



深度学习 概述

万永权

目录

CONTENTS

1. 深度学习的发展历史
2. MP模型
3. 感知机
4. 深度学习的基本方法
5. 深度学习的编程框架



本章目标

- ◆ 了解人工神经网络的发展历程。
- ◆ 理解感知机的工作原理和局限性。
- ◆ 掌握前馈神经网络的结构、前馈神经网络的基本原理、前馈传播、反向传播算法



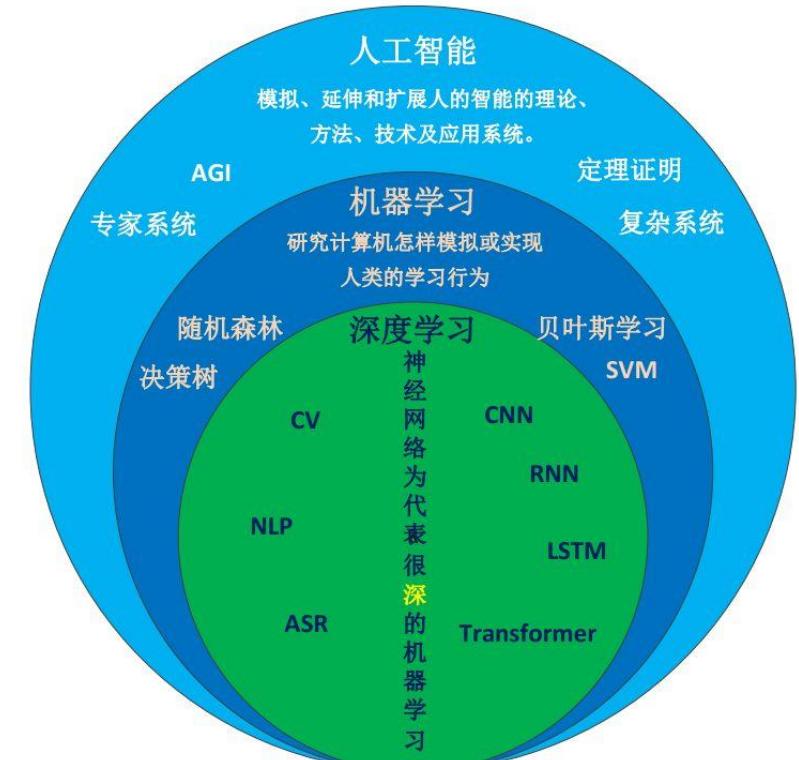
概述

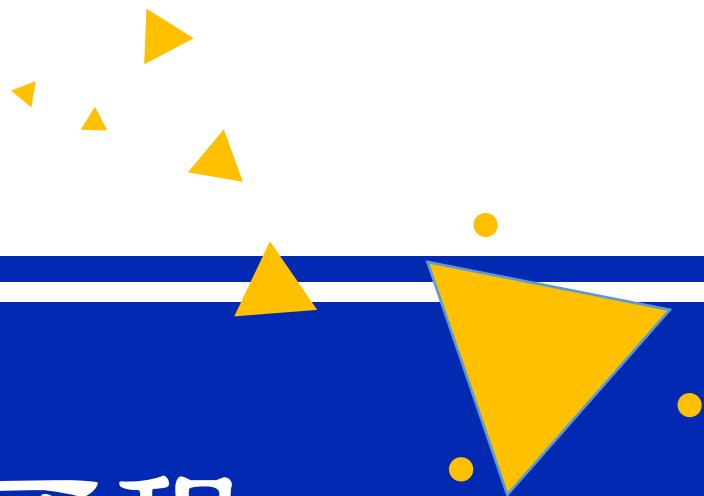
- 深度学习(Deep Learning, DL)是机器学习(Machine Learning, ML)领域中一个重要分支。
- 深度学习的概念起源于人工神经网络 (Artificial Neural Network, ANN)
- 深度学习是学习样本数据的内在规律和表示层次，这些学习过程中获得的信息对诸如文字、图像和声音等数据的解释有很大的帮助。它的最终目标是让机器能够像人一样具有分析学习能力，能够识别文字、图像和声音等数据。
- 本章主要介绍深度学习的核心计算模型—**人工神经网络模型：前馈神经网络**

人工神经网络

■ 人工智能、机器学习与深度学习关系

- 机器学习是人工智能研究领域的一部分，是实现人工智能的算法。
- 人工神经网络算法是众多机器学习算法中的一个分支，又可以分为浅层神经网络（实现浅层学习）和深层神经网络（实现深度学习）。
- 由于深度学习算法处理出来的结果具有非常高的准确度，达到或接近人的智能程度，从而掀起了当前人工智能新高潮。





人工神经网络发展历程

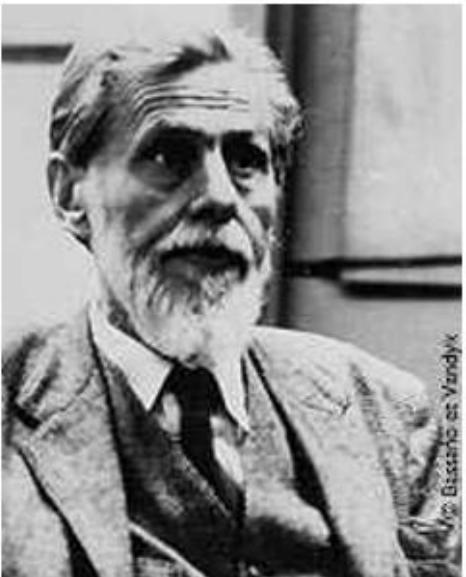


人工神经网络

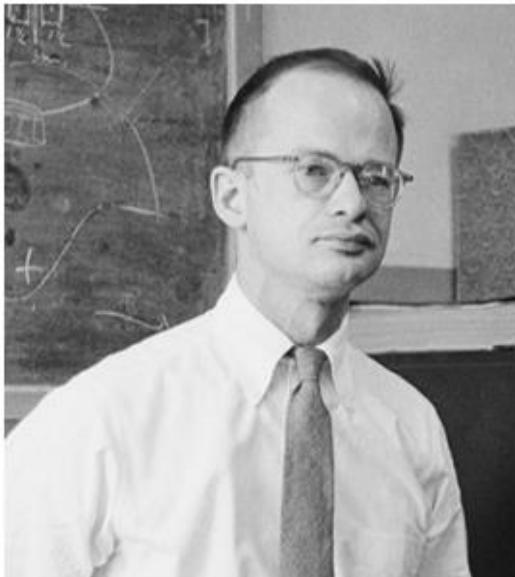
- ◆ 人工神经网络 (ANN, Artificial Neural Network) 20世纪40年代起源，20世纪80年代逐步兴起，基本思想是使用计算机模拟人脑的神经网络来实现机器学习。
- ◆ 在机器学习领域，人工神经网络一般简称为神经网络或者神经计算。具有学习能力，而且经常是大量的神经元模型并行工作。
- ◆ 神经网络主要包括神经元、拓扑结构、训练规则三个要素，可以看作是按照一定规则连接起来的多个神经元构成的系统。各神经元能够根据经验自适应学习，同时还能并行化处理。
- ◆ 神经网络算法也是深度学习的基础。

MP模型

- ◆ 第一个神经网络模型M-P模型出现于1943年，是由神经生理学家Warren McCulloch和数学家Walter Pitts共同提出的，并由他们名字的首字母共同命名。



(1) Warren McCulloch



(2) Walter Pitts

图 10.1 M-P 神经模型的提出者

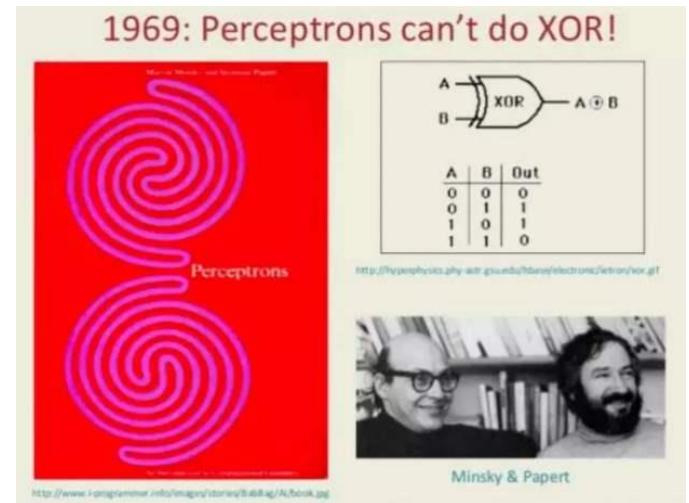


人工神经网络

- ◆ 1949年，赫布提出了赫布理论，即“突触前神经元向突触后神经元的持续重复的刺激，可以导致突触传递效能的增加。”即：突触前给的刺激越强，突触后的反应越大。
- ◆ 1957年，美国心理学家Rosenblatt在M-P模型基础上，提出了具有自学习能力的神经网络模型——**感知器**模型，神经网络从纯理论走向了实际的工程应用。这是第一个人工神经网络。
- ◆ 感知机本质是一种仅包含输入层和输出层的二元线性分类器，也称为**单层神经网络**。感知机的输入层可以包含多个单元，而输出层只有一个单元。
- ◆ 神经网络默认的方向是从输入到输出，信息可以看成是向前传递的，因此从输入到输出进行信息传递的多层神经网络也称为**前馈型神经网络**。

人工神经网络

- ◆ 与MP模型**不同的是**，感知机模型可以利用学习算法自动更新输入的权重和阈值。此后，神经网络的研究进入了第一次高潮。
- ◆ 1969年，[明斯基《感知机：计算几何导论》](#)指出了感知机的局限性，使人工智能的连接主义研究流派陷入了长达10多年的低谷期。
 1. 感知机无法处理“异或”问题；
 2. 当时的计算机无法支持处理大型人工神经网络所需要的计算能力。
- ◆ 1974年，哈佛大学的韦伯斯在其博士论文中提出利用反向传播算法训练多层神经网络，但几乎无人了解这项研究成果。



人工神经网络

- ◆ 1980年代末，**迈克尔·乔丹**(Michael I. Jordan)和**杰弗里·埃尔曼**(Jeffrey Elman)提出了简单循环神经网络，拉开了循环神经网络（Recurrent Neural Network，简称RNN）的研究序幕。
- ◆ 1982年，**霍普菲尔德教授**提出Hopfield神经网络，可用于实现联想记忆和优化计算，在旅行商问题上获得了突破。尤其是1984年又用模拟集成电路实现了Hopfield神经网络，有力地推动了神经网络的研究，**连接主义迎来了第二次高潮**。
- ◆ 受此启发，**辛顿**于1984年提出了一种随机型的Hopfield网络，即**玻尔兹曼机**。它是一种**反馈神经网络**，借鉴了模拟退火法的思想，具有一定的“跳出局部最优”的能力。**玻尔兹曼机的特点**是：
 - 一是包含显层与隐层两层结构，显层代表输入和输出，隐层则被理解为数据的内部表达；
 - 二是其神经元是布尔型的，即只能取0或1值。

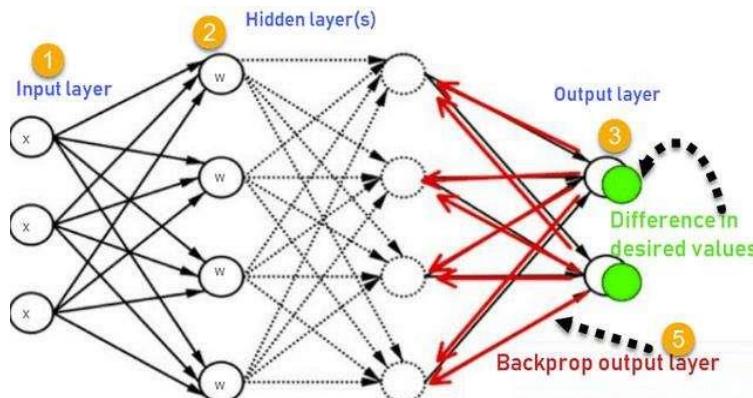


人工神经网络

- ◆ 1986年,心理学家大卫·鲁梅尔哈特 (David Rumelhart) 、**杰弗里·辛顿**(Geoffrey E. Hinton和罗纳德·威廉姆斯(Ronald Williams)在自然杂志上发表一篇突破性的论文《通过**反向传播**算法实现表征学习》,清晰论证了“误差反向传播”算法是切实可操作的训练多层神经网络的方法。
- ◆ 彻底扭转明斯基《感知机》一书带来的负面影响,多层神经网络的有效性终于再次得到学术界的普遍认可,从而将神经网络的研究推向了新的高潮。

人工神经网络

◆ 1989年，**杨乐昆**将反向传播算法引入了**卷积神经网络**（CNN），并将其成功应用于美国邮政、银行支票上手写数字的识别（1998年）。LeNet-5模型成为现代CNN的基础。杨乐昆被誉为“**卷积神经网络之父**”。





人工神经网络

- ◆ 但BP算法存在梯度消失、梯度爆炸等问题，且当时算力不足，BP算法只适合于训练浅层神经网络。
- ◆ 1995年，万普尼克于提出了支持向量机，SVM可通过核技巧将非线性问题转换成线性问题，其理论基础清晰、证明完备、可解释性好，获得了广泛的认同。
- ◆ 同时，统计机器学习专家从理论角度怀疑NN的泛化能力，使得神经网络的研究第二次陷入低谷。

梯度消失与梯度爆炸问题

■ 梯度消失与梯度爆炸的原因

在训练过程中，需要计算最终的误差评估函数对所有神经元权值参数的偏导数，而根据链式传递法则可知，误差评估函数对前面层次神经元的权值参数的偏导数是由误差评估函数对后面各层神经元输出计算公式的偏导数的连续乘积计算得来，当存在非常多网络层次时，各层神经元函数的偏导数连续相乘的结果就会接近0或者非常大，这两种情况分别称为梯度消失和梯度爆炸。



人工神经网络

- 梯度消失问题阻碍了人工神经网络的进一步发展。
- 1995~2006年，支持向量机和其它更简单的方法在机器学习领域的流行度逐渐超过了人工神经网络。
- 当时的计算机性能和数据规模不足以支持训练大规模的人工神经网络。
- 20世纪90年代中期，统计学习理论和以支持向量机为代表的机器学习模型开始兴起。



人工神经网络

◆进入21世纪，有了大数据和算力，**神经网络的研究迎来了第三次高潮**。2006年，辛顿教授等人首次提出了“深度信念网络”（Deep Belief Network），它是由多个受限玻尔兹曼机（Restricted Boltzmann Machine）串联堆叠而组成的一个深度网络。深度信念网络在分类任务上的性能超过了传统经典的浅层学习模型（如支持向量机），引起了学术圈的广泛关注。



深度网络时代：第三代神经网络（2006~至今）

◆ 深度学习的提出：

2006年，DL元年

◆ 2006年，加拿大多伦多大学教授、机器学习领域的泰斗Geoffrey Hinton在《科学》上发表论文提出深度学习主要观点：

- ✓ 多隐层的人工神经网络具有优异的特征学习能力，学习得到的特征对数据有更本质的刻画，从而有利于可视化或分类；
- ✓ 深度神经网络在训练上的难度，可以通过“逐层初始化”（layer-wise pre-training）来有效克服，逐层初始化可通过无监督学习实现的。



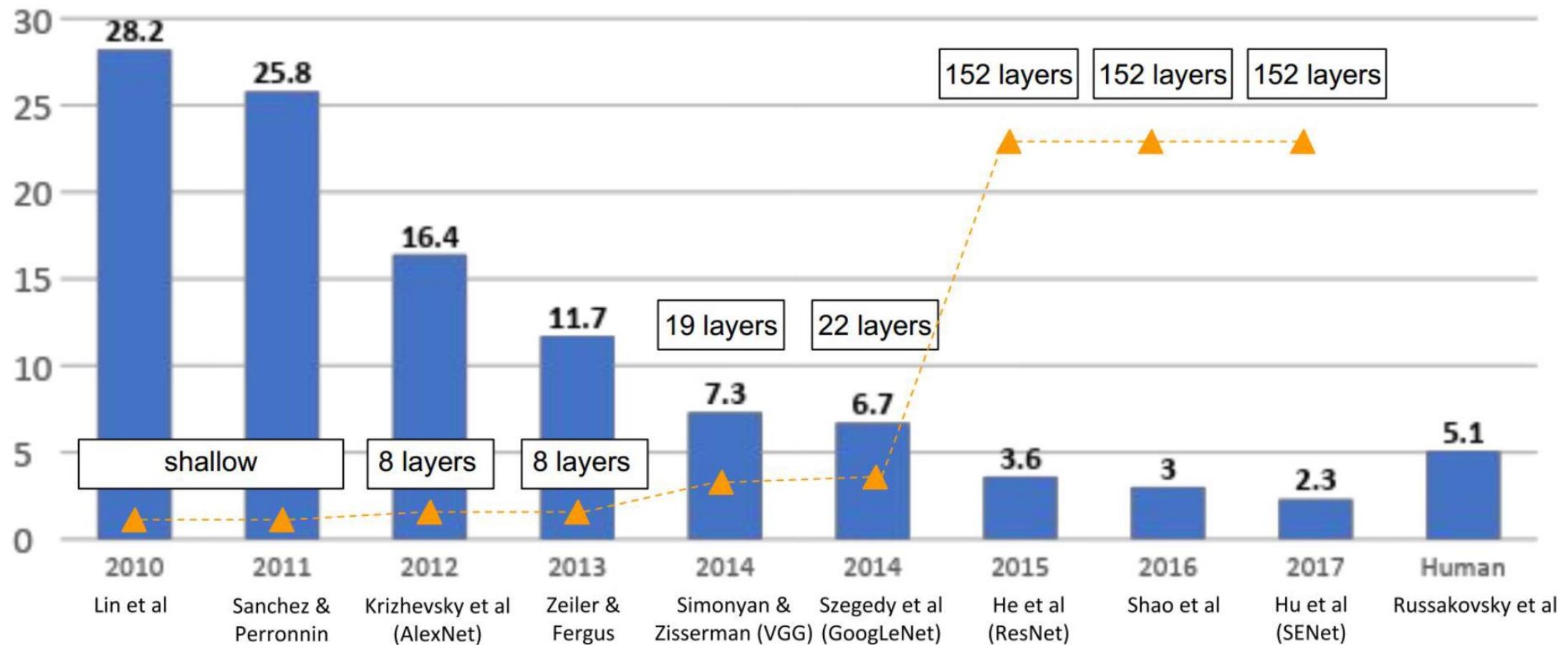
人工神经网络

- ◆ 随着神经网络层数的不断加深，辛顿将这种深层神经网络上的学习方法命名为“**深度学习**”。从此，**连接主义研究学派**开始大放异彩。
- ◆ 通常，将包含多个（大于3即可）隐藏层的人工神经网络称为**深度神经网络**，在这样的网络上学习的过程称为**深度学习**。
- ◆

- 
- ◆ 2012年，辛顿教授及其学生Alex提出AlexNet模型，在ILSVRC比赛的图像分类组一举夺魁。从此深度学习在学术圈和产业界均得到广泛关注。
 - ◆ 深度学习发展至今，神经网络的层数不断加深，模型性能不断提升，甚至在图像分类任务上的能力已超过了人类。
 - ◆ 2010至2017年间，ImageNet图像分类的top-5（即排名前5的分类标签）错误率从28%降到了3%，目标识别的平均准确率从23%上升到了66%。
 - ◆ 此外，深度学习方法在自然语言处理、机器翻译、无人驾驶、语音识别、生物信息学、医学影像分析与金融大数据分析等方面也都有广泛的、成熟的应用。

ILSVRC – ImageNet Large Scale Visual Recognition Competition

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners





人工神经网络

第三代神经网络（2006~至今）：深度网络时代

深度学习分为两个时期：快速发展期（2006~2012）与爆发期（2012~至今）

2006年：DL元年：Hinton提出了深层网络训练中梯度消失问题的解决方案

-----无监督预训练对权值进行初始化+有监督训练微调

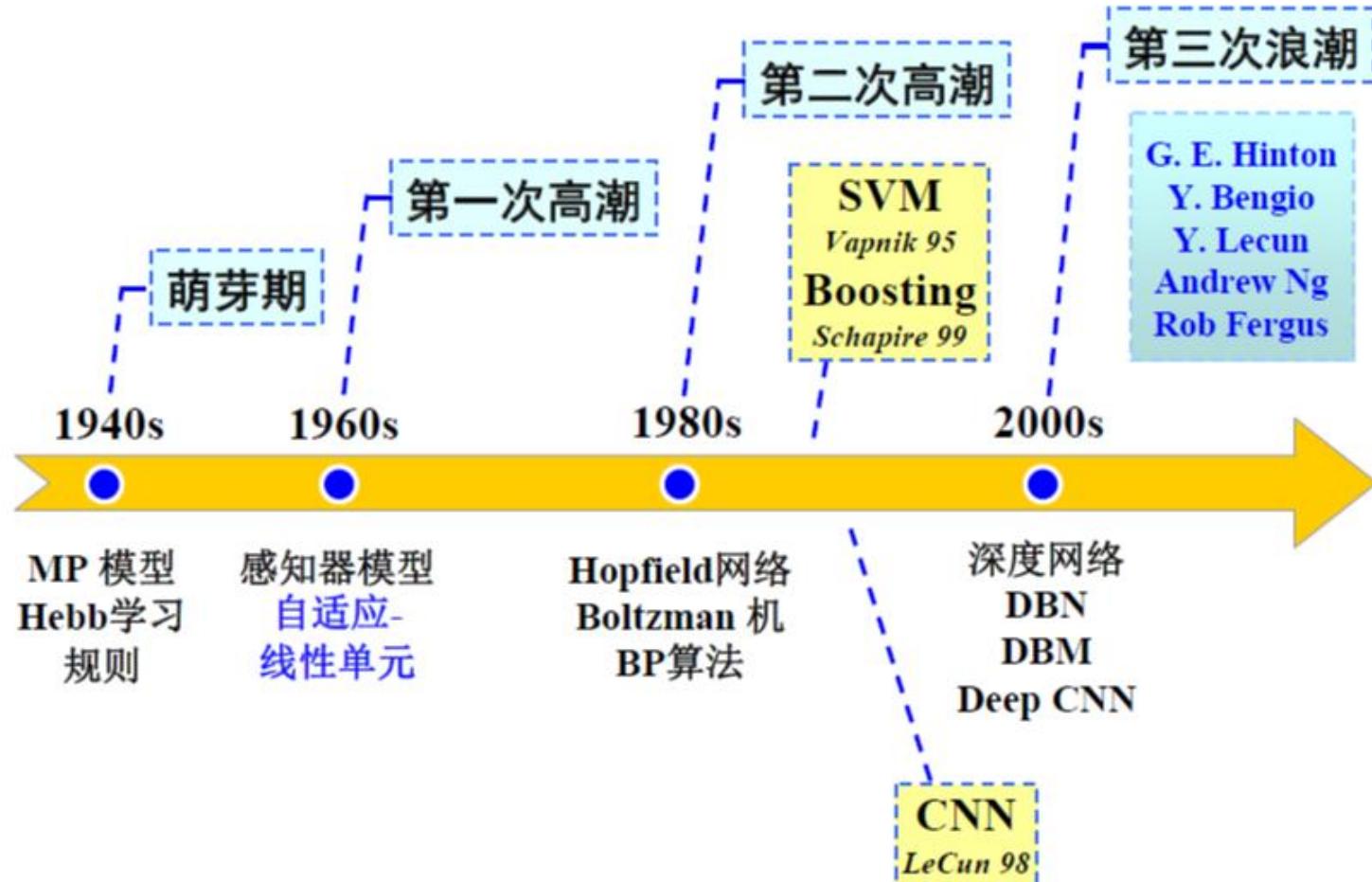
2011年：ReLU激活函数被提出，该激活函数能够有效的抑制梯度消失问题

2012年：CNN网络AlexNet在ImageNet图像识别比赛一举夺得冠军，超过第二名（SVM方法）
的分类性能

2013,2014,2015年：DL的网络结构，训练方法，GPU硬件的不断进步；Hinton, LeCun, Bengio论证Loss的局部极值问题对于深层网络来说影响可以忽略；DeepResidualNet发明。

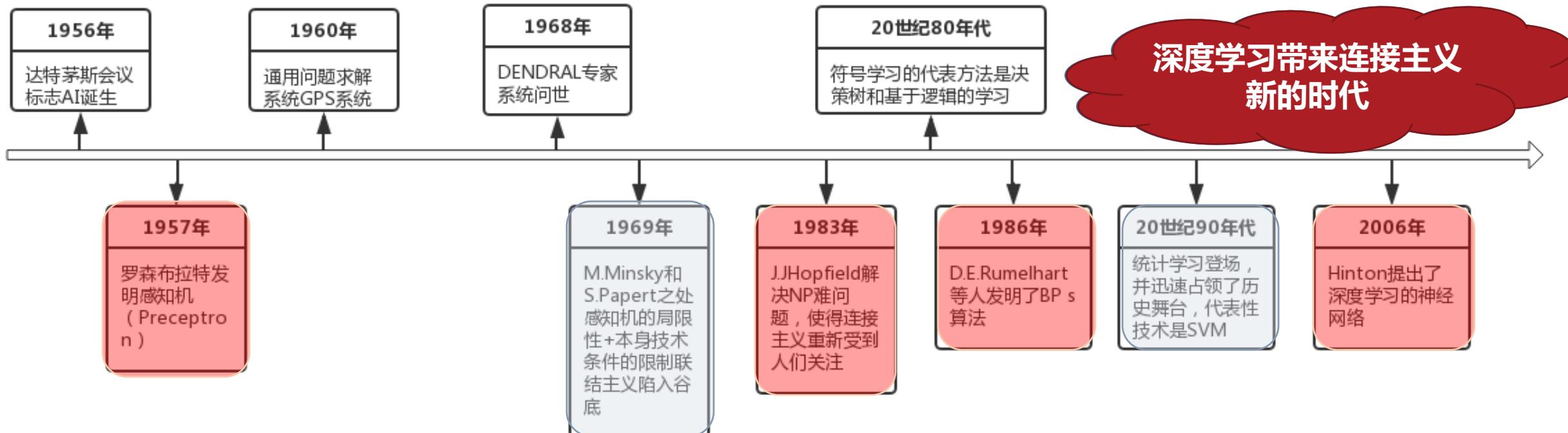
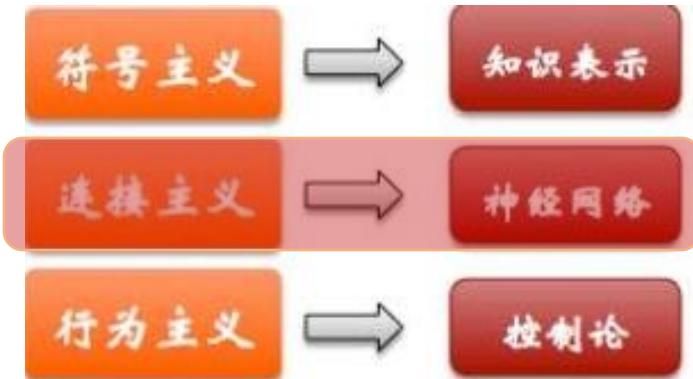


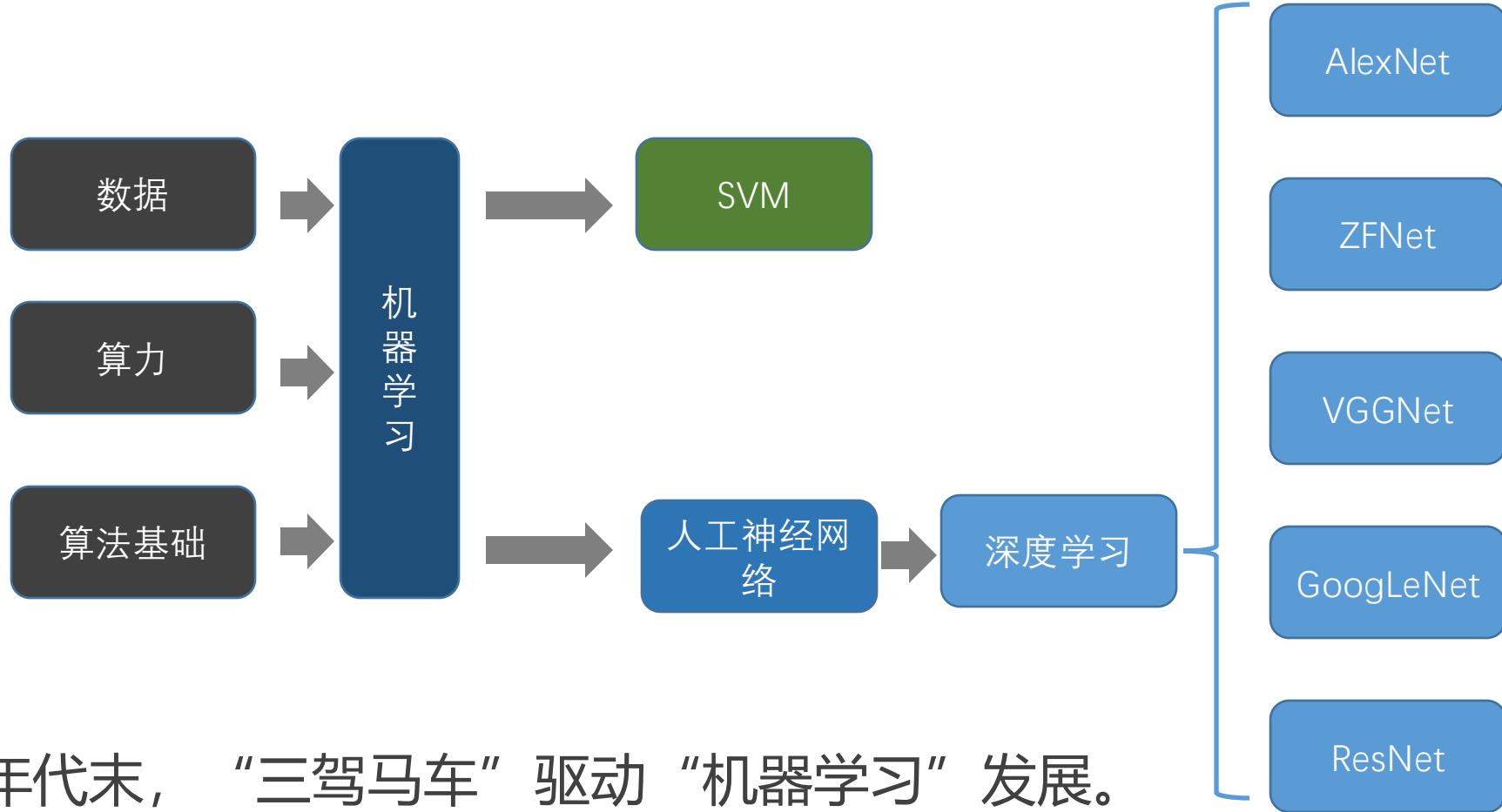
ANN 发展史



发展历程

人工智能的多种流派





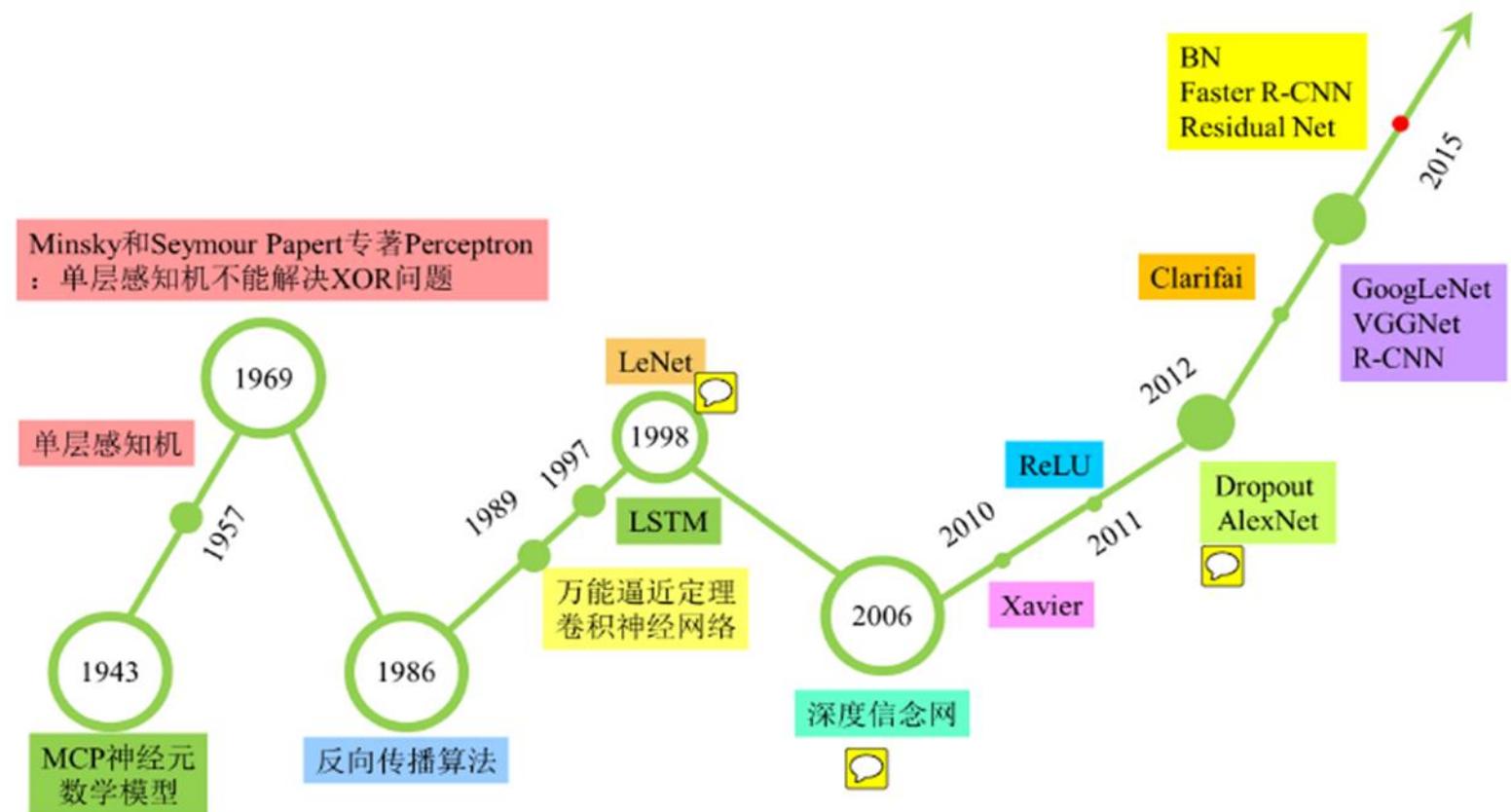
- 20世纪80年代末，“三驾马车”驱动“机器学习”发展。
- 2006前后，SVM独占鳌头，2012年后是深度学习的天下。

发展历程-神经网络发展

神经网络发展

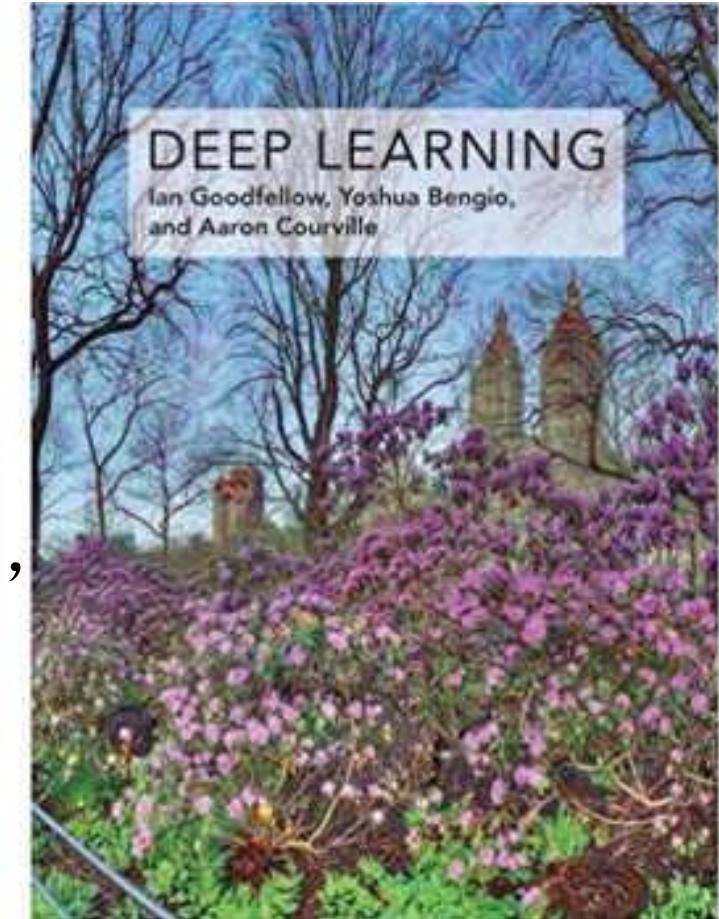
在深度学习从06年崛起之前经历了两个低谷；

这两个低谷也将神经网络的发展分为了三个不同的阶段



图书推荐

- ◆AI圣经！深度学习领域奠基于此的经典畅销书！长期位居美国亚马逊AI和机器学习类图书榜首！所有数据科学家和机器学习从业者的必读图书！特斯拉CEO埃隆·马斯克等国内外众多专家推荐！
- ◆封面特色：由艺术家Daniel Ambrosi提供的中央公园杜鹃花步道梦幻景观。在Ambrosi的亿级像素全景图上，应用Joseph Smarr（Google）和Chris Lamb（NVIDIA）修改后的Google DeepDream开源程序，创造了Daniel Ambrosi的“幻景”。



MP模型、感知机、神经网络



连接主义学派

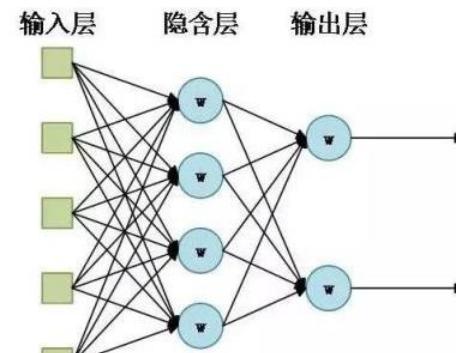


又称 **仿生学派** 或 **生理学派**

- 认为人的思维基元是**神经元**, 而不是符号处理过程

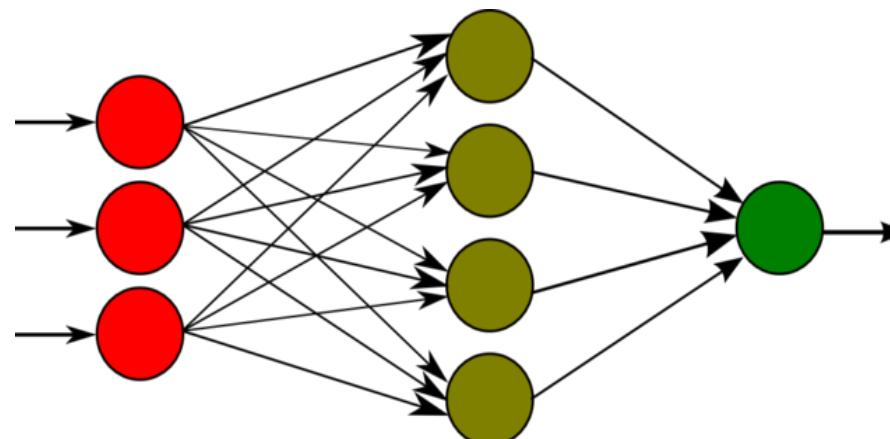


- 核心: 智能的本质是**连接机制**。
- 原理: 神经网络及神经网络间的连接机制和学习算法。



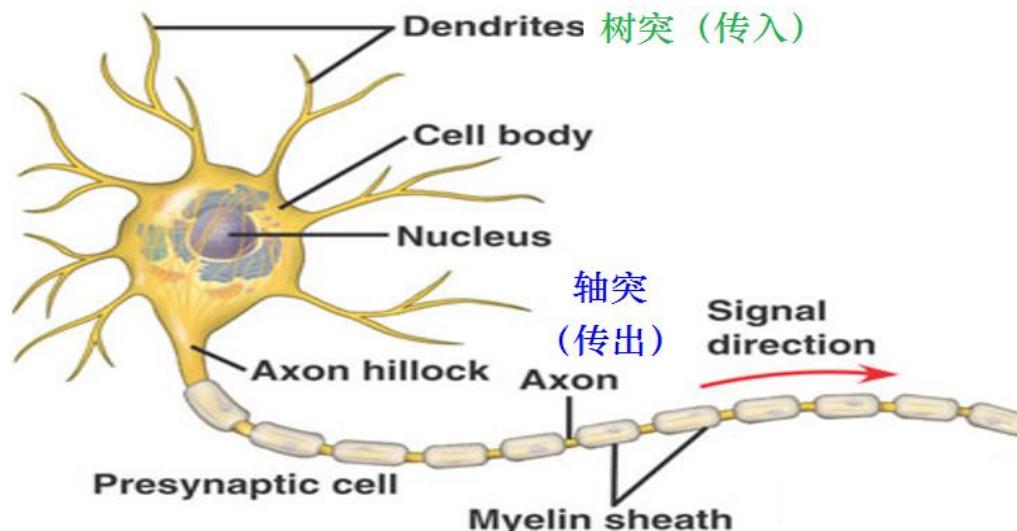
人工神经网络

- ◆ 科学家受到人脑或生物神经元结构和学习机制的启发，提出了人工神经元的**数学模型**，又在此基础上增加了学习机制，研制出了可运行的**感知机**，即**单层人工神经网络**。最后，将若干个感知机连接在一起，形成了**人工神经网络**。
- ◆ 人工神经网络，简称**神经网络**（Neural Network, NN）或类神经网络，是模拟人脑或生物神经网络的学习机制而建立起来的一种运算模型。它由大量简单的信息处理单元按照一定拓扑结构相互连接而组成人工网络。



神经网络

- ◆ 生物学家早在20世纪初就发现了生物神经元的结构，**神经元**（Neuron）也称为**神经细胞**。
- ◆ 典型的神经元结构可分为**细胞体**和**细胞突起**。
- **细胞体**中的细胞膜上有各种受体和离子通道，受体可与相应的化学神经递质结合，引起膜内外**电位差**发生改变，产生相应的生理活动：**兴奋或抑制**。

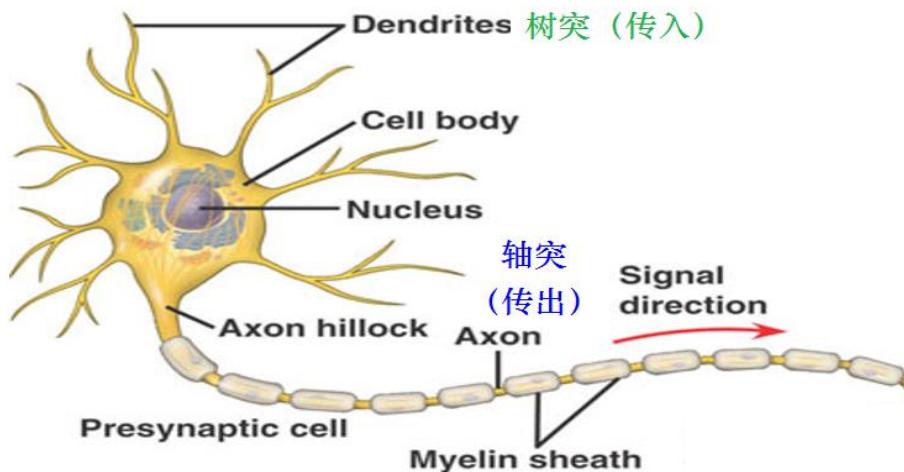




生物神经网络



- 细胞突起是由细胞体延伸出来的细长部分，又可分为**树突**和**轴突**。
- **树突**：可以接收刺激并将兴奋**传入**细胞体，每个神经元可以有1或多个树突。
- **轴突**：可以把自身的兴奋状态**传出**到另一个神经元或其它组织。每个神经元只有一个轴突。

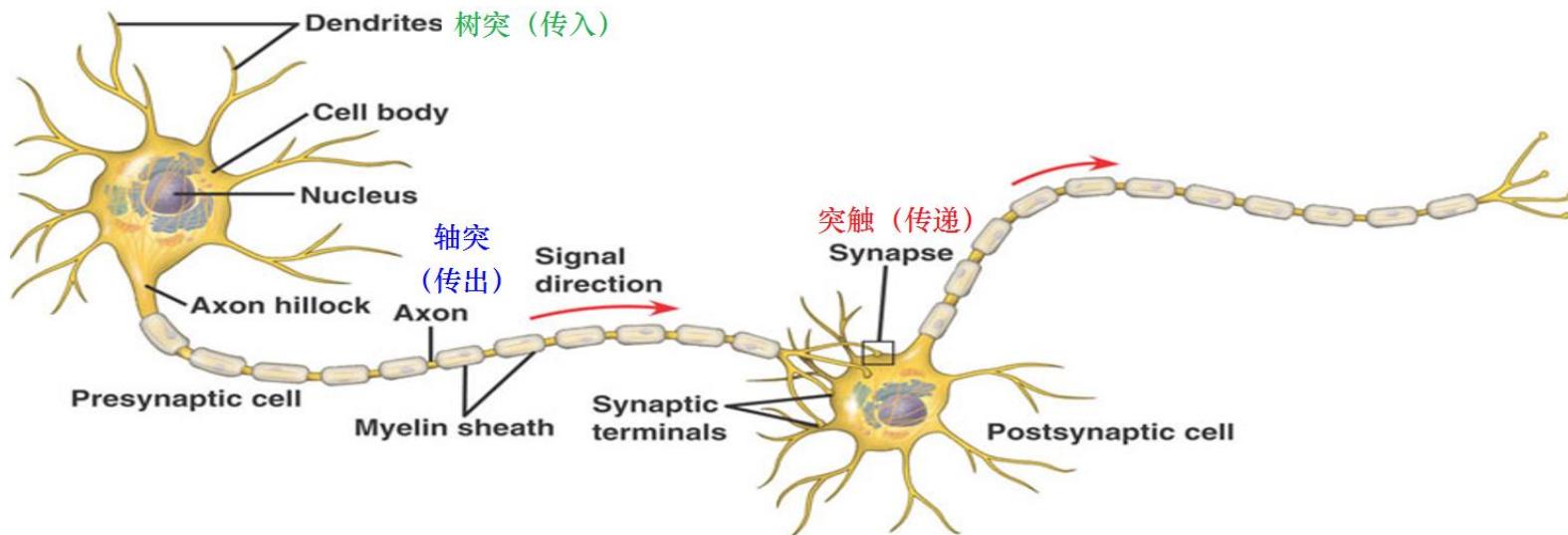




生物神经网络



- 神经元可以接收其它神经元的信息，也可以发送信息给其它神经元。
- 神经元之间通过突触来传递信息，从而形成一个神经网络，即神经系统。
- 一个神经元可视为一种只有两种状态的细胞：兴奋和抑制。





生物神经元的两种状态

- ◆ 兴奋状态
 - ◆ 当传入的神经冲动使细胞膜电位升高超过一个“阈值”时，该细胞就会被激活，进入兴奋状态，产生神经冲动，并由轴突经过突触输出；
- ◆ 抑制状态
 - ◆ 当传入的神经冲动使细胞膜电位下降到低于一个“阈值”时，该细胞就进入抑制状态，不输出神经冲动。

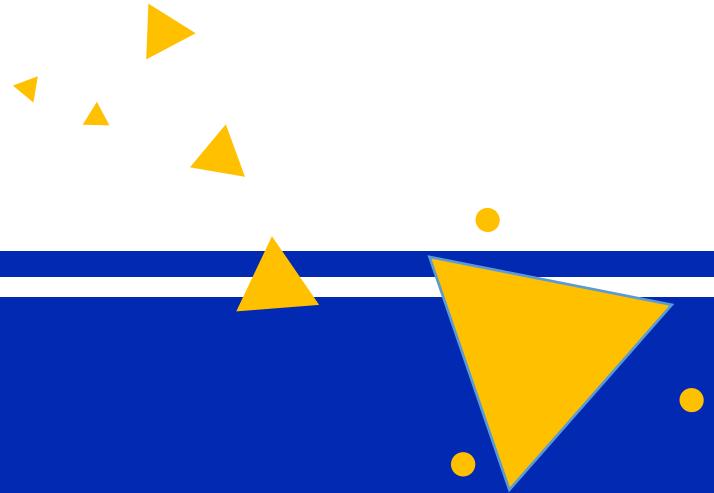


人工神经网络



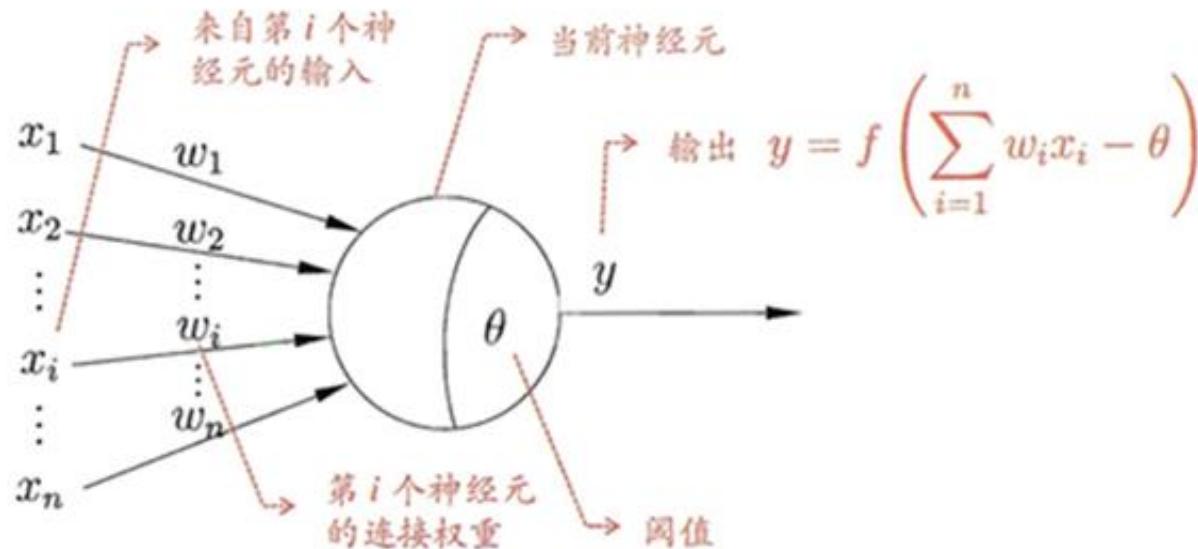
- 人工神经网络是为模拟人类神经网络而设计的一种计算模型，它从结构、实现机理和功能上模拟了人类神经网络。
- 人工神经网络与生物神经网络类似，由多个节点（人工神经元）互相连接而成，可以用来对数据之间的复杂关系进行建模。
- 不同节点之间的连接被赋予了不同的权重。
- 每个节点代表一种特定函数，来自其它节点的信息经过相应的加权计算，输入到一个激活函数中并得到一个值（兴奋或抑制）。

MP 模型



M-P模型

- ◆ 1943年, 神经生理学家沃伦·麦卡洛克(Warren S. McCulloch)和数理逻辑学家沃尔特·皮茨(Walter Pitts)提出了神经元的数学模型, 称之为**M-P模型**。该模型是模仿生物神经系统的结构和工作原理。



MP模型

- ◆ 对于某一个神经元*i*, 输入($x_1, x_2, x_3, \dots, x_n$)。神经元权值($w_{i1}, w_{i2}, w_{i3}, \dots, w_{in}$), 用X表示输入向量, 用W表示权重向量, 即:

$$X = (x_1, x_2, x_3, \dots, x_n)$$

$$W = \begin{bmatrix} w_{i1} \\ w_{i2} \\ w_{i3} \\ \vdots \\ w_{in} \end{bmatrix}$$

- ◆ 则神经元*i*的输出

$$y_i = f(XW)$$

神经元数学模型—MP模型

- ◆ MP模型可以接收多路输入信号。
- ◆ 假设一个神经元同时接收的 n 个输入信号用向量 $X = (x_1, x_2, x_3, \dots, x_n)$ 表示，则所有输入信号的线性组合称为该神经元的净输入，记为 $u \in \mathcal{R}$ ，计算公式如下：

$$u = \sum_{j=1}^n w_j x_j + \theta$$

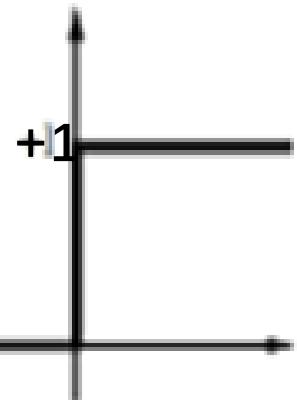
其中， w_1, \dots, w_n 为权值， $\theta \in \mathcal{R}$ 为偏置。

- ◆ 净输入 u 会被函数 $f(x)$ 转换为输出值 y ，该函数称为**激活函数**。

- ◆ 在MP模型中，激活函数 $f(x)$ 采用**非线性的阶跃函数**，其表达式如下：

$$y = f(u) \quad f(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

- ◆ **阶跃函数的缺陷**：不连续、不平滑，它在0点处的导数是无穷大，除了0点处之外，导数都是0，这意味着：若学习算法采用基于梯度的优化方法，是不可行的。





MP模型

- ◆ 在MP模型中**引入激活函数的目的是**：用于模拟生物神经元的工作机制，当电位高于一个设定的阈值时，则进入兴奋状态，输出信号；否则进入抑制状态，不输出信号。
- ◆ MP模型中的输入和输出数据只能是二值化数据0或1，而且网络中的权重、阈值等参数都需要**人为设置**，无法从数据中学习得到。
- ◆ MP模型的激活函数是一个简单的阶跃函数。
- ◆ MP模型只能处理一些简单的分类任务，例如线性二分类问题，但无法解决线性不可分问题。

M-P模型实现与门电路逻辑运算的应用案例

当设置 $w_1=0.5$, $w_2=0.5$, $\theta=0.8$ 时, 就可以用来实现与门电路的逻辑运算:

1) x_1, x_2 分别输入 0, 0 时, M-P 模型会输出值 0, 计算如下:

$$y=f(w_1x_1+w_2x_2-\theta)=f(0.5*0+0.5*0-0.8)=f(-0.8)=0;$$

2) x_1, x_2 分别输入 0, 1 时, M-P 模型会输出值 0, 计算如下:

$$y=f(w_1x_1+w_2x_2-\theta)=f(0.5*0+0.5*1-0.8)=f(-0.3)=0;$$

3) x_1, x_2 分别输入 1, 0 时, M-P 模型会输出值 0, 计算如下:

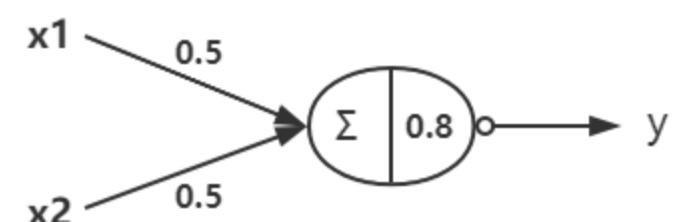
$$y=f(w_1x_1+w_2x_2-\theta)=f(0.5*1+0.5*0-0.8)=f(-0.3)=0;$$

4) x_1, x_2 分别输入 1, 1 时, M-P 模型会输出值 1, 计算如下:

$$y=f(w_1x_1+w_2x_2-\theta)=f(0.5*1+0.5*1-0.8)=f(0.2)=1。$$

表 3.1 与门真值表

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

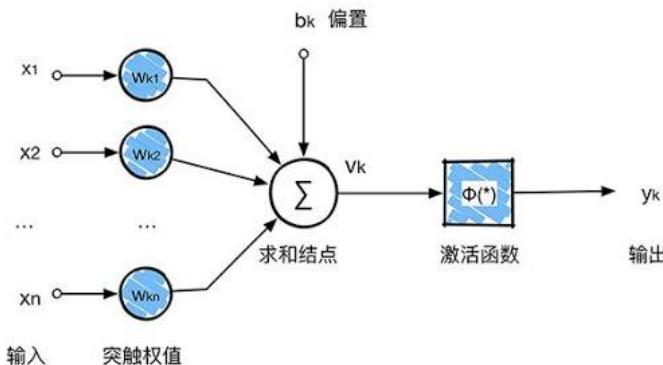


感知机

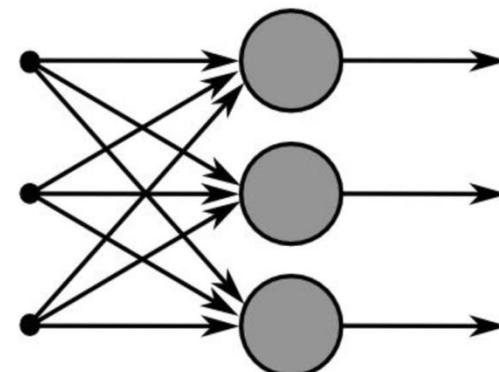
感知机

◆ **简单感知机**:简单感知机模型实际上仍然是M-P模型的结构。它是一种单层感知机模型,一层为输入层(只负责接收输入信号,无信息处理能力),另一层具有计算单元,可以通过采用监督学习来逐步增强模式划分的能力,达到学习的目的。

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



神经元的堆叠和连接



(a) 单层神经网络



感知机

- ◆ 由一层神经元构成的神经网络称为**感知机**，也称为**单层神经网络**。
- ◆ 1957年提出的感知机是**第一个工程实现的人工神经网络**，可以运行感知机学习算法来训练模型。
- ◆ 感知机是一种简单的**非线性神经网络**，是人工神经网络的基础。
- ◆ 感知机只包含输入层和输出层，其输入层可以包含多个单元，而输出层只有一个单元。
- ◆ 感知机通过采用**有监督学习**来逐步增强模式分类的能力，达到学习的目的。



感知机

- ◆ Rosenblatt在M-P模型基础上，给出了两层感知器的收敛定理，提出了具有自学习能力的神经网络模型——感知器模型，神经网络从纯理论走向了实际的工程应用。
- ◆ 神经网络默认的方向是从输入到输出，信息可以看成是向前传递的，因此从输入到输出进行信息传递的多层神经网络也称为多层次前馈型神经网络。

感知机学习过程

■ 感知机学习过程和权值更新规则：

第一步，随机初始化权值 $W(w_0, w_1, w_2, \dots, w_n)$ ，为了描述的统一，用 w_0 代替 b 。

第二步，输入一个样本 $(1, x_1, x_2, \dots, x_n)$ 和对应的期望结果 y 。其中 1 与 w_0 相乘表示偏置值，传入神经元。二分类中，一般用 $y=1$ 表示一类，用 $y=0$ 表示另一类，（也有的用 -1 表示另一类别，都不影响计算）。

$$f\left(\sum_{i=0}^n w_i \cdot x_i\right)$$

第三步，根据公式计算感知机输出结果为 $y_{out} =$

第四步，若该点被分类错误，则存在误差 $(\varepsilon = y - y_{out} \neq 0)$ ，以误差为基础对每个权值 w_i ($0 \leq i \leq n$) 按以下规则进行调整（称为学习规则）：
 $\Delta w_i = \eta * (y - y_{out}) * x_i$

$$w_i \leftarrow w_i + \Delta w_i$$

第五步，如果所有的样本分类的输出均正确即成功分类，则训练过程结束。只要有任何一个样本输出错误，那么都将导致权值调整，并且再次逐个输入所有样本进行训练。

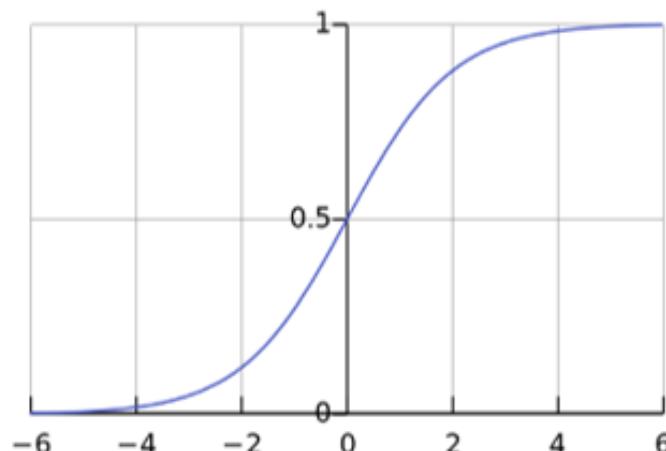


感知机与MP模型的异同点

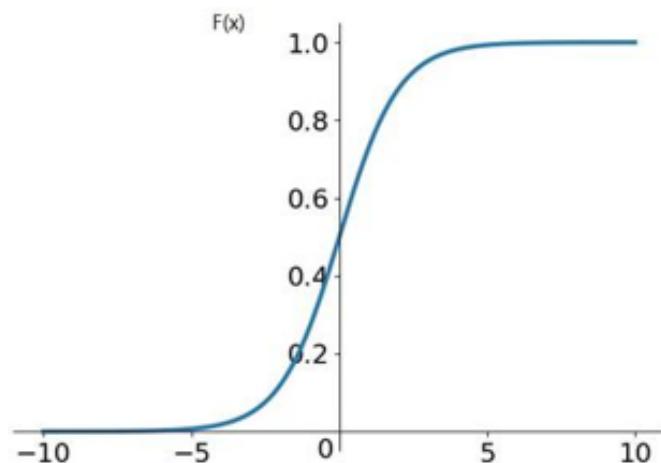
- ◆ 从本质上说，感知机与MP模型没有太大的区别，两者的结构相同，计算过程也相同，都能完成线性可分的二分类任务，也都无法解决线性不可分问题。
- ◆ 但MP模型与感知机的不同之处在于：
 - ① MP模型没有“学习”的机制，其权值 w 和偏置 θ 都是人为设定的；而感知机引入了“学习”的概念，权值 w 和偏置 θ 是通过学习得到的，并非人为设置的，在一定程度上模拟了人脑的“学习”功能，这也是两者最大的区别。
 - ② 两者采用的激活函数不同，MP模型采用阶跃函数作为激活函数，而感知机通常采用sigmoid函数作为激活函数。

激活函数

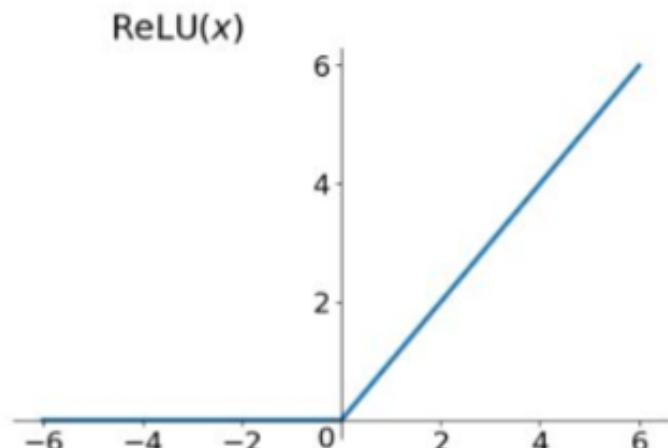
- ◆ 为了避免单纯的线性组合，一般在每一层的输出后面都增加一个函数变换（常见的如sigmoid、tanh、ReLU函数等等），这个函数称为**激活函数**。



(a) Sigmoid



(b) Tanh



(c) ReLu

图 10.5 常用的激活函数

1. sigmoid函数

◆ sigmoid函数，现在专指logistic函数。

◆ 特点：具有平滑性、连续性、单调性和渐近性，且连续可导。

◆ 2012年前，sigmoid是最常用的非线性激活函数，其输出值为(0,1)，表示概率或输入的归一化。

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

◆ sigmoid 函数的求导公式如下：

$$\begin{aligned}\frac{\partial \sigma(x)}{\partial x} &= -\frac{1}{(1+e^{-x})^2} \cdot e^{-x} \cdot (-1) = \frac{e^{-x}}{(1+e^{-x})^2}, \\ &= \frac{1}{1+e^{-x}} \cdot \left(1 - \frac{1}{1+e^{-x}}\right) = \sigma(x)(1 - \sigma(x))\end{aligned}$$

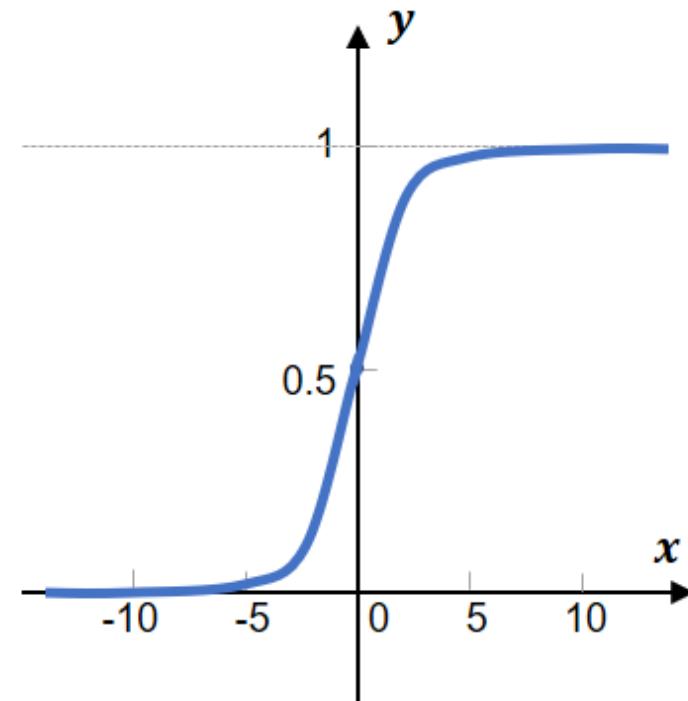


图5.3 sigmoid函数图像

◆ sigmoid函数的**优点**：

➤ 平滑、易于求导，其导数可直接用函数的输出计算，简单高效。

➤ sigmoid函数很好地解释了神经元在受到刺激的情况下是否被激活和向后传递的情景。

当取值接近0时，几乎没有被激活；当取值接近1时，几乎完全被激活。



1. sigmoid函数

◆ sigmoid函数的缺点：

- (1) 当输入的绝对值大于某个阈值时，会快速进入饱和状态（即函数值趋于1或 -1，不再有显著的变化，梯度趋于0），会出现梯度消失的情况，权重无法再更新，会导致算法收敛缓慢，甚至无法完成深层网络的训练。因此在一些现代的神经网络中，sigmoid函数逐渐被ReLU激活函数取代。
- (2) sigmoid函数公式中有幂函数，计算耗时长，在反向传播误差梯度时，求导运算涉及除法。
- (3) sigmoid函数的输出恒大于 0，非零中心化，在多层神经网络中，可能会造成后面层神经元的输入发生偏置偏移，导致梯度下降变慢。

2. tanh函数

- ◆ tanh函数是sigmoid函数的一个变形，称为双曲正切函数。
- ◆ tanh函数的值域为(-1,1)，改进了sigmoid变化过于平缓的问题，且其输出是零中心化的，解决了sigmoid函数的偏置偏移问题。
- ◆ tanh函数的数学表达式： $y = f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2\sigma(2x) - 1$
- ◆ tanh可看作是在纵轴方向上放大到2倍并向下平移的sigmoid函数。
- ◆ tanh函数的导数公式：

$$\frac{\partial y}{\partial x} = -\frac{(e^x + e^{-x})(e^x + e^{-x}) - (e^x - e^{-x})(e^x - e^{-x})}{(e^x + e^{-x})^2} = 1 - y^2$$

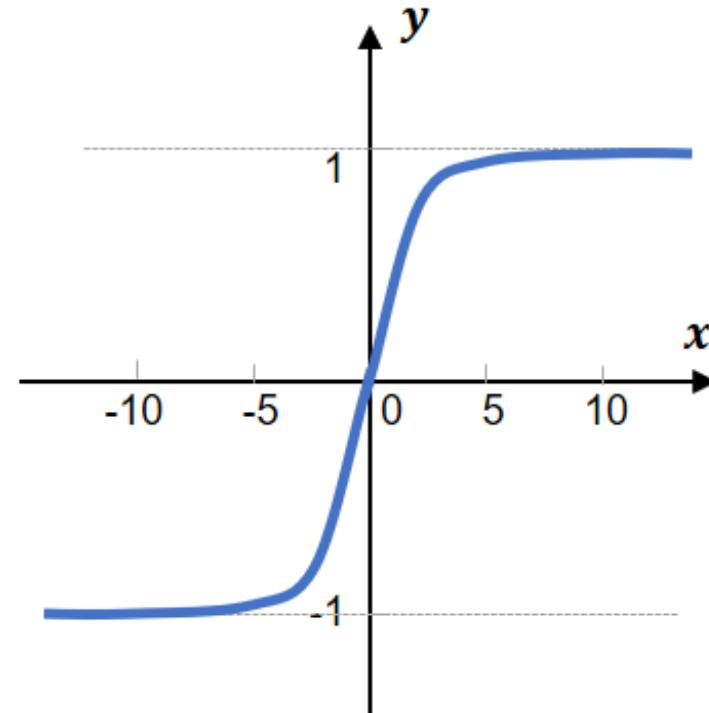


图5.4 tanh 函数图像

- ◆ tanh函数的**优点**：tanh 在线性区的梯度更大，能加快神经网络的收敛。
- ◆ tanh函数的**缺点**：其两端的梯度也趋于零，依旧存在梯度消失的问题，同时，幂运算也会导致计算耗时长。

感知机的学习过程

权值调整是让误差减小为依据的，因此能够有效的学习到合理的权值参数。还有其他特征：

1. 对某个样本错误进行权值参数调整，可能引起其他原本分类正确的点出错。所以每次调整之后，要把所有样本都要重新进行依次计算。
2. 合适的学习率对于训练来说非常重要。学习率大，调整力度大，可能让分割直线调整过头，引起其他点错误，从而引起分割直线来回振荡调整。学习率小，整力度小，可能要反复训练多轮。
3. 一些点误差是1，一些点误差是-1，会让分隔直线来回移动。可以把所有点都计算一遍，把所有点的误差求和，然后对权值进行一次调整，实现每次调整让总体误差最小。
4. 能线性分割这些点的直线非常多，根据初始参数和学习率，最终得到的权值参数结果不唯一。
5. 如果这些点本身不是用一条直线可以分割的（非线性可分），即没有一条直线能把这些点正确分开，那么这个学习过程无法终止。因此，它无法解决线性不可分问题。



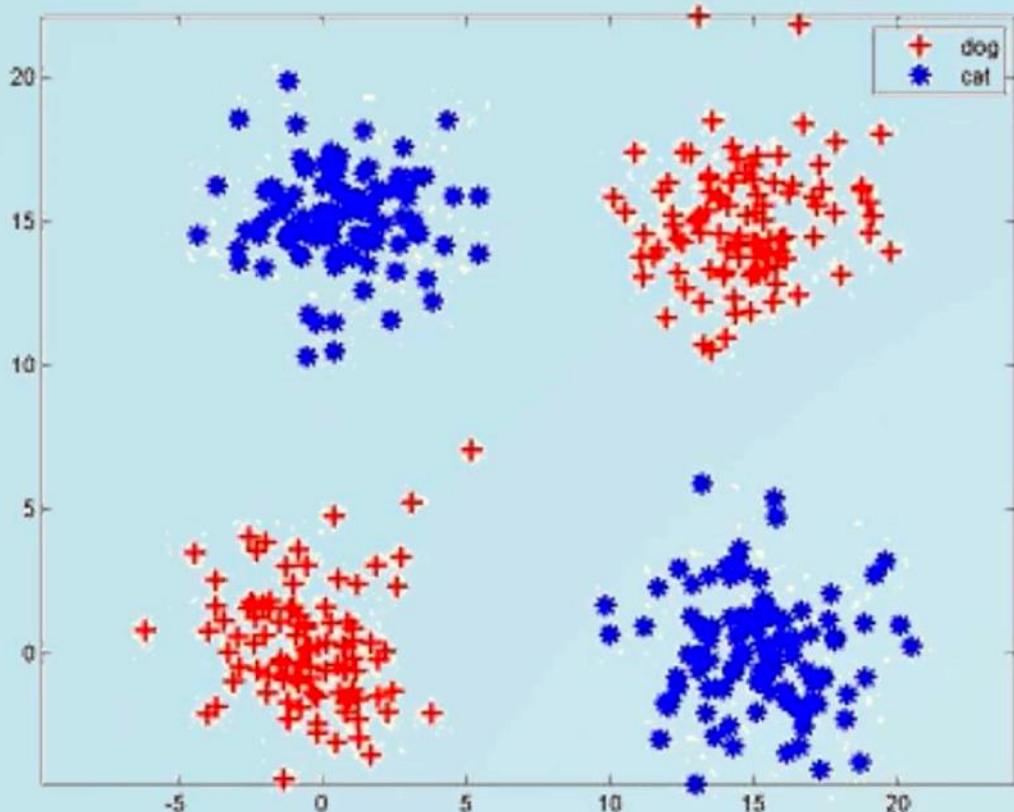
感知机

◆ 感知机算法实质上是一种赏罚过程：

- 对正确分类的模式则“赏”，实际上是“不罚”，即权向量不变。
- 对错误分类的模式则“罚”，需要对权向量进行相应的更新。
- 当用全部模式样本训练过一轮以后，只要有一个模式是判别错误的，则需要进行下一轮迭代，即用全部模式样本再训练一次。
- 如此不断反复直到全部模式样本进行训练都能得到正确的分类结果为止。

感知机的缺陷

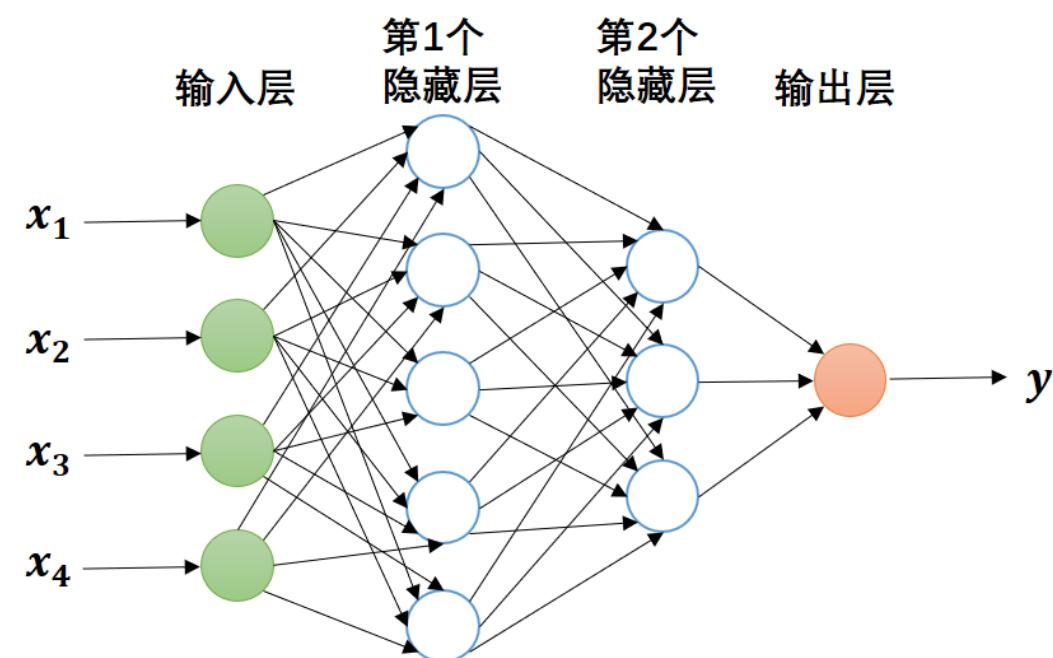
1969年：Minsky和Papert指出感知机的缺陷：仅能解决一阶谓词逻辑，即**只能完成线性划分**，对于非线性或者其他分类会遇到很多困难，就连简单的 XOR（异或）问题都解决不了。



对于左图中的分类问题，单层感知机模型无法找到一个超平面将两类样本区分开。

多层神经网络结构

- ◆ 单个人工神经元的结构简单，功能有限。
- ◆ 若想完成复杂的功能，就需要将许多人工神经元按照一定的拓扑结构相互连接在一起，相互传递信息，协调合作。
- ◆ 组成神经网络的所有神经元是分层排列的，一个神经网络包括输入层、**隐藏层**（也称为**隐层**、**隐含层**）和输出层。
- ◆ 每个网络只能有一个**输入层**和一个**输出层**，却可以有**0个或多个隐藏层**，每个隐藏层上可以有若干个神经元。
- ◆ 只有输入层与输出层的神经元可以与外界相连，外界无法直接触及隐藏层，故而得名“**隐藏层**”。





多层神经网络结构

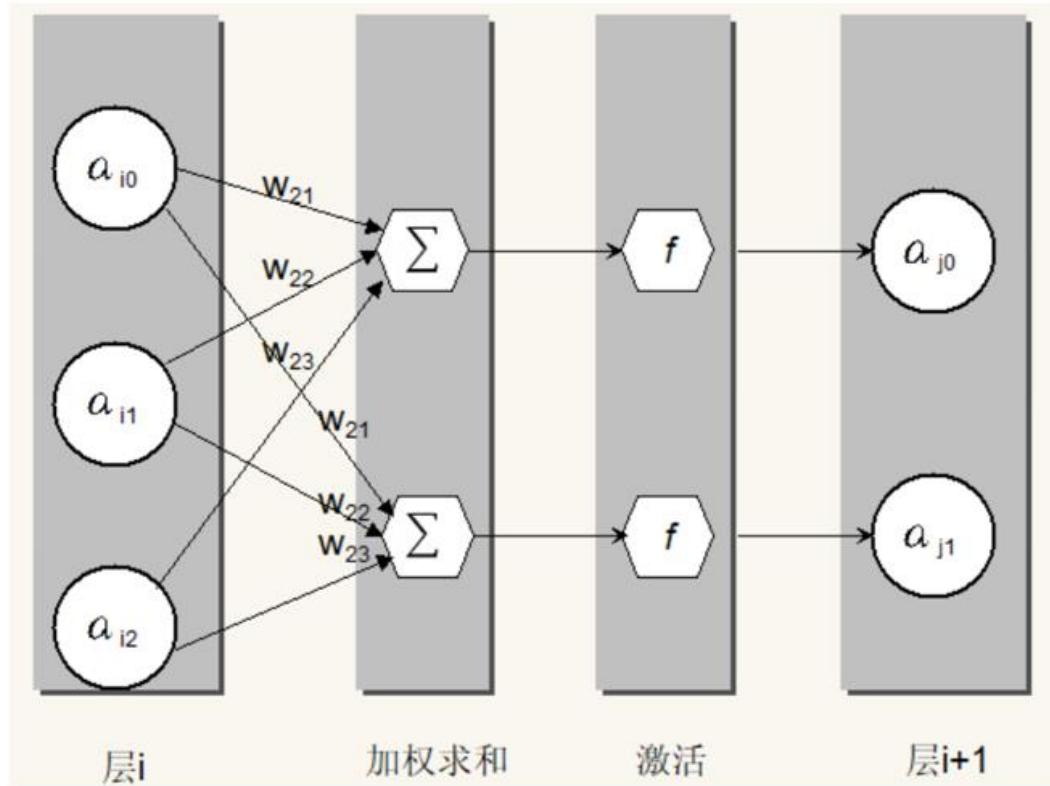
- ◆ 包含至少一个隐藏层的神经网络称为**多层神经网络**。
- ◆ 在包含神经元个数最多的一层，神经元的个数称为该**神经网络的宽度**。
- ◆ 除输入层外，其他层的层数称为**神经网络的深度**，即等于隐藏层数加1（输出层）。
- ◆ 感知机没有隐藏层，只有输入层和输出层，因此感知机的层数为1，称为**单层神经网络**。
- ◆ 人工神经网络的行为并非各个神经元行为的简单相加，而是具有学习能力的、行为复杂的非线性系统，既可以提取和表达样本的高维特征，又可以完成复杂的预测任务。



多层神经网络结构

- ◆ 多层神经网络的每个隐藏层后面都有一个**非线性的激活函数**。
- ◆ 这里激活函数的作用比感知机中作为激活函数的阶跃函数的作用要大得多，因为激活函数是对所有输入信号的线性组合结果进行非线性变换，而且多层神经网络就有多个激活函数。
- ◆ **激活函数最主要的作用**是向模型中加入非线性元素，用以解决非线性问题。
- ◆ 一般在同一个网络中使用同一种激活函数。

有激活层的神经网络模型



- ◆ 由于激活层的存在，输入数据在加权求和后，没有直接向前传递，而是先经过激活运算，然后再传递到j层。激活函数能起到“激活”神经网络的作用。

- ◆如果一个神经网络中，每一层中每个神经元都和下一层的所有神经元相连，则这个神经网络是全连接的（full connected），称为**全连接神经网络**。

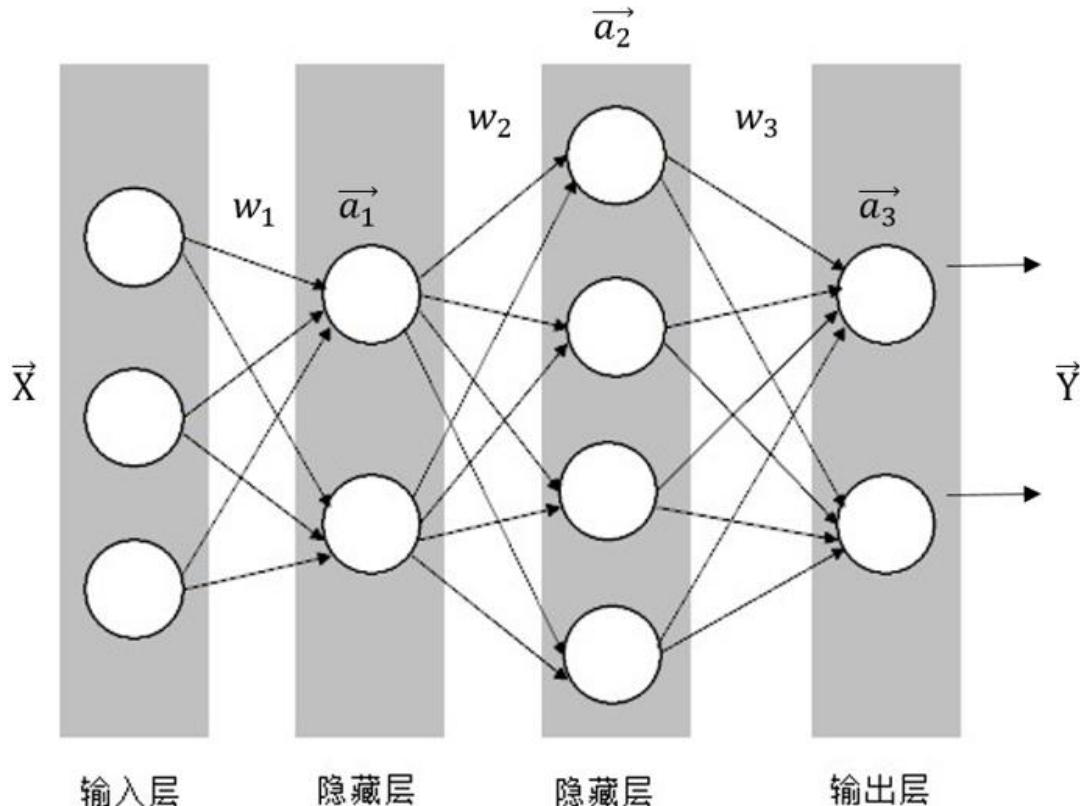


图 10.4 有两个隐藏层的神经网络

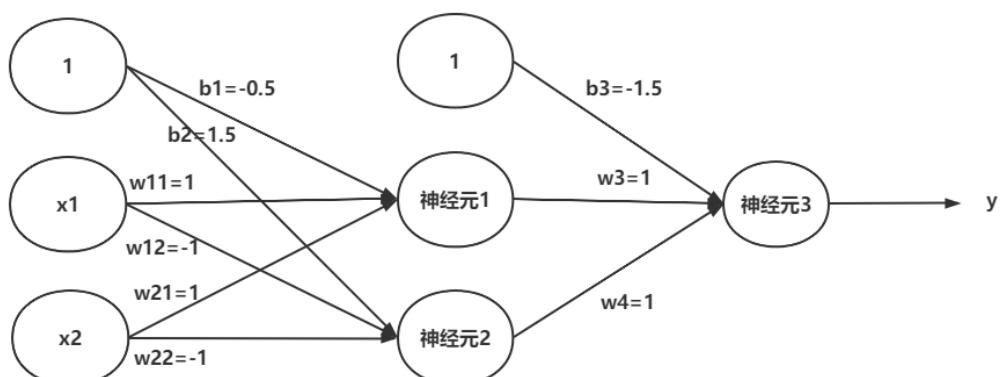
多层感知机

■ 多层感知机实现异或运算：

表 3.4 异或逻辑运算真值表

x ₁	x ₂	y
0	0	0
0	1	1
1	0	1
1	1	0

二层感知机结构如下图所示，设置权值参数 $w_{11}=1, w_{21}=1, w_{12}=-1, w_{22}=-1, w_3=1, w_4=1, b_1=-0.5, b_2=1.5, b_3=-1.5$ ，激活函数 $f(x)$ 为阶跃函数。



将样本 $x_1, x_2 = (0, 0)$ 输入模型，输出 $y=0$ ，计算如下：

$$\text{神经元1输出 } s_1 = f(w_{11}x_1 + w_{21}x_2 + b_1) = f(1*0 + 1*0 - 0.5) = 0$$

$$\text{神经元2输出 } s_2 = f(w_{12}x_1 + w_{22}x_2 + b_2) = f(-1*0 + -1*0 + 1.5) = 1$$

$$\text{神经元3输出 } y = f(w_3s_1 + w_4s_2 + b_3) = f(1*0 + 1*1 - 1.5) = 0$$

将样本 $x_1, x_2 = (0, 1)$ 输入模型，输出 $y=1$ ，计算如下：

$$s_1 = f(w_{11}x_1 + w_{21}x_2 + b_1) = f(1*0 + 1*1 - 0.5) = 1$$

$$s_2 = f(w_{12}x_1 + w_{22}x_2 + b_2) = f(-1*0 + -1*1 + 1.5) = 1$$

$$y = f(w_3s_1 + w_4s_2 + b_3) = f(1*1 + 1*1 - 1.5) = 1$$

将样本 $x_1, x_2 = (1, 0)$ 输入模型，输出 $y=1$ ，计算如下：

$$s_1 = f(w_{11}x_1 + w_{21}x_2 + b_1) = f(1*1 + 1*0 - 0.5) = 1$$

$$s_2 = f(w_{12}x_1 + w_{22}x_2 + b_2) = f(-1*1 + -1*0 + 1.5) = 1$$

$$y = f(w_3s_1 + w_4s_2 + b_3) = f(1*1 + 1*1 - 1.5) = 1$$

将样本 $x_1, x_2 = (1, 1)$ 输入模型，输出 $y=0$ ，计算如下：

$$s_1 = f(w_{11}x_1 + w_{21}x_2 + b_1) = f(1*1 + 1*1 - 0.5) = 1$$

$$s_2 = f(w_{12}x_1 + w_{22}x_2 + b_2) = f(-1*1 + -1*1 + 1.5) = 0$$

$$y = f(w_3s_1 + w_4s_2 + b_3) = f(1*1 + 1*0 - 1.5) = 0$$

多层神经网络结构

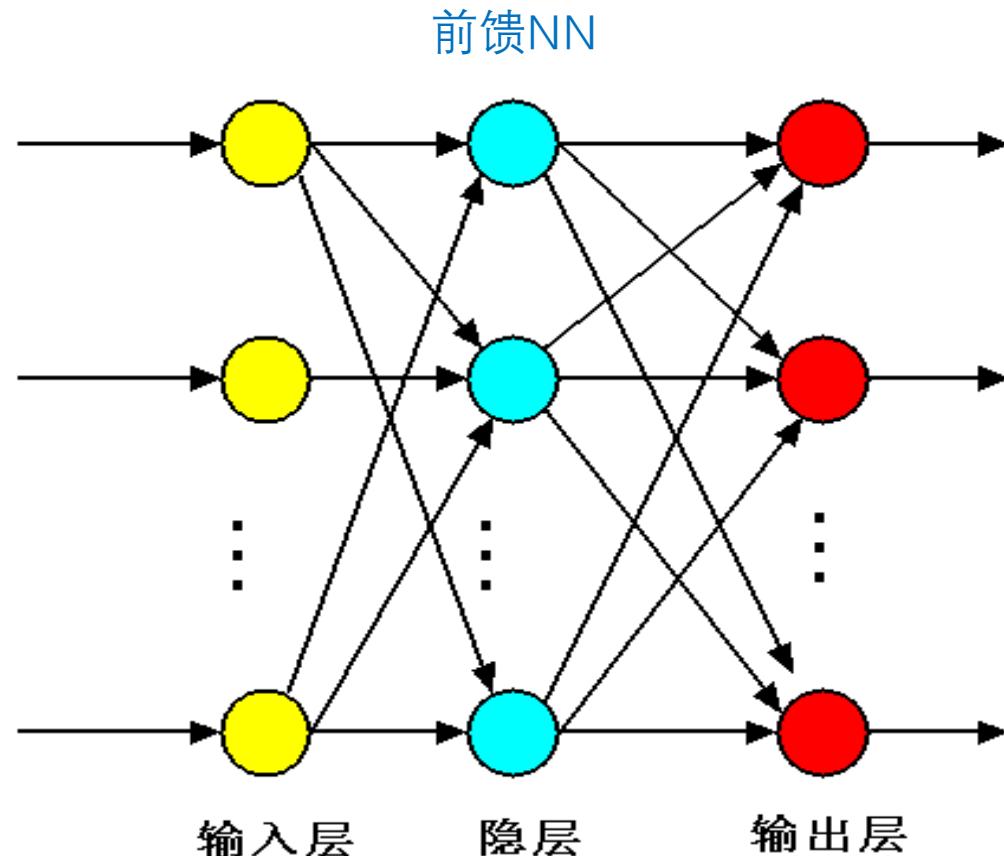
根据**网络层之间的连接方式**，又可以将多层人工神经网络分为**前馈神经网络**和**反馈神经网络**。

1. 前馈神经网络

- ◆ 前馈神经网络 (Feedforward Neural Network) 是一种多层神经网络，其中每个神经元只与其相邻层上的神经元相连，接收前一层的输出，并输出给下一层，即第 i 层神经元以第 $i-1$ 层神经元的输出作为输入，第 i 层神经元的输出作为第 $i+1$ 层神经元的输入。
- ◆ 同层的神经元之间没有连接。
- ◆ 整个网络中的信息是单向传递的，即只能按一个方向从输入层到输出层的方向传播，没有反向的信息传播，可以用一个**有向无环图**表示。

5.2.4 多层神经网络结构

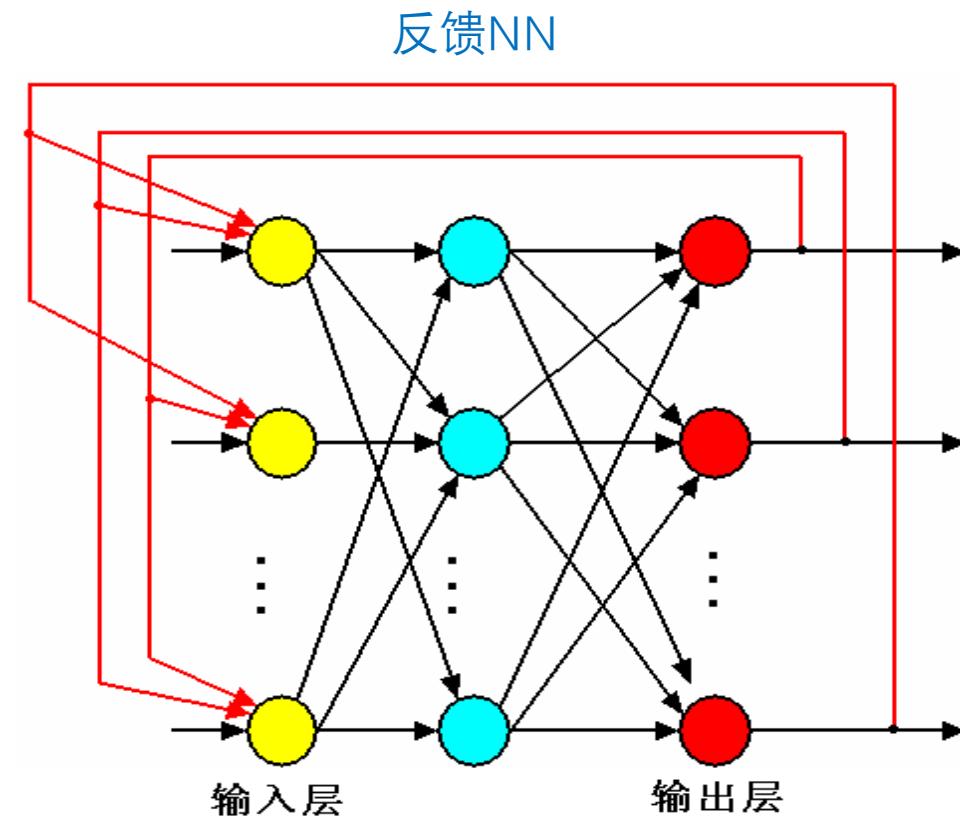
- ◆ 若前馈神经网络采用全连接方式，则称为**前馈全连接神经网络**。
- ◆ 前馈神经网络的**优点**：网络结构简单，易于实现。
- ◆ 前馈全连接神经网络的**缺点**是：当网络很深时，参数量巨大，计算量大，训练耗时。
- ◆ 前馈神经网络包括：**BP神经网络**和**卷积神经网络**。



5.2.4 多层神经网络结构

2. 反馈神经网络

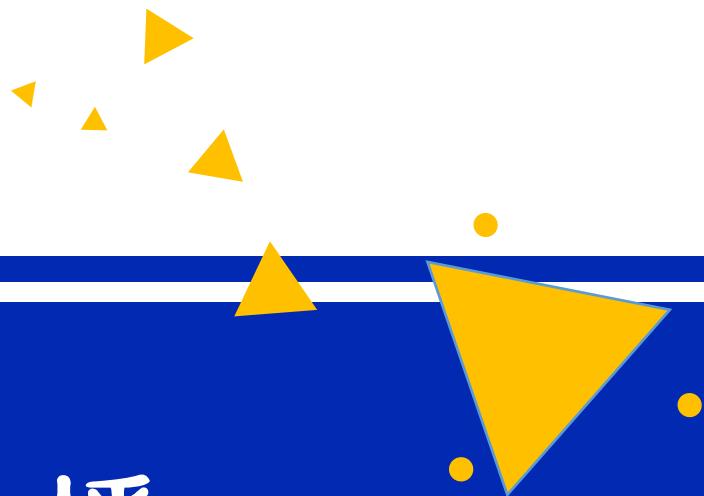
- ◆ 反馈神经网络 (Feedback Neural Network) 是一种反馈动力学系统。
- ◆ 在这种网络中，有些神经元不但可以接收其前一层上神经元的信息，还可以接收来自于其后面上神经元的信息。
- ◆ 神经元的连接可以形成**有向循环**。
- ◆ 反馈神经网络中的**信息既可以单向传递，也可以双向传递**，且神经元具有记忆功能，在不同时刻具有不同的状态，能建立网络的内部状态，可展现动态的时间特性。





A Neural Network Playground

<http://playground.tensorflow.org/>



正向传播和反向传播

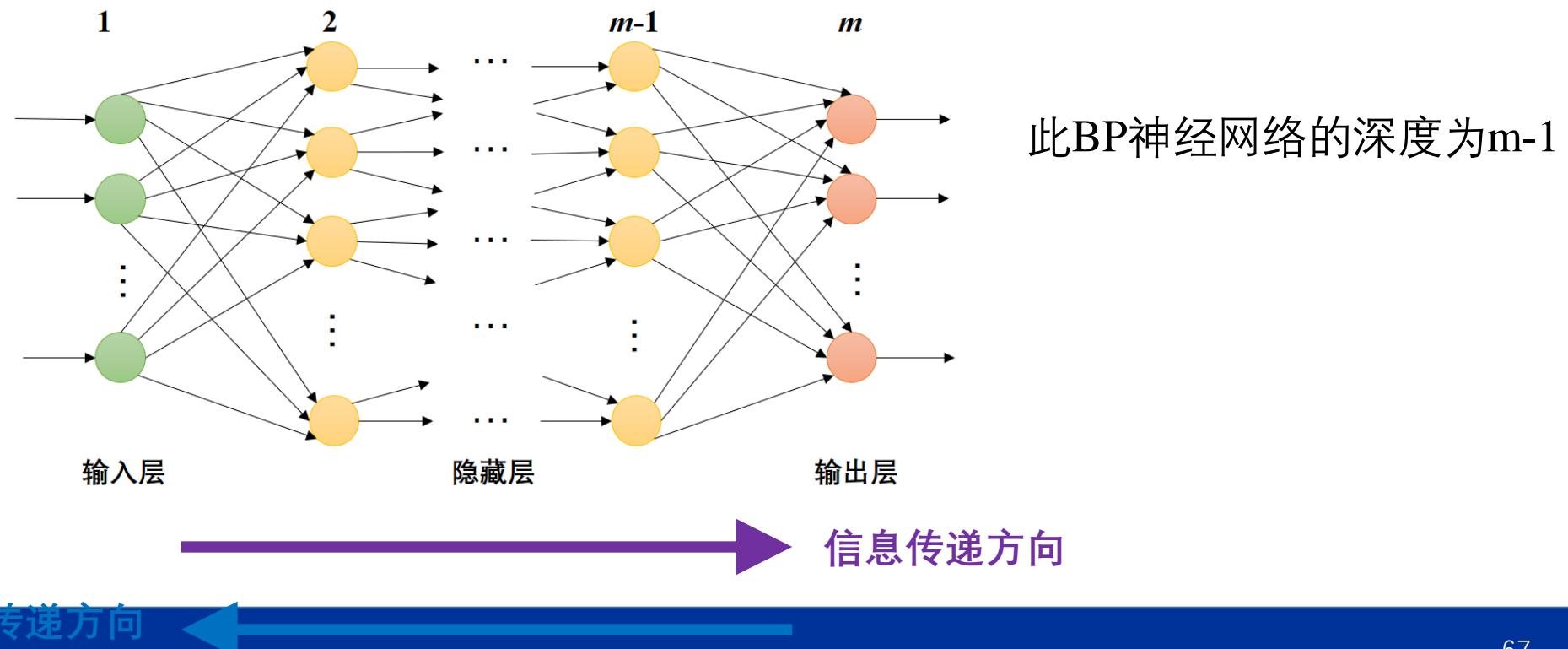


5.3 BP神经网络及其学习算法

- ◆ 多层前馈神经网络的表达能力比单层感知机要强得多。
- ◆ 要训练多层前馈神经网络，**单层感知机的学习算法是远远不够的**，需要更强大的学习算法。
- ◆ 1974年Werbos提出了BP算法；1986年辛顿等人又重新独立地提出了BP算法。
- ◆ 迄今为止，最成功的多层神经网络学习算法就是**反向传播（Back-Propagation, BP）算法**。
- ◆ BP算法能解决对参数逐一求偏导、计算效率低下的问题。
- ◆ 目前，学术界和产业界依然在用BP算法训练神经网络。

5.3.1 BP神经网络的结构

- ◆ 由于前馈神经网络大多采用反向传播学习算法来进行训练模型，故被称为**BP神经网络**。
- ◆ BP神经网络是一种**多层前馈神经网络**。
- ◆ 每个节点就是一个神经元，神经元之间带有箭头的连线表示信息传递的方向。



5.3.1 BP神经网络的结构

假设神经网络中第 $k-1$ 层中有 p_{k-1} 个神经元，令

- ◆ y_j^{k-1} 表示第 $k-1$ 层中第 j 个神经元的输出，
- ◆ w_{ji}^k 表示第 $k-1$ 层的第 j 个神经元与第 k 层的第 i 个神经元之间的连接权值，
- ◆ θ^k 表示第 k 层的偏置，
- ◆ u_i^k 表示第 k 层的第 i 个神经元的净输入，
- ◆ 则 u_i^k 的计算表达式为： $u_i^k = \sum_{j=1}^{p_{k-1}} w_{ji}^k y_j^{k-1} + \theta^k$, $k = 2, \dots, m$
- ◆ 为使公式整齐，令 $w_{0i}^k = \theta^k$, $y_0^{k-1} = 1$ ，则有： $u_i^k = \sum_{j=0}^{p_{k-1}} w_{ji}^k y_j^{k-1}$, $k = 2, \dots, m$
- ◆ 令 $f(\cdot)$ 表示激活函数， y_i^k 表示第 k 层的第 i 个神经元的输出，其值为： $y_i^k = f(u_i^k)$

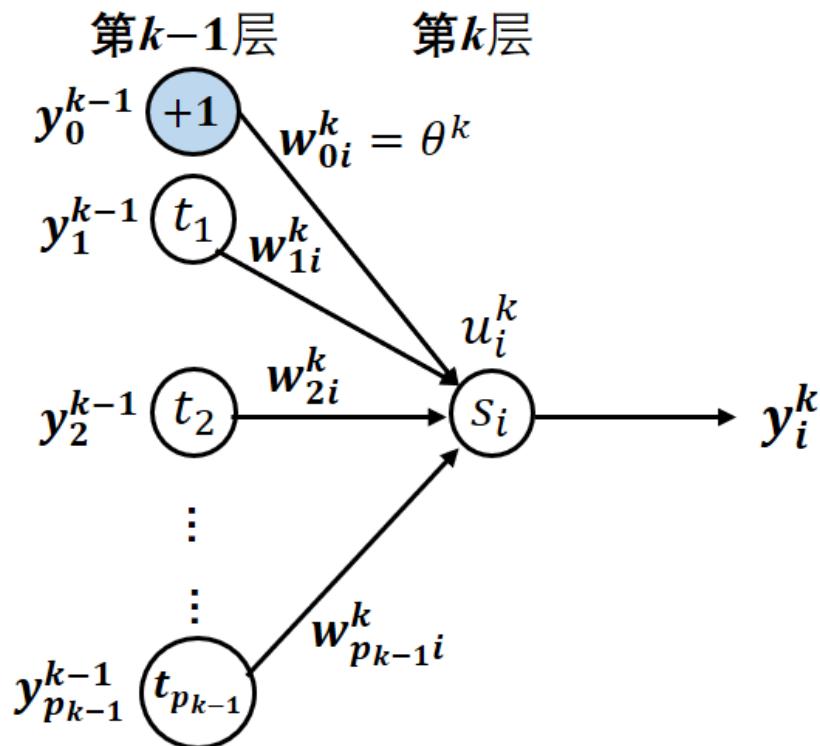


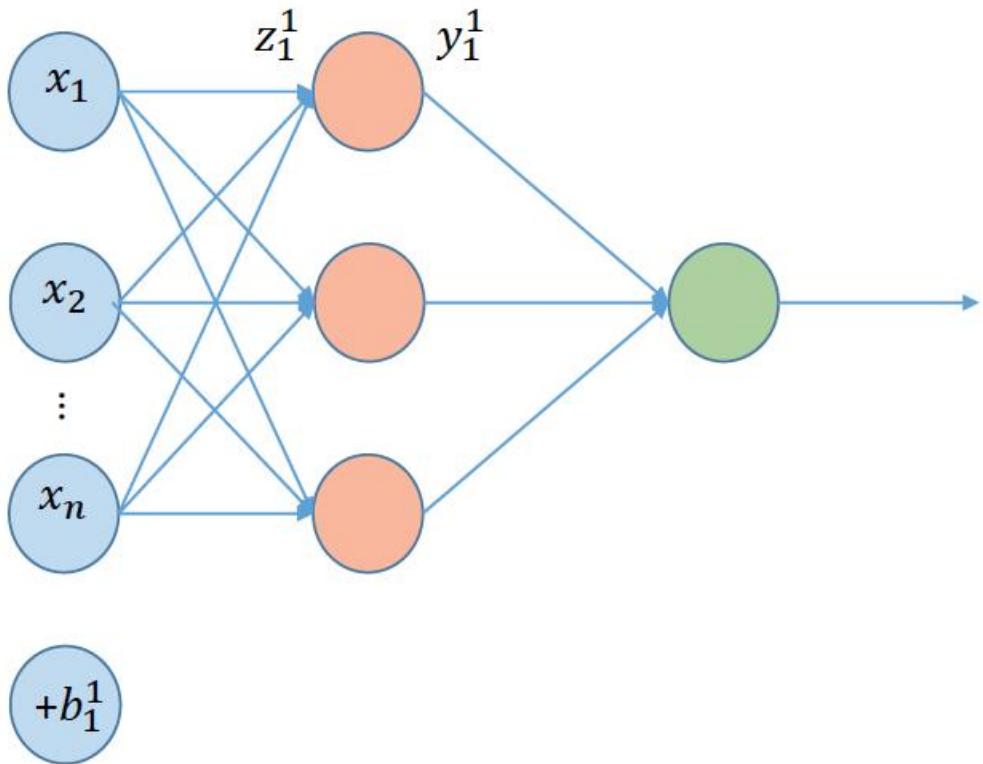
图5.7 BP网络中相邻两层的各变量之间关系



参数学习

★ 信号正向传播

- 假定第 l 层神经元状态为 z^l , 经激活函数后的输出值为 y^l

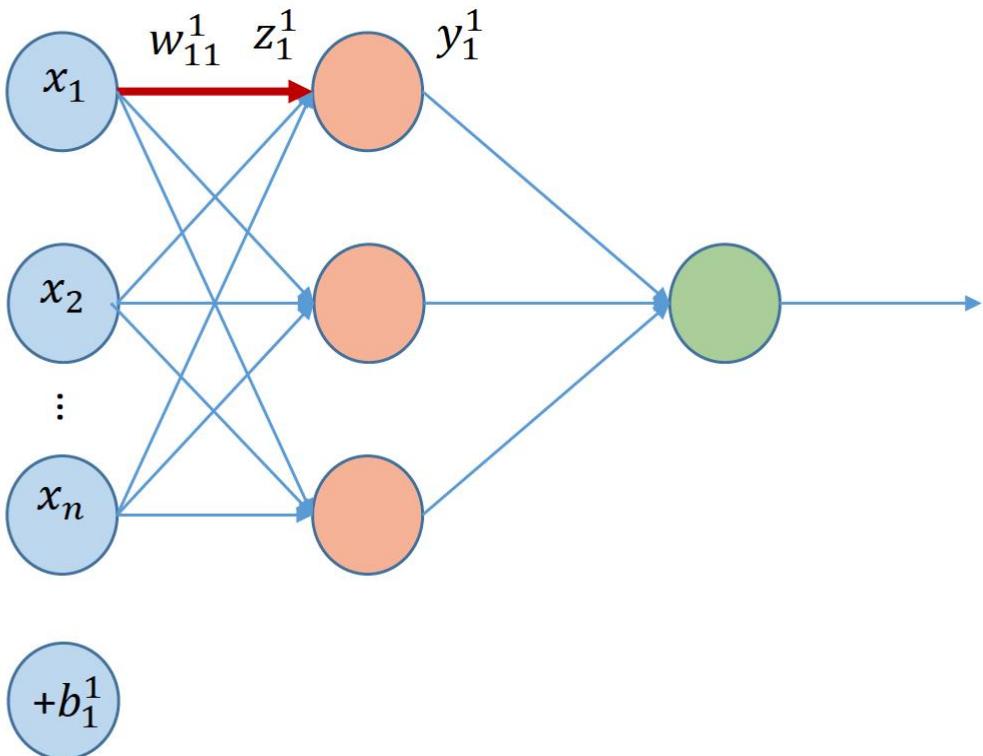




参数学习

★ 信号正向传播

- 假定第 l 层神经元状态为 z^l , 经激活函数后的输出值为 y^l



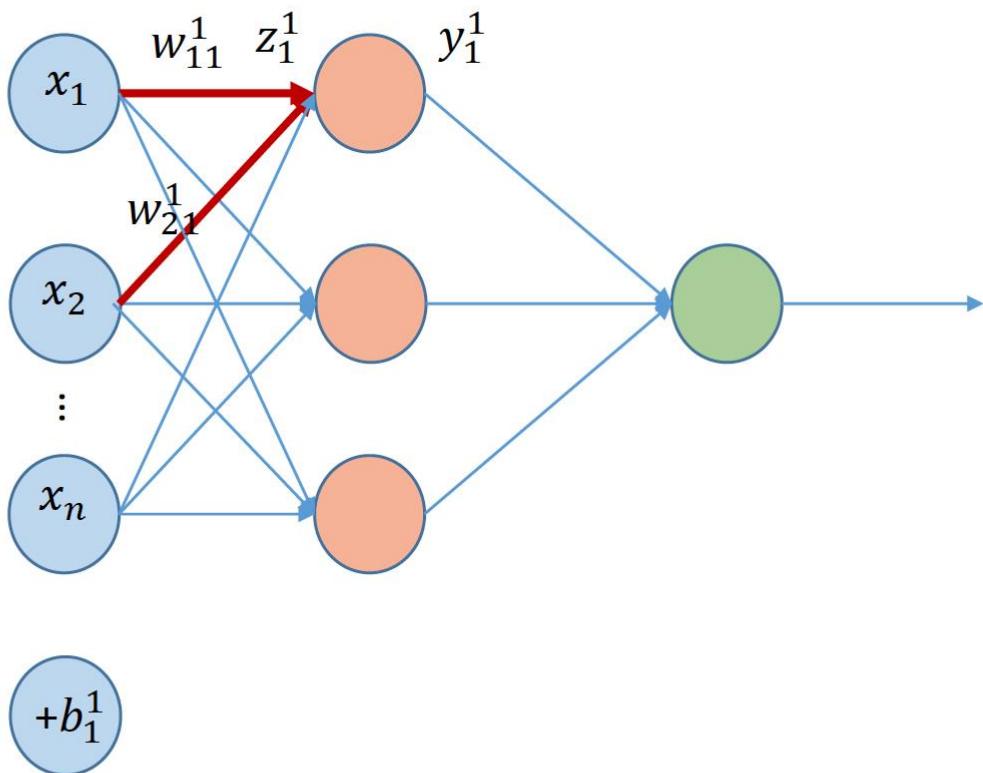
$$\bullet \ z_1^1 = w_{11}^1 x_1$$



参数学习

★ 信号正向传播

- 假定第 l 层神经元状态为 z^l , 经激活函数后的输出值为 y^l



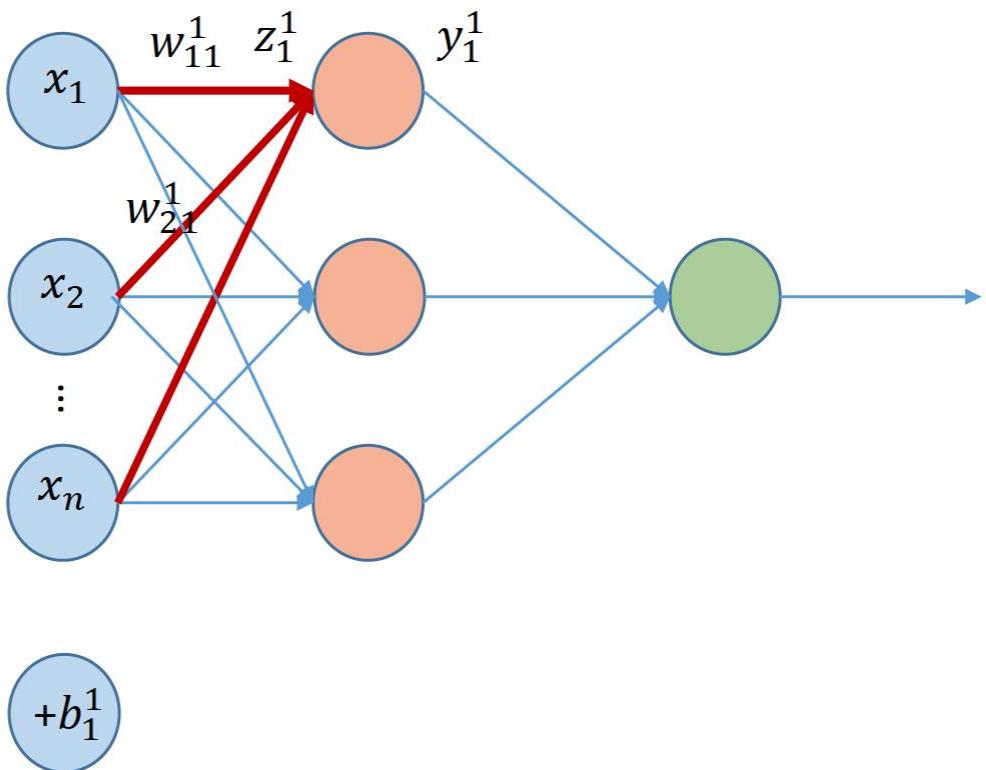
$$\bullet \ z_1^1 = w_{11}^1 x_1 + w_{21}^1 x_2$$



参数学习

★ 信号正向传播

- 假定第 l 层神经元状态为 z^l , 经激活函数后的输出值为 y^l



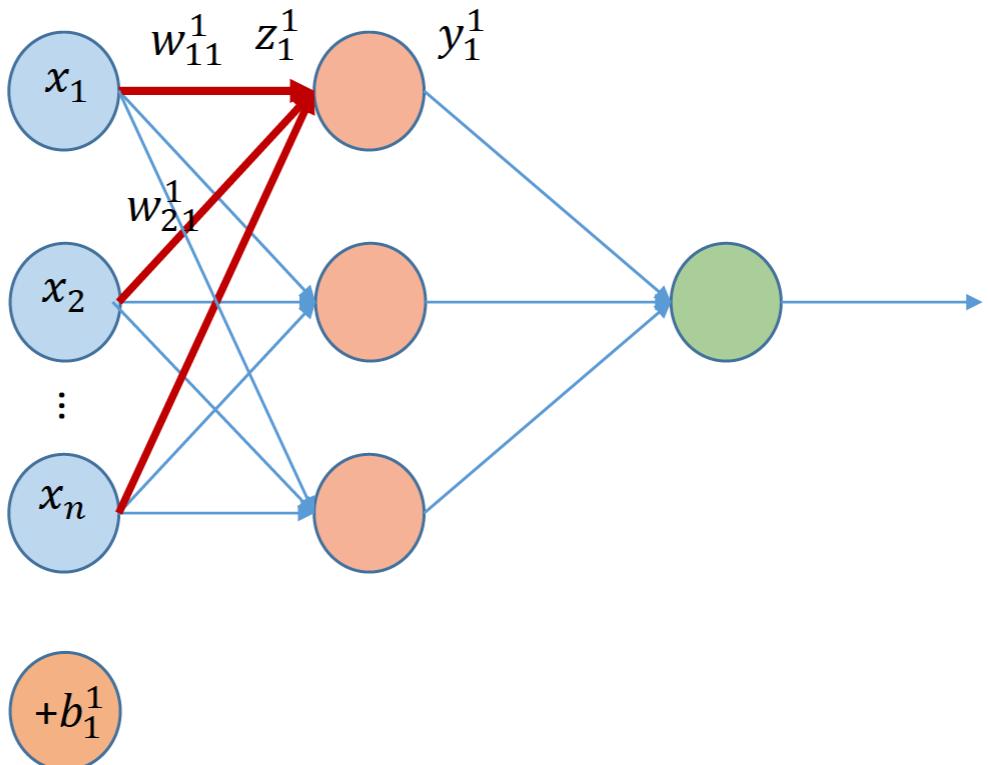
$$\bullet \ z_1^1 = w_{11}^1 x_1 + w_{21}^1 x_2 + \dots + w_{n1}^1 x_n$$



参数学习

★ 信号正向传播

- 假定第 l 层神经元状态为 z^l , 经激活函数后的输出值为 y^l



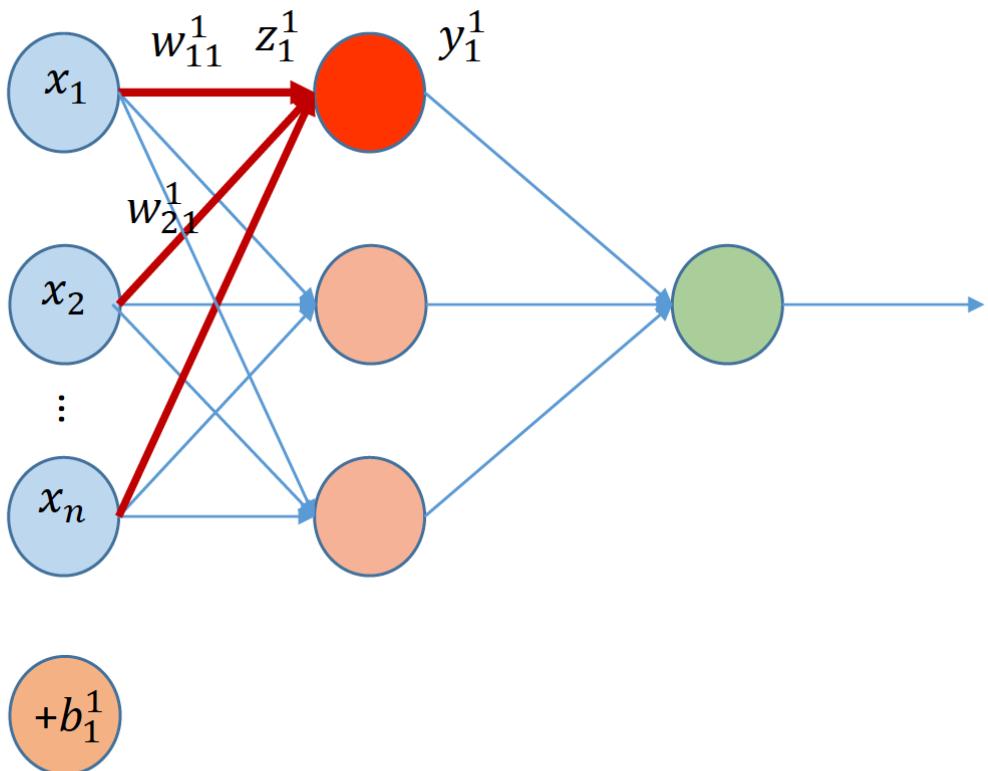
$$\bullet \ z_1^1 = w_{11}^1 x_1 + w_{21}^1 x_2 + \dots + w_{n1}^1 x_n + b_1^1$$



参数学习

信号正向传播

- 假定第 l 层神经元状态为 z^l , 经激活函数后的输出值为 y^l



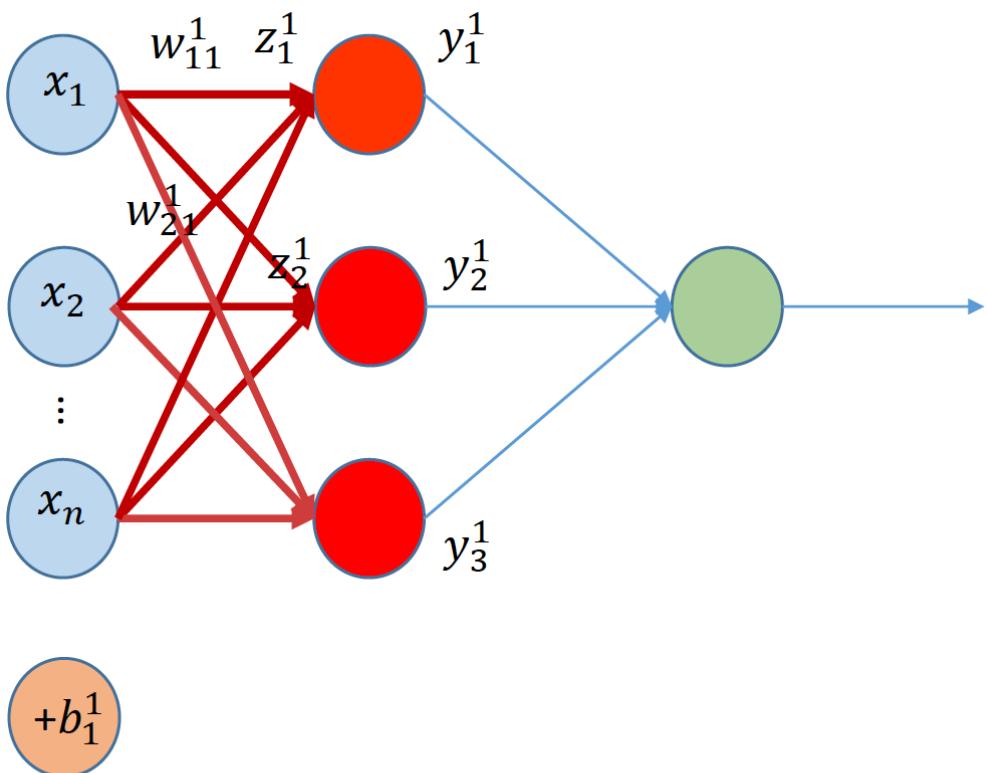
- $z_1^1 = w_{11}^1 x_1 + w_{21}^1 x_2 + \cdots + w_{n1}^1 x_n + b_1^1$
 - $y_1^1 = f(z_1^1) = f(w_{11}^1 x_1 + w_{21}^1 x_2 + \cdots + w_{n1}^1 x_n + b_1^1)$



参数学习

★ 信号正向传播

- 假定第 l 层神经元状态为 z^l , 经激活函数后的输出值为 y^l



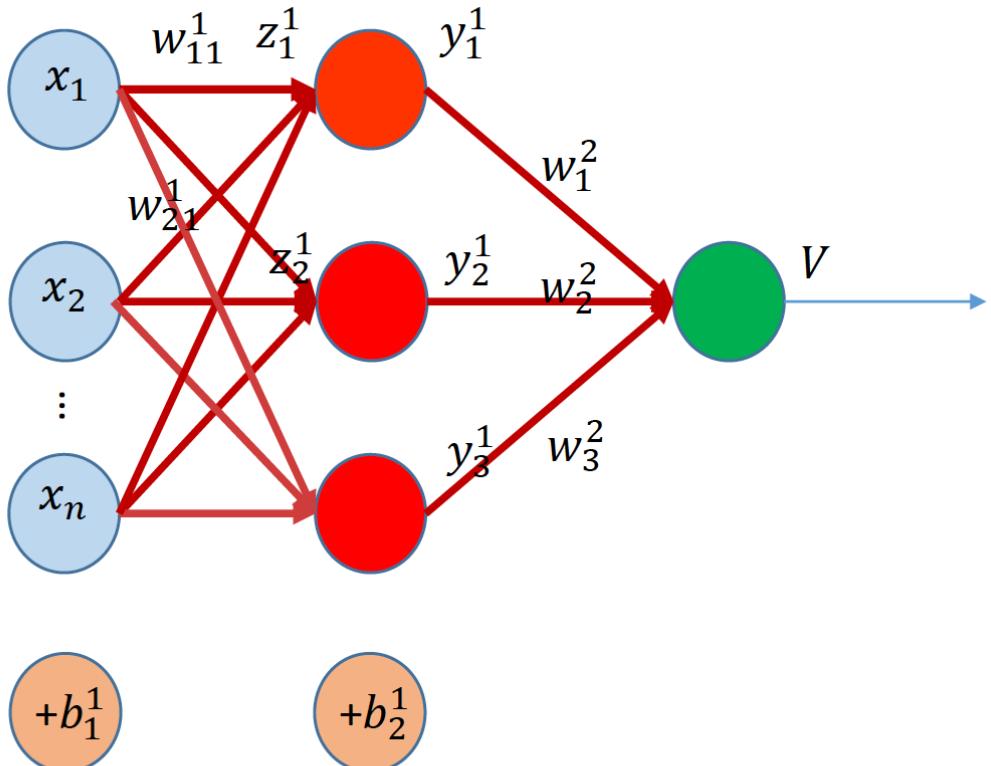
- $z_1^1 = w_{11}^1 x_1 + w_{21}^1 x_2 + \cdots + w_{n1}^1 x_n + b_1^1$
 - $y_1^1 = f(z_1^1) = f(w_{11}^1 x_1 + w_{21}^1 x_2 + \cdots + w_{n1}^1 x_n + b_1^1)$
 - $y_2^1 = f(z_2^1) = f(w_{12}^1 x_1 + w_{22}^1 x_2 + \cdots + w_{n2}^1 x_n + b_2^1)$
 - $y_3^1 = f(z_3^1) = f(w_{13}^1 x_1 + w_{23}^1 x_2 + \cdots + w_{n3}^1 x_n + b_3^1)$



参数学习

★ 信号正向传播

- 假定第 l 层神经元状态为 z^l , 经激活函数后的输出值为 y^l



- $V = f(w_1^2 y_1^1 + w_2^2 y_2^1 + \dots + w_3^2 y_3^1 + b_2)$

其中：

 - $y_1^1 = f(z_1^1) = f(w_{11}^1 x_1 + w_{21}^1 x_2 + \dots + w_{n1}^1 x_n + b_1^1)$
 - $y_2^1 = f(z_2^1) = f(w_{12}^1 x_1 + w_{22}^1 x_2 + \dots + w_{n2}^1 x_n + b_2^1)$
 - $y_3^1 = f(z_3^1) = f(w_{13}^1 x_1 + w_{23}^1 x_2 + \dots + w_{n3}^1 x_n + b_3^1)$

5.3.2 BP学习算法

- ◆ 神经网络学习，也称为神经网络训练，是指利用训练数据集不断地修改神经网络的所有连接权值和偏置值，使神经网络的实际输出尽可能地逼近真实值。
- ◆ BP学习算法是一种有监督学习。
- ◆ 权重 w 和偏置 θ 是未知的，需要采用损失函数来指导学习算法来更新这些参数。
- ◆ 假设有 $m-1$ 层BP神经网络 F ，其输入数据 $X = [x_1, x_2, \dots, x_{p_1}]^T$ (p_1 为输入层神经元的个数)，输出数据 $Y = [y_1^m, y_2^m, \dots, y_{p_m}^m]^T$ (p_m 为输出层神经元的个数)。
- ◆ BP神经网络看成是一个从输入到输出的非线性映射 $Y=F(X)$ 。
- ◆ 要学习的网络参数：
 - 由第 $k-1$ 层的第 j 个神经元到第 k 层的第 i 个神经元的连接权值 $\{w_{ji}^k, k=2, \dots, m; i=1, \dots, p_k; j=1, \dots, p_{k-1}\}$ (p_k 表示第 k 层中神经元的个数)
 - 第 k 层的偏置 $\{\theta^k, k=2, \dots, m\}$

5.3.2 BP学习算法

◆ 假设选择神经网络的实际输出与期望输出之间的误差平方和作为损失函数，其

数学表达式为： $E = \frac{1}{2} \sum_{j=1}^{p_m} (y_j - y_j^m)^2$ (公式5.10)

- m 是输出层的层号，即输出层是第 m 层；
- p_m 是输出层中神经元的个数；
- y_j^m 是输出层第 j 个神经元的实际输出；
- y_j 是输出层第 j 个神经元的期望输出。

◆ BP学习算法的目的：使得目标函数（即损失函数） E 达到极小值。

◆ 损失函数的值越小，表示神经网的预测结果越接近真实值。

5.3.2 BP学习算法

BP算法的学习过程由正向传播和反向传播两个阶段组成，算法的实现过程如下。

第1步：用随机值初始化神经网络 F 的权重 W 和偏置 θ ；令 $l=1$ 。

第2步：若 $l \leq N$ (N 为训练样本总数)，向 F 输入第 l 个样本的 p_1 维向量 $X_l = (x_{l1}, \dots, x_{lp_1})$ ，经过一次**正向传播**，得到预测结果 $Y_l^m = F(X_l)$ ， Y_l^m 为一个 p_m 维的向量 $(y_{l1}^m, \dots, y_{lp_m}^m)$ ；否则，算法停止。

第3步：已知第 l 个样本的期望输出为 $Y_l = (y_{l1}, \dots, y_{lp_m})$ ，**计算输出层的损失函数** E 的值。

$$E = \frac{1}{2} \sum_{j=1}^{p_m} (y_j - y_j^m)^2$$

第4步：若 E 值小于设定的阈值，则 $l=l+1$ ，转向第2步；否则，进行**反向传播**，采用梯度下降法，依次计算每个权重和偏置的修正值，公式为：

$$\Delta w_{ji}^k = -\eta \frac{\partial E}{\partial w_{ji}^k}, \quad \Delta \theta^k = -\eta \frac{\partial E}{\partial \theta^k} \quad (\eta > 0) \quad (5.11), \quad \text{其中 } \eta \text{ 是学习率，一般小于0.5。}$$

第5步：采用公式 (5.11) 计算出的修正值，依次**更新所有权重和偏置**；

第6步：转向第2步。

BP算法的具体执行过程

假设输入样本为 $(x_1=0.05, x_2=0.1)$ ，期望输出值为 $(y_1=0.03, y_2=0.05)$ 。

1. 正向传播

(1) 由输入层向隐藏层传播信息。

➤ 隐藏层中第一个神经元 h_1 的输出为：

$$out_{h_1} = \sigma(net_{h_1}) = \frac{1}{1+e^{-net_{h_1}}} = 0.595$$

➤ 隐藏层中第一个神经元 h_2 的输出为：

$$out_{h_2} = \sigma(net_{h_2}) = \frac{1}{1+e^{-net_{h_2}}} = 0.599$$

(2) 由隐藏层向输出层传播信息。

输出层中两个神经元的输出分别为：

$$\left\{ \begin{array}{l} y_1^3 = out_{o_1} = \frac{1}{1+e^{-net_{o_1}}} = 0.694 \\ y_2^3 = out_{o_2} = \frac{1}{1+e^{-net_{o_2}}} = 0.718 \end{array} \right.$$

◆ 显然，实际输出为 $Y^3=[0.694, 0.718]$ ，与期望输出 $Y=[0.03, 0.05]$ 相差较大。

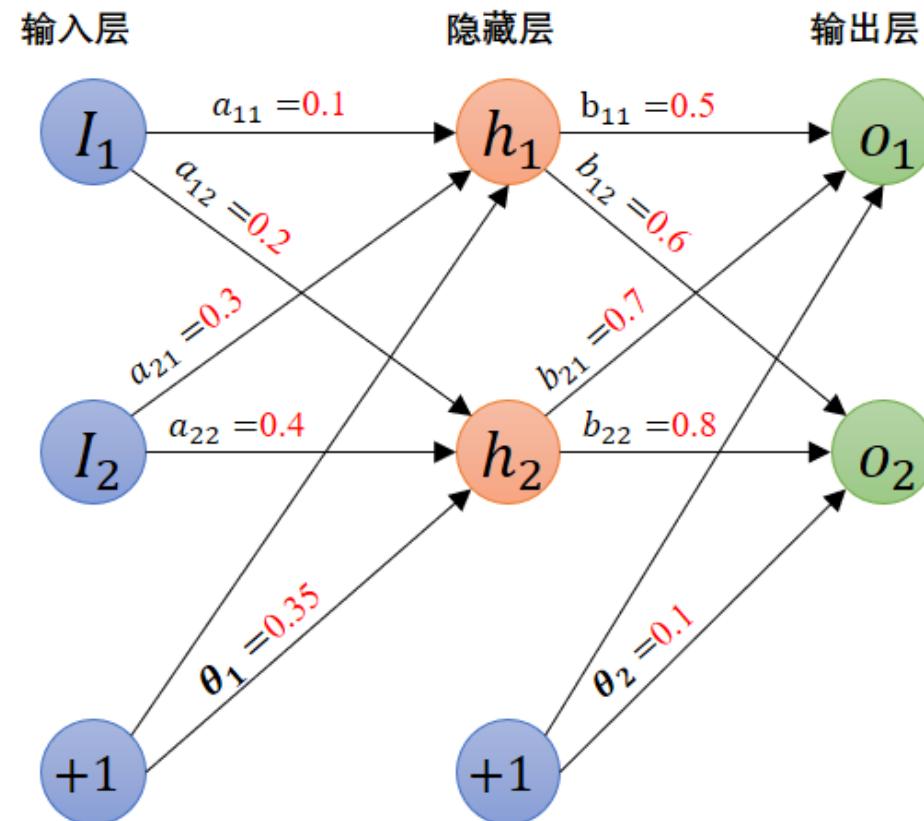
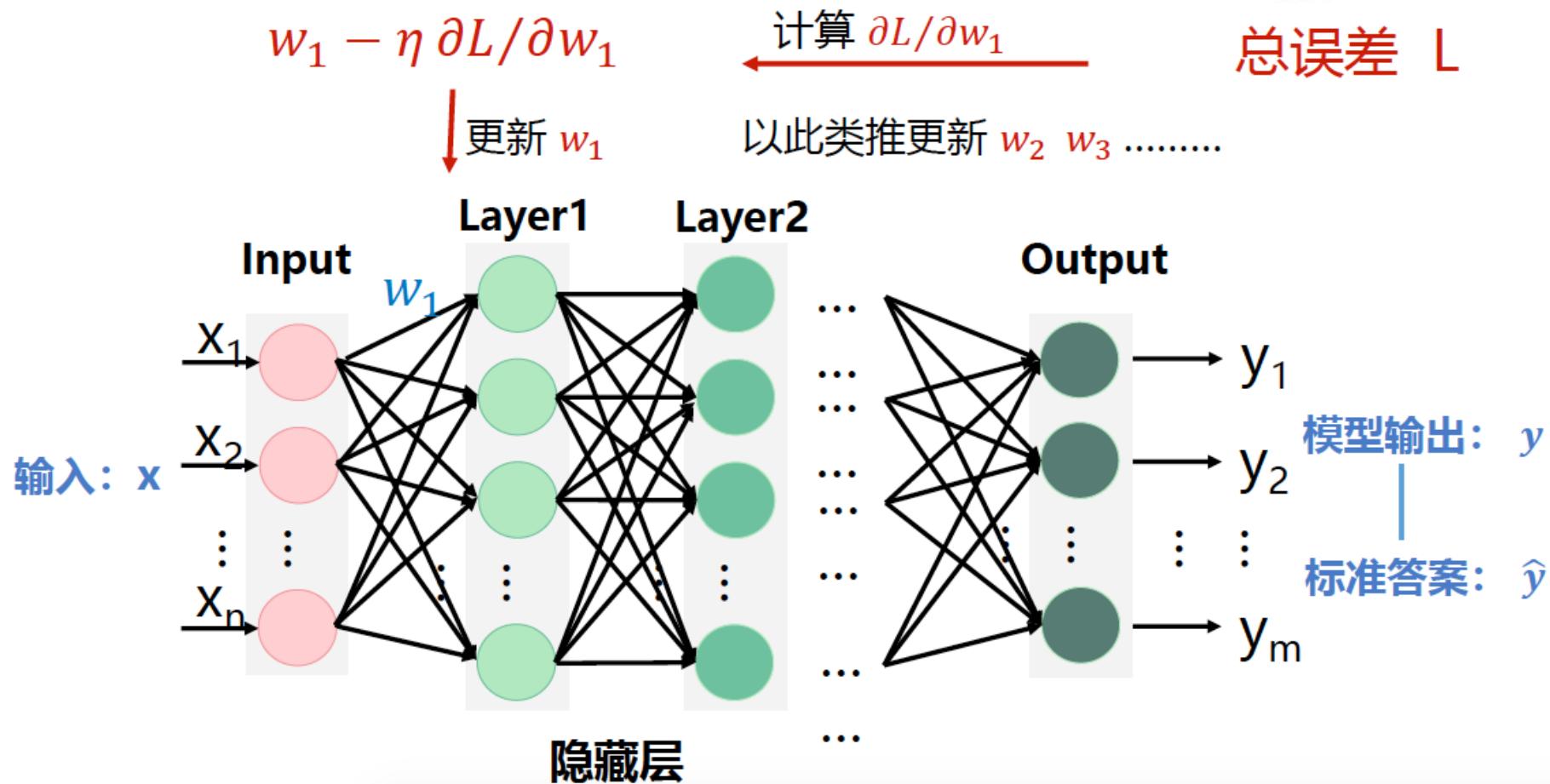


图5.8 BP神经网络示例

参数学习

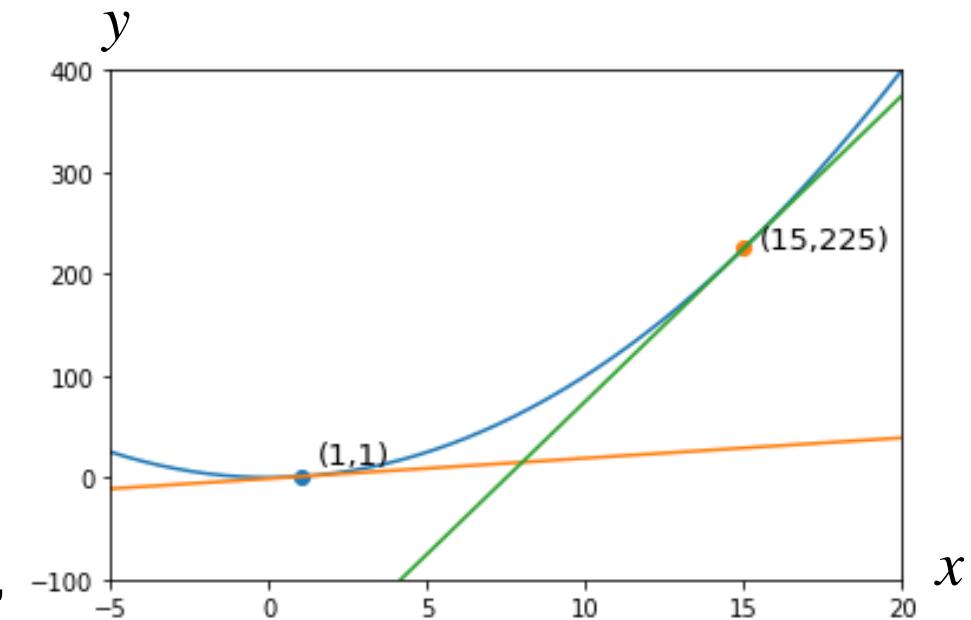
★ 反向传播算法

只剩一个问题：
怎么求 梯度 $\partial L / \partial w$?



多层次神经网络--梯度

- 函数在某点的导数表示函数值在这一点附近的变化率。
- 以一元二次函数为例，右图是该函数在二维平面中的图像，并在图上分别画出了 $(1, 1)$ 和 $(15, 225)$ 两点处的切线，每个点位置的导数值也就是在该点切线的斜率。该函数的导数用 y' 表示，根据求导公式有：在点 $x=15$ 处的导数值为30；点 $x=1$ 处的导数值为2。



多层次神经网络--梯度

■ 梯度的求解

多元函数的梯度的求解与其偏导数有关，形式上看，梯度就是把多元函数的各个变量的偏导数放在一起构成的向量。

以二元函数 $f(x, y)$ 为例，其梯度表达式为

$$\text{grad}(f) = \left(\frac{\partial f(x, y)}{\partial x}, \frac{\partial f(x, y)}{\partial y} \right)$$

练：尝试计算 $f(x, y) = x^2 + y^2 + xy$ 的梯度

多层次神经网络--梯度

■ 偏导数

以二元函数 $z = f(x, y)$ 为例，用 $\frac{\partial f(x,y)}{\partial x}$ 表示函数 $f(x,y)$ 关于 x 的偏导数，其几何含义就是在函数 $z = f(x, y)$ 的三维几何图中， y 固定为某个值时（ y 方向不变），函数值 z 沿 x 轴方向的变化率；用 $\frac{\partial f(x,y)}{\partial y}$ 表示函数 f 关于 y 的偏导数，指在 x 固定为某个值时（ x 方向不变），函数值 z 沿 y 轴方向的变化率。

■ 偏导数计算

以二元函数 $z = f(x, y) = x^2 + y^2 + xy$ 为例，

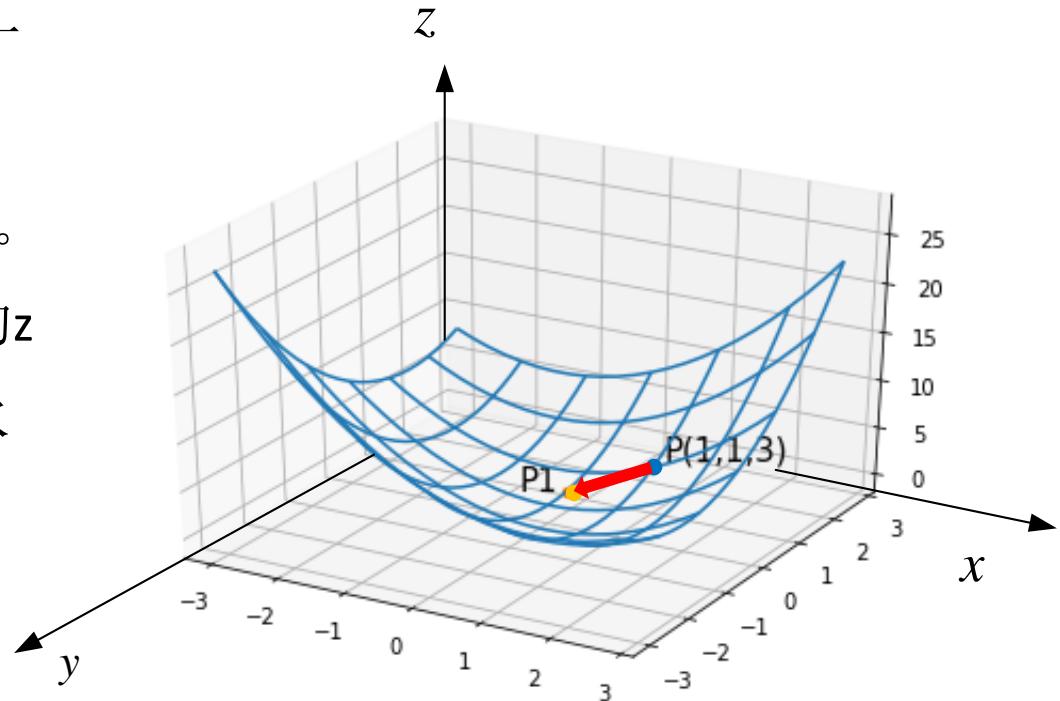
计算 z 对 x 的偏导数，就是把 y 看作常数，对 x 求导得 $\frac{\partial f(x,y)}{\partial x} = 2x + y$ ；

计算 z 对 y 的偏导数，就是把 x 看作常数，对 y 求导得 $\frac{\partial f(x,y)}{\partial y} = x + 2y$ 。

多层神经网络--梯度

■ 函数值随自变量的变化而变化

当给P点的x和y分别加上一个数值 Δx 和 Δy ，得到一个新点P1，其在水平面上的坐标为 $(x+\Delta x, y+\Delta y)$ ，那么在三维图中可以标记点P1 $(x+\Delta x, y+\Delta y, Z_{new})$ 。这个过程可以看作从P点移动到了P1点，如果P1的z值大于P，则说是在往水池高处走，反之则是往水池底部走，差值为高度变化值。



■ 梯度方向是函数值增加最快的方向

2. 反向传播

- ◆ **反向学习的目的**: 利用梯度下降法更新网络中的参数, 使得损失函数达到极小值。
- ◆ **反向学习的过程**: 误差信息从输出层经过隐藏层传向输入层, 逐层使权值沿损失函数的负梯度方向更新。

➤ 计算损失函数: $E = \frac{1}{2} \sum_{j=1}^{p_m} (y_j - y_j^m)^2$

➤ 神经元的激活函数一般选择非线性的sigmoid函数, 则各个权重 w_{ji}^k 的修正量 Δw_{ji}^k 可通过 E 求偏导获得, 计算公式如下。

$$\Delta w_{ji}^k = -\eta \frac{\partial E}{\partial w_{ji}^k} = -\eta d_i^{k+1} y_j^k \quad (\eta > 0)$$

$$d_i^m = y_i^m(1 - y_i^m)(y_i^m - y_i)$$

$$d_i^k = y_i^k(1 - y_i^k) \sum_{j=1}^{p_{k+1}} d_j^{k+1} w_{ji}^{k+1} \quad (k = m-1, m-2, \dots, 2)$$

- k 为层号
- d_i^{k+1} 表示第 $k+1$ 层的误差信号
- 注意: d_i^{k+1} 不是误差 Δw_{ji}^k , 两者差了一个第 k 层第 j 个神经元的输出项 y_j^k ,

- ◆ 更新误差的过程是一个**始于输出层向输入层传播的递归过程**, 故称为**反向传播学习算法**。

BP算法中反向传播阶段的具体执行过程 (1)

(1) 计算总误差 $E = \frac{1}{2} \sum_{j=1}^{p_m} (y_j - y_j^m)^2$ p_m 为输出层的神经元个数, 本例中为2。

$$= \frac{1}{2} [(y_1 - y_1^m)^2 + (y_2 - y_2^m)^2]$$

$$= \frac{1}{2} [(0.03 - 0.694)^2 + (0.05 - 0.718)^2] = 0.444$$

(2) 从输出层向隐藏层反向更新学习参数。

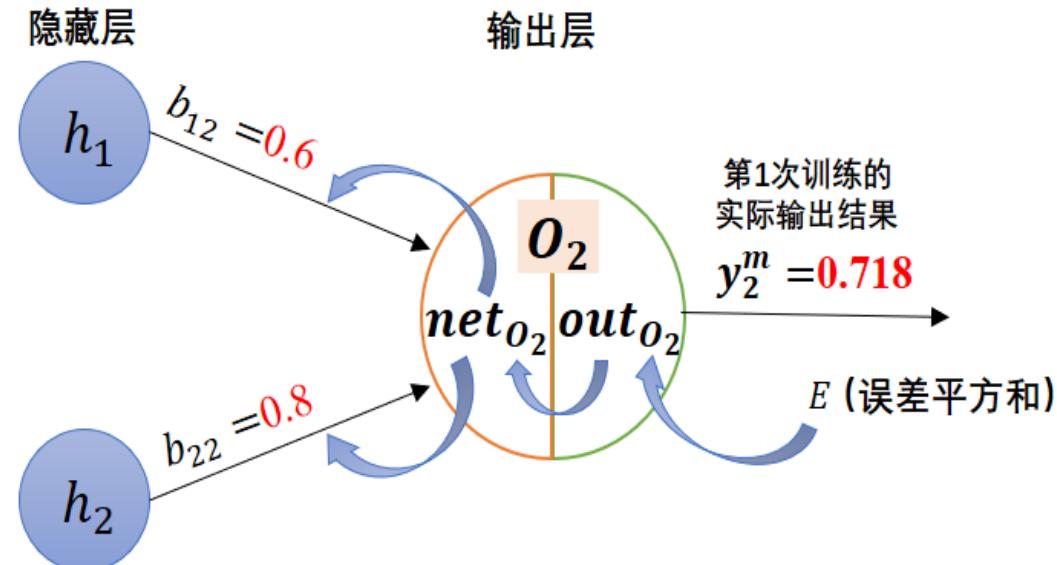
从输出层到隐藏层, 共有5个学习参数需要更新, 分别为 b_{11} 、 b_{12} 、 b_{21} 、 b_{22} 、 θ_2 。

① 计算权重的修正量。

以 b_{22} 为例, 按照链式法则计算总误差 E 对 b_{22} 的偏导: $\Delta b_{22} = \frac{\partial E}{\partial b_{22}} = \frac{\partial E}{\partial out_{O_2}} * \frac{\partial out_{O_2}}{\partial net_{O_2}} * \frac{\partial net_{O_2}}{\partial b_{22}} = 0.081$

设定学习率 η 为 0.5, 更新 b_{22} , 有: $b_{22}^{new} = b_{22} - \eta * \Delta b_{22} = 0.8 - 0.5 * 0.081 = 0.76$

同理可计算得到: $b_{11}^{new} = 0.458$, $b_{12}^{new} = 0.560$, $b_{21}^{new} = 0.658$ 。



BP算法中反向传播阶段的具体执行过程 (2)

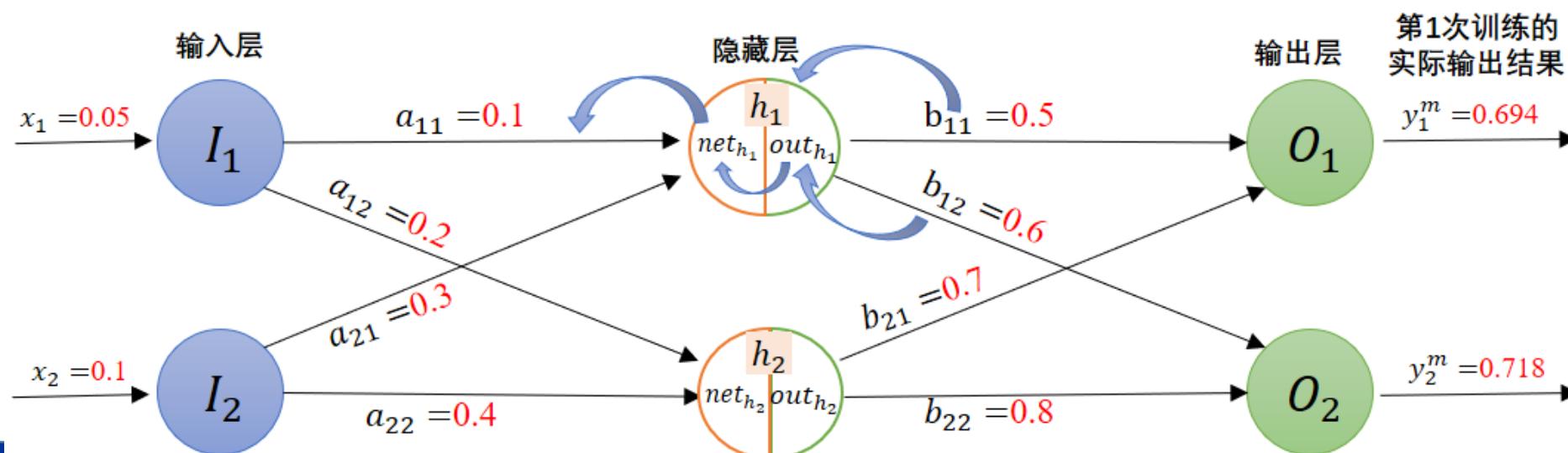
② 计算偏置项的修正量。

$$\Delta\theta_2 = \frac{\partial E}{\partial\theta_2} = \sum_{i=1}^{p_m} \left(\frac{\partial E}{\partial out_{O_i}} * \frac{\partial out_{O_i}}{\partial net_{O_i}} * \frac{\partial net_{O_i}}{\partial\theta_2} \right) = 0.174$$

求得更新后的偏置: $\theta_2^{new} = \theta_2 - \eta * \Delta\theta_2 = 0.1 - 0.5 * 0.174 = 0.913$

(3) 从隐藏层向输入层反向更新学习参数。

从隐藏层到输入层, 共有5个参数需要更新, 分别为 a_{11} 、 a_{12} 、 a_{21} 、 a_{22} 、 θ_1 ,



BP算法中反向传播阶段的具体执行过程 (3)

◆ 这些学习参数的更新方法与“从输出层向隐藏层反向更新学习参数”类似，但稍有区别：在计算 Δa_{11} 时， a_{11} 的值不仅要受到 h_1 反向传过来的误差的影响，还会受到 O_1 和 O_2 传过来的误差的影响。

$$\Delta a_{11} = \frac{\partial E}{\partial a_{11}} = \frac{\partial E}{\partial out_{h_1}} * \frac{\partial out_{h_1}}{\partial net_{h_1}} * \frac{\partial net_{h_1}}{\partial a_{11}}$$

第1项： $\frac{\partial E}{\partial out_{h_1}} = \boxed{\frac{\partial E}{\partial out_{O_1}} * \frac{\partial out_{O_1}}{\partial net_{O_1}}} * \frac{\partial net_{O_1}}{\partial out_{h_1}} + \boxed{\frac{\partial E}{\partial out_{O_2}} * \frac{\partial out_{O_2}}{\partial net_{O_2}}} * \frac{\partial net_{O_2}}{\partial out_{h_1}}$ 这4项在第(2)步的权值更新中均已有计算值。

容易计算得到： $\frac{\partial net_{O_1}}{\partial out_{h_1}} = 0.5$, $\frac{\partial net_{O_2}}{\partial out_{h_1}} = 0.6$, 所以有： $\frac{\partial E}{\partial out_{h_1}} = 0.151$

第2项： $\frac{\partial out_{h_1}}{\partial net_{h_1}} = \sigma(net_{h_1})(1 - \sigma(net_{h_1})) = 0.241$.

第3项： $\frac{\partial net_{h_1}}{\partial a_{11}} = \frac{\partial}{\partial a_{11}}(out_{I_1} * a_{11} + out_{I_2} * a_{21} + \theta_1) = 0.05$

综上所述， $\Delta a_{11} = \frac{\partial E}{\partial a_{11}} = 0.151 * 0.241 * 0.05 = 0.002$

更新 a_{11} ，可得： $a_{11}^{new} = a_{11} - \eta * \Delta a_{11} = 0.099$ 。

同理可得， $a_{12}^{new} = 0.199$, $a_{21}^{new} = 0.298$, $a_{22}^{new} = 0.398$ 。

偏置的更新方法与第(2)步中的方法相同，可求得： $\theta_1^{new} = 0.307$ 。



5.3.2 BP学习算法

- ◆ 第1个训练样本 ($x_1=0.05, x_2=0.1$) , 目标输出为[0.03, 0.05], 实际输出为[0.694, 0.718], 总误差为**0.444**;
- ◆ 完成第1轮参数更新;
- ◆ 继续将第1个训练样本 ($x_1=0.05, x_2=0.1$) 输入到已更新参数的神经网络中, 进行第二次训练, 得到实际输出为[0.667, 0.693], 总误差为**0.44356**;
- ◆ 可见, 总误差在逐渐减小, 随着迭代次数的增加, 输出值会越来越接近目标值, 直到总误差小于设定的阈值, 再输入下一个训练样本, 继续训练, 直到输入所有训练样本训练完毕为止。



梯度下降法

人们通常使用均方误差来衡量预测值与真实值之间的差距，此处考虑单输出的情况，即：

$$e = |\mathbf{y} - \mathbf{y}_{true}|^2$$

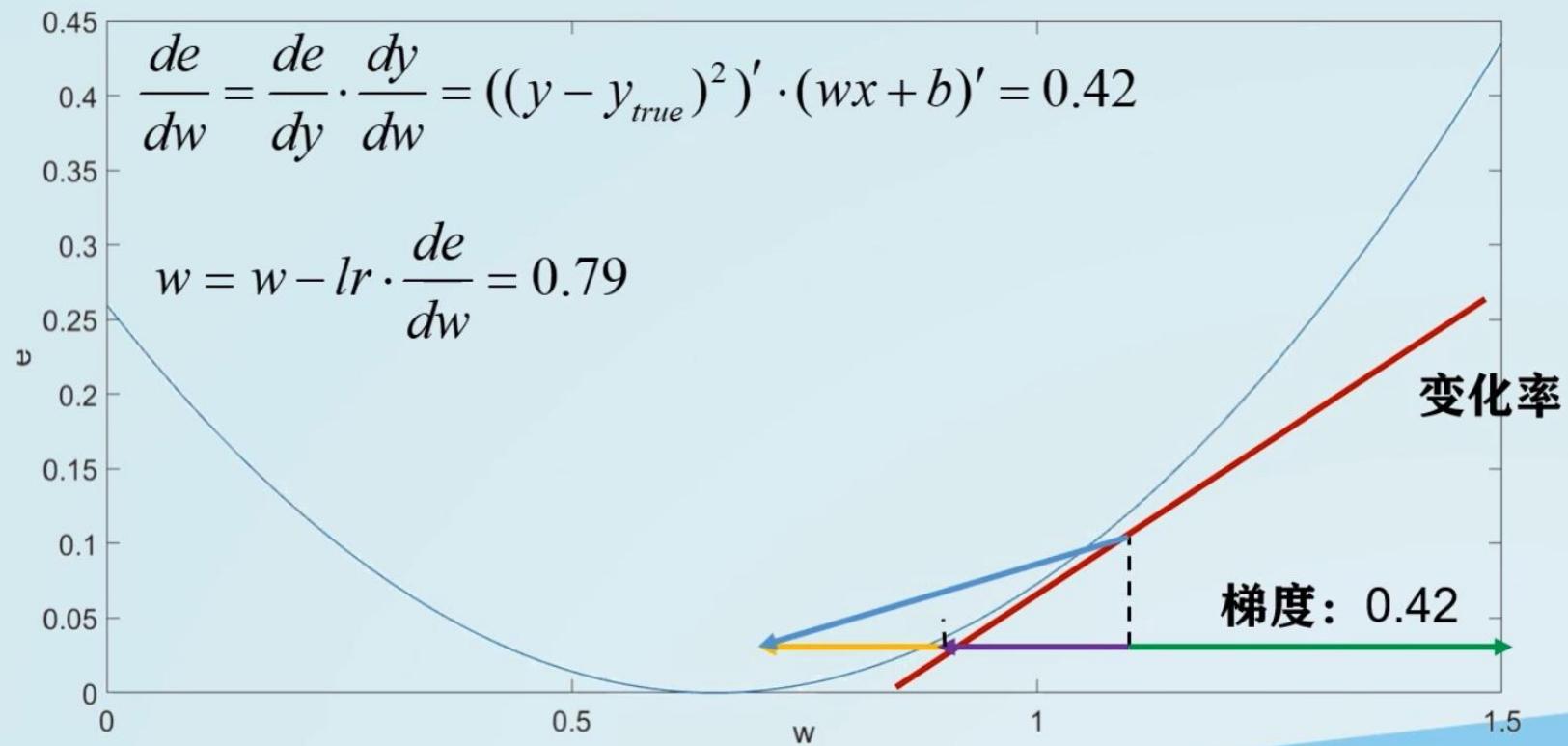
对于神经网络模型：

$$\mathbf{y} = \mathbf{w}\mathbf{x} + \mathbf{b}$$

我们训练神经网络的目的实际上是通过确定合适的 w, b 使得误差 e 尽可能地变小；
要想知道 w, b 是如何影响误差 e 的，我们可以用误差 e 对 w, b 进行求导；
导数的方向为误差 e 上升最快的方向，我们只需将 w, b 向着导数的相反方向进行调整，即可使误差 e 减小——梯度下降。

例题

若预测值: $y = wx + b = 1.78$, 真实值 $y_{true} = 1.51$, 其中 $w=1$, $b=1$, $x=0.78$, 试用梯度下降法对 w 进行调整, 假设学习率 $lr=0.5$ 。

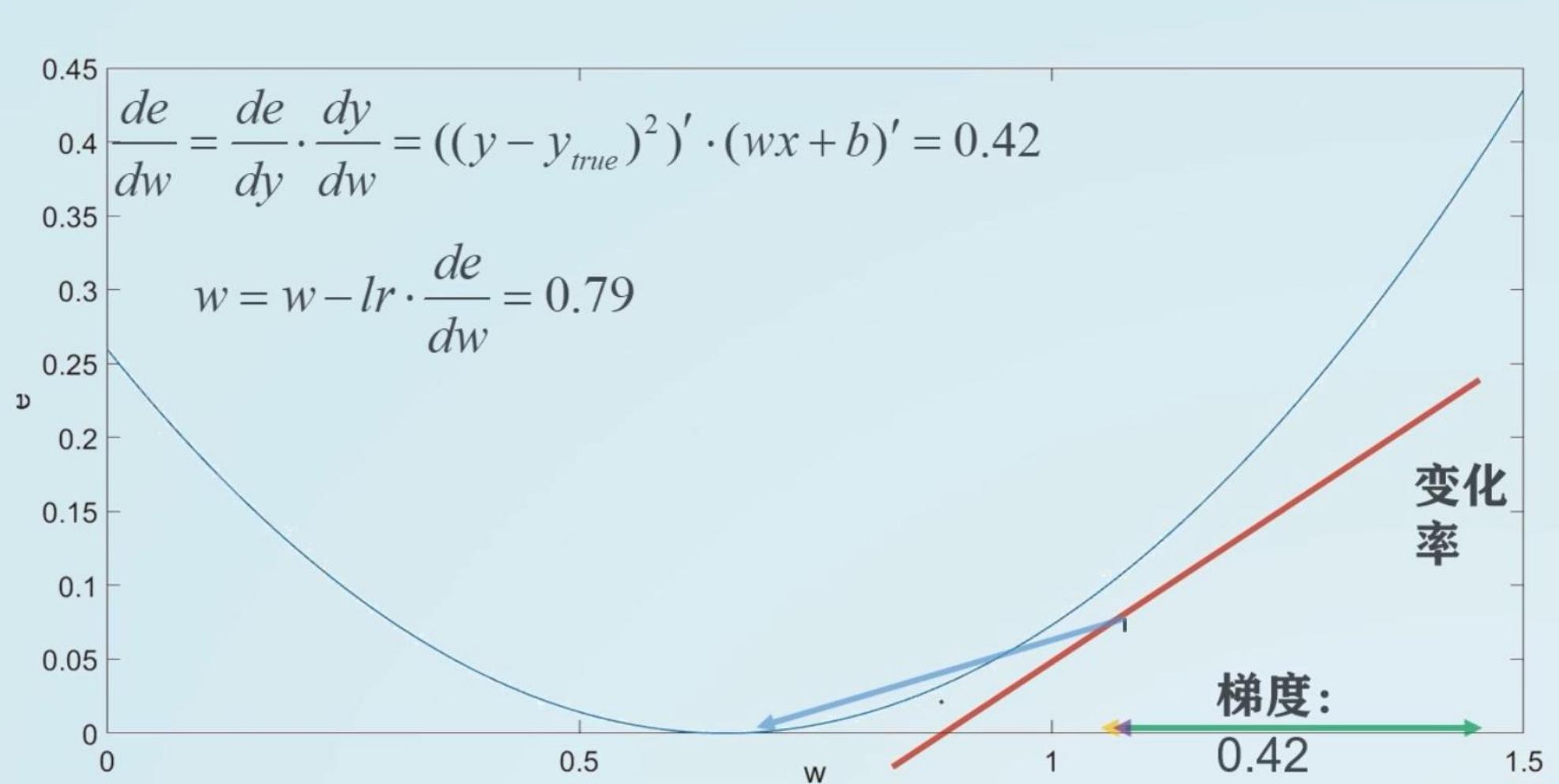


注意：

为了使 w 能较快地收敛于一个使函数 e 极小的解， lr 值的选择是很重要的。

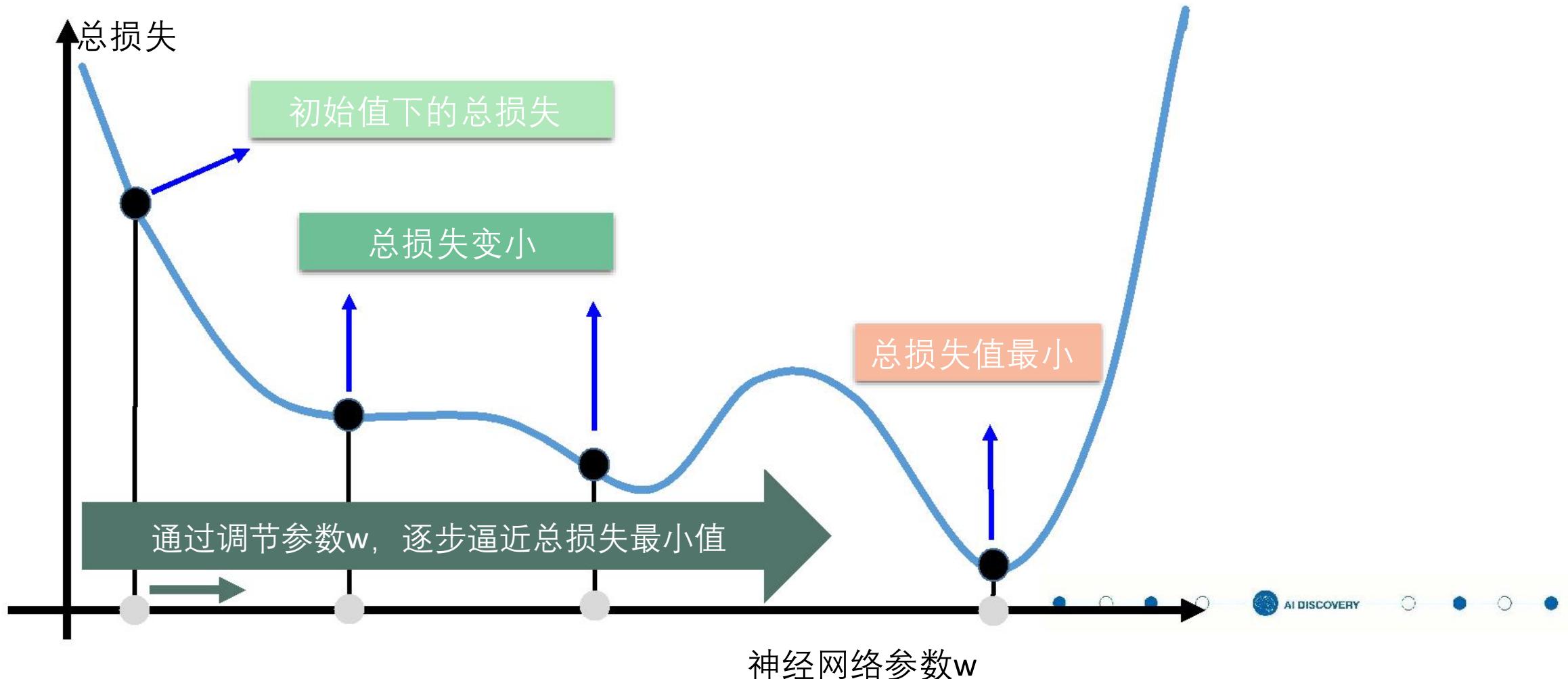
若 lr 值太小，则收敛太慢；

若 lr 值太大，则搜索可能过头，引起发散。

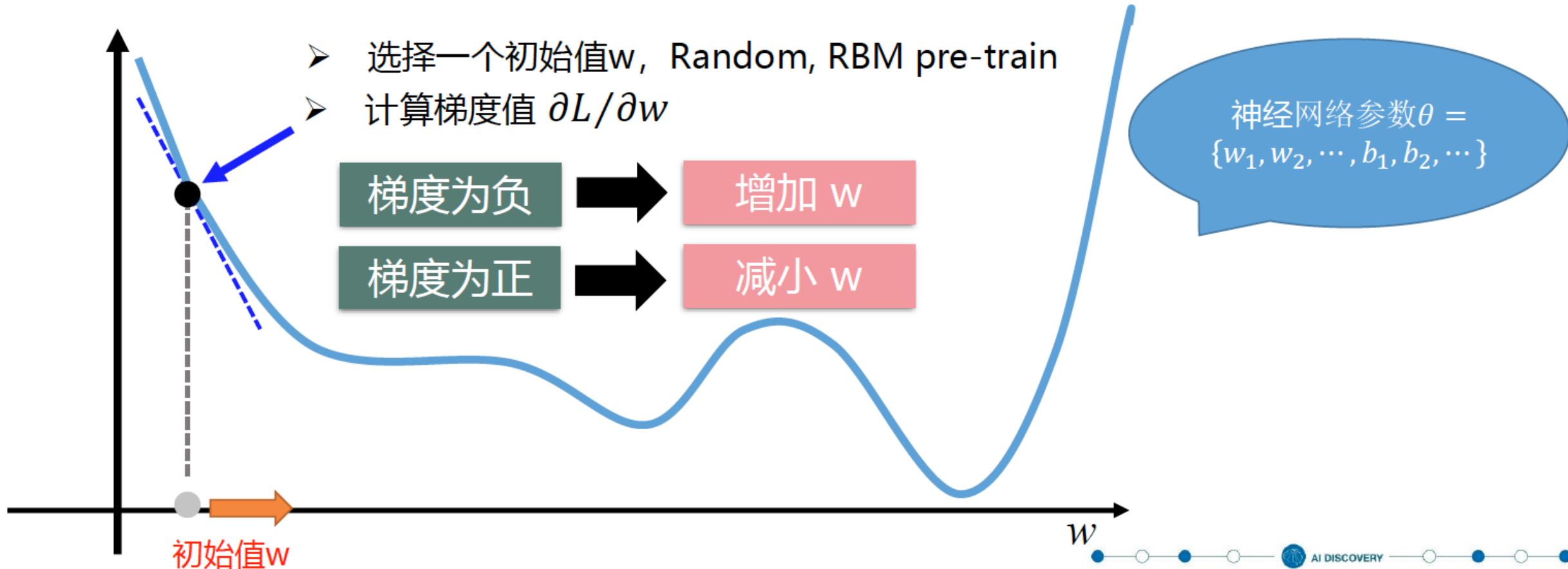


参数学习

★ 梯度下降法

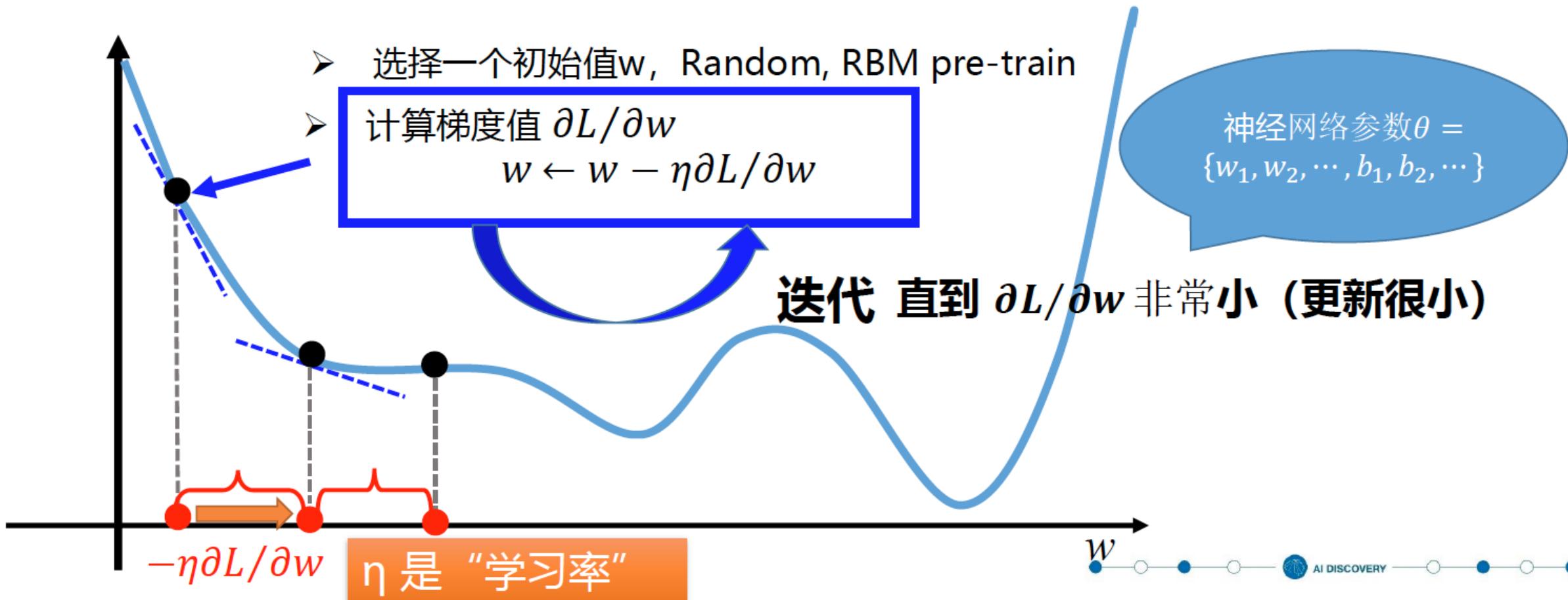


★ 梯度下降法



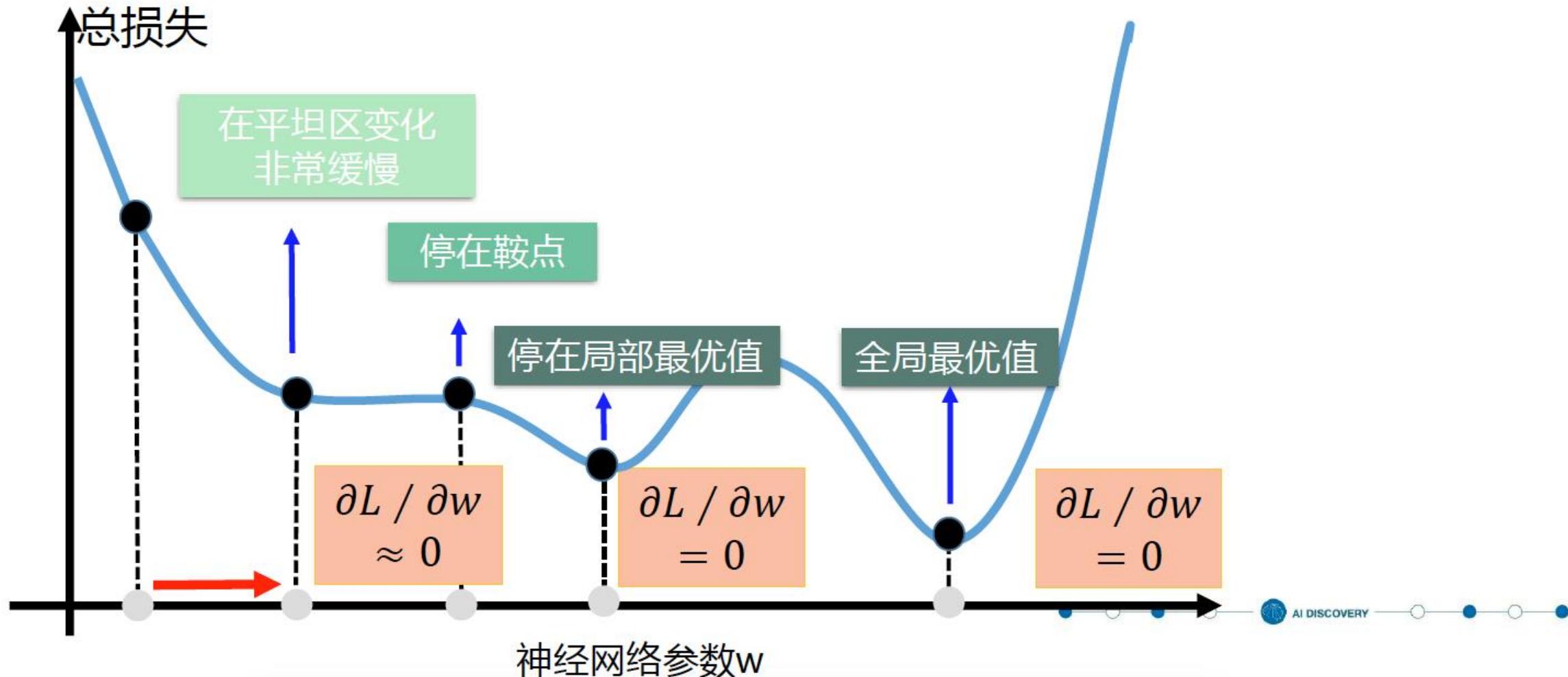
参数学习

★ 梯度下降法



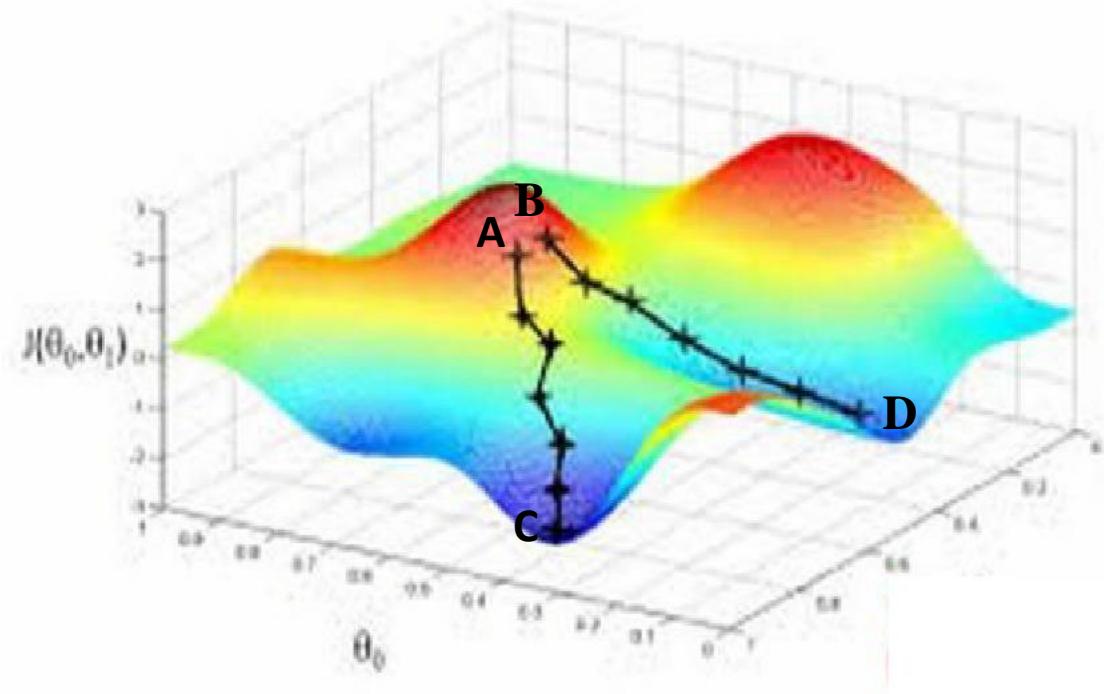
参数学习

★ 梯度下降法



参数学习

★ 初始值影响



例如：

A → C

B → D

选取不同的初始值，可能到达不同的局部最小值



反向传播算法学习过程

- (1) 选择一组训练样本，每一个样本由输入信息和期望的输出结果两部分组成。
- (2) 从训练样本集中取一样本，把输入信息输入到网络中。
- (3) 分别计算经神经元处理后的各层结点的输出。
- (4) 计算网络的实际输出和期望输出的误差。
- (5) 从输出层反向计算到第一个隐层，并按照某种能使误差向减小方向发展的原则，调整网络中各神经元的连接权值。
- (6) 对训练样本集中的每一个样本重复(3)-(5)的步骤，直到对整个训练样本集的误差达到要求时为止。



BP神经网络及学习算法的局限性

- (1) BP 学习算法是有监督学习, **需要大量带标签的训练数据。**
- (2) BP神经网络中的**参数量大**, 收敛速度慢, 需要较长的训练时间, **学习效率低。**
- (3) BP 学习算法采用梯度下降法更新学习参数, 容易**陷入局部极值**, 从而**找不到全局最优解**。
- (4) 尚无理论指导如何选择网络隐藏层的层数和神经元的个数, 一般是根据经验或通过反复实验确定。因此, **网络往往存在很大的冗余性**, 在一定程度上也增加了网络学习的负担。

深度学习使用步骤

★ 建立模型

- 选择什么样的网络结构
- 选择多少层数，每层选择多少神经元

★ 损失函数

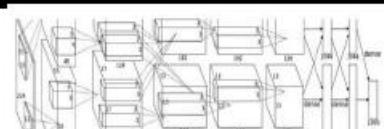
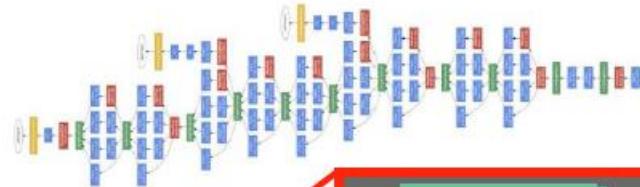
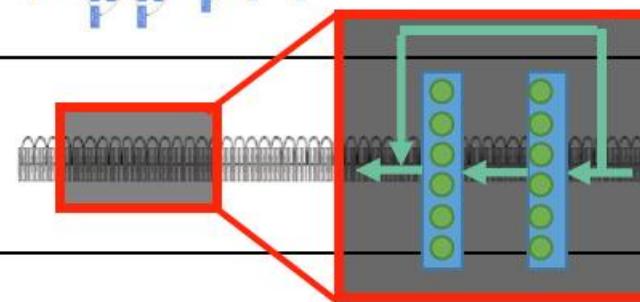
- 选择常用损失函数，平方误差，交叉熵....

★ 参数学习

- 梯度下降
- 反向传播算法

建立模型

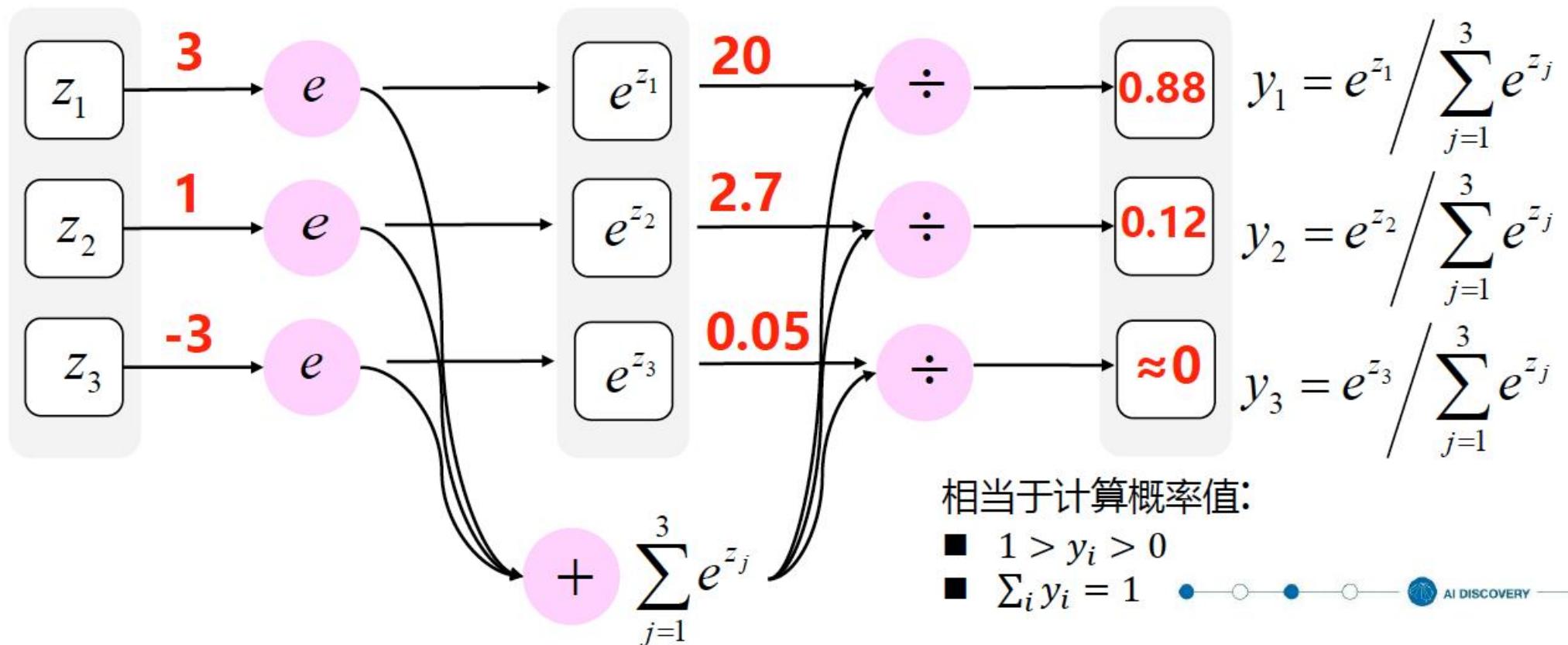
例子

模型	层数	结构	错误率
AlexNet (2012)	8层		16.4%
VGG (2014)	19层		6.7%
GoogleNet (2014)	22层		6.7%
Residual Net (2015)	152层		3.57%

建立模型

★ 输出层

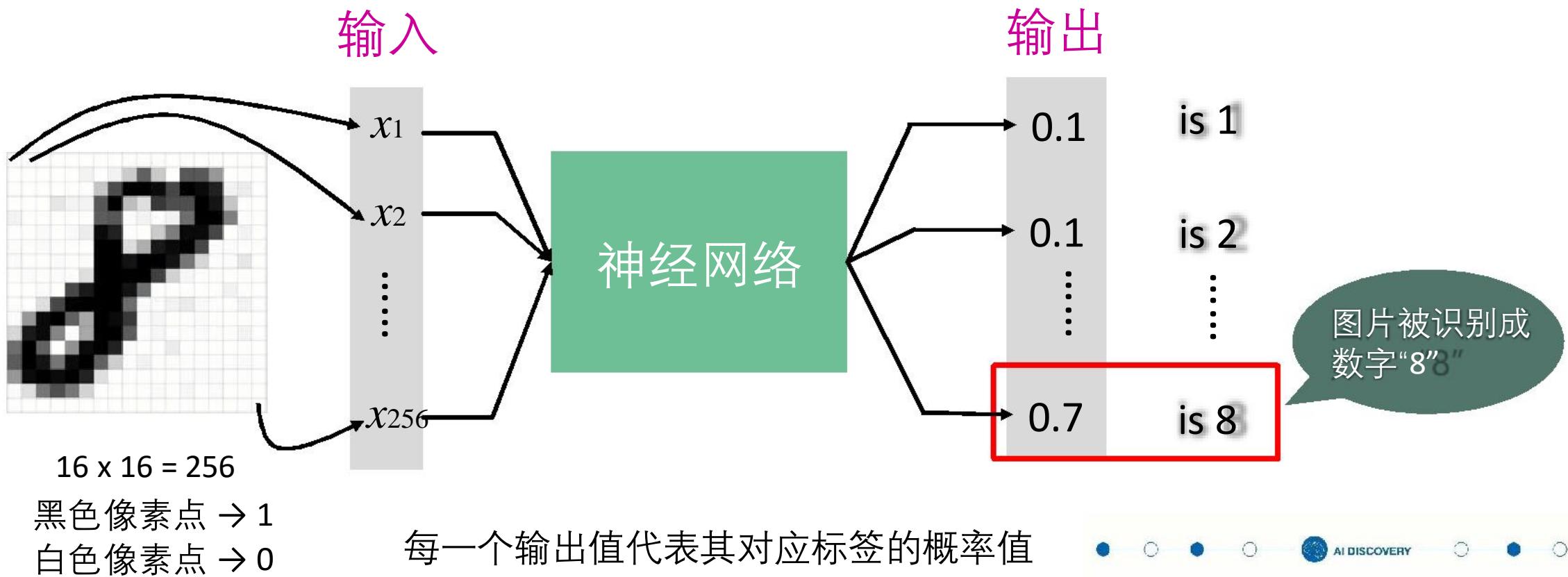
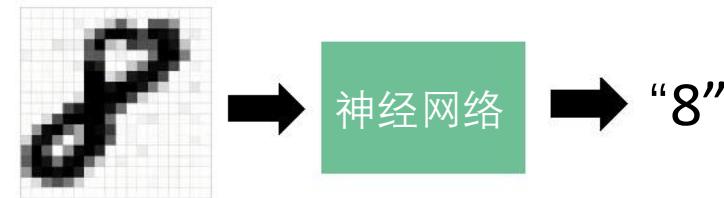
- 常用softmax函数作为输出层激活函数：容易理解、便于计算



建立模型

AI DISCOVERY

应用示例：手写识别

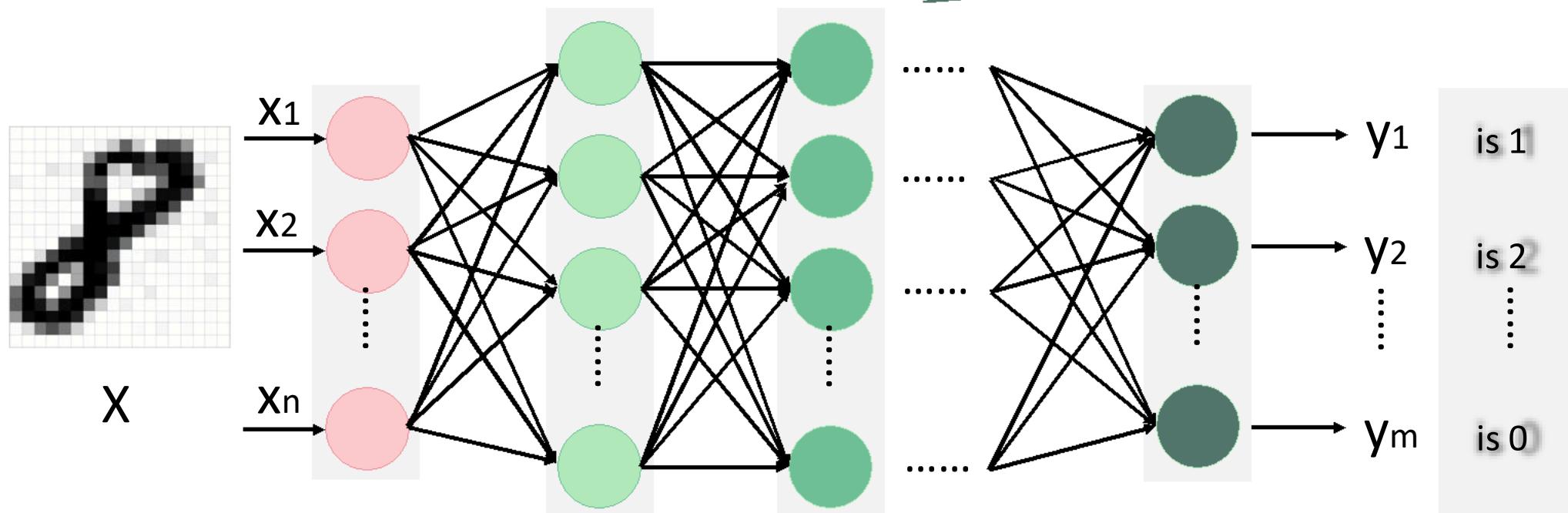


建立模型

AI DISCOVERY

★ 应用示例：手写体识别

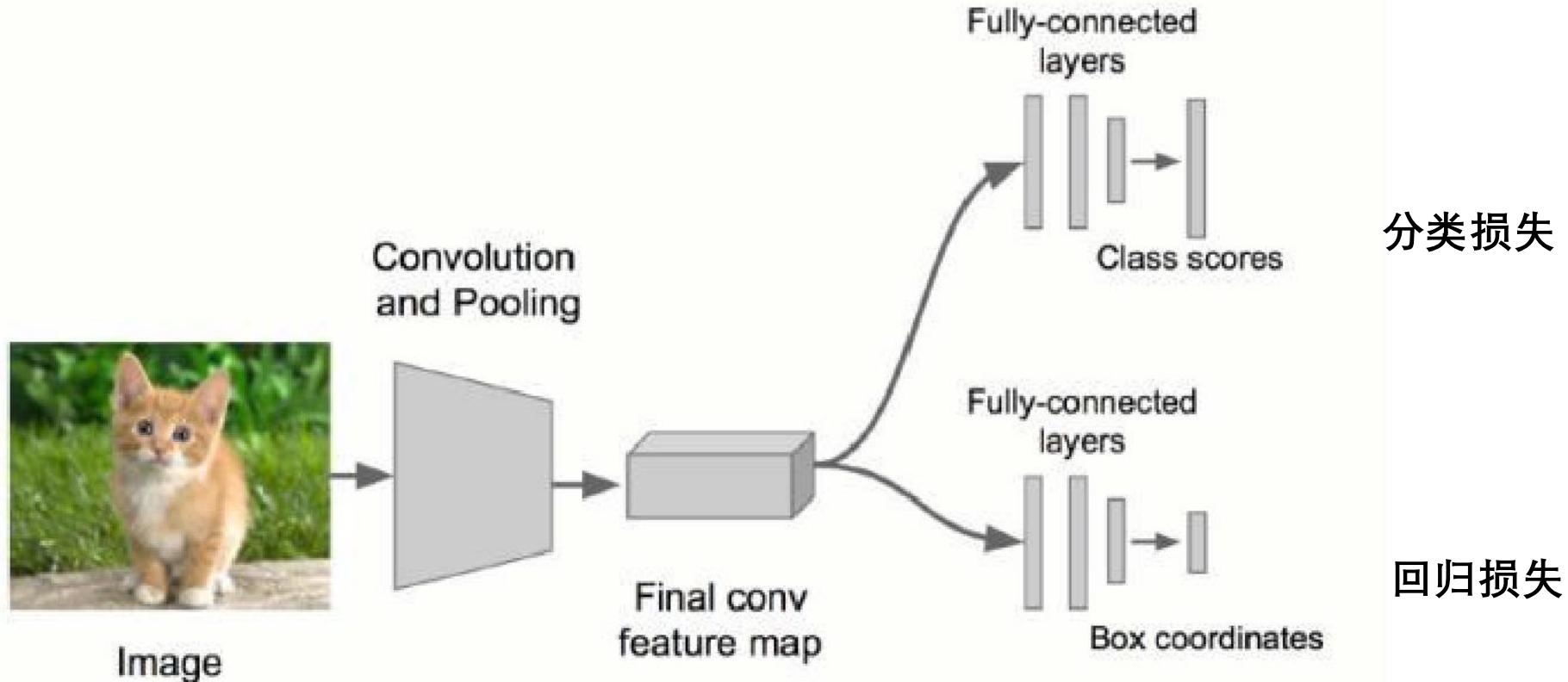
问题：应该设置多少层，多少结点？
是否需要选择其他网络结构如
CNN/RNN？



设置合适的网络结构：层数和结点个数、激活函数

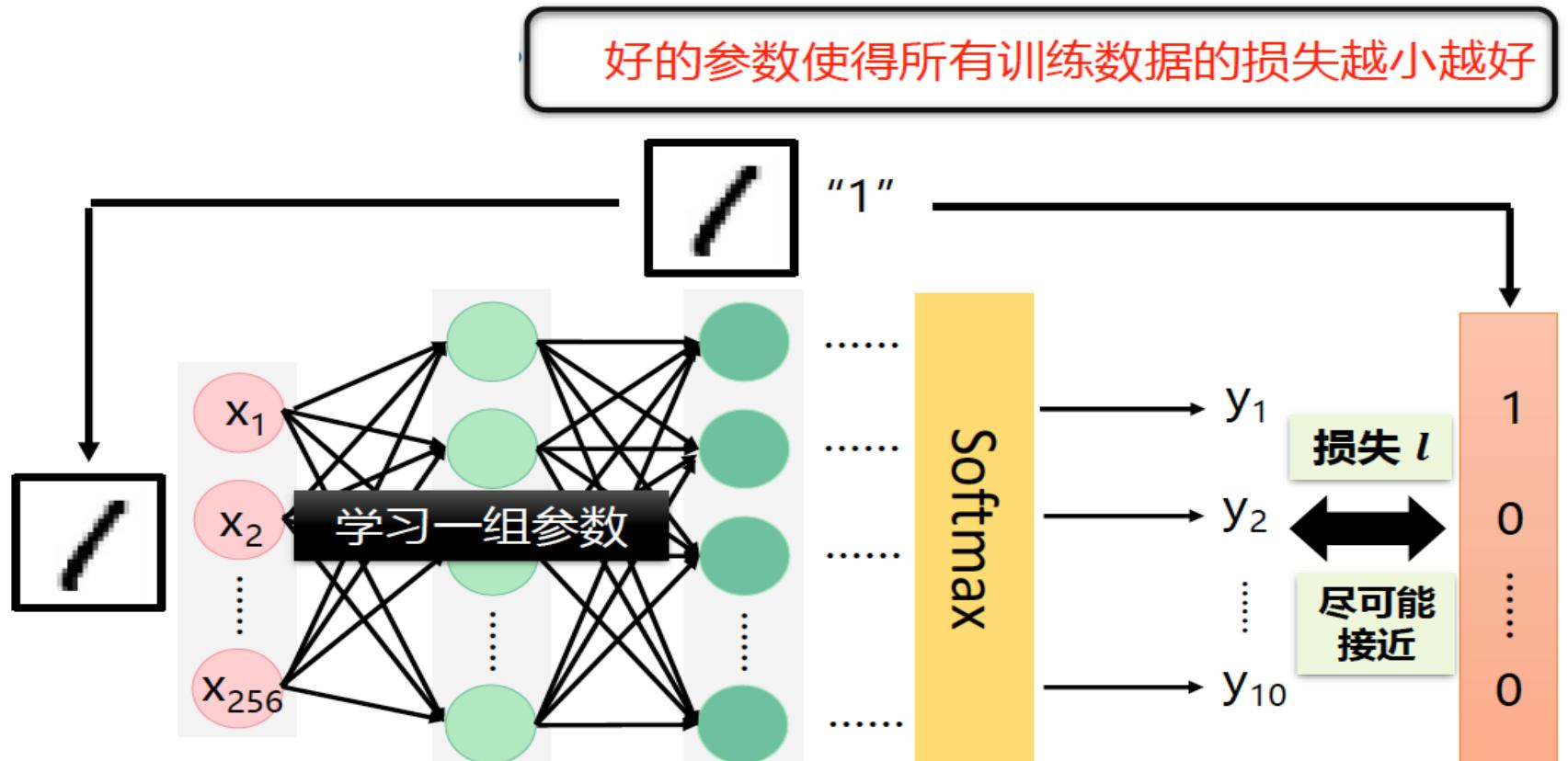
AI DISCOVERY

损失函数



损失函数的设计依赖于具体的任务

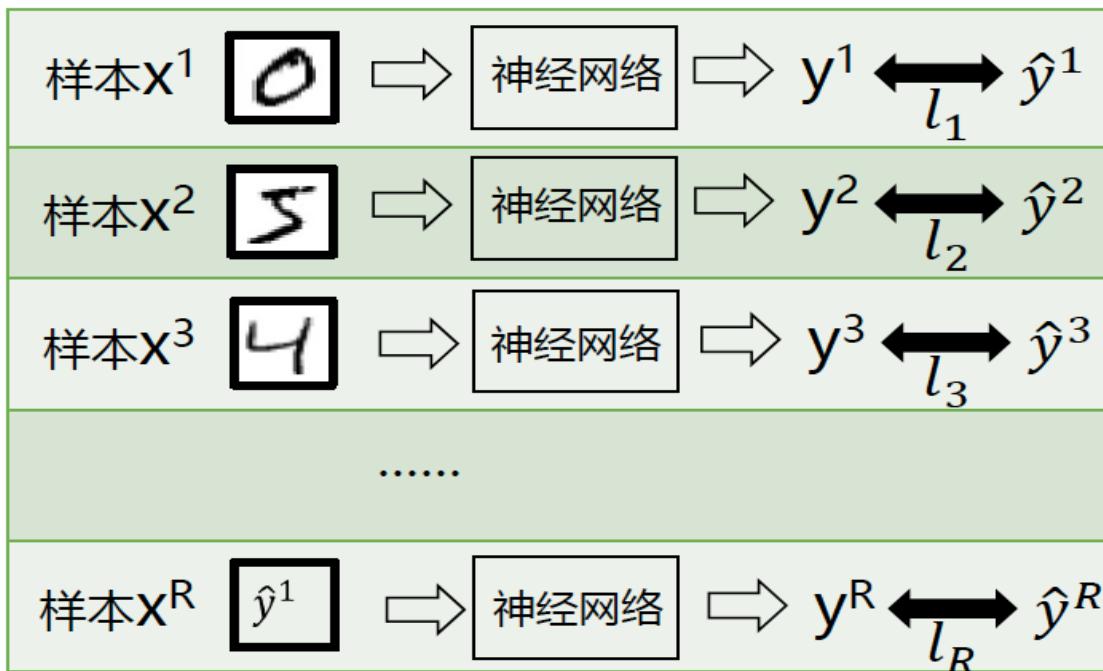
损失函数



常用损失函数：平方损失函数、交叉熵损失函数

损失函数

对所有训练数据：



总损失：

$$L = \sum_{r=1}^R l_r$$

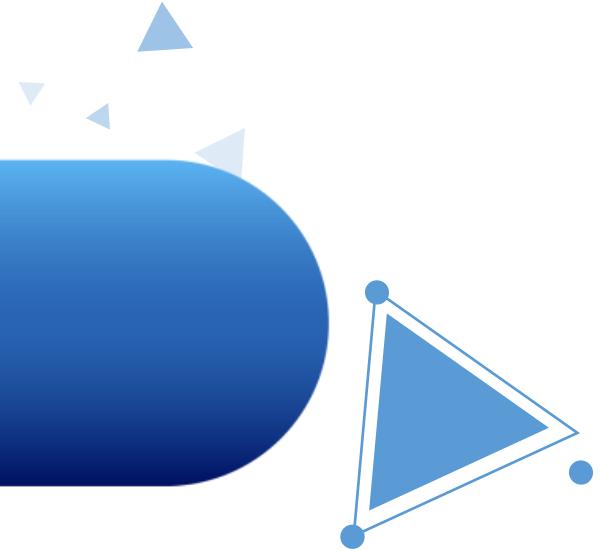
尽可能小

找到一个函数使得总损失 L 最小

即确定参数使得总损失 L 最小



深度学习框架



热门深度学习库

基于 Github 和 Stack Overflow 上的活跃度以及 Google 搜索结果，The Data Incubator 最近制作了一个热门深度学习库的排名。

其中值 1 表示高于平均值的一个标准偏差（平均值为 0）, 越高说明越受欢迎。

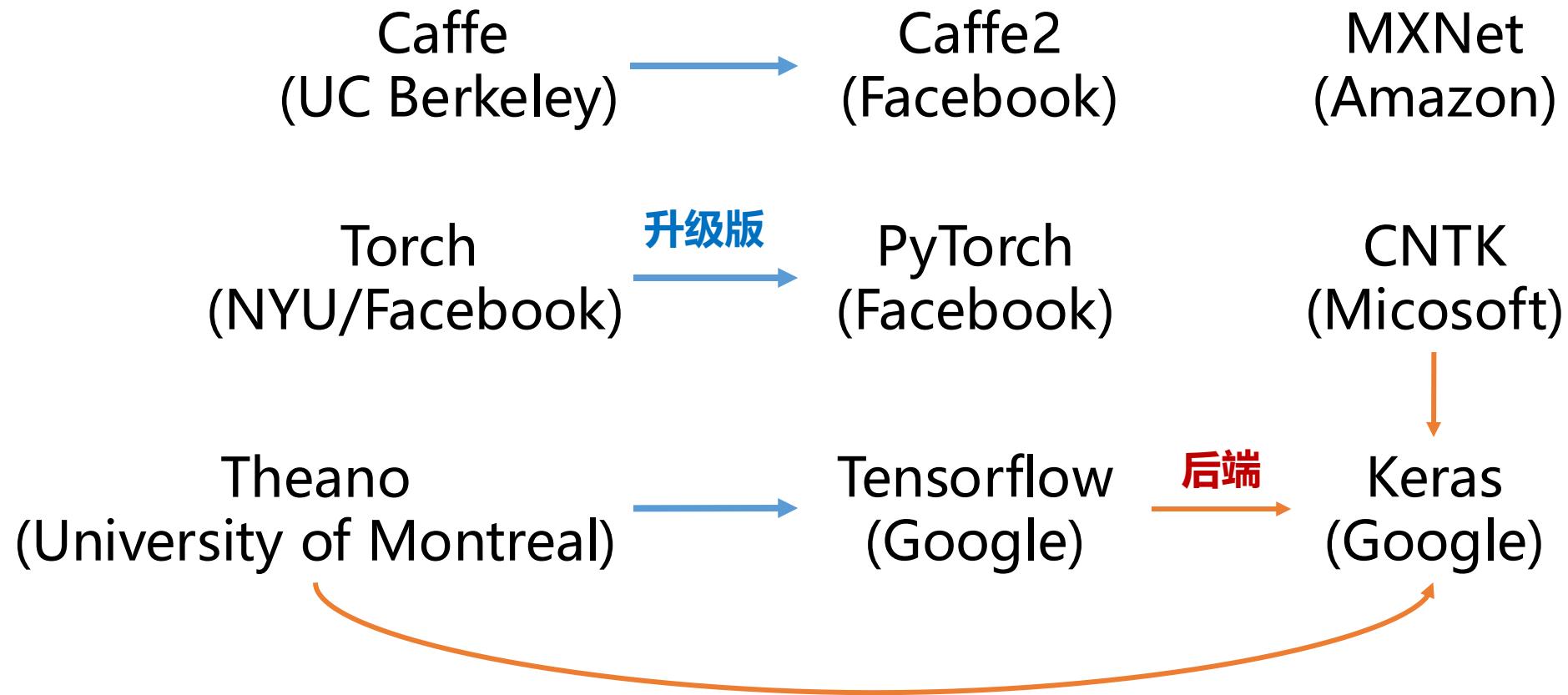
Library	Rank	Overall	Github	Stack Overflow	Google Results
tensorflow	1	10.87	4.25	4.37	2.24
keras	2	1.93	0.61	0.83	0.48
caffe	3	1.86	1.00	0.30	0.55
theano	4	0.76	-0.16	0.36	0.55
pytorch	5	0.48	-0.20	-0.30	0.98
sonnet	6	0.43	-0.33	-0.36	1.12
mxnet	7	0.10	0.12	-0.31	0.28
torch	8	0.01	-0.15	-0.01	0.17
cntk	9	-0.02	0.10	-0.28	0.17
dlib	10	-0.60	-0.40	-0.22	0.02
caffe2	11	-0.67	-0.27	-0.36	-0.04
chainer	12	-0.70	-0.40	-0.23	-0.07
paddlepaddle	13	-0.83	-0.27	-0.37	-0.20
deeplearning4j	14	-0.89	-0.06	-0.32	-0.51
lasagne	15	-1.11	-0.38	-0.29	-0.44
bigdl	16	-1.13	-0.46	-0.37	-0.30
dynet	17	-1.25	-0.47	-0.37	-0.42
apache singa	18	-1.34	-0.50	-0.37	-0.47
nvidia digits	19	-1.39	-0.41	-0.35	-0.64
matconvnet	20	-1.41	-0.49	-0.35	-0.58

操作语言

平台名称	操作语言
Theano	Python, C++, CUDA
Tensorflow	Python, C++, CUDA
Caffe/Caffe 2	Python, C++, CUDA, Matlab
Torch(PyTorch)	Lua(Python)
MxNet	Python, C++, Julia, Matlab, R, Scala
CNTK	C++
Keras	Python, R



总结





国内的深度学习框架

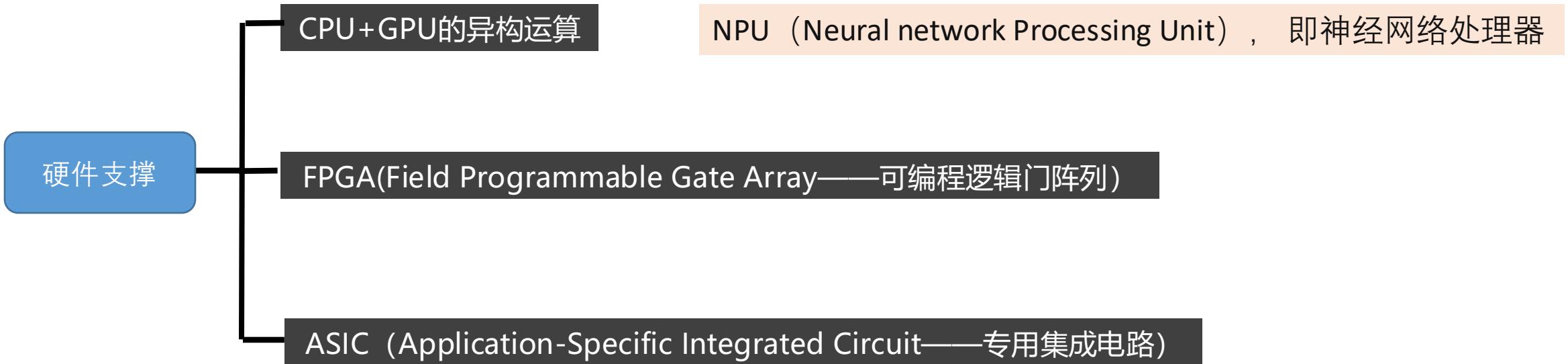
- ◆ 飞桨PaddlePaddle
- ◆ 以百度多年的深度学习技术研究和业务应用为基础，集核心框架、基础模型库、端到端开发套件、丰富的工具组件、星河社区于一体，是中国首个自主研发、功能丰富、开源开放的产业级深度学习平台。
- ◆ <https://www.paddlepaddle.org.cn/>



A Neural Network Playground

<http://playground.tensorflow.org/>

深度学习的硬件支撑



[通俗易懂告诉你CPU/GPU/TPU/NPU...都是什么意思?](#)



总结

- ◆什么是ANN
- ◆MP模型
- ◆感知机
- ◆正向传播
- ◆反向传播
- ◆开发框架