

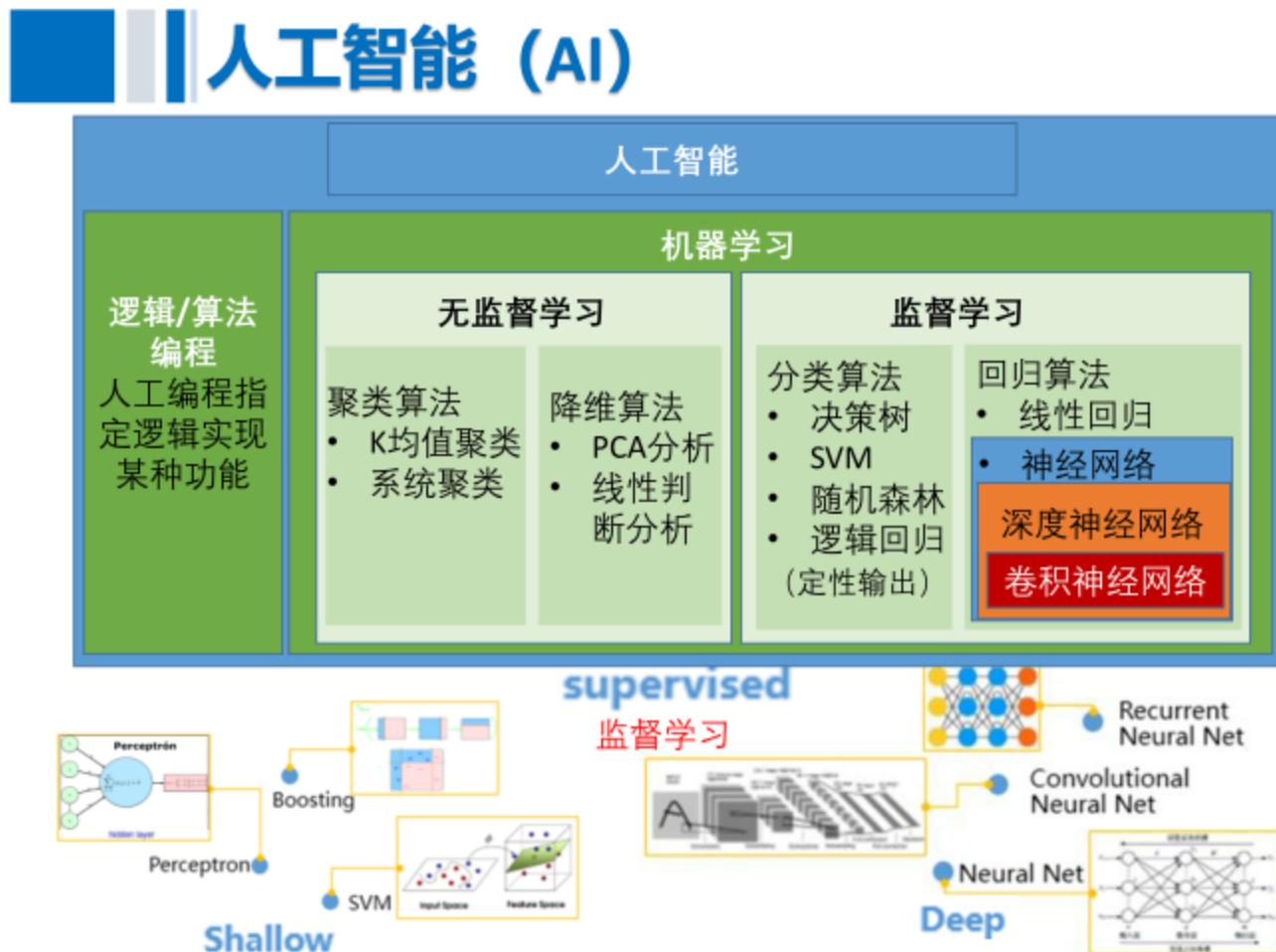


# 人工神经网络

李明磊

南京航空航天大学电子信息工程学院  
E-mail: minglei\_li@nuaa.edu.cn

1



2

## 神经网络

### ◆ 1. 人类的大脑

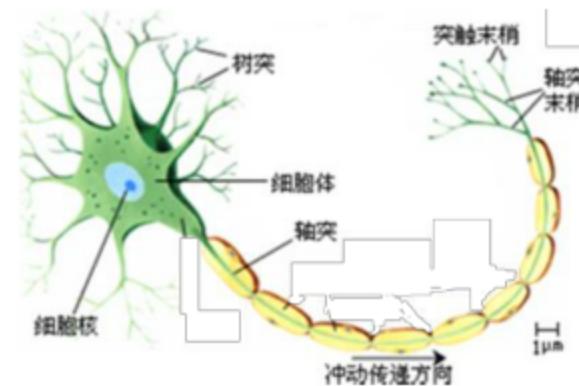
- 二十世纪初，科学家通过新的细胞染色方法得知人的大脑是一个由数以十亿计的细胞构成的无比复杂的网络。人脑平均包含超过 $10^{15}$ 条内部连接，即一立方厘米中就超过 $10^9$ 条连接。
- 人脑内部结构十分复杂，总体上可以看做是一个分区的并行处理器，能够完成记忆、理解、学习、推理、判断以及控制等复杂的工作



1 立方毫米脑组织的成像数据  
数以万计的重建神经元、数百万个神经元片段、1.3亿个带注释的突触、104个校对过的细胞

### ◆ 2. 生物神经元

- 神经元从外部形态结构上看是由细胞胞体(soma)，以及树突(dendrite)和轴突(axon)这两类突起组成的。通常树突接受刺激信息，并向胞体传送，经胞体整合后由轴突传出。



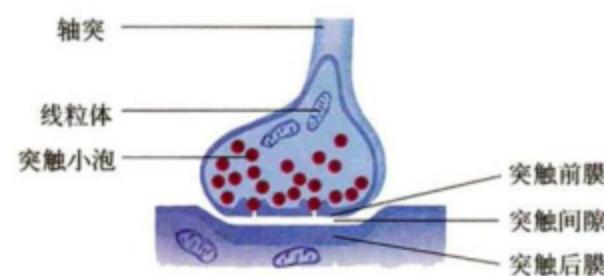
参考资料：东北大学 信息学院，陈东岳、王晓哲

3

## 神经网络

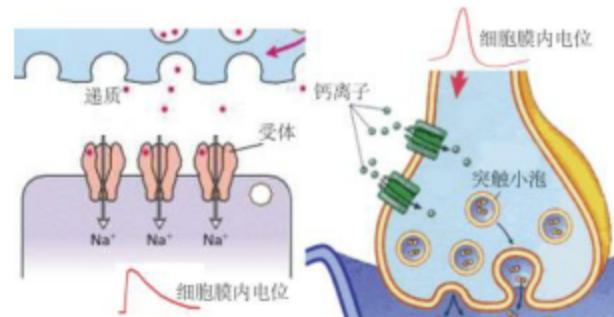
### ◆ 3. 突触

- 突触(synapse)是神经元之间进行信息传递的特异性功能接触部位。借助于突触，神经元相互联系构成了神经系统。突触的形态如图所示。突触根据其功能可以分为兴奋型(Excitatory)突触和抑制型(Inhibitory)突触。



### ◆ 4. 神经元的电活动

- 所谓神经元的电活动就是在信息传递过程中神经元的膜电位变化的过程。其最主要的行为表现为动作电位的发放，也就是我们常说的神经元的电脉冲。



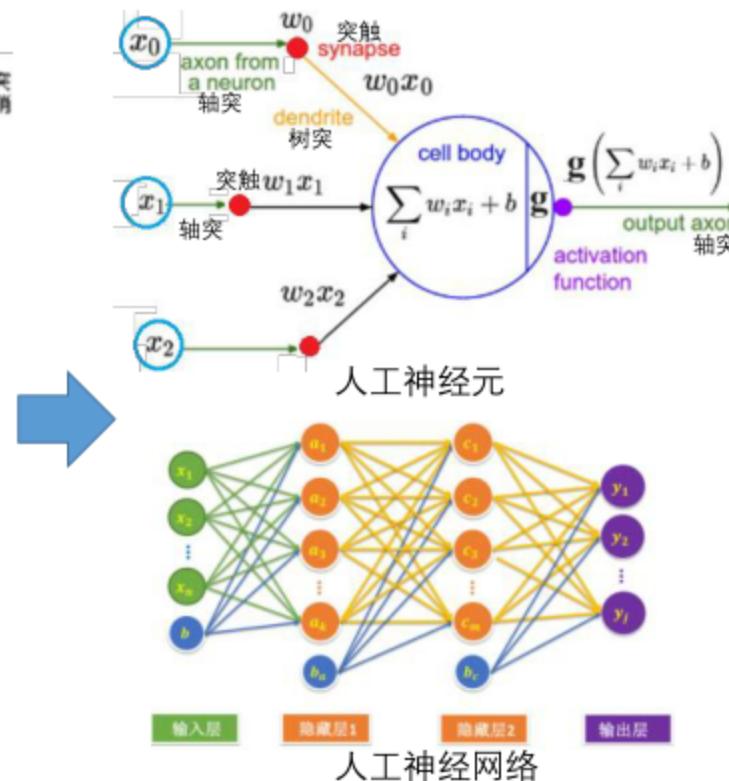
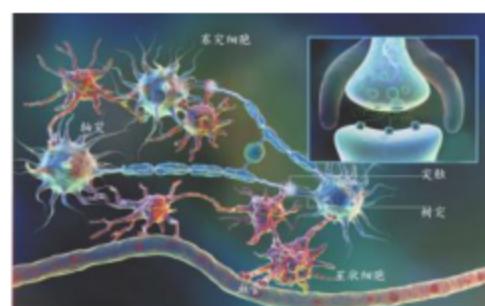
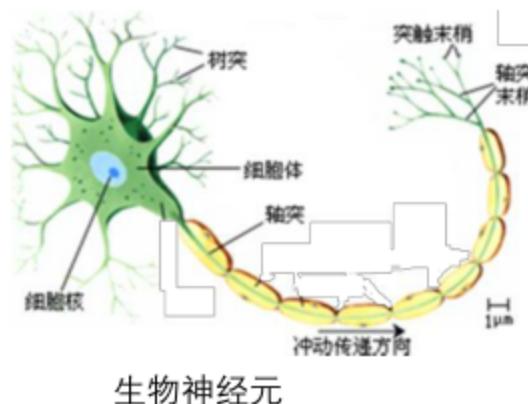
参考资料：东北大学 信息学院，陈东岳、王晓哲

4

# 人工神经网络

## ◆ 5. 人工神经网络

● 模拟生物神经网络由简单的处理单元(神经元)组成的大规模并行分布式处理器

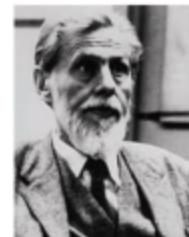


5

# AI发展

1943年，心理学家McCulloch和数学家Pitts参考了生物神经元的结构，发表了抽象的神经元模型MP。

1949年心理学家Hebb提出了Hebb学习率，认为人脑神经细胞的突触（也就是连接）上的强度上可以变化的。于是计算科学家们开始考虑用调整权值的方法来让机器学习。这为后面的学习算法奠定了基础。限于当时的计算机能力，直到接近10年后，第一个真正意义的神经网络才诞生。



Warren McCulloch



Walter Pitts



Donald Olding Hebb

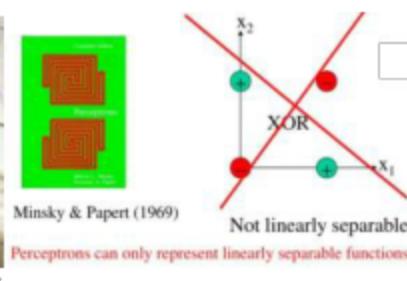
1958年，计算科学家Rosenblatt提出了由两层神经元组成的神经网络。起名“感知器” (Perceptron)

感知器是当时首个可以学习的人工神经网络。Rosenblatt现场演示了其学习识别简单图像的过程，在当时的社会引起了轰动。人们认为已经发现了智能的奥秘，许多学者和科研机构纷纷投入到神经网络的研究中。美国军方大力资助了神经网络的研究，并认为神经网络比“原子弹工程”更重要。



1969年，人工智能领域的马文·明斯基 (Marvin Minsky) 出版《Perceptron》一书，用详细的数学证明了感知器的弱点。神经网络的研究陷入了冰河期 (“AI winter”)

- 一层感知器只能解决线性问题；
- 要解决非线性问题（包括分段线性问题），比如异或 (XOR) 问题，我们需要多层感知器 (MLP)；
- 但是，目前没有MLP可用的训练算法。
- 所以，神经网络是不够实用的。



## AI发展

Minsky说过单层神经网络无法解决异或问题。但是当增加一个计算层以后，两层神经网络不仅可以解决异或问题，而且具有非常好的非线性分类效果。  
不过两层神经网络的计算是一个问题，没有一个较好的解法。

1986年，Rumelhart和Hinton等人提出了**反向传播（Backpropagation, BP）算法**，解决了两层神经网络所需要的复杂计算量问题，从而带动了业界使用两层神经网络研究的热潮。大量的教授神经网络的教材都是重点介绍两层（带一个隐藏层）神经网络的内容。



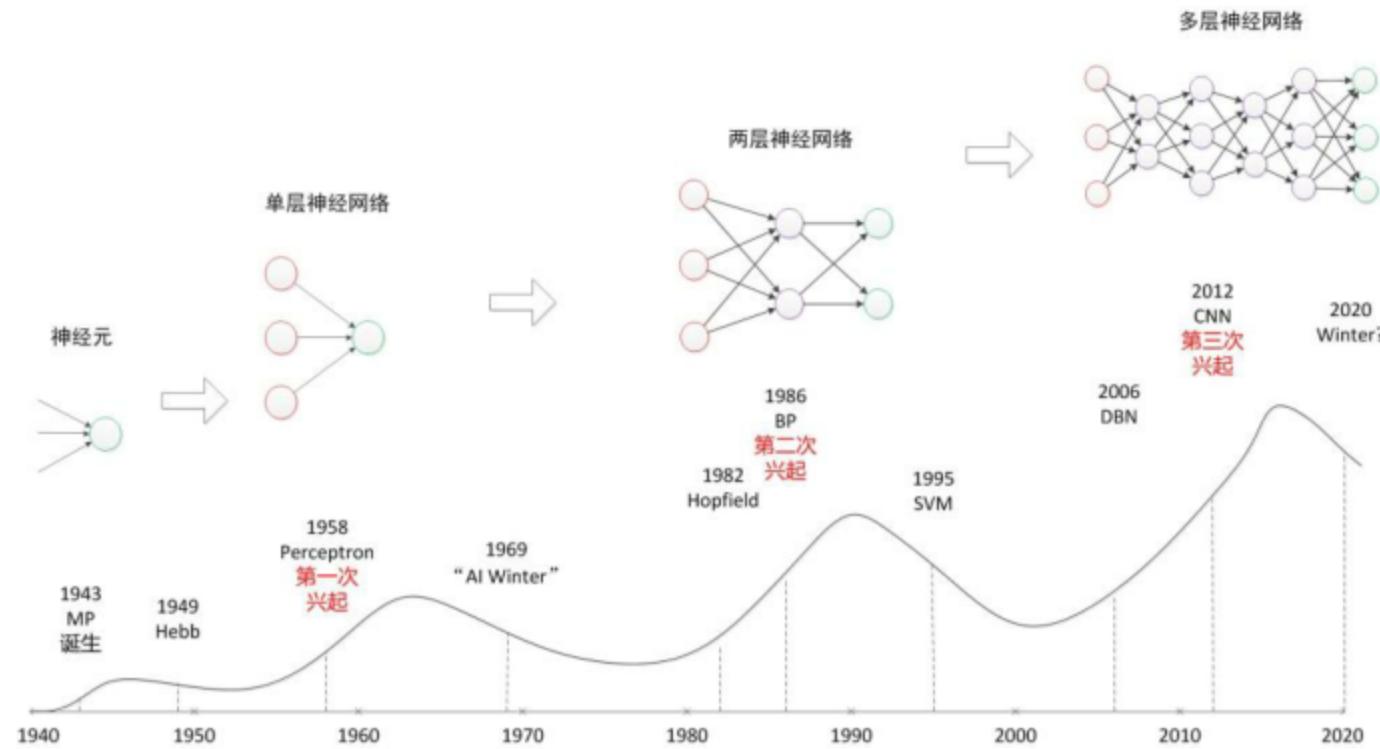
David Rumelhart、Geoffrey Hinton

1986年，David Rumelhart、Geoffrey Hinton和Ronald Williams发表了著名的文章 *Learning representations by back-propagating errors* 《通过误差反向传播进行表示学习》，回应了Minsky在1969年发出的挑战。尽管不是唯一得到这个发现的小组（其他人包括Parker, 1985; LeCun, 1985），但是这篇文章本身得益于其清晰的描述，开启了神经网络新一轮的高潮。

**BP算法**是基于一种“简单”的思路：不是（如感知器那样）用误差本身去调整权重，而是用误差的导数（梯度）。

7

## AI发展

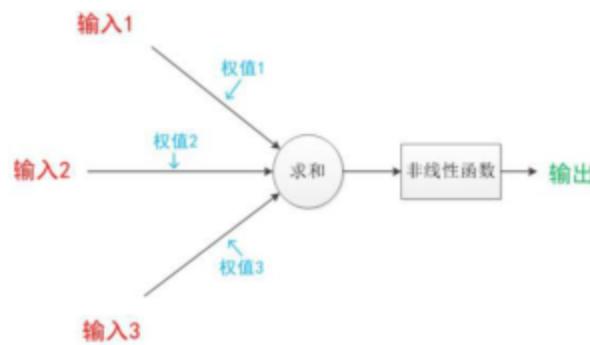


参考：神经网络——最易懂最清晰的一篇文章-CSDN博客

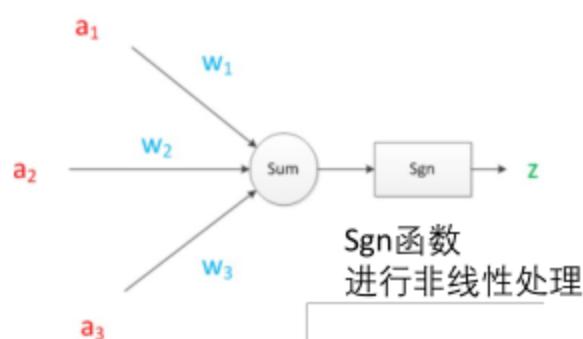
8

## 神经元模型

### 神经元模型



连接是神经元中最重要的东西。每一个连接上都有一个权重。  
一个神经网络的训练算法就是让权重的值调整到最佳，以使得整个网络的预测效果最好。

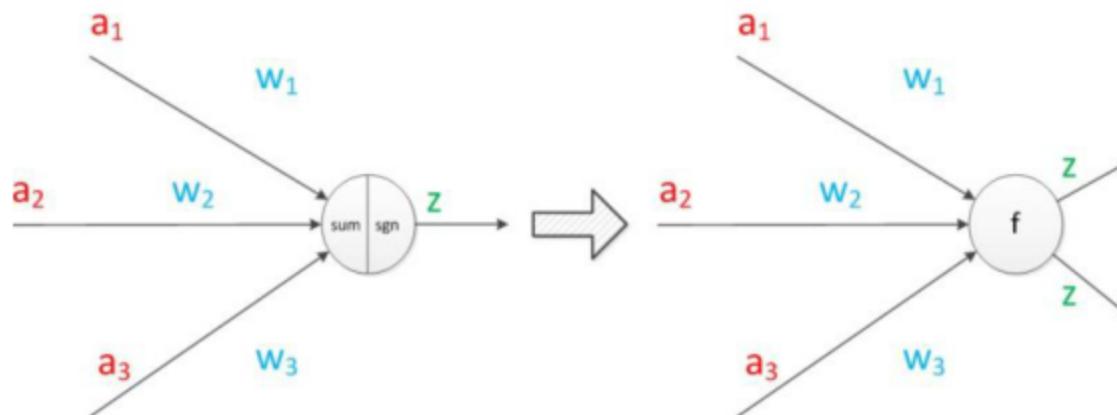


将神经元图中的所有变量用符号表示，写出输出的计算公式

$$z = g(a_1 * w_1 + a_2 * w_2 + a_3 * w_3)$$

10

## 神经元模型



神经元可以看作一个计算与存储单元。

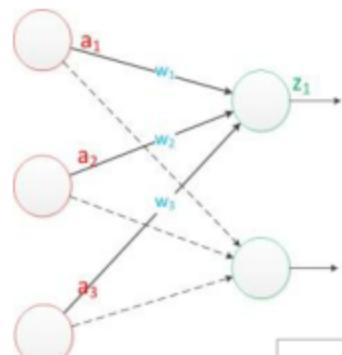
- **计算**是神经元对它的输入进行计算功能。
- **存储**是神经元会暂存计算结果，并传递到下一层。

描述网络中的某个“神经元”时，会用“**单元**”(unit)来指代。  
由于神经网络的表现形式是一个有向图，有时也会用“**节点**”(node)来表达。

11

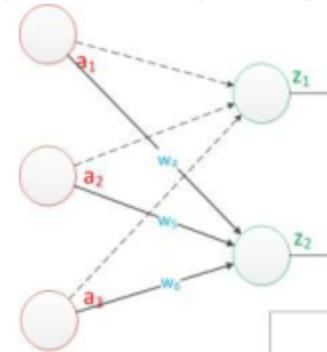
## 神经元模型

一个神经元的输出可以向多个神经元传递

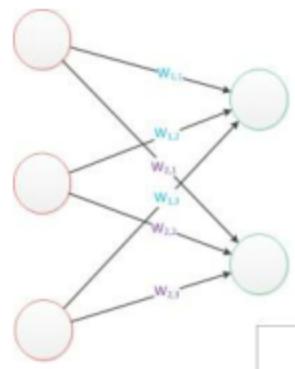


$$z_1 = g(a_1 * w_1 + a_2 * w_2 + a_3 * w_3)$$

单层神经网络(z1和z2)



$$z_2 = g(a_1 * w_4 + a_2 * w_5 + a_3 * w_6)$$



$$z_1 = g(a_1 * w_{1,1} + a_2 * w_{1,2} + a_3 * w_{1,3})$$

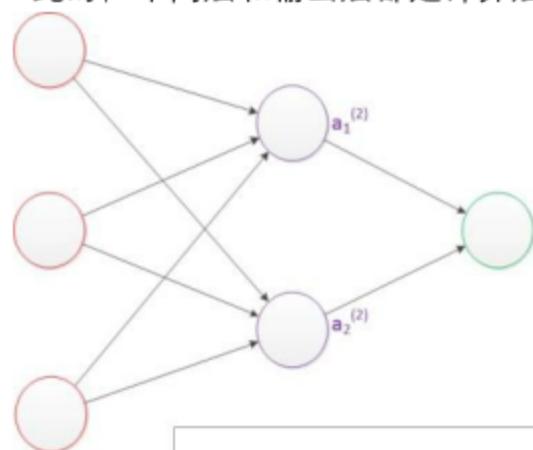
$$z_2 = g(a_1 * w_{2,1} + a_2 * w_{2,2} + a_3 * w_{2,3})$$

输入的变量  $[a_1, a_2, a_3]^T$  用向量  $\mathbf{a}$  来表示  
方程的左边是  $[z_1, z_2]^T$ ，用向量  $\mathbf{z}$  来表示。  
系数矩阵  $\mathbf{W}$  (2行3列)  
公式可以改写成： $g(\mathbf{W} \cdot \mathbf{a}) = \mathbf{z}$   
这个公式就是神经网络中从前一层计算后一层的矩阵运算。

12

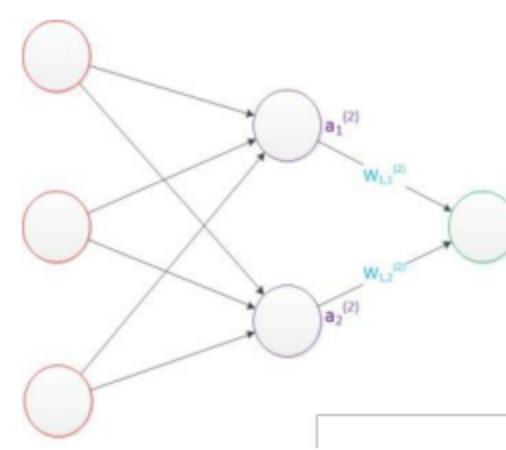
## 两层神经网络 (多层感知器)

两层神经网络除了包含一个输入层，一个输出层以外，还增加了一个中间层。  
此时，中间层和输出层都是计算层。



$$a_1^{(2)} = g(a_1^{(1)} * w_{1,1}^{(1)} + a_2^{(1)} * w_{1,2}^{(1)} + a_3^{(1)} * w_{1,3}^{(1)})$$

$$a_2^{(2)} = g(a_1^{(1)} * w_{2,1}^{(1)} + a_2^{(1)} * w_{2,2}^{(1)} + a_3^{(1)} * w_{2,3}^{(1)})$$



$$\mathbf{z} = g(a_1^{(2)} * w_{1,1}^{(2)} + a_2^{(2)} * w_{1,2}^{(2)})$$

用上标来区分不同层次之间的变量

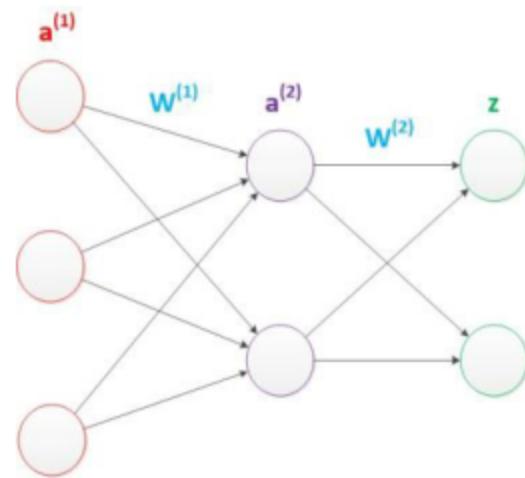
例如， $a_x^{(y)}$  代表第  $y$  层的第  $x$  个结点。  $z_1, z_2$  变成了  $a_1^{(2)}, a_2^{(2)}$ 。

计算最终输出  $z$  的方式是利用了中间层  $a_1^{(2)}, a_2^{(2)}$  和第二个权值矩阵计算得到的。

13

## ■ ■ ■ 两层神经网络（多层感知器）

假设网络的预测目标是一个向量结果，那么与前面类似，只需要在“输出层”再增加节点即可。



两层神经网络

矩阵运算公式，神经网络中从前一层计算后一层

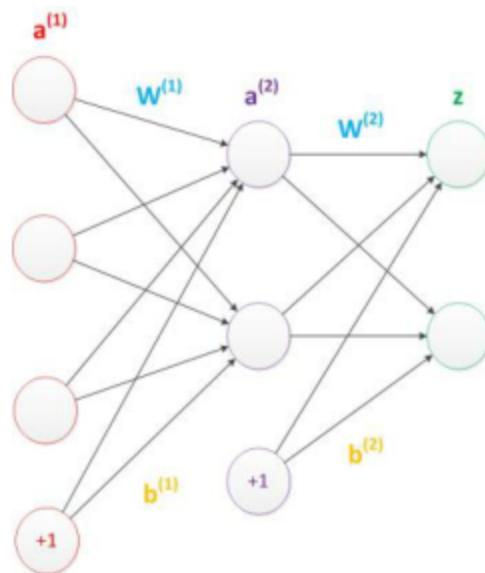
$$g(W^{(1)} * a^{(1)}) = a^{(2)}$$

$$g(W^{(2)} * a^{(2)}) = z$$

矩阵运算表达简洁、不受节点数影响

14

## ■ ■ ■ 两层神经网络（多层感知器）



- 在神经网络的每个层次中，除了输出层以外，都会含有一个偏置单元 (bias unit)。
- 这些节点是默认存在的。它本质上是一个只含有存储功能，且存储值永远为1的单元。
- 偏置单元与后一层的所有节点都有连接，我们设这些参数值为向量b，称之为偏置。

考虑了偏置以后的一个神经网络的矩阵运算

$$g(W^{(1)} * a^{(1)} + b^{(1)}) = a^{(2)}$$

$$g(W^{(2)} * a^{(2)} + b^{(2)}) = z$$

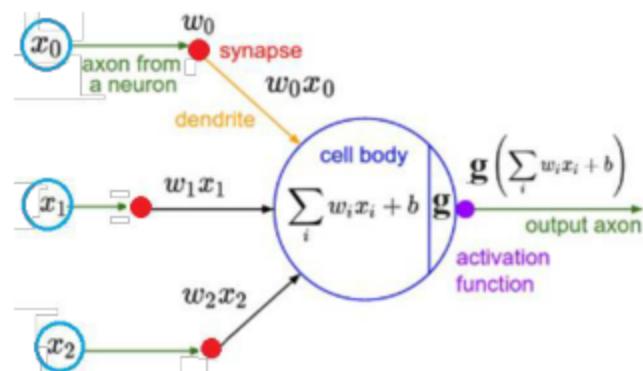
之后，不再使用sgn函数作为函数g，而是使用平滑函数sigmoid作为函数g。  
把函数g也称作激活函数 (active function)。

与单层神经网络不同。理论证明了，  
两层神经网络可以无限逼近任意连续函数。

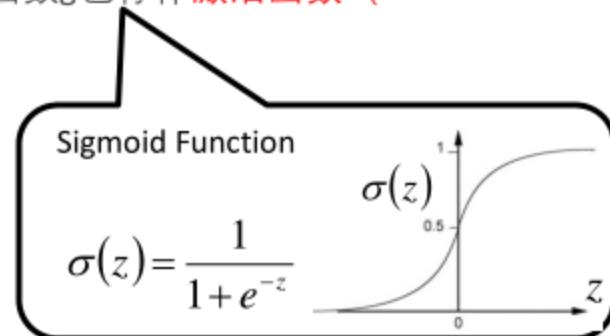
15



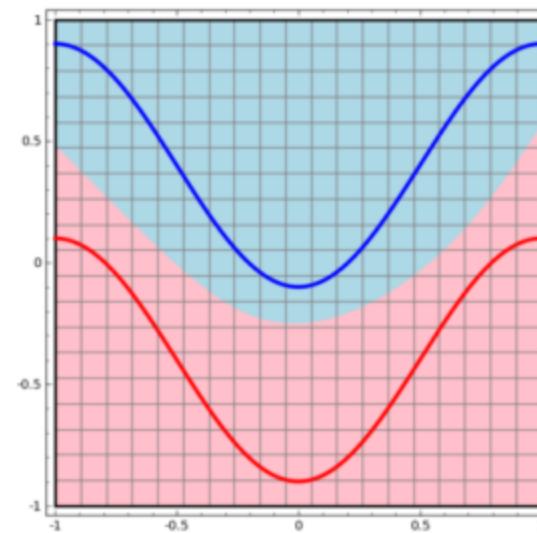
## 两层神经网络 (多层感知器)



使用平滑函数 **sigmoid** 作为函数  $g$   
把函数  $g$  也称作 **激活函数 (active function)**



与单层神经网络不同。理论证明，两层神经网络可以无限逼近任意连续函数。也就是说，面对复杂的非线性分类任务，两层神经网络可以分类的很好。

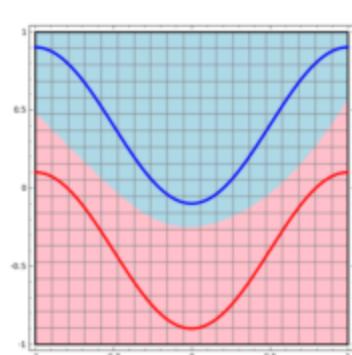


16

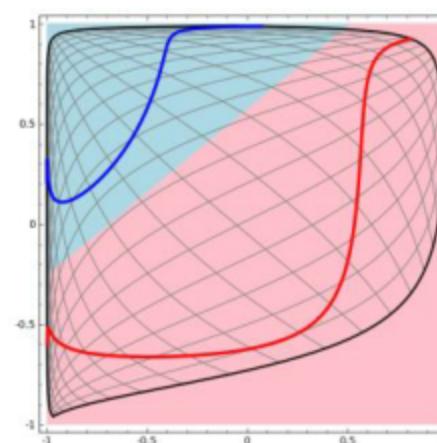


## 两层神经网络 (多层感知器)

- **单层网络只能做线性分类任务** (因为感知机是广义线性模型。不管有无激活函数，单层神经网络的决策边界都是线性的。)
- 这点可以从逻辑回归模型的**决策函数**看出，决策函数  $Y=\text{sigmoid}(wx + b)$ ，当  $wx+b>0, Y>0.5$ ; 当  $wx+b<0, Y<0.5$ ，以  $wx+b$  这条线可以区分开  $Y=0$  或  $1$
- 两层神经网络中的最后一层也是线性分类层



两层神经网络 (决策分界)



把输出层的决策分界单独拿出来看

输出层的决策分界仍然是直线。

从输入层到隐藏层时，数据发生了**空间变换**。隐藏层对原始的数据进行了一个空间变换，使其可以被线性分类，然后输出层的决策分界划出了一个线性分类分界线，对其进行分类。

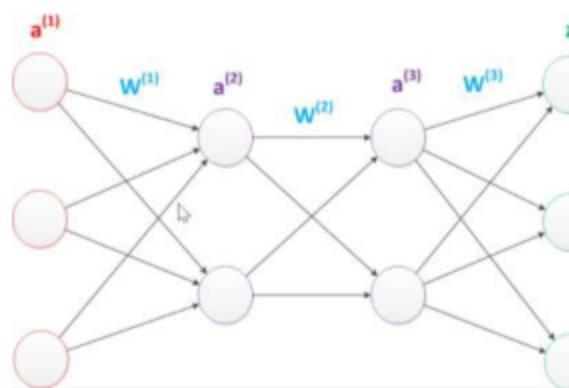
17

## 多层神经网络

在两层神经网络的输出层后面，继续添加层次。

原来的输出层变成中间层，新加的层成为新的输出层。

所以可以得到下图：



$$g(W^{(1)} * a^{(1)}) = a^{(2)}$$

$$g(W^{(2)} * a^{(2)}) = a^{(3)}$$

$$g(W^{(3)} * a^{(3)}) = z$$

多层神经网络中，输出也是按照一层一层的方式来计算。从最外面的层开始，算出所有单元的值以后，再继续计算更深一层。只有当前层所有单元的值都计算完毕以后，才会算下一层。有点像计算向前不断推进的感觉。所以这个过程叫做“正向传播”。

18

## 反向传播



David Rumelhart、Geoffery Hinton

1986年，David Rumelhart、Geoffrey Hinton和Ronald Williams发表了著名的文章 *Learning representations by back-propagating errors* 《通过误差反向传播进行表示学习》，回应了Minsky在1969年发出的挑战。尽管不是唯一得到这个发现的小组（其他人包括Parker, 1985；LeCun, 1985），但是这篇文章本身得益于其清晰的描述，开启了神经网络新一轮的高潮。

**BP算法**是基于一种“简单”的思路：

不是（如感知器那样）用误差本身去调整权重，而是用**误差的导数（梯度）**。

让神经网络根据输出值  $h_{\theta}(x)$  和真实值  $y$  之间的差别，反向更新参数矩阵  $\theta$ ，使之拟合得更好，这就是反向传播。

反向传播算法的启示是数学中的**链式法则**

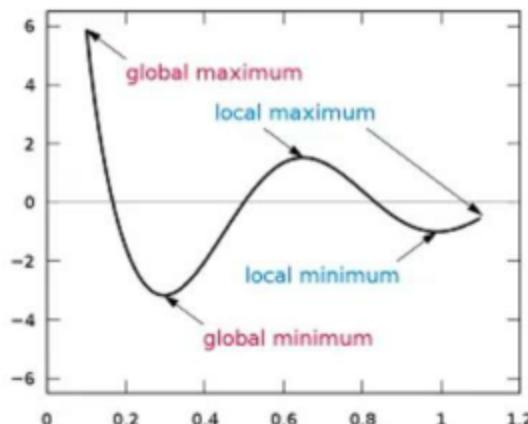
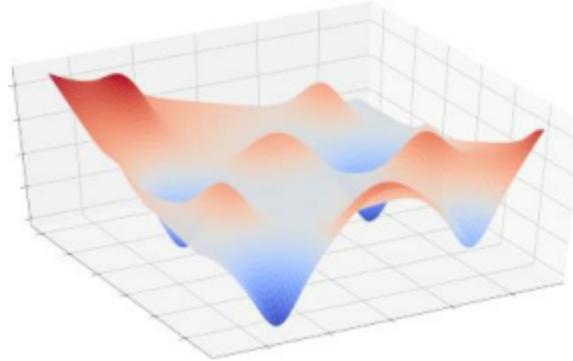
早期神经网络的研究人员努力从**生物学**中得到启发，

但从BP算法开始，研究者们更多地从**数学**上寻求问题的最优解。

19

## 反向传播

对于链式求导的**梯度下降算法**，物理学的解释是在一个误差构成的“能量函数”图上，沿着山坡最陡峭的路线下行，直到达到一个稳定的极小值，也即“收敛”点。



梯度下降法可以找到局部最小值

**反向传播算法**是利用了神经网络的结构进行的计算。

- 不是一次计算所有参数的梯度，而是从后往前。
- 首先计算输出层的梯度，然后是第二个参数矩阵的梯度，接着是中间层的梯度，再然后是第一个参数矩阵的梯度，最后是输入层的梯度。
- 计算结束以后，所要的两个参数矩阵的梯度就都有了。

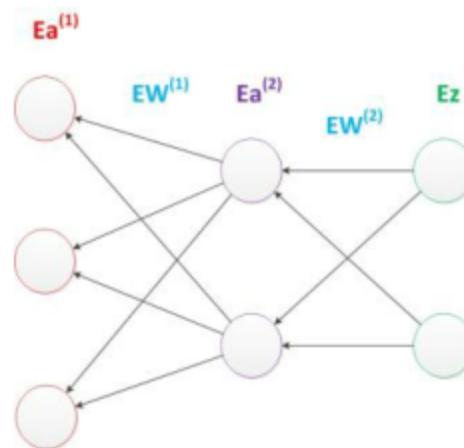
[【官方双语】反向传播超简明教程，解析神经网络参数优化的核心技术\\_bilibili](#)

20

## 反向传播

[反向传播算法可视化展示\\_bilibili](#)

反向传播算法可以直观的理解为下图。梯度的计算从后往前，一层层反向传播。前缀E代表着相对导数的意思。



**优化问题**只是训练中的一个部分。机器学习问题之所以称为学习问题，而不是优化问题，就是因为它不仅要求数据在训练集上求得一个较小的误差，在测试集上也要表现好。因为模型最终是要部署到没有见过训练数据的真实场景。

提升模型在测试集上的预测效果的主题叫做**泛化 (generalization)**，相关方法被称作**正则化 (regularization)**。神经网络中常用的泛化技术有权重衰减等。

21

## 反向传播

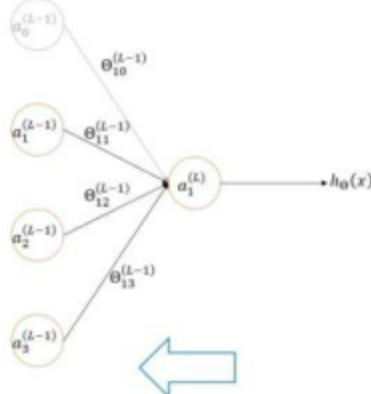
### 反向传播 (Back Propagation, BP)

让神经网络根据输出值  $h_{\Theta}(x)$  和真实值  $y$  之间的差别，反向更新参数矩阵  $\Theta$ ，使之拟合得更好，这就是反向传播。

对于只有一个输出值、一个样本、一共  $L$  层的神经网络，以平方误差代价函数为例：

$$J(\Theta) = (h_{\Theta}(x) - y)^2 = (a_1^{(L)} - y)^2$$

我们希望求取的是它对每一个参数的偏导  $\frac{\partial J}{\partial \Theta_{ij}^{(l)}}$ ，以进行梯度下降。如果直接把  $h_{\Theta}(x)$  展开计算偏导数，这太复杂了！但我们有一个巧妙的工具：链式法则，可利用它把偏导数一层层剥开，最终求出。



先考虑最后一层与倒数第二层的关系，  
计算一下最后一层参数的偏导数

22

## 反向传播

为了方便，记线性组合的结果为  $z_1^{(L)} = \Theta_1^{(L-1)} a^{(L-1)}$ ，这样有  $a_1^{(L)} = g(z_1^{(L)})$ 。由链式法则有

$$\frac{\partial J}{\partial \Theta_{1j}^{(L-1)}} = \frac{\partial z_1^{(L)}}{\partial \Theta_{1j}^{(L-1)}} \cdot \frac{\partial a_1^{(L)}}{\partial z_1^{(L)}} \cdot \frac{\partial J}{\partial a_1^{(L)}}$$

分别计算这三个式子，得到

$$\frac{\partial z_1^{(L)}}{\partial \Theta_{1j}^{(L-1)}} = a_j^{(L-1)}$$

$$\frac{\partial a_1^{(L)}}{\partial z_1^{(L)}} = \frac{\partial g(z_1^{(L)})}{\partial z_1^{(L)}} = g'(z_1^{(L)})$$

$$\frac{\partial J}{\partial a_1^{(L)}} = 2(a_1^{(L)} - y)$$

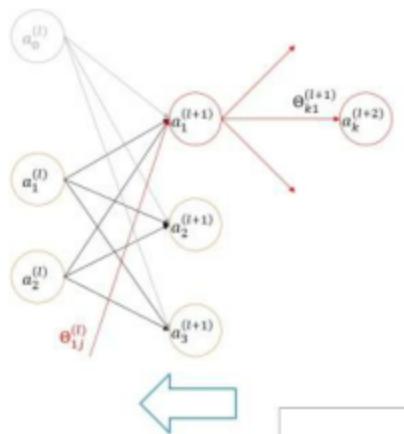
因此，我们得到了

$$\frac{\partial J}{\partial \Theta_{1j}^{(L-1)}} = 2a_j^{(L-1)} g'(z_1^{(L)}) (a_1^{(L)} - y)$$

23

## 反向传播

对任意层参数的偏导数



易知

$$\frac{\partial J}{\partial \theta_{ij}^{(l)}} = \frac{\partial z_i^{(l+1)}}{\partial \theta_{ij}^{(l)}} \cdot \frac{\partial a_i^{(l+1)}}{\partial z_i^{(l+1)}} \cdot \frac{\partial J}{\partial a_i^{(l+1)}}$$

注意  $J$  对  $a_1^{(l+1)}$  求偏导比较复杂，因为  $a_1^{(l+1)}$  会通过多种途径影响后面的第  $l+2$  层，此时用链式法则的话需要求和，即

$$\begin{aligned} \frac{\partial J}{\partial a_1^{(l+1)}} &= \sum_k \frac{\partial z_k^{(l+2)}}{\partial a_1^{(l+1)}} \cdot \frac{\partial a_k^{(l+2)}}{\partial z_k^{(l+2)}} \cdot \frac{\partial J}{\partial a_k^{(l+2)}} \\ &= \sum_k \Theta_{k1}^{(l+1)} g'(z_k^{(l+2)}) \frac{\partial J}{\partial a_k^{(l+2)}} \end{aligned}$$

上式是一个递推式。可以求出  $J$  对于任何一激活值  $a_i^{(l)}$  的偏导，从而求出  $J$  对任何一参数  $\Theta_{ij}^{(l)}$  的偏导。

### 反向传播的公式整理

将上述结论整理如下：

$$\frac{\partial J}{\partial \theta_{ij}^{(l)}} = a_j^{(l)} g'(z_i^{(l+1)}) \frac{\partial J}{\partial a_i^{(l+1)}}$$

$$\frac{\partial J}{\partial a_i^{(l)}} = \begin{cases} \sum_k \Theta_{ki}^{(l)} g'(z_k^{(l+1)}) \frac{\partial J}{\partial a_k^{(l+1)}}, & l \neq L \\ 2(a_i^{(L)} - y_i), & l = L \end{cases}$$

注意到公共的部分，我们记

$$\delta_i^{(l)} = g'(z_i^{(l)}) \frac{\partial J}{\partial a_i^{(l)}} = \frac{\partial J}{\partial z_i^{(l)}}$$

那么  $\delta$  的递推式为

$$\delta_i^{(l)} = \begin{cases} g'(z_i^{(l)}) \sum_k \Theta_{ki}^{(l)} \delta_k^{(l+1)}, & l \neq L \\ 2g'(z_i^{(L)}) (a_i^{(L)} - y_i), & l = L \end{cases}$$

可用矩阵的形式记为

$$\delta^{(l)} = \begin{cases} ((\Theta^{(l)})^T \delta^{(l+1)}) \odot g'(z^{(l)}), & l \neq L \\ 2(a^{(L)} - y) \odot g'(z^{(L)}), & l = L \end{cases}$$

【这里的  $\odot$  符号是哈达玛积 (Hadamard product)，表示两个向量对应元素相乘，结果仍为向量。】

最终用  $\frac{\partial J}{\partial \theta_{ij}^{(l)}} = a_j^{(l)} \delta_i^{(l+1)}$  就可求出每个参数的梯度。

24

## 训练过程

从两层神经网络开始，神经网络的研究人员开始使用机器学习相关的技术进行神经网络的训练。例如用大量的数据（1000-10000左右），使用算法进行优化等等，从而使得模型训练可以获得性能与数据利用上的双重优势。

机器学习模型训练的目的，就是使得参数尽可能的与真实的模型逼近。

具体做法：

- a) 首先给所有参数赋上随机值。使用这些随机生成的参数值，来预测训练数据中的样本。样本的预测目标为  $Y_p$ ，真实目标为  $Y$ 。那么，定义一个值  $loss$ ，损失 ( $loss$ ) 计算公式如下。

$$loss = \frac{1}{2} (Y_p - Y)^2$$

训练目标就是使对所有训练数据的损失和尽可能的小。

- a) 如果将先前的神经网络预测的矩阵公式带入到  $Y_p$  中（因为有  $z = Y_p$ ），那么可以把损失写为关于参数 (parameter) 的函数，这个函数称之为 **损失函数 (loss function)**。
- b) 下面的问题就是求：如何优化参数，能够让损失函数的值最小。这个问题就被转化为一个优化问题。

- 优化问题求解常用方法就是高等数学中的求导，但是这里的问题由于参数不止一个，求导后计算导数等于0的运算量很大，所以一般来说解决这个优化问题使用的是 **梯度下降算法**。
- 梯度下降算法每次计算参数在当前的梯度，然后让参数向着梯度的反方向前进一段距离，不断重复，直到梯度接近零时截止。一般这个时候，所有的参数恰好达到使损失函数达到一个最低值的状态。
- 在神经网络模型中，由于结构复杂，每次计算梯度的代价很大。因此还需要使用 **反向传播** 算法。

## 训练过程

首先随机初始化神经网络中的所有参数  $\Theta$ . 注意, 不能简单地全部初始化为零, 否则根据对称性, 神经网络会退化成一条链。

对于  $m$  个样本点  $(x^{(1)}, y^{(1)})$ 、 $(x^{(2)}, y^{(2)}) \dots, (x^{(m)}, y^{(m)})$ , 训练的伪代码如下:

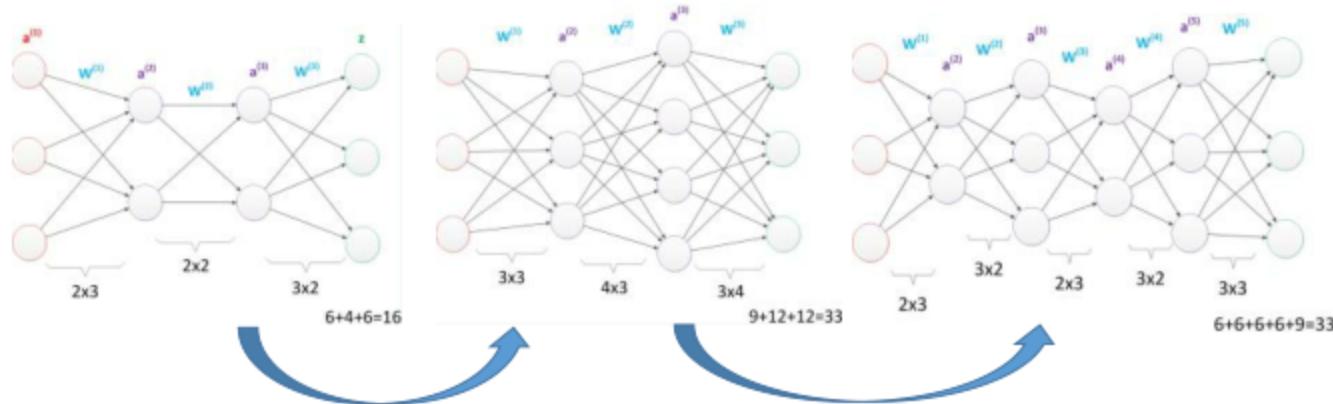
```

1. 循环直至收敛:
2.   令  $\Delta_{ij}^{(l)} = 0$  (for all  $i, j, l$ , 下同)           // 统计对每个参数的调整量
3.   for index = 1 to m:
4.     令  $a^{(1)} = x^{(index)}$                                 // 输入层赋值
5.     使用前向传播计算所有  $a^{(l)}$  (for  $l = 2$  to  $L$ )
6.     使用反向传播计算所有  $\delta^{(l)}$  (for  $l = L$  to 2)
7.      $\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$       // 累加每个样本产生的偏导数
8.   令  $D_{ij}^{(l)} = \begin{cases} \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \theta_{ij}^{(l)}, & \text{if } j \neq 0 \\ \frac{1}{m} \Delta_{ij}^{(l)}, & \text{if } j = 0 \end{cases}$           // 对各个样本求平均, 并给予非零次项正则化调整
9.    $\theta_{ij}^{(l)} := \theta_{ij}^{(l)} - \alpha D_{ij}^{(l)}$                   // 调整参数

```

## 多层感知器

讨论一下多层神经网络中的参数



虽然层数保持不变, 但是第二个神经网络的参数数量却是第一个神经网络的接近两倍之多, 从而带来了更好的表示能力。

虽然参数数量仍然是 33, 但却有 4 个中间层, 是原来层数的接近两倍。这意味着一样的参数数量, 可以用更深的层次去表达。

与两层层神经网络不同, 多层神经网络中的层数增加了很多, 从而拥有更深入的表示特征, 以及更强的函数模拟能力。

## 多层感知器

BP算法大获成功，引起了人们对“**连接主义**”方法的极大兴趣。

数以百计的新模型被提出来，比如**Hopfield**网络、**自组织特征映射（SOM）**网络、**双向联想记忆（BAM）**、**卷积神经网络**、**循环神经网络**、**玻尔兹曼机**等。物理学家也带来了很多新方法和新概念，如**协同学习**、**模拟退火**、**随机场**、**平均场**和各种从统计物理学中借鉴过来的概念。其实后来深度学习复兴时代的很多算法，都是在那时候就已经被提出来了。

**更强的函数模拟能力**是由于随着层数的增加，整个网络的参数就越多。而神经网络其实本质就是模拟特征与目标之间的真实关系函数的方法，更多的参数意味着其模拟的函数可以更加的复杂，可以有更多的容量去拟合真正的关系。

**更深入的表示特征**可以这样理解，随着网络的层数增加，每一层对于前一层次的抽象表示更深入。在神经网络中，每一层神经元学习到的是前一层神经元值的更抽象的表示。例如第一个隐藏层学习到的是“边缘”的特征，第二个隐藏层学习到的是由“边缘”组成的“形状”的特征，第三个隐藏层学习到的是由“形状”组成的“图案”的特征，最后的隐藏层学习到的是由“图案”组成的“目标”的特征。通过抽取更抽象的特征来对事物进行区分，从而获得更好的区分与分类能力。

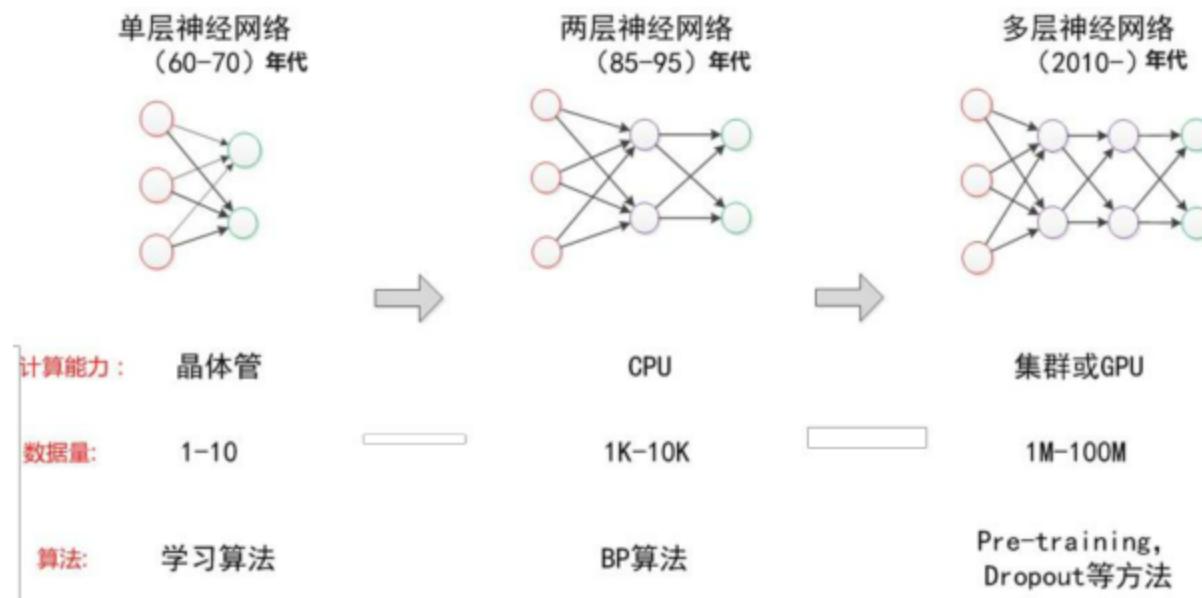
28

## 多层感知器

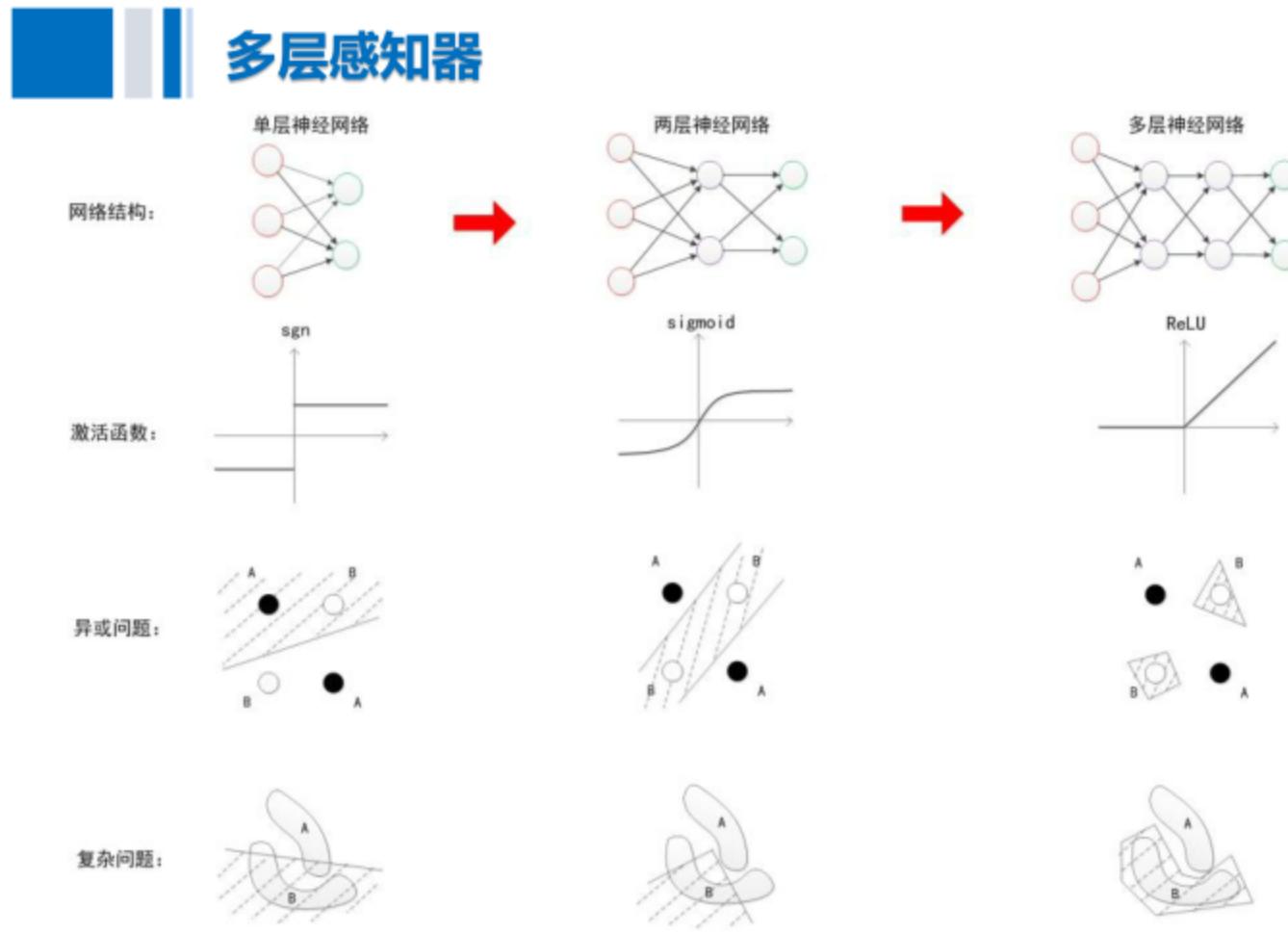
通常设计神经网络要把它设计得深一点。

这与人类解决问题时的**模块化**（modularization）思想是一致的。

比如说第一二层隐藏层负责粗分类，第三四层隐藏层负责细分类，整体效率会更高。



29



30

## ◆ 参考资料

### 1、斯坦福大学吴恩达机器学习系列课程

<https://www.bilibili.com/video/BV164411b7dx?from=search&seid=12860939662404817484>

### 2、台湾大学李宏毅机器学习课程

<https://www.bilibili.com/video/BV13x411v7US?from=search&seid=12860939662404817484>

### 3、3Blue1Brown

[https://space.bilibili.com/88461692?spm\\_id\\_from=333.788.b\\_765f7570696e666f.1](https://space.bilibili.com/88461692?spm_id_from=333.788.b_765f7570696e666f.1)