 \times \log(\widehat{y}_1) + y_2 \times \log(\widehat{y}_2) + y_3 \times \log(\widehat{y}_3))$$

- 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：

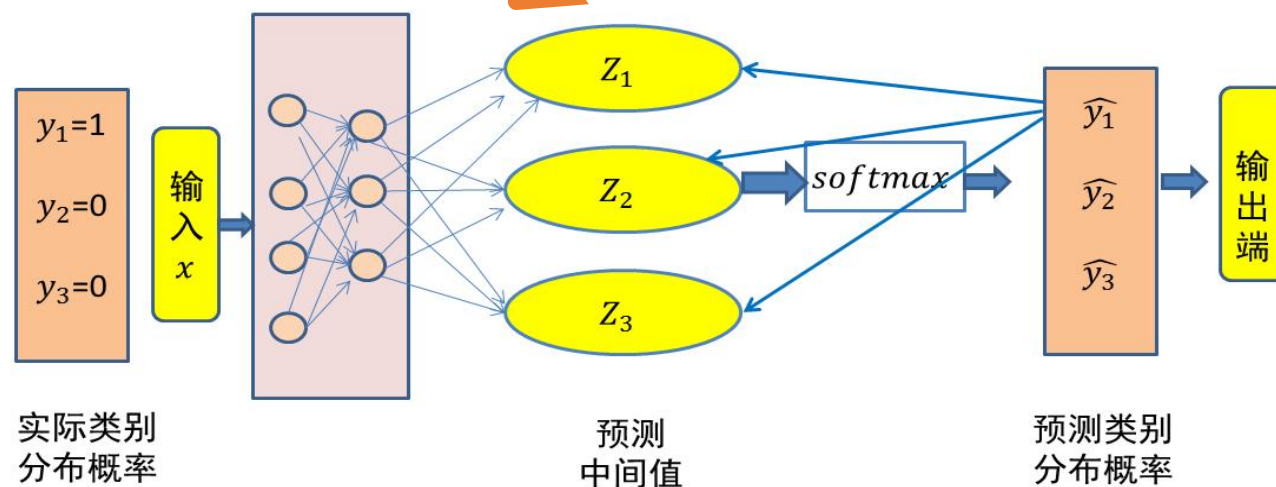
$$-y_1 \times \log(\widehat{y}_1) = -\log(\widehat{y}_1)$$

- 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 \hat{y}_1 传递给 z_1

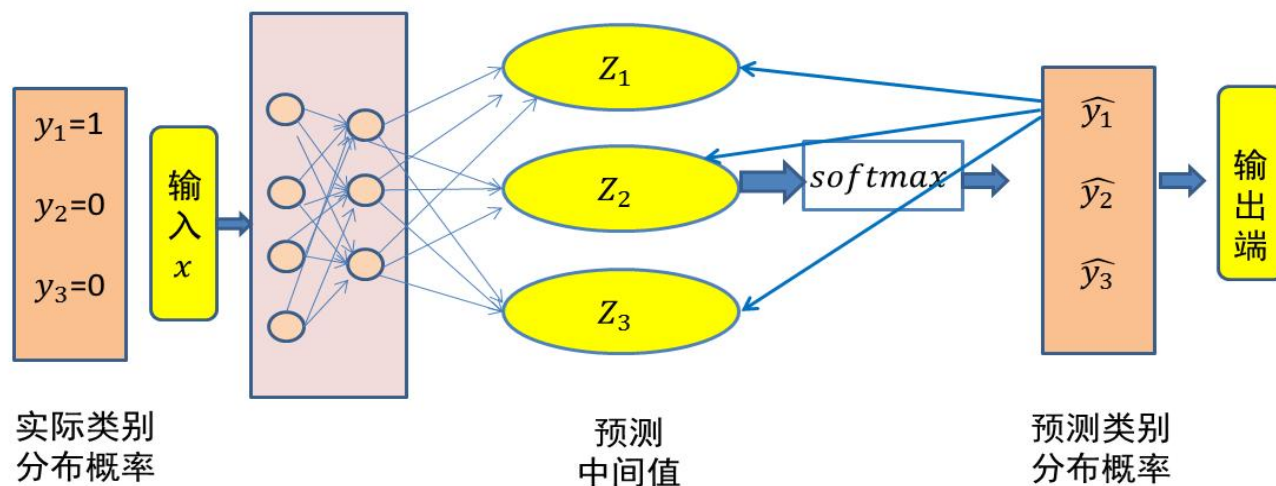
$$\frac{\partial \hat{y}_1}{\partial z_1}$$



常用的损失函数--交叉熵损失函数

- 据交叉熵计算的误差从 \hat{y}_1 传递给 z_1

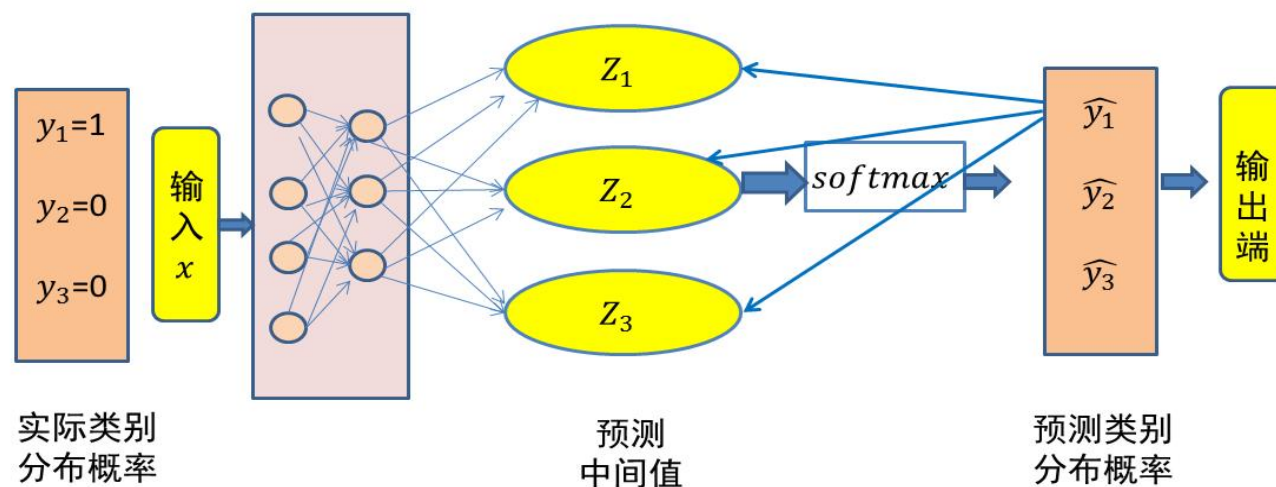
$$\begin{aligned}\frac{\partial \hat{y}_1}{\partial z_1} &= \frac{\partial}{\partial z_1} \left(\frac{e^{z_1}}{\sum_k e^{z_k}} \right) = \frac{(e^{z_1})' \times \sum_k e^{z_k} - e^{z_1} \times e^{z_1}}{(\sum_k e^{z_k})^2} \\ &= \frac{e^{z_1}}{\sum_k e^{z_k}} - \frac{e^{z_1}}{\sum_k e^{z_k}} \times \frac{e^{z_1}}{\sum_k e^{z_k}} = \hat{y}_1(1 - \hat{y}_1)\end{aligned}$$



常用的损失函数--交叉熵损失函数

- 据交叉熵计算的误差从 \hat{y}_1 传递给 z_1

$$\frac{\partial \hat{y}_1}{\partial z_1} = \hat{y}_1(1 - \hat{y}_1)$$



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

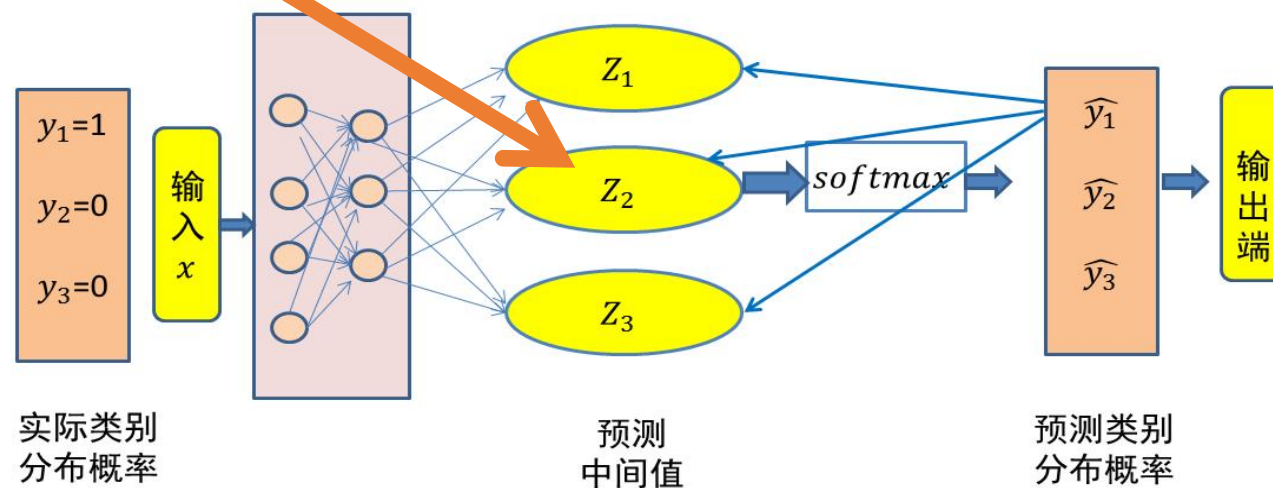
$$\begin{aligned}\frac{\partial \widehat{y}_1}{\partial z_1} &= \frac{\partial}{\partial z_1} \left(\frac{e^{z_1}}{\sum_k e^{z_k}} \right) = \frac{(e^{z_1})' \times \sum_k e^{z_k} - e^{z_1} \times e^{z_1}}{(\sum_k e^{z_k})^2} \\ &= \frac{e^{z_1}}{\sum_k e^{z_k}} - \frac{e^{z_1}}{\sum_k e^{z_k}} \times \frac{e^{z_1}}{\sum_k e^{z_k}} = \widehat{y}_1(1 - \widehat{y}_1)\end{aligned}$$

- 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 \hat{y}_1 传递给 z_2 (z_3 的推导同 z_2)

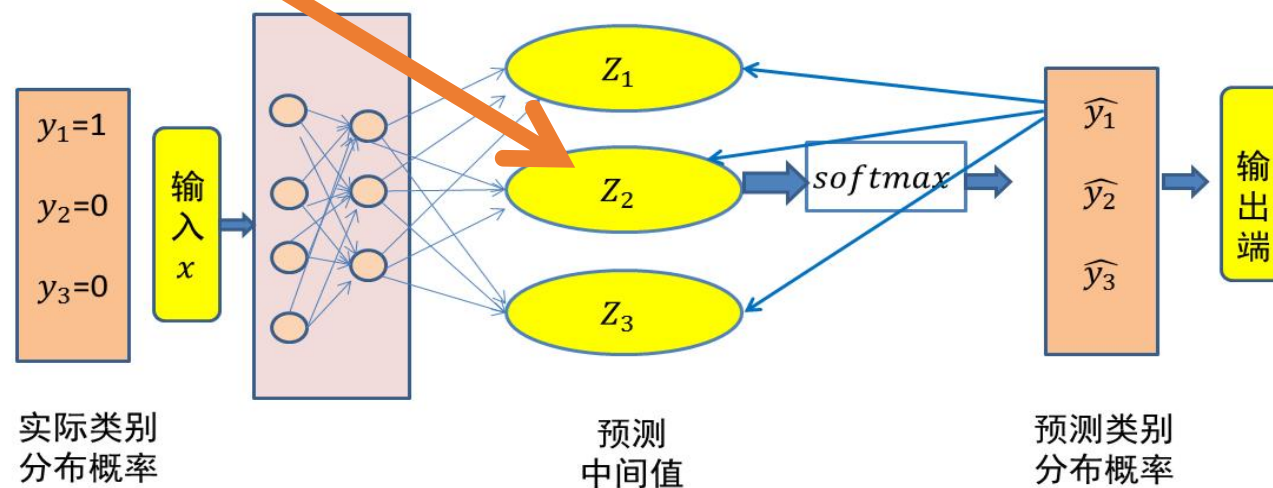
$$\frac{\partial \hat{y}_1}{\partial z_2} :$$



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)

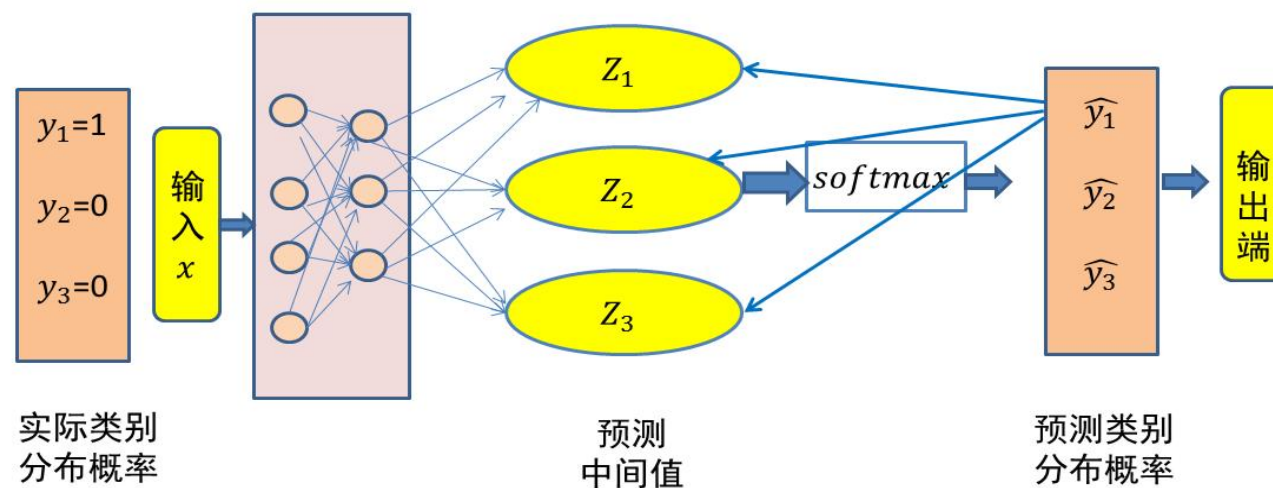
$$\frac{\partial \widehat{y}_1}{\partial z_2} = \frac{\partial}{\partial z_2} \left(\frac{e^{z_1}}{\sum_k e^{z_k}} \right) = \frac{0 \times \sum_k e^{z_k} - e^{z_1} \times e^{z_2}}{(\sum_k e^{z_k})^2} = - \frac{e^{z_1}}{\sum_k e^{z_k}} \times \frac{e^{z_2}}{\sum_k e^{z_k}} = - \widehat{y}_1 \widehat{y}_2$$



常用的损失函数--交叉熵损失函数

- 据交叉熵计算的误差从 \hat{y}_1 传递给 z_2 (z_3 的推导同 z_2)

$$\frac{\partial \hat{y}_1}{\partial z_2} = \leftarrow \text{orange arrow} \rightarrow = -\hat{y}_1 \hat{y}_2$$



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)

$$\frac{\partial \widehat{y}_1}{\partial z_2} = \frac{\partial}{\partial z_2} \left(\frac{e^{z_1}}{\sum_k e^{z_k}} \right) = \frac{0 \times \sum_k e^{z_k} - e^{z_1} \times e^{z_2}}{(\sum_k e^{z_k})^2} = - \frac{e^{z_1}}{\sum_k e^{z_k}} \times \frac{e^{z_2}}{\sum_k e^{z_k}} = - \widehat{y}_1 \widehat{y}_2$$

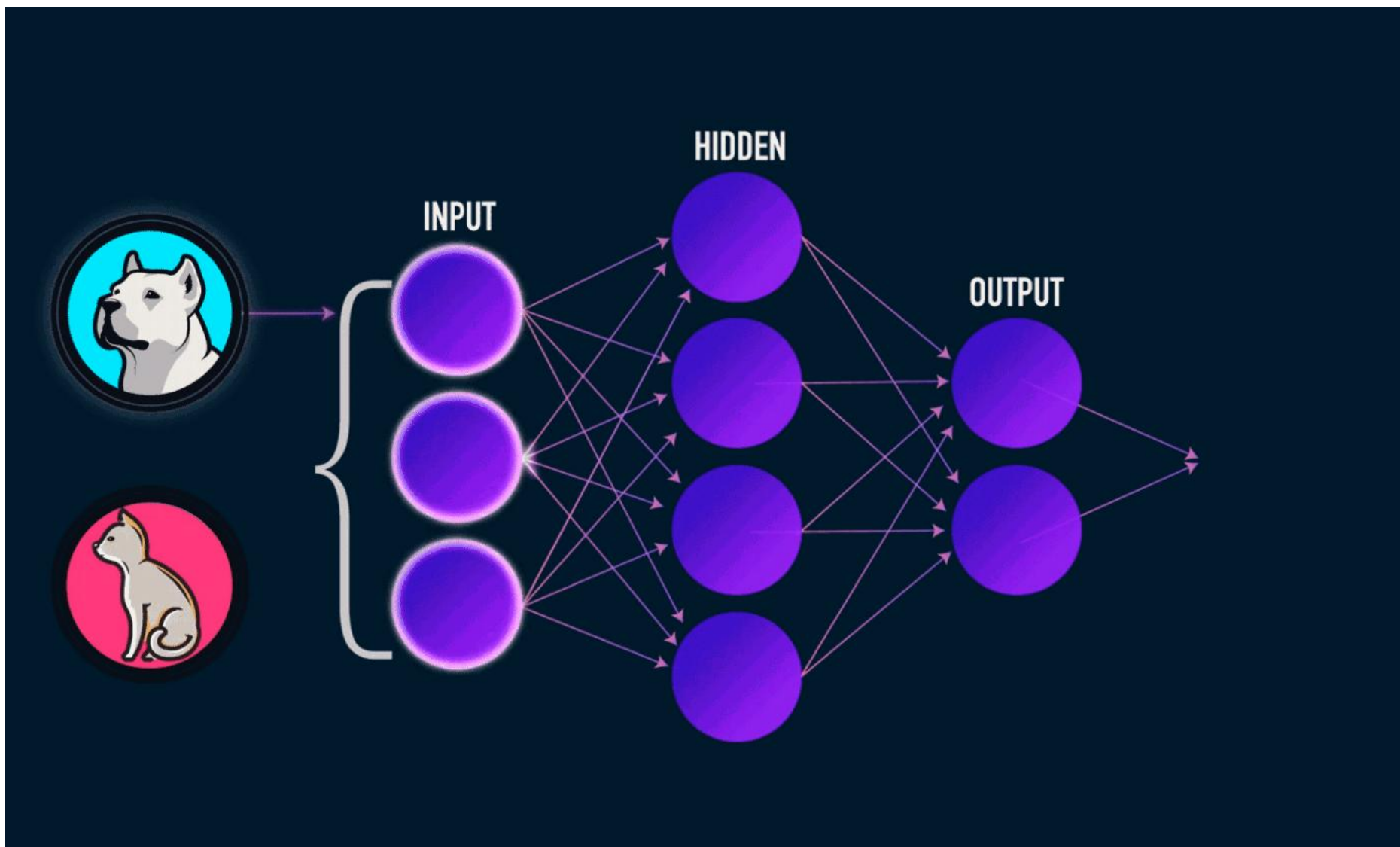
- 注意：在推导中应用了 $(e^x)' = e^x$ 、 $(\frac{v}{u})' = \frac{v'u - vu'}{u^2}$ 等公式
- 交叉熵损失函数导数为 $-\frac{1}{\widehat{y}_1}$ ，相乘结果为 \widehat{y}_2 。这意味对于除第一个输出节点以外的节点进行偏导，模型预测输出就是交叉熵损失函数相对于其他节点的偏导。**在 z_1 、 z_2 和 z_3 得到偏导结果后**，再通过**链式法则**（后续介绍）将损失误差继续往输入端传递

常用的损失函数

- 交叉熵损失函数

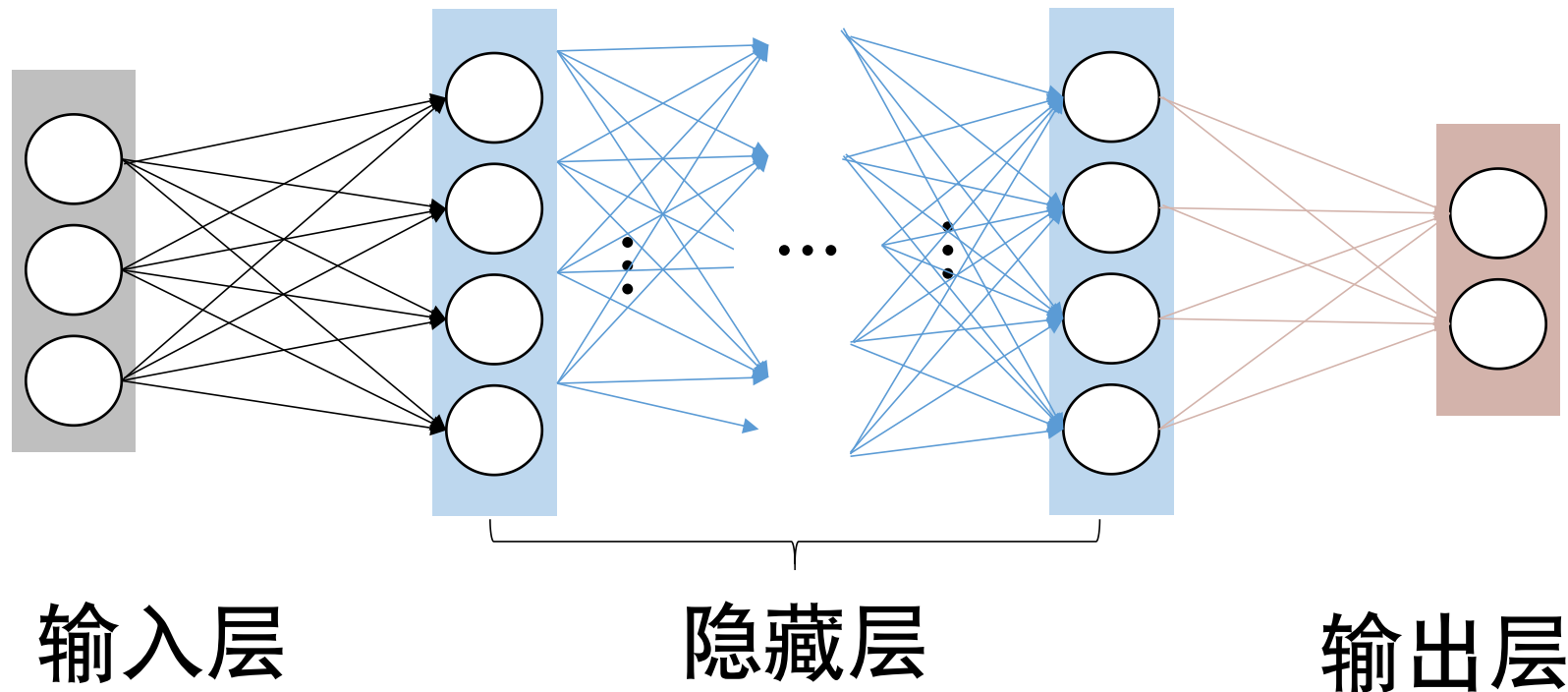
- 上面的例子中，假设预测中间值 (z_1, z_2, z_3) 经过Softmax映射后为 $(0.34, 0.46, 0.20)$ 。由于输入数据 x 属于第一类，显然这个输出不理想而需要对模型参数进行优化。如果选择交叉熵损失函数来优化模型，则 (z_1, z_2, z_3) 这一层的偏导为 $(0.34 - 1, 0.46, 0.20)$ 。

前馈神经网络 (feedforward neural network)



前馈神经网络 (feedforward neural network)

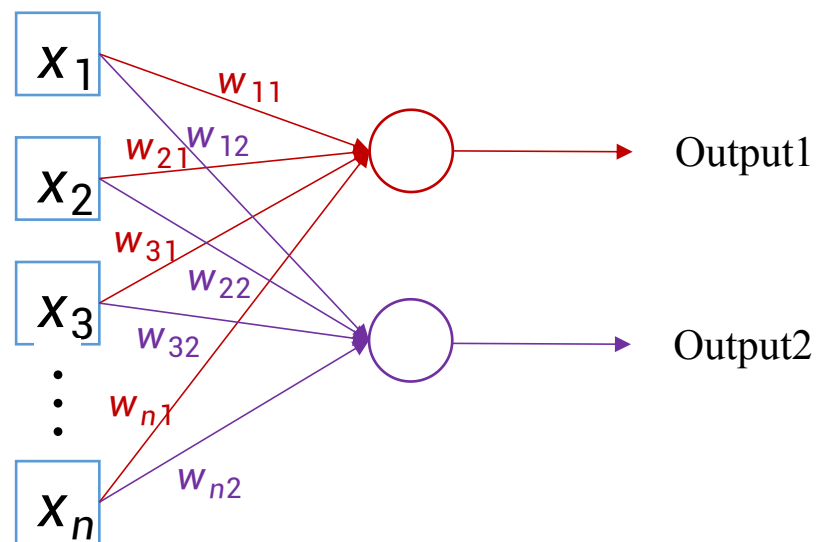
- 各神经元接受前一级输入并输出到下一级，模型中没有反馈层间“全连接”，即两个相邻层之间的神经元完全成对连接，但层内的神经元不相互连接。



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

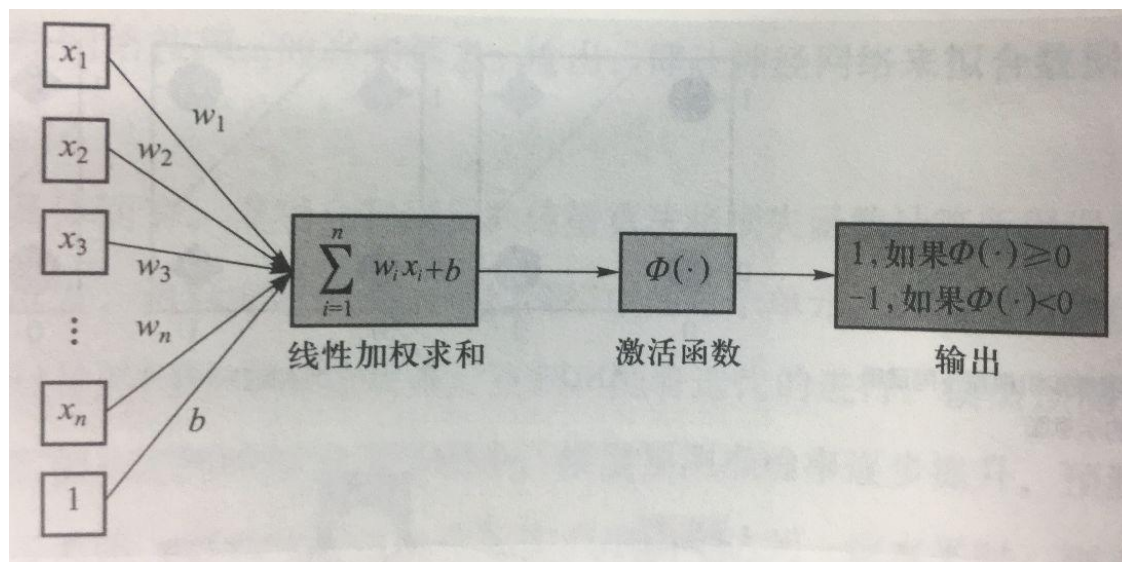
- 无隐藏层，只有输入层/输出层
- 无法拟合复杂的数据



$w_{ij} (1 \leq i \leq n, 1 \leq j \leq 2)$ 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 (**bias**) 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。 W 参数可学习。

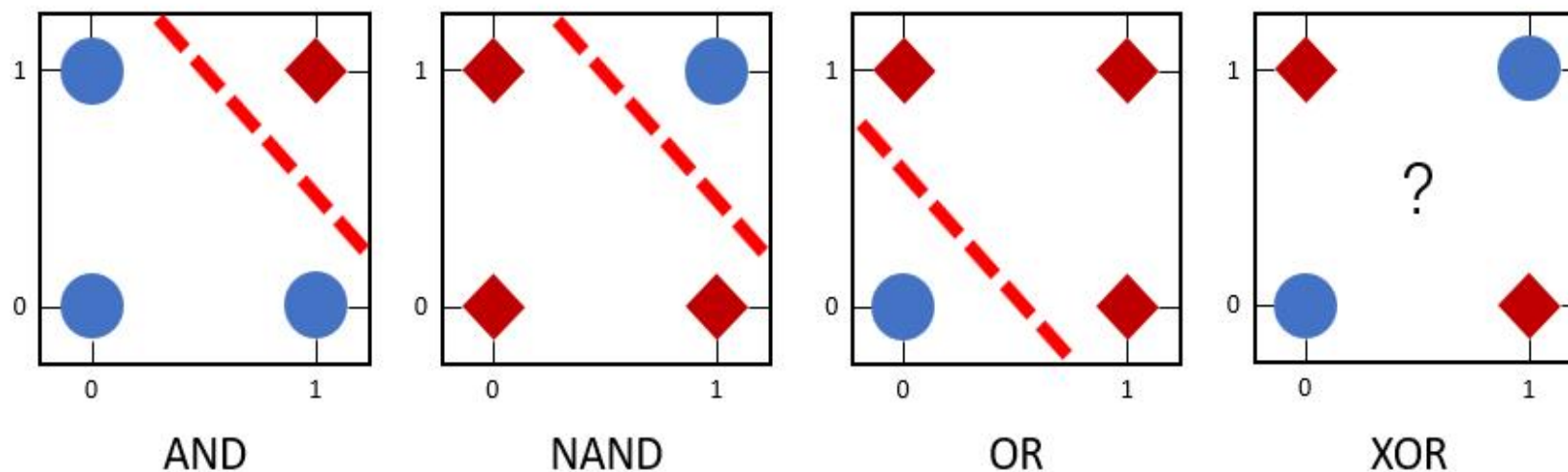


感知机模型

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

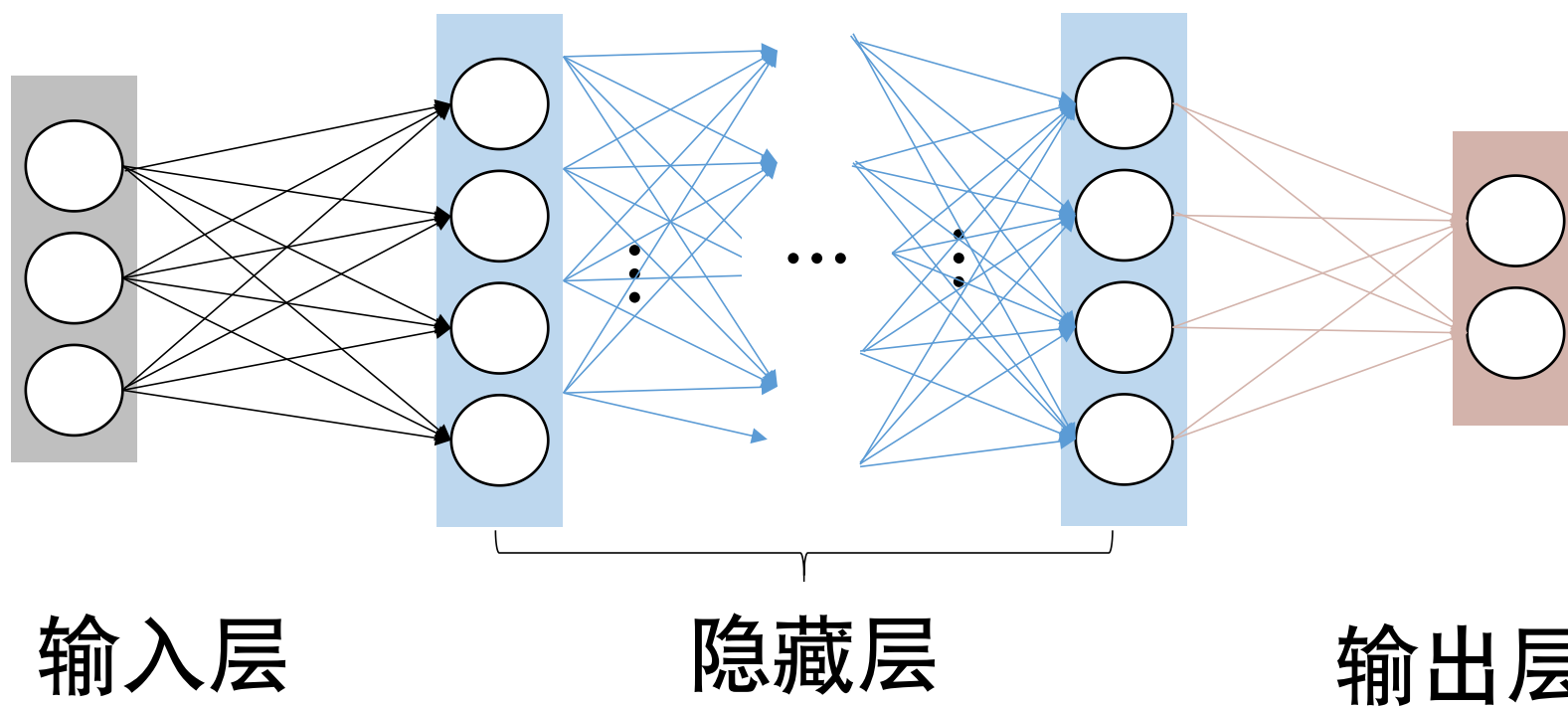
- 图中，逻辑与(AND)、逻辑与非(NAND)和逻辑或(OR)为线性可分函数，所以可利用单层感知机来模拟这些逻辑函数。
- 逻辑异或 (XOR) 是非线性可分的，因此单层感知机无法模拟



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

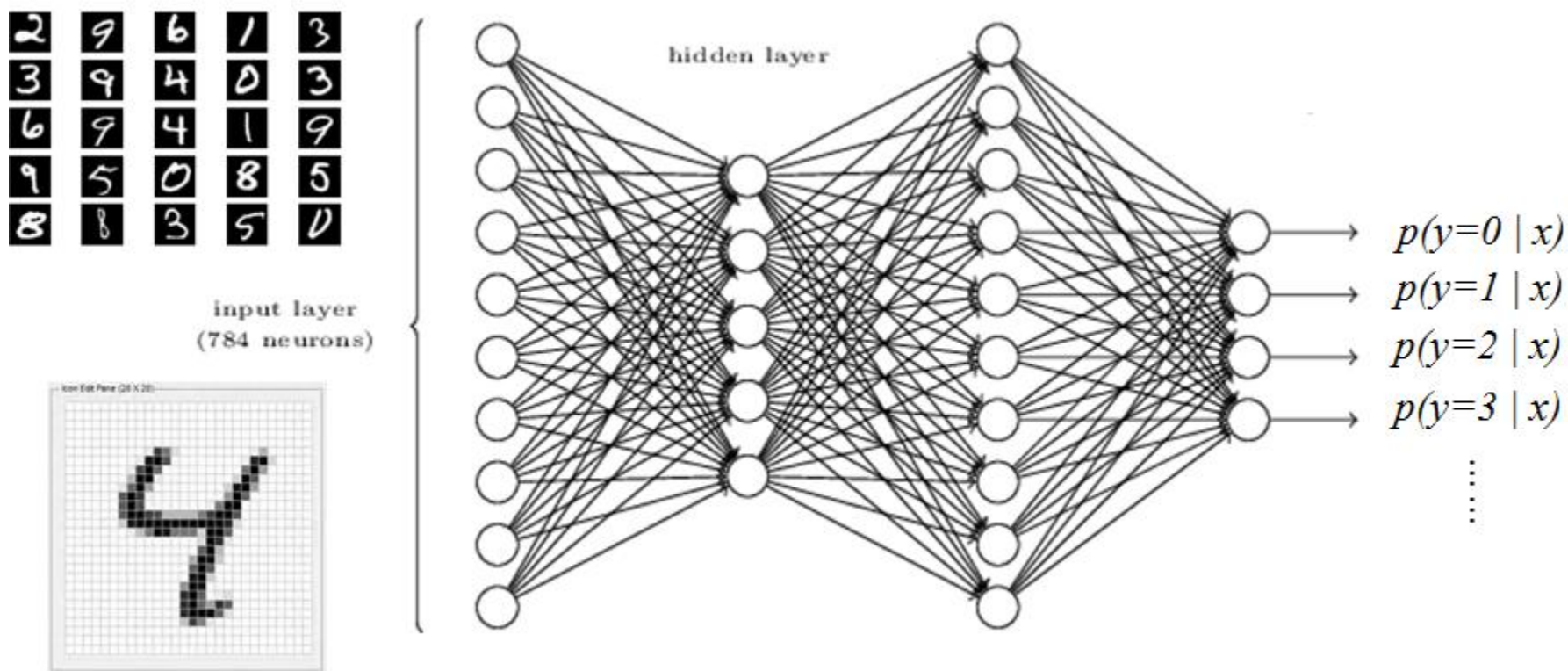
感知机模型： 多层感知机

- **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？

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

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

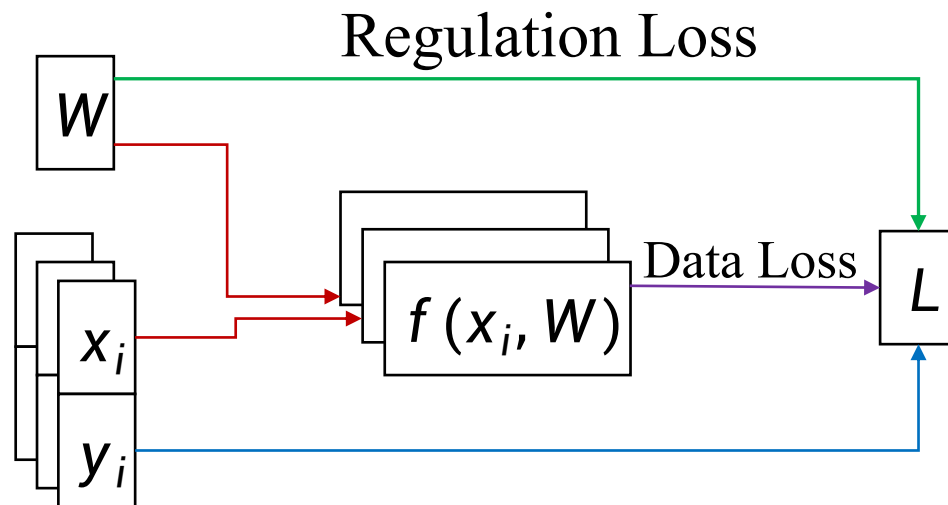
- 损失函数来估量模型预测值与真实值之间的差距

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right) \quad \leftarrow \begin{array}{l} \text{分类正确样本的} \\ \text{Softmax值所占比重越} \\ \text{大，其Loss越小} \end{array}$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \quad \leftarrow \begin{array}{l} \text{分类正确样本置信} \\ \text{度比分类错误样本} \\ \text{置信度至少大1} \end{array}$$

问题：如何优化网络参数？

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



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 f 在 x 处梯度为：

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



- 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
- 批量梯度下降、随机梯度下降、**小批量梯度下降**

问题：如何优化网络参数？

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

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

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

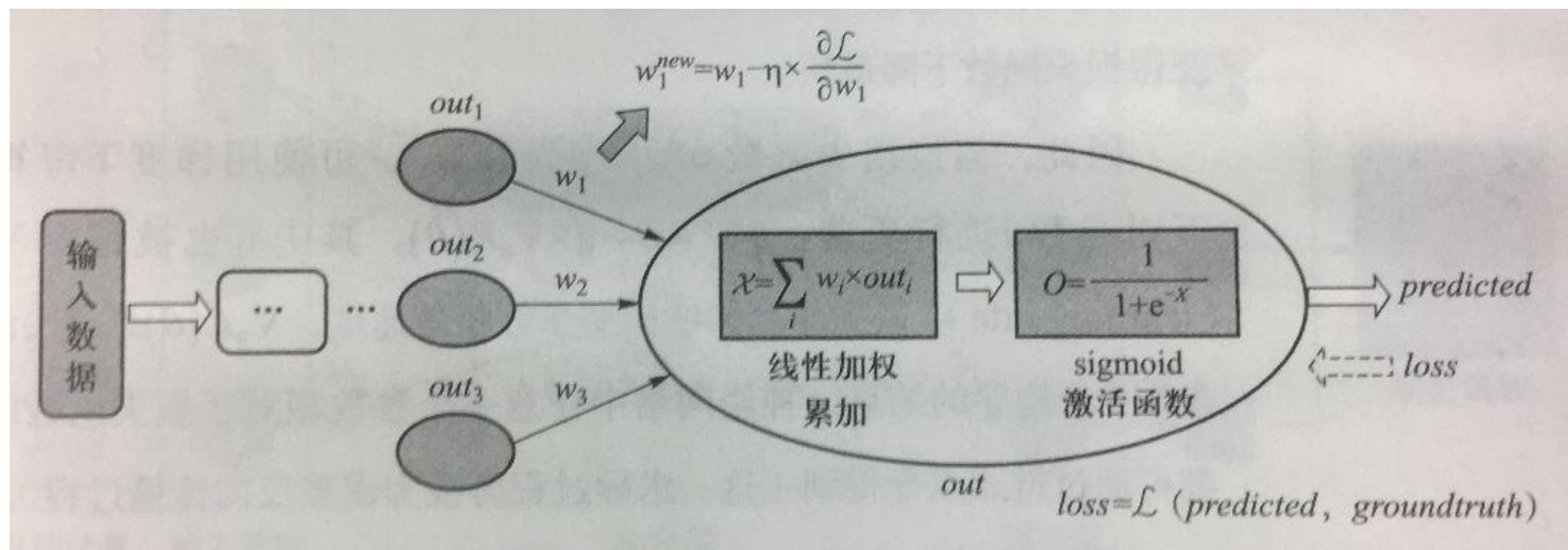
- 因为 $(\nabla f(\mathbf{x}))^T \Delta \mathbf{x} = \|\nabla f(\mathbf{x})\| \|\Delta \mathbf{x}\| \cos \theta$, 为了最小化损失函数, 可以沿着梯度的反方向 (i.e., $\theta = \pi$) 来寻找参数取值

$$f(\mathbf{x} + \Delta \mathbf{x}) - f(\mathbf{x}) = \|\nabla f(\mathbf{x})\| \|\Delta \mathbf{x}\| \cos \theta = - \underbrace{\alpha}_{\text{步长}} \underbrace{\|\nabla f(\mathbf{x})\|}_{\text{函数偏导}}$$

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

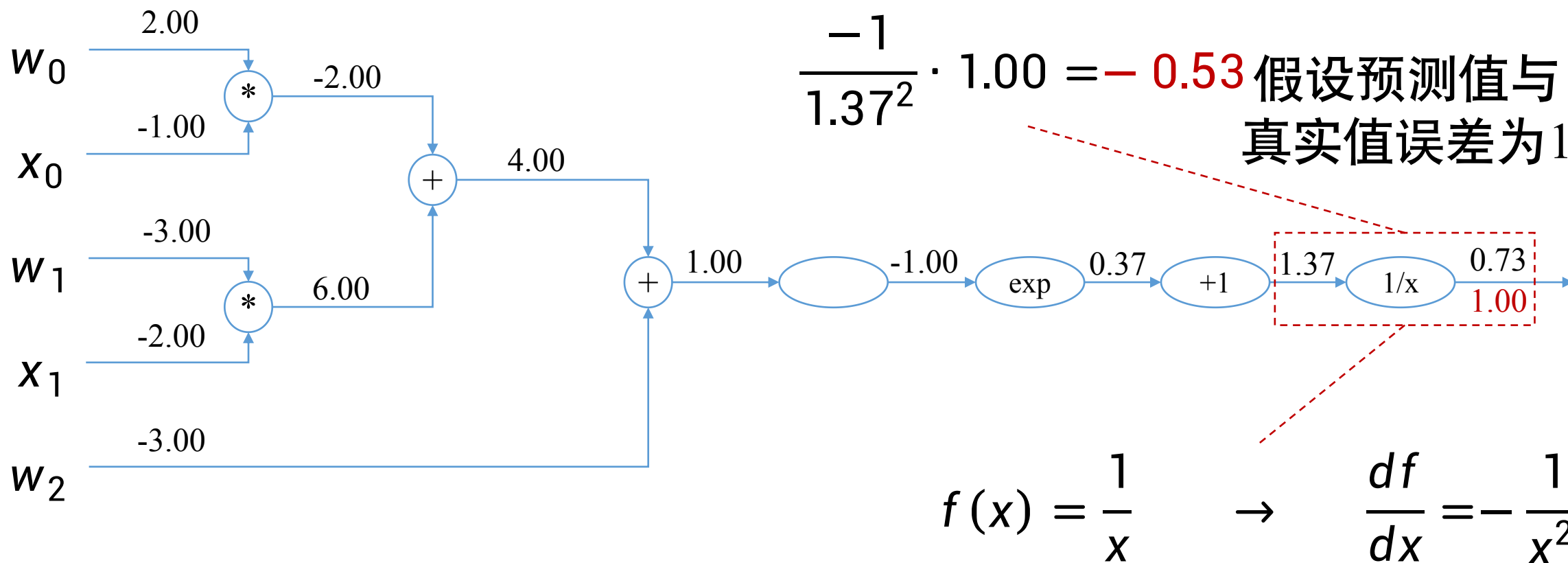
- 将误差从后向前传递，将误差分摊给各层所有单元，从而获得各层单元所产生的误差，进而依据这个误差来让各层单元负起各自责任、修正各单元参数。



链式求导与模型参数更新示意图

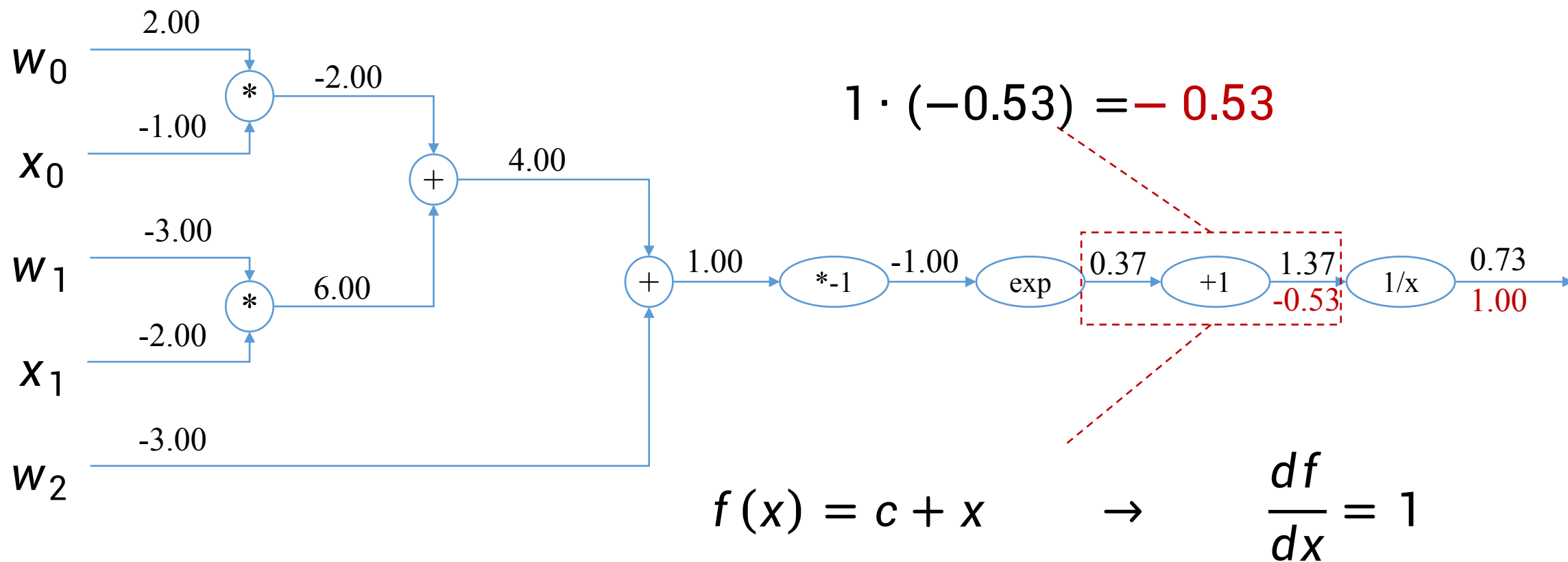
误差反向传播例子

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2x_2)}}$$



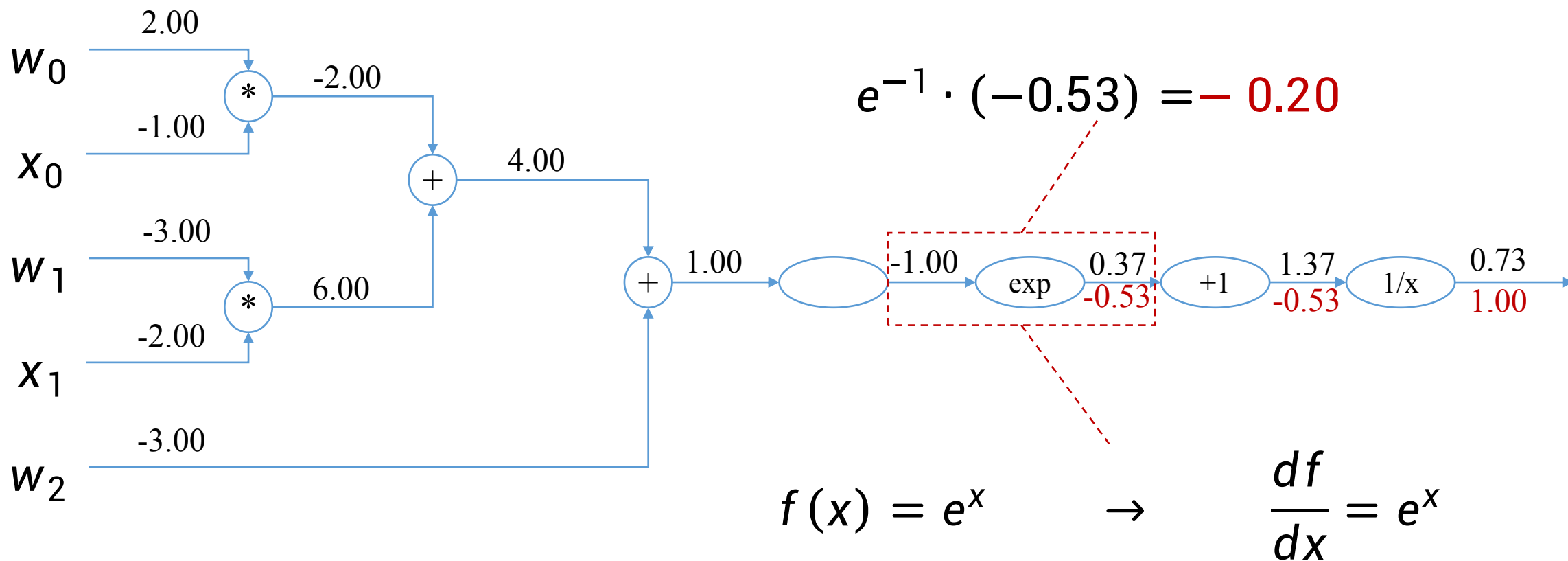
误差反向传播例子

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2x_2)}}$$



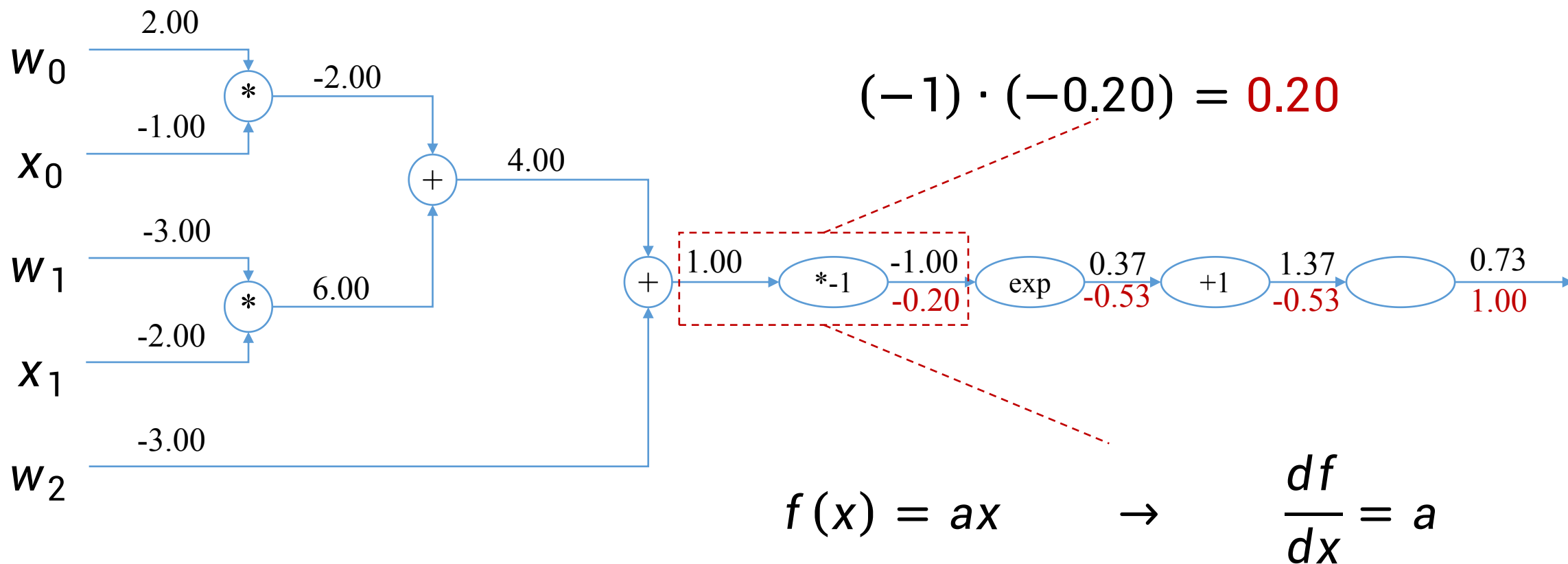
误差反向传播例子

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2x_2)}}$$



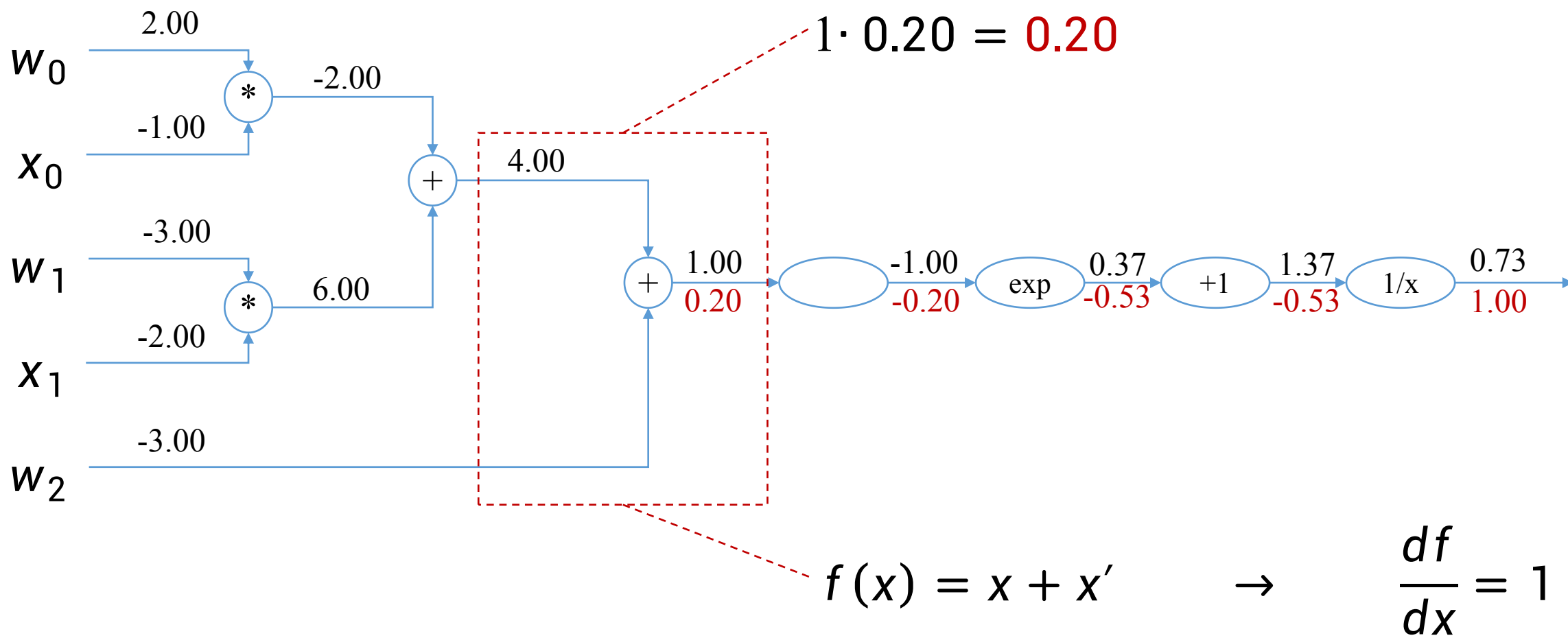
误差反向传播例子

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2x_2)}}$$



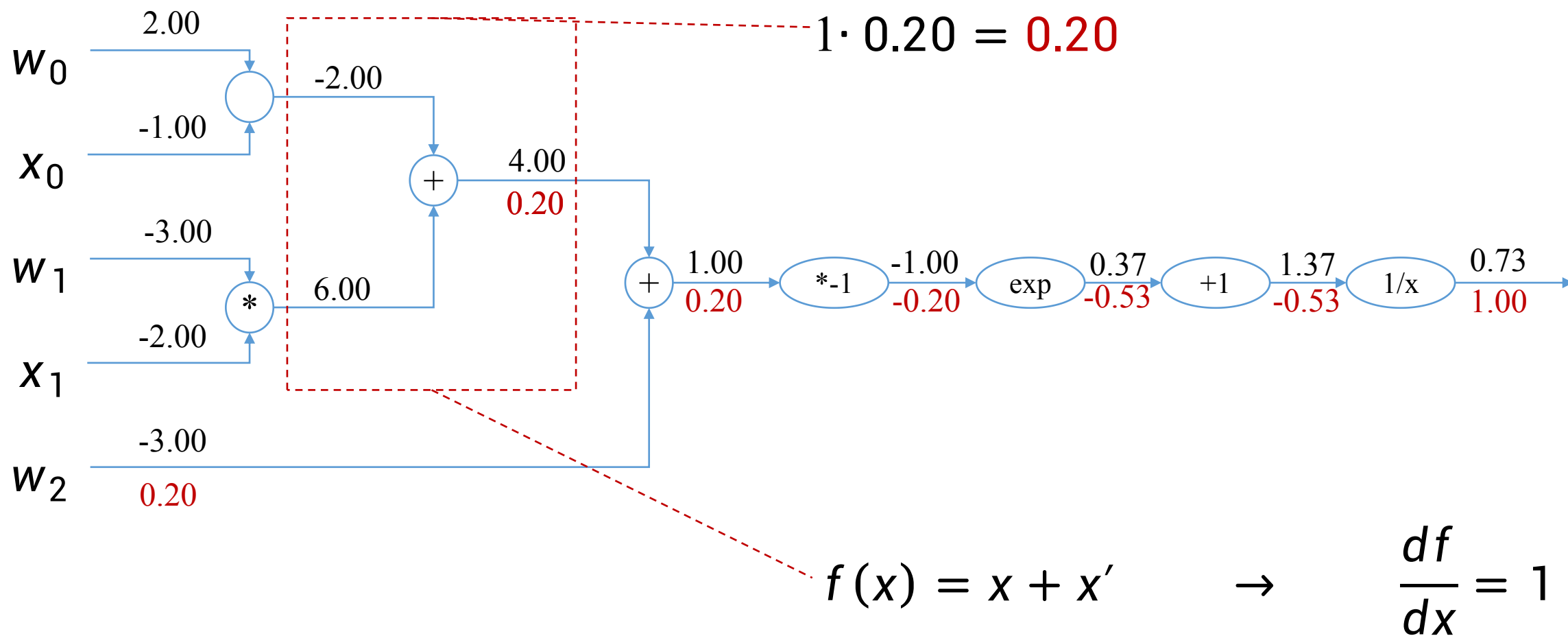
误差反向传播例子

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2x_2)}}$$



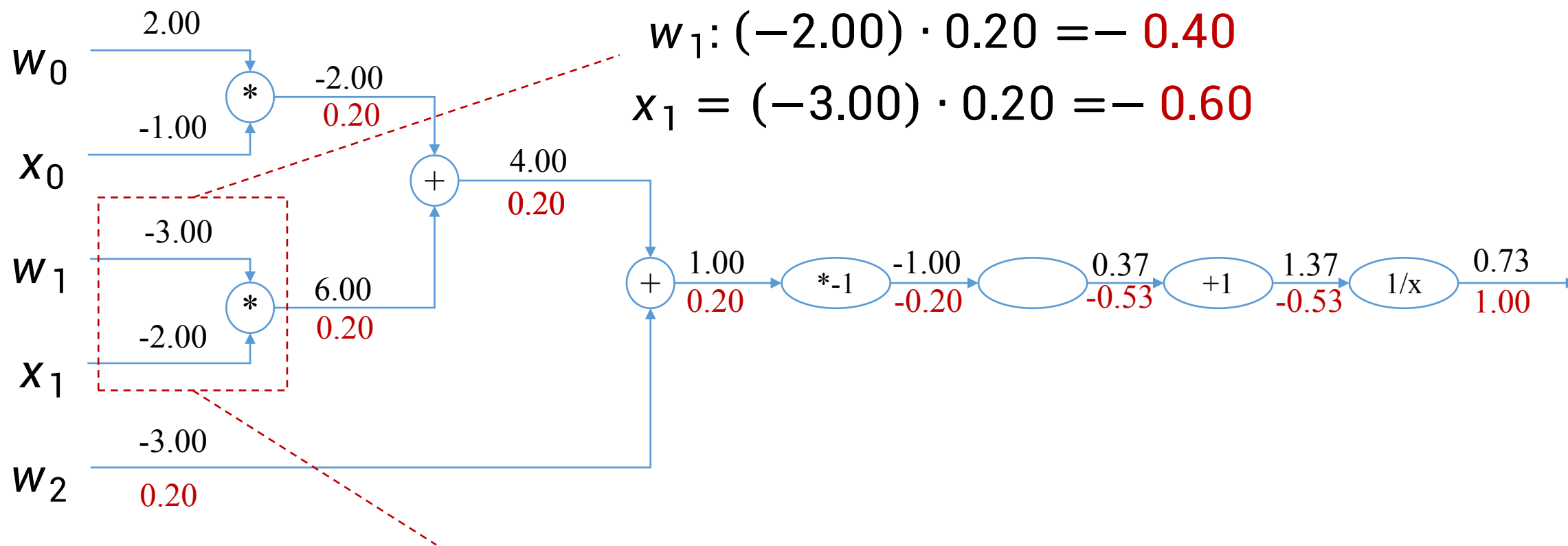
误差反向传播例子

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



误差反向传播例子

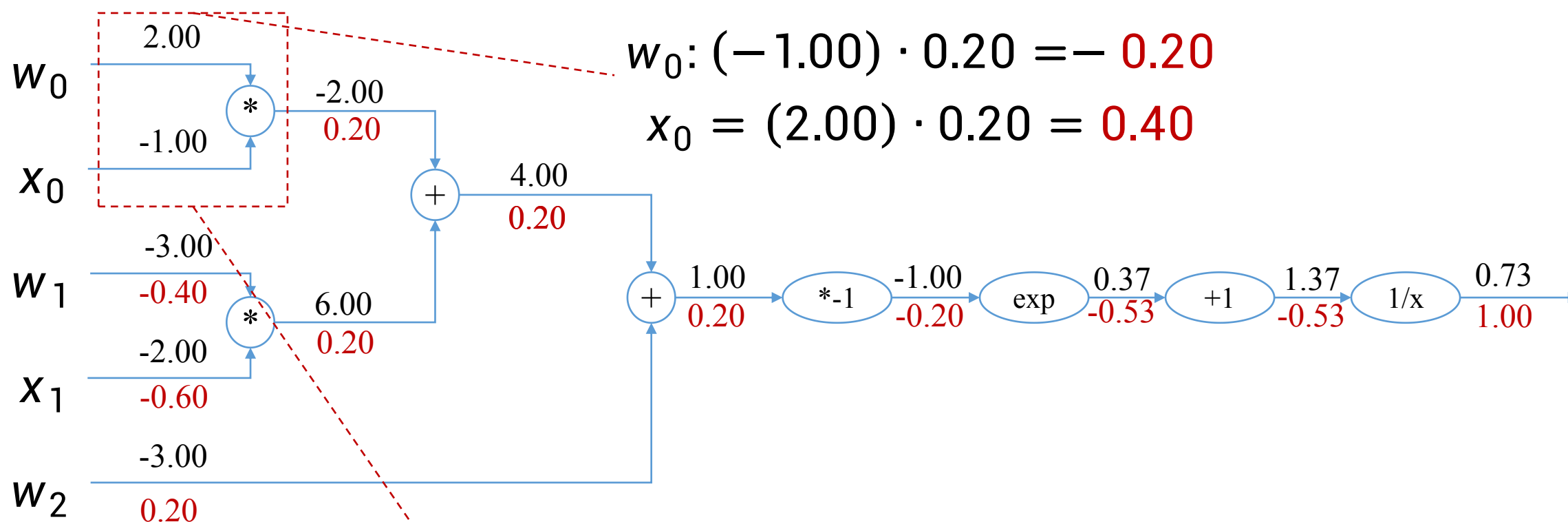
$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2x_2)}}$$



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

误差反向传播例子

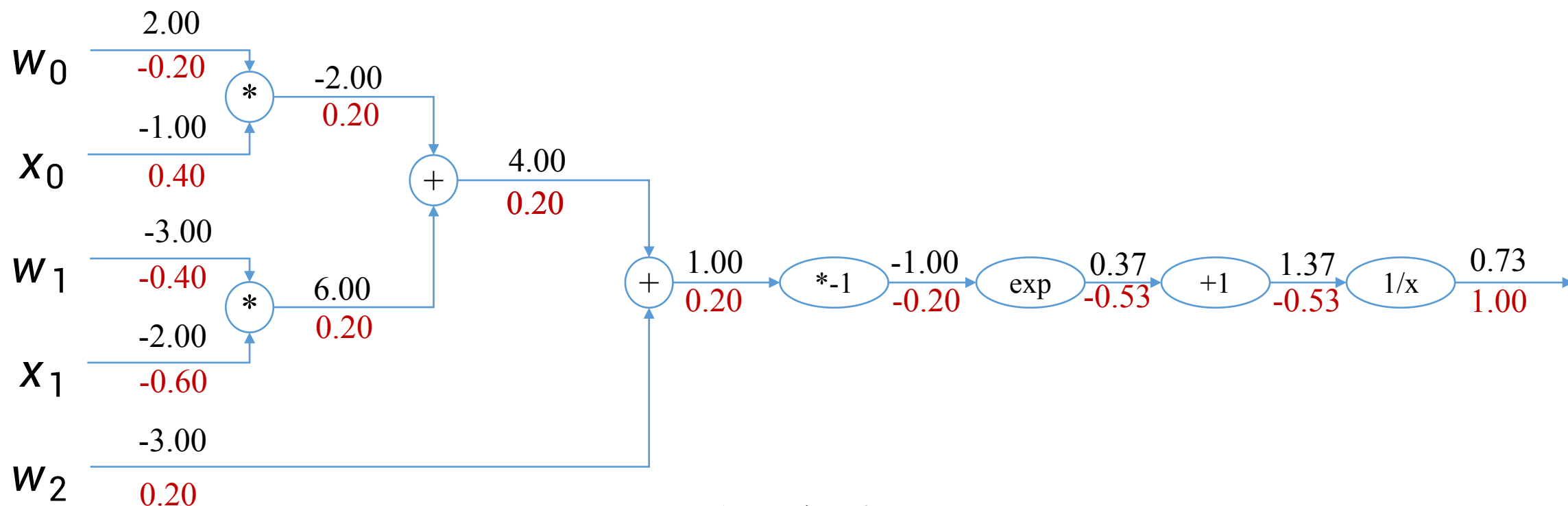
$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2x_2)}}$$



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

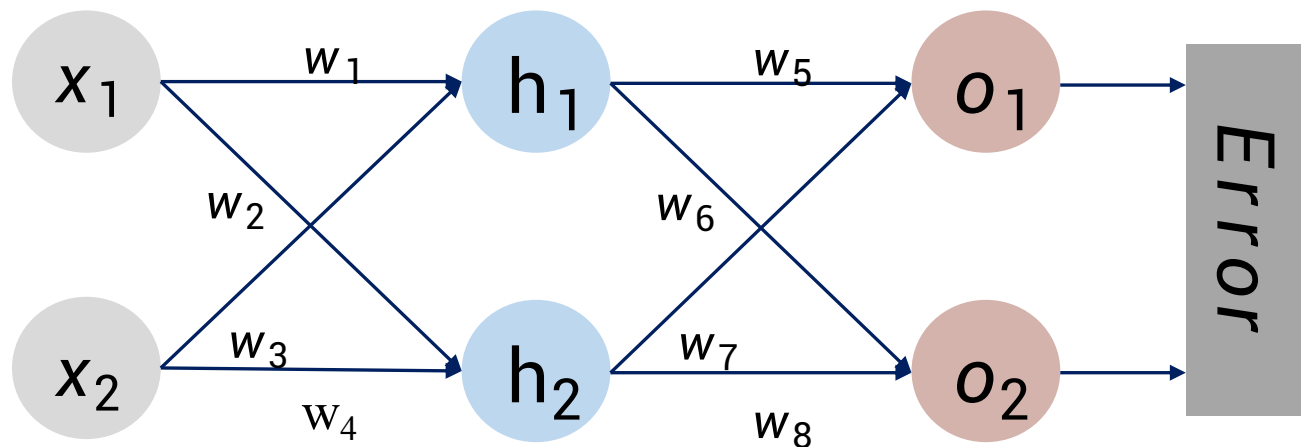
误差反向传播例子

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

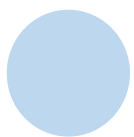


误差反向传播

误差反向传播：前馈神经网络



Input Layer



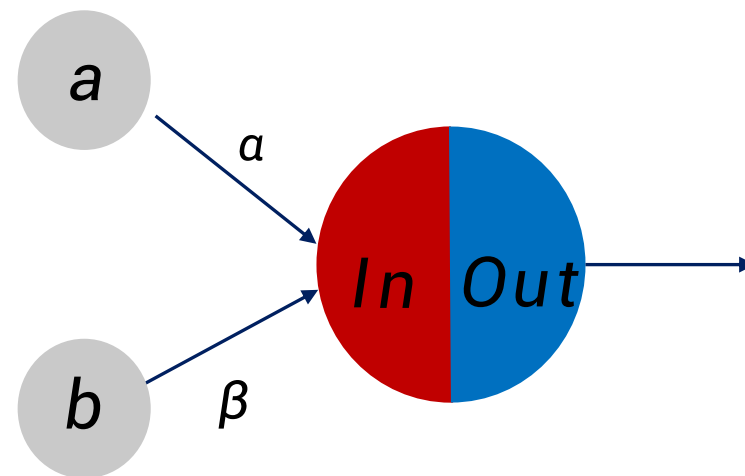
Hidden Layer



Output Layer



Weight

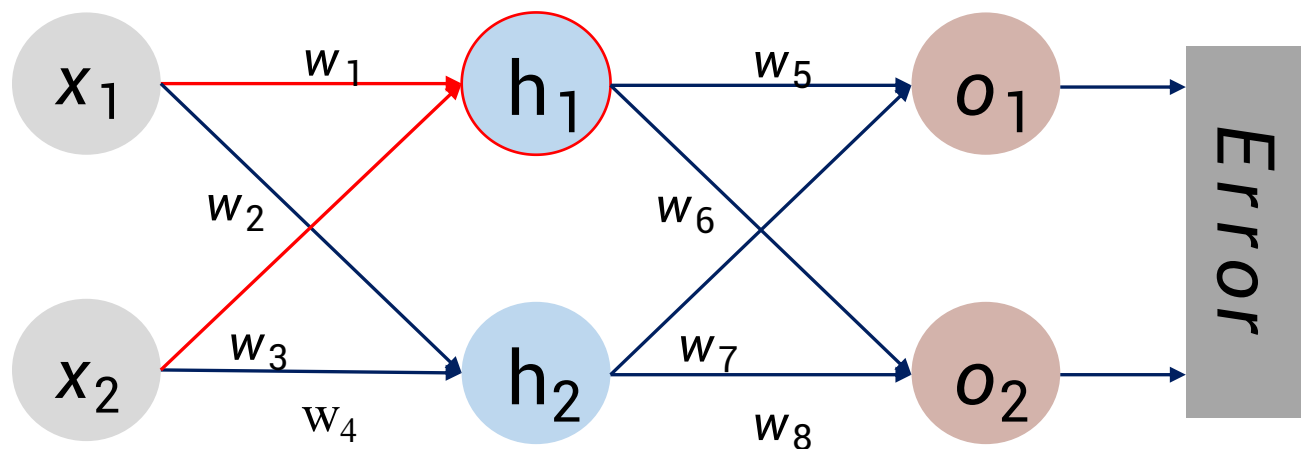


$$In = a * a + \beta * b$$

$$Out = Sigmoid(In)$$

误差反向传播：前馈神经网络

• 前向传播

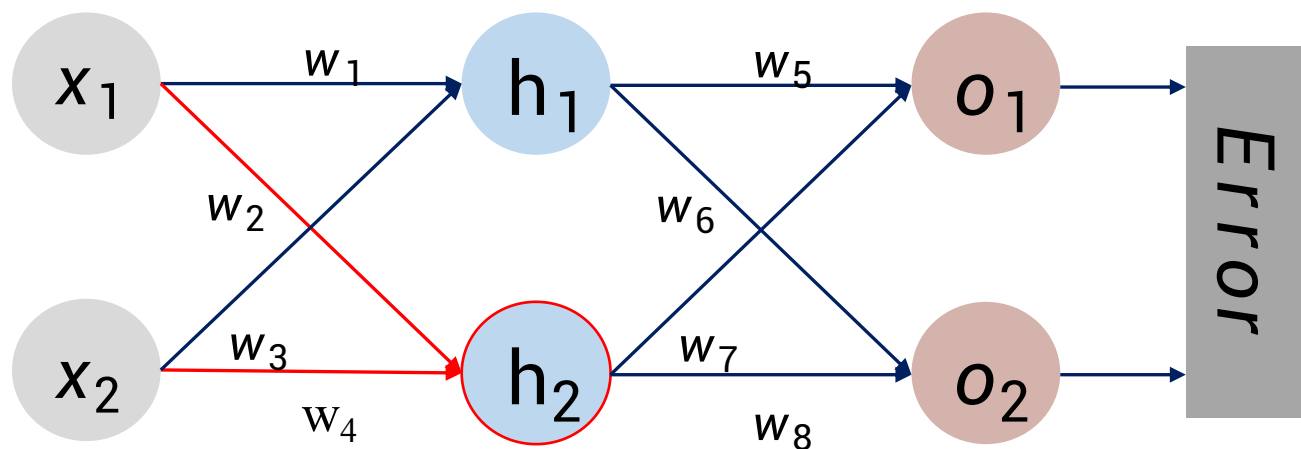


$$In_{h_1} = w_1 * x_1 + w_3 * x_2$$

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

误差反向传播：前馈神经网络

• 前向传播

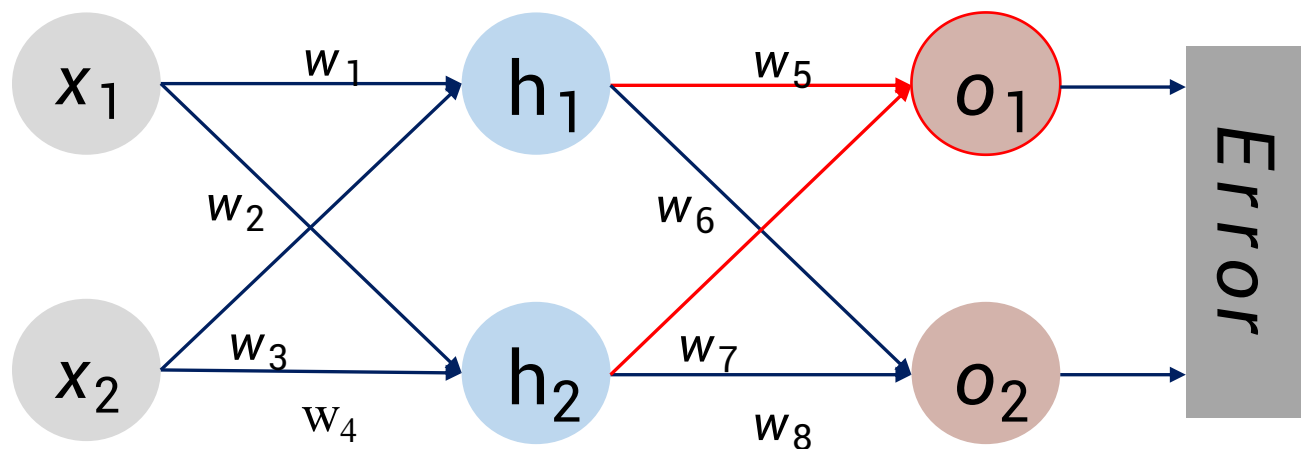


$$In_{h_2} = w_2 * x_1 + w_4 * x_2$$

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

误差反向传播：前馈神经网络

• 前向传播

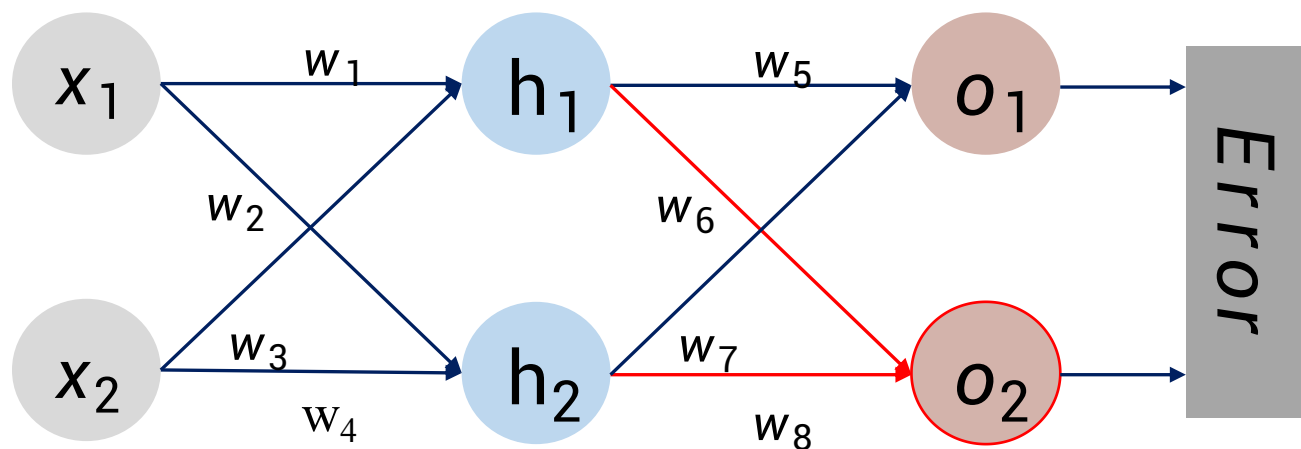


$$In_{o_1} = w_5 * h_1 + w_7 * h_2$$

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

误差反向传播：前馈神经网络

• 前向传播

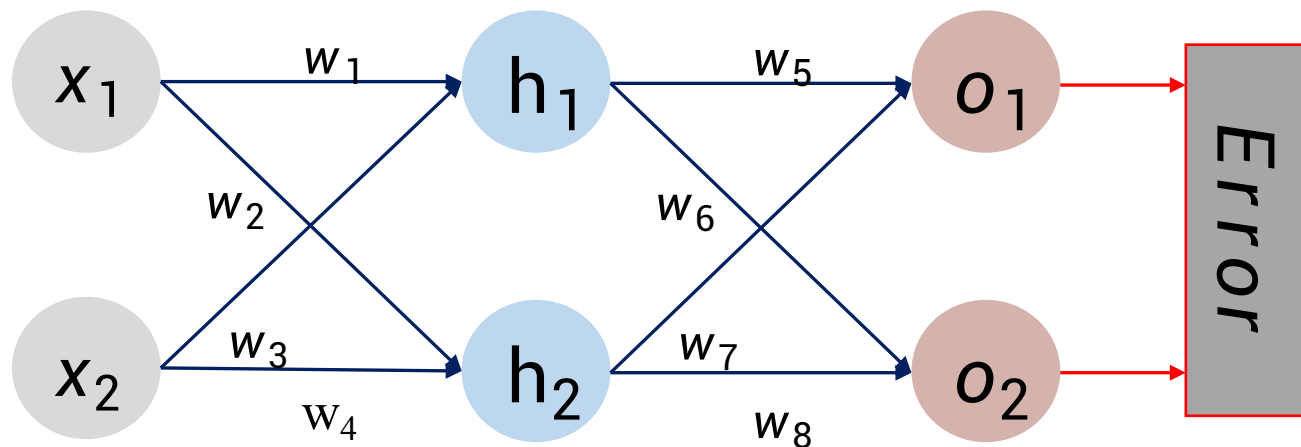


$$In_{o_2} = w_6 * h_1 + w_8 * h_2$$

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

误差反向传播：前馈神经网络

• 前向传播



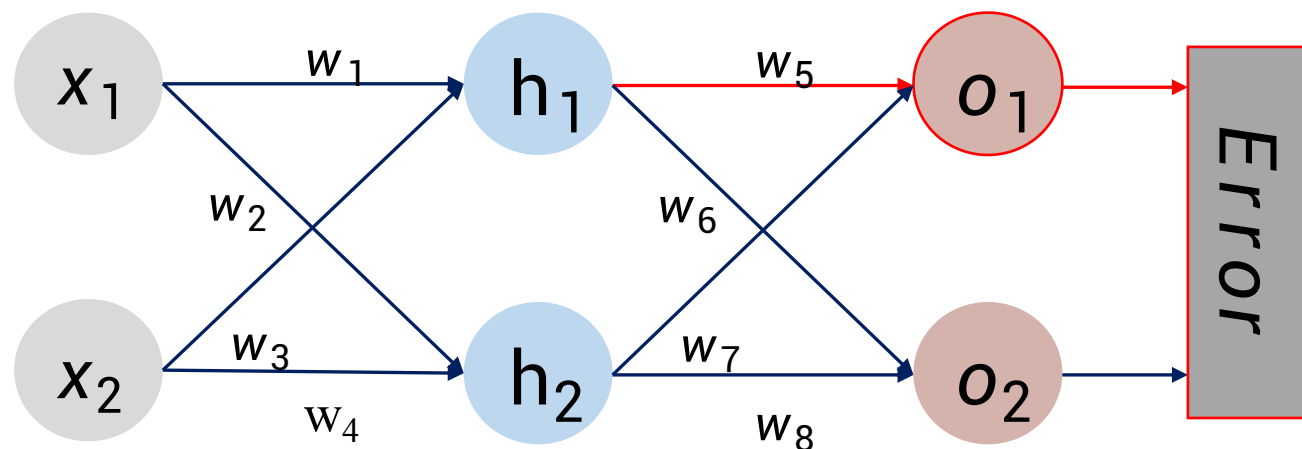
$$Error = \frac{1}{2} \sum_{i=1}^2 (o_i - y_i)^2$$

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)



$$\bar{\delta}_5 = \frac{\partial Error}{\partial w_5} = \frac{\partial Error}{\partial o_1} * \frac{\partial o_1}{\partial \ln_{o_1}} * \frac{\partial \ln_{o_1}}{\partial w_5}$$

where,

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

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

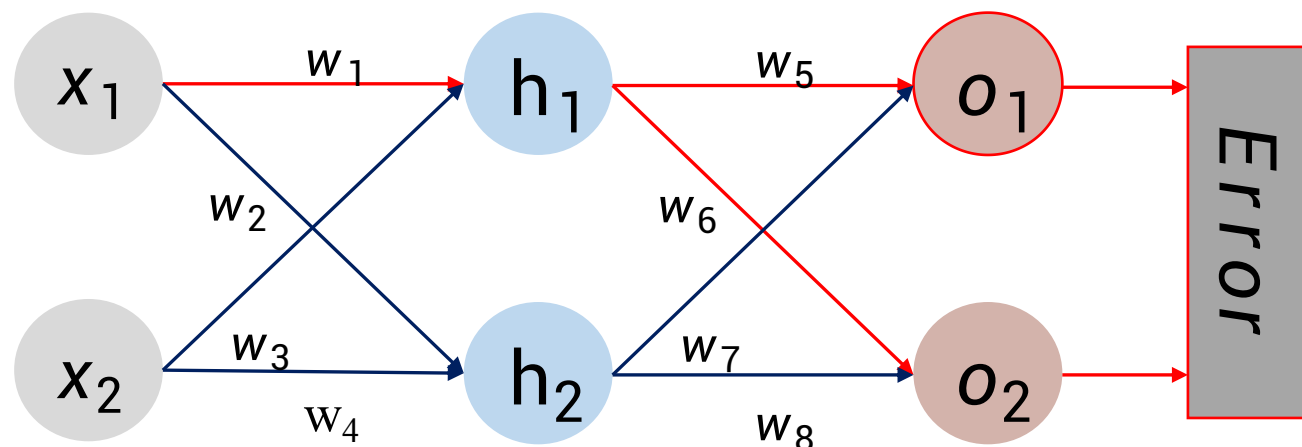
$$\frac{\partial \ln_{o_1}}{\partial w_5} = h_1 \quad \leftarrow \quad \ln_{o_1} = w_5 * h_1 + w_7 * h_2$$

误差反向传播：前馈神经网络

- 反向传播：梯度计算

- 计算 $w_1 \sim w_4$ 的梯度

(以 w_1 为例)



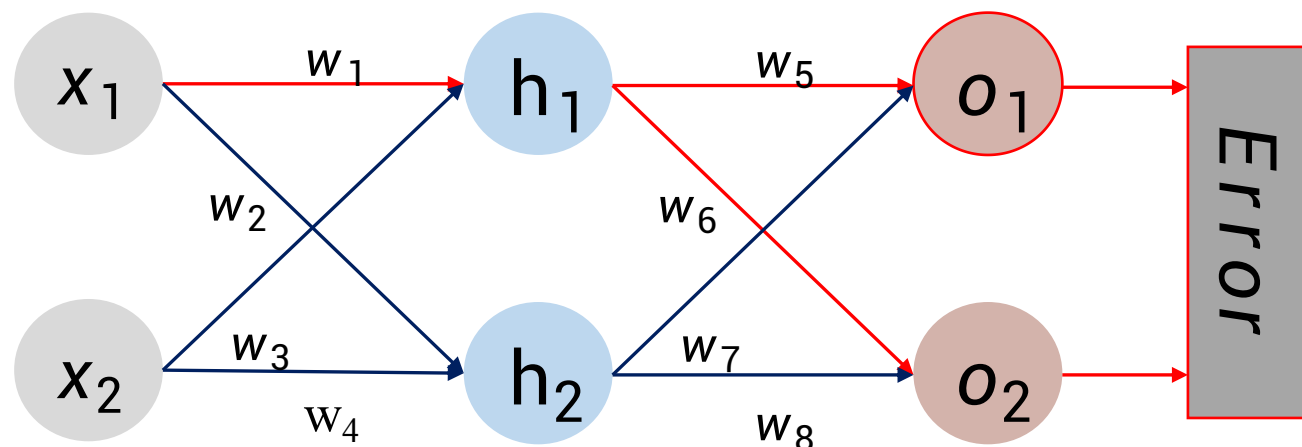
$$\begin{aligned}\delta_1 &= \frac{\partial \text{Error}}{\partial w_1} = \frac{\partial \text{Error}}{\partial o_1} * \frac{\partial o_1}{\partial w_1} + \frac{\partial \text{Error}}{\partial o_2} * \frac{\partial o_2}{\partial w_1} \\&= \frac{\partial \text{Error}}{\partial o_1} * \frac{\partial o_1}{\partial \ln o_1} * \frac{\partial \ln o_1}{\partial h_1} * \frac{\partial h_1}{\partial \ln h_1} * \frac{\partial \ln h_1}{\partial w_1} + \frac{\partial \text{Error}}{\partial o_2} * \frac{\partial o_2}{\partial \ln o_2} * \frac{\partial \ln o_2}{\partial h_1} * \frac{\partial h_1}{\partial \ln h_1} * \frac{\partial \ln h_1}{\partial w_1} \\&= \left(\frac{\partial \text{Error}}{\partial o_1} * \frac{\partial o_1}{\partial \ln o_1} * \frac{\partial \ln o_1}{\partial h_1} + \frac{\partial \text{Error}}{\partial o_2} * \frac{\partial o_2}{\partial \ln o_2} * \frac{\partial \ln o_2}{\partial h_1} \right) * \frac{\partial h_1}{\partial \ln h_1} * \frac{\partial \ln h_1}{\partial w_1} \\&= \left(\frac{\partial \text{Error}}{\partial o_1} * \frac{\partial o_1}{\partial \ln o_1} * w_5 + \frac{\partial \text{Error}}{\partial o_2} * \frac{\partial o_2}{\partial \ln o_2} * w_6 \right) * \frac{\partial h_1}{\partial \ln h_1} * \frac{\partial \ln h_1}{\partial w_1}\end{aligned}$$

误差反向传播：前馈神经网络

- 反向传播：梯度计算

- 计算 $w_1 \sim w_4$ 的梯度

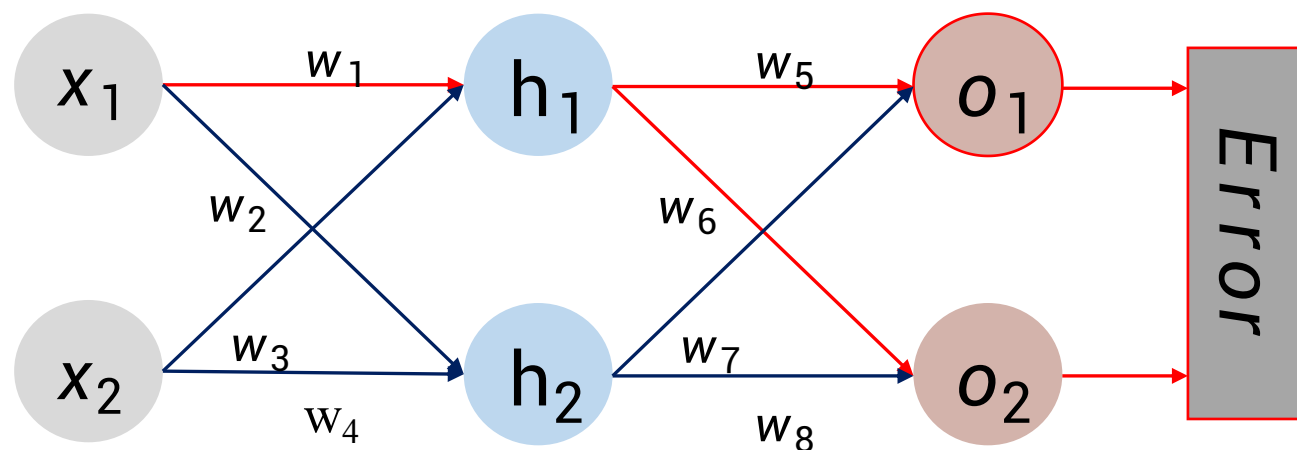
(以 w_2 为例)



$$\begin{aligned}\delta_2 &= \frac{\partial \text{Error}}{\partial w_2} = \frac{\partial \text{Error}}{\partial o_1} * \frac{\partial o_1}{\partial w_2} + \frac{\partial \text{Error}}{\partial o_2} * \frac{\partial o_2}{\partial w_2} \\&= \frac{\partial \text{Error}}{\partial o_1} * \frac{\partial o_1}{\partial \ln o_1} * \frac{\partial \ln o_1}{\partial h_2} * \frac{\partial h_2}{\partial \ln h_2} * \frac{\partial \ln h_2}{\partial w_2} + \frac{\partial \text{Error}}{\partial o_2} * \frac{\partial o_2}{\partial \ln o_2} * \frac{\partial \ln o_2}{\partial h_2} * \frac{\partial h_2}{\partial \ln h_2} * \frac{\partial \ln h_2}{\partial w_2} \\&= \left(\frac{\partial \text{Error}}{\partial o_1} * \frac{\partial o_1}{\partial \ln o_1} * \frac{\partial \ln o_1}{\partial h_2} + \frac{\partial \text{Error}}{\partial o_2} * \frac{\partial o_2}{\partial \ln o_2} * \frac{\partial \ln o_2}{\partial h_2} \right) * \frac{\partial h_2}{\partial \ln h_2} * \frac{\partial \ln h_2}{\partial w_2} \\&= \left(\frac{\partial \text{Error}}{\partial o_1} * \frac{\partial o_1}{\partial \ln o_1} * w_7 + \frac{\partial \text{Error}}{\partial o_2} * \frac{\partial o_2}{\partial \ln o_2} * w_8 \right) * \frac{\partial h_2}{\partial \ln h_2} * \frac{\partial \ln h_2}{\partial w_2}\end{aligned}$$

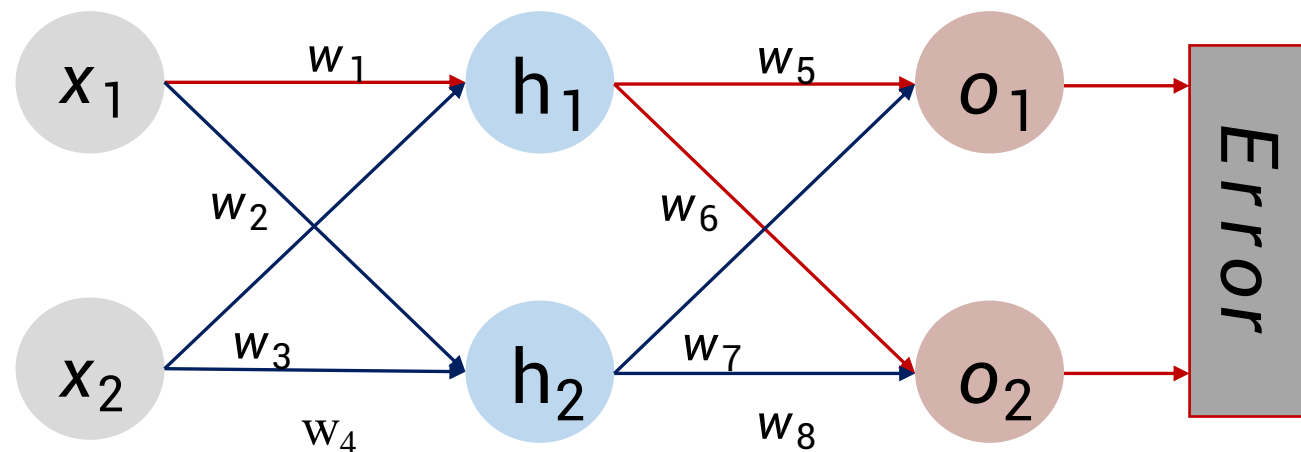
误差反向传播：前馈神经网络

- 反向传播：参数更新
 - 其中 η 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



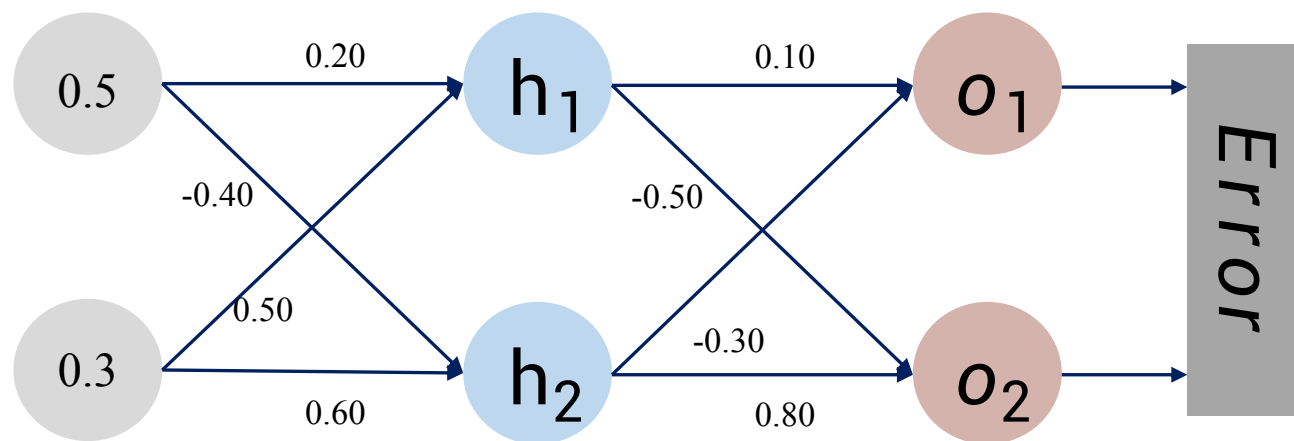
2. 更新参数，其中 η 被称为学习率

$$w'_i = w_i - \eta * \delta_i$$

原有参数 步长 传递误差

误差反向传播：前馈神经网络

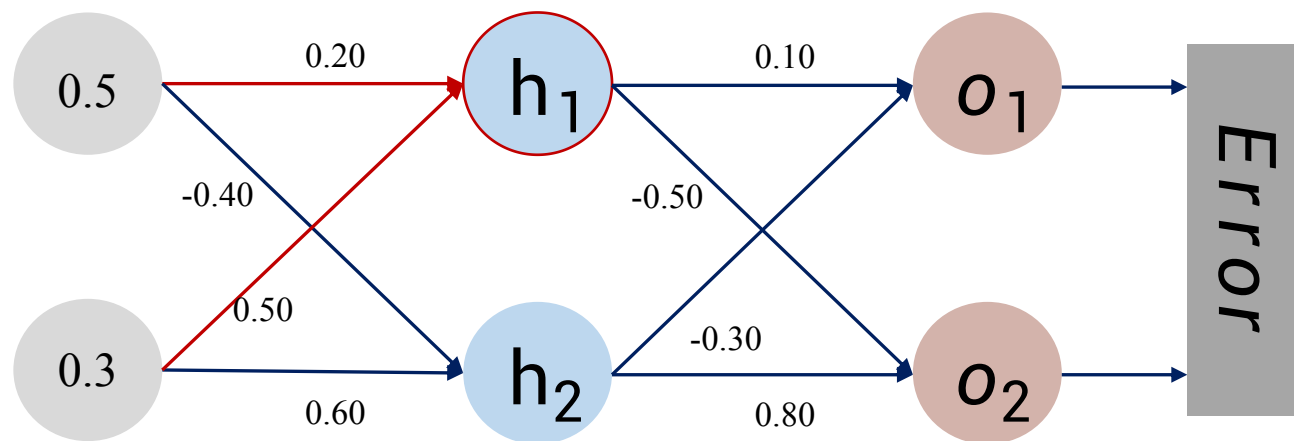
参数初始化



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

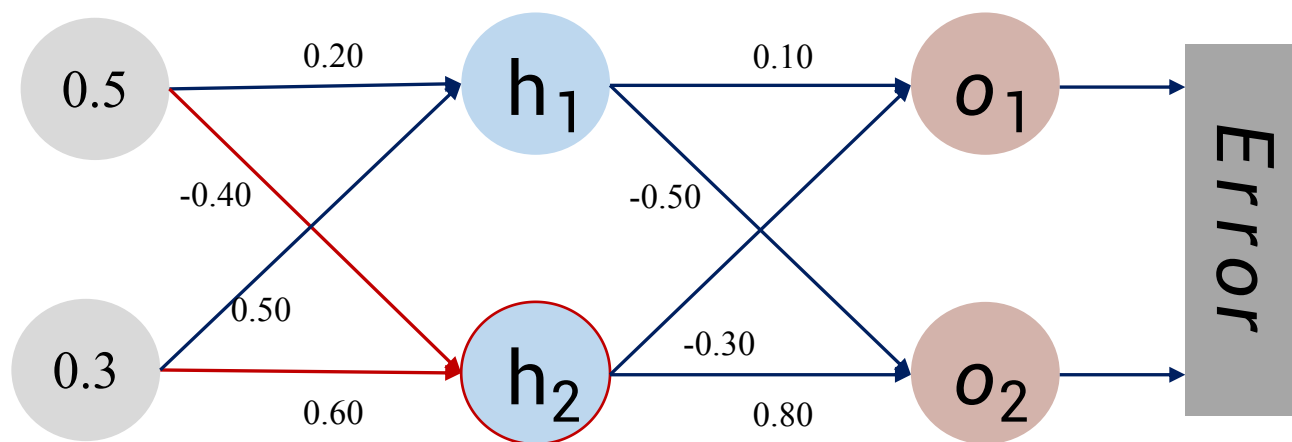
前向传播



$$h_1 = \text{sigmoid}(0.20 * 0.50 + 0.50 * 0.30) = 0.56$$

误差反向传播：前馈神经网络

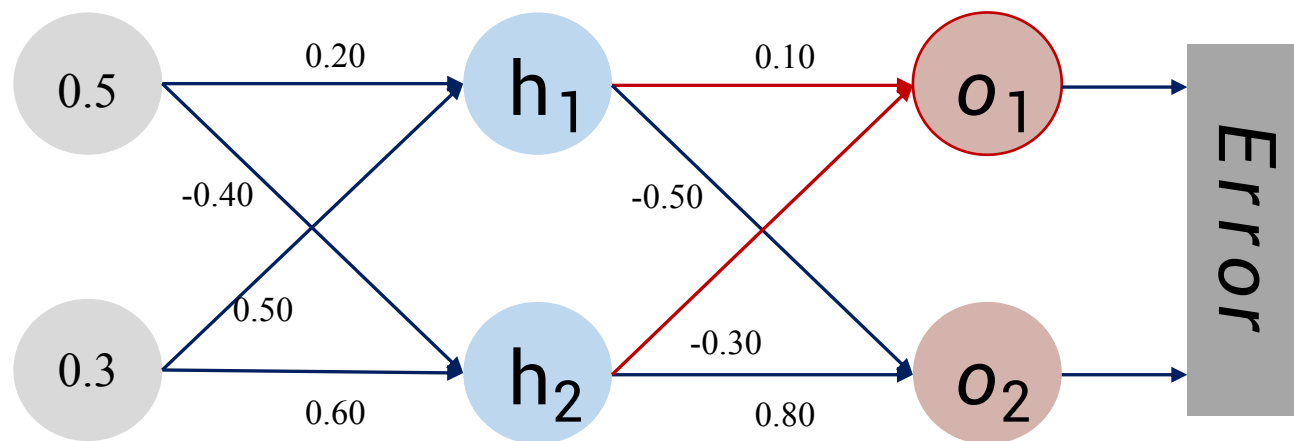
前向传播



$$h_2 = \text{sigmoid}(-0.40 * 0.50 + 0.60 * 0.30) = 0.50$$

误差反向传播：前馈神经网络

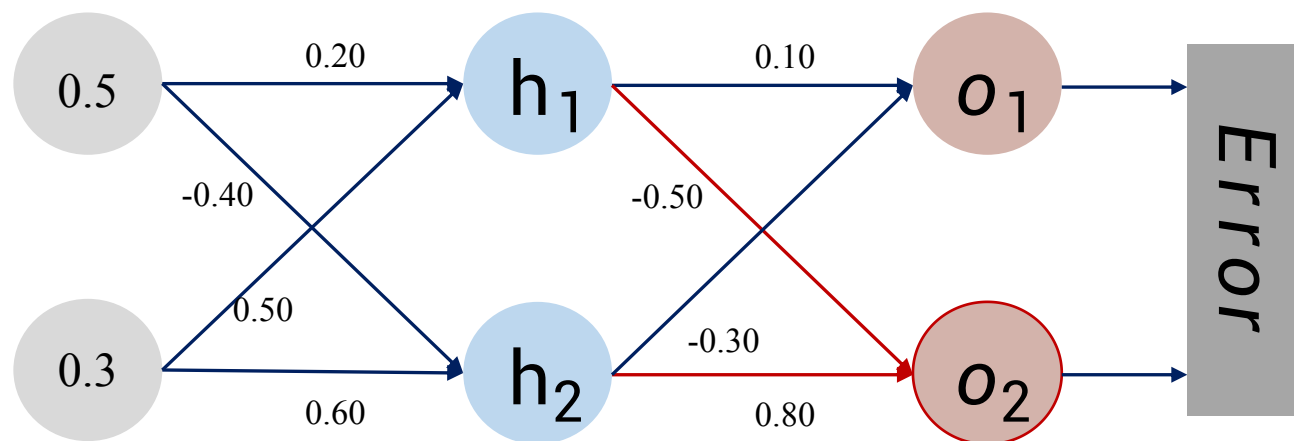
前向传播



$$o_1 = \text{sigmoid}(0.10 * 0.56 + (-0.30 * 0.50)) = 0.48$$

误差反向传播：前馈神经网络

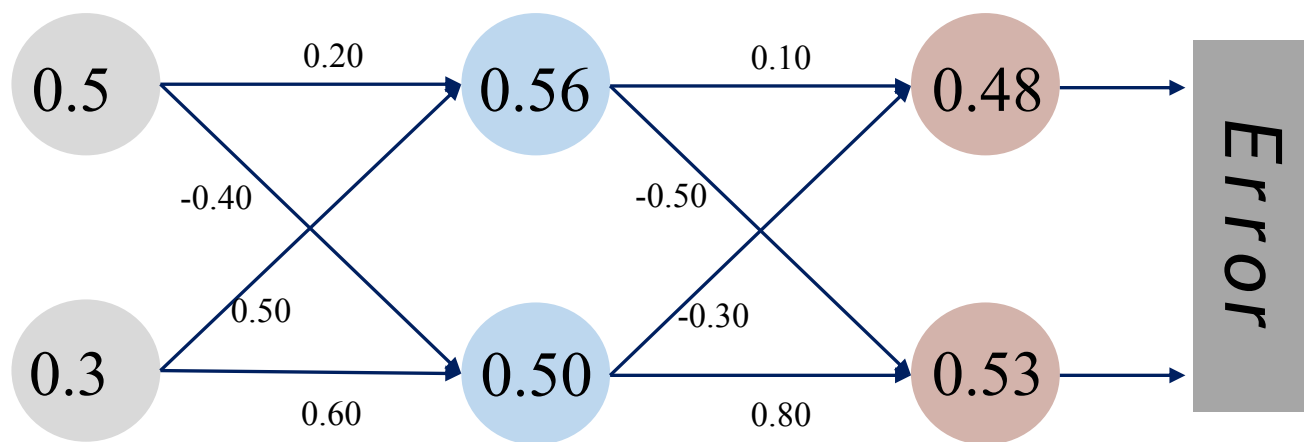
前向传播



$$o_2 = \text{sigmoid}(-0.50 * 0.56 + 0.80 * 0.50) = 0.53$$

误差反向传播：前馈神经网络

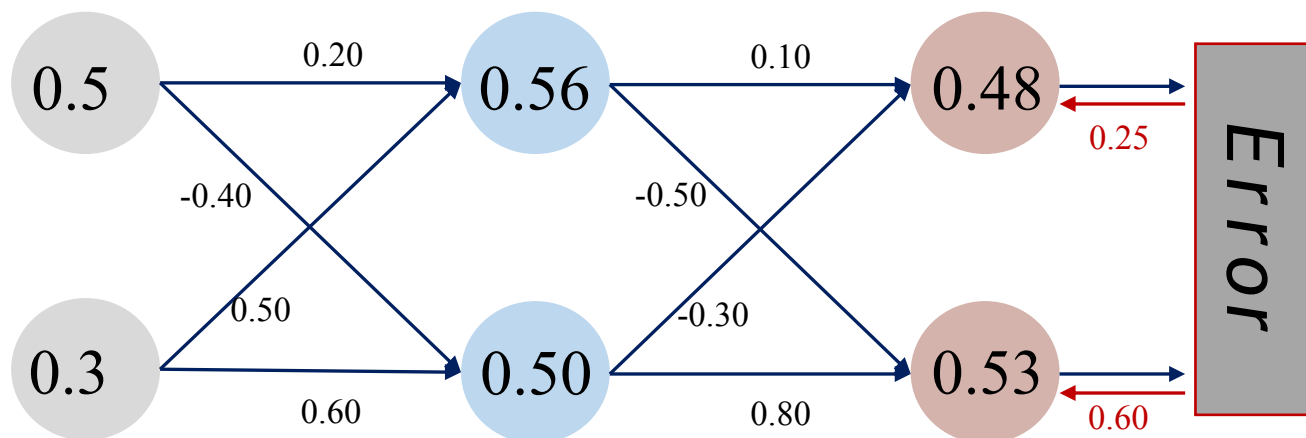
反向传播（假设学习速率 Learning Rate $\eta = 1$ ）



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

误差反向传播：前馈神经网络

梯度计算

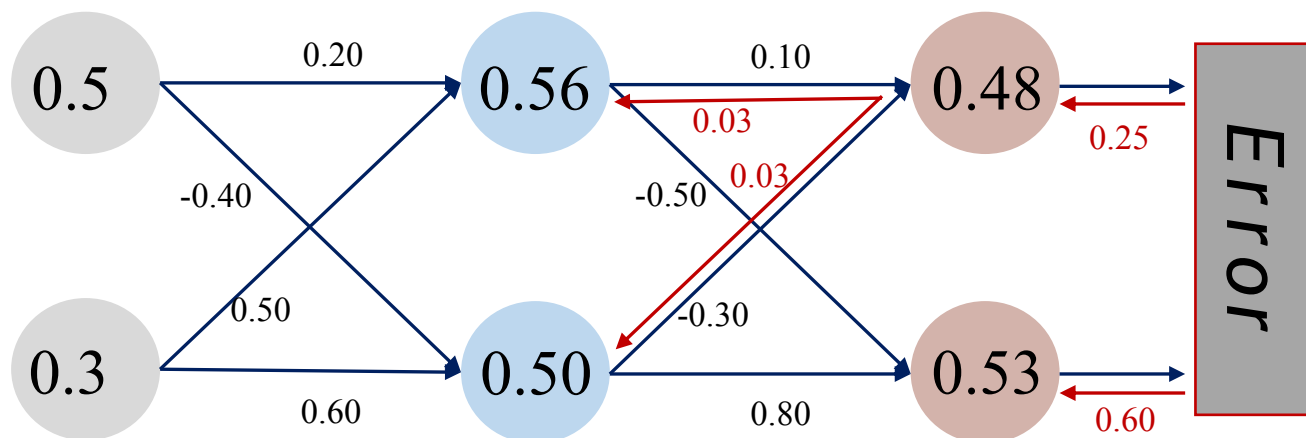


$$\delta_{o1} = \frac{\partial Error}{\partial o_1} = o_1 - 0.23 = 0.25$$

$$\delta_{o2} = \frac{\partial Error}{\partial o_2} = o_2 - (-0.07) = 0.60$$

误差反向传播：前馈神经网络

梯度计算

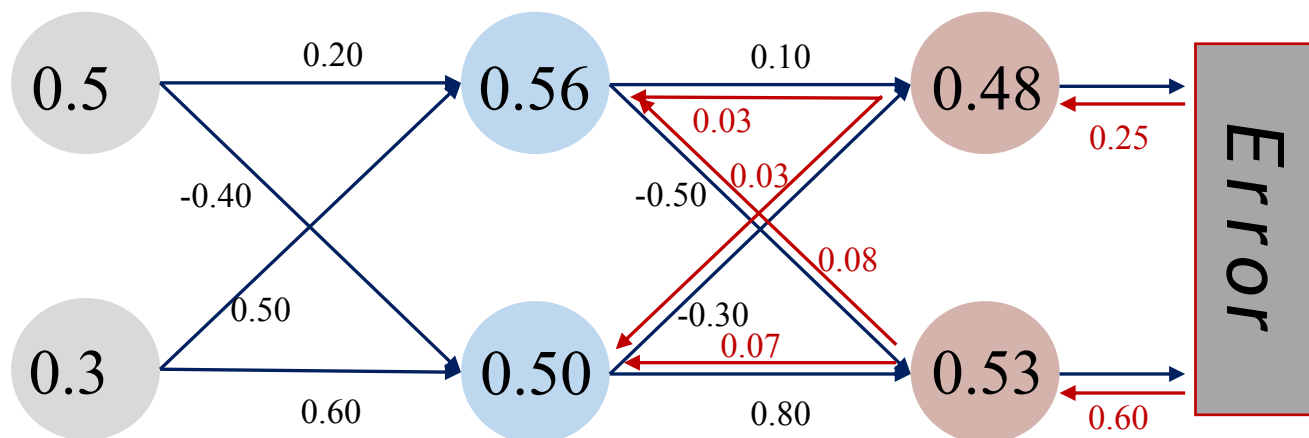


$$\delta_5 = \frac{\partial Error}{\partial w_5} = \frac{\partial Error}{\partial o_1} * \frac{\partial o_1}{\partial n_{o1}} * \frac{\partial n_{o1}}{\partial w_5} = 0.25 * 0.48 * (1 - 0.48) * 0.56 = 0.03$$

$$\delta_7 = \frac{\partial Error}{\partial w_7} = \frac{\partial Error}{\partial o_1} * \frac{\partial o_1}{\partial n_{o1}} * \frac{\partial n_{o1}}{\partial w_7} = 0.25 * 0.48 * (1 - 0.48) * 0.50 = 0.03$$

误差反向传播：前馈神经网络

梯度计算

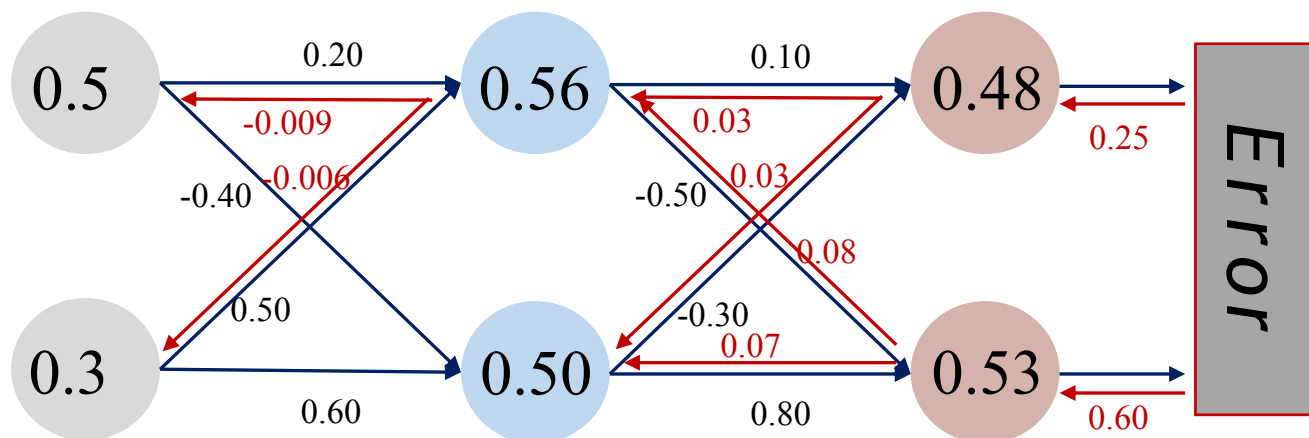


$$\delta_6 = \frac{\partial Error}{\partial w_6} = \frac{\partial Error}{\partial o_2} * \frac{\partial o_2}{\partial n_{o_2}} * \frac{\partial n_{o_2}}{\partial w_6} = 0.60 * 0.53 * (1 - 0.53) * 0.56 = 0.08$$

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

误差反向传播：前馈神经网络

梯度计算

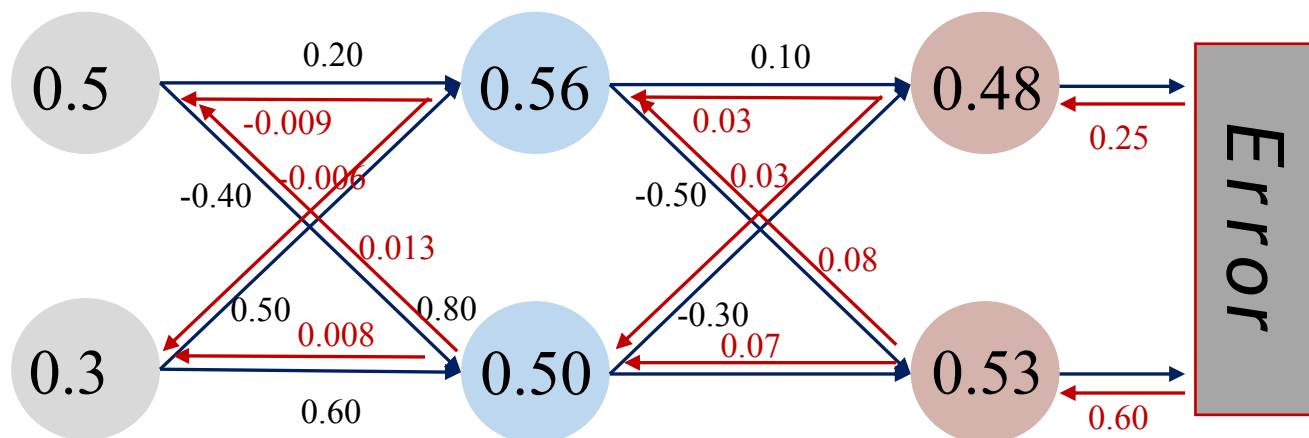


$$\begin{aligned}\delta_1 &= \frac{\partial Error}{\partial w_1} = \left(\frac{\partial Error}{\partial o_1} * \frac{\partial o_1}{\partial \ln_{o_1}} * w_5 + \frac{\partial Error}{\partial o_2} * \frac{\partial o_2}{\partial \ln_{o_2}} * w_6 \right) * \frac{\partial h_1}{\partial \ln_{h_1}} * \frac{\partial \ln_{h_1}}{\partial w_1} \\ &= (0.25 * 0.48 * (1 - 0.48) * 0.1 + 0.6 * 0.53 * (1 - 0.53) * (-0.5)) * 0.56 * (1 - 0.56) * 0.50 \\ &= -0.009\end{aligned}$$

$$\begin{aligned}\delta_3 &= \frac{\partial Error}{\partial w_3} = \left(\frac{\partial Error}{\partial o_1} * \frac{\partial o_1}{\partial \ln_{o_1}} * w_5 + \frac{\partial Error}{\partial o_2} * \frac{\partial o_2}{\partial \ln_{o_2}} * w_6 \right) * \frac{\partial h_1}{\partial \ln_{h_1}} * \frac{\partial \ln_{h_1}}{\partial w_3} \\ &= (0.25 * 0.48 * (1 - 0.48) * 0.1 + 0.6 * 0.53 * (1 - 0.53) * (-0.5)) * 0.56 * (1 - 0.56) * 0.30 \\ &= -0.006\end{aligned}$$

误差反向传播：前馈神经网络

梯度计算



$$\begin{aligned}\delta_2 &= \frac{\partial Error}{\partial w_2} = \left(\frac{\partial Error}{\partial o_1} * \frac{\partial o_1}{\partial \ln_{o_1}} * w_7 + \frac{\partial Error}{\partial o_2} * \frac{\partial o_2}{\partial \ln_{o_2}} * w_8 \right) * \frac{\partial h_2}{\partial \ln_{h_2}} * \frac{\partial \ln_{h_2}}{\partial w_2} \\ &= (0.25 * 0.48 * (1 - 0.48) * (-0.3) + 0.6 * 0.53 * (1 - 0.53) * 0.8) * 0.50 * (1 - 0.50) * 0.5 = 0.013\end{aligned}$$

$$\begin{aligned}\delta_4 &= \frac{\partial Error}{\partial w_4} = \left(\frac{\partial Error}{\partial o_1} * \frac{\partial o_1}{\partial \ln_{o_1}} * w_7 + \frac{\partial Error}{\partial o_2} * \frac{\partial o_2}{\partial \ln_{o_2}} * w_8 \right) * \frac{\partial h_2}{\partial \ln_{h_2}} * \frac{\partial \ln_{h_2}}{\partial w_4} \\ &= (0.25 * 0.48 * (1 - 0.48) * (-0.3) + 0.6 * 0.53 * (1 - 0.53) * 0.8) * 0.50 * (1 - 0.50) * 0.3 = 0.008\end{aligned}$$

误差反向传播：前馈神经网络

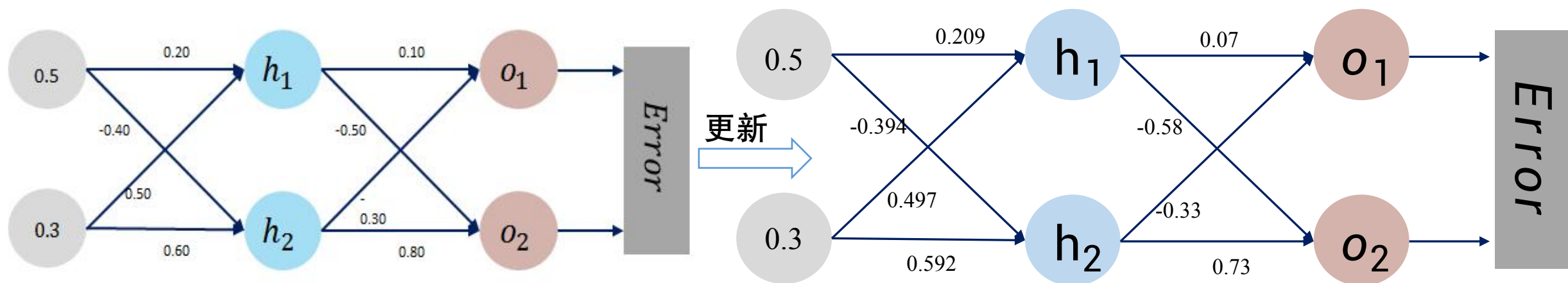
参数更新

$$w'_i = w_i - \eta * \delta_i$$

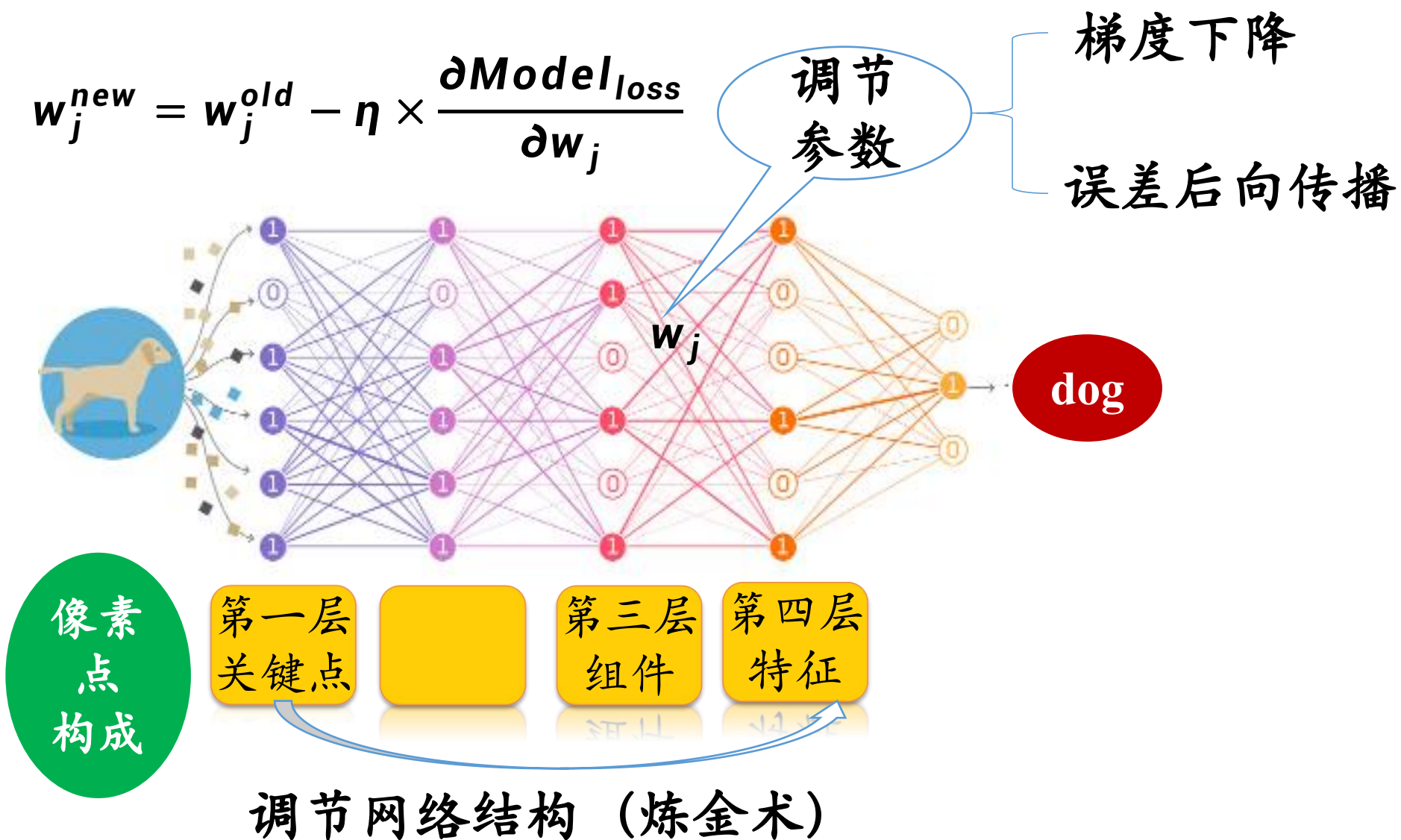
原有
参数

步长

传递
误差



机器学习的能力在于拟合和优化



谢谢!