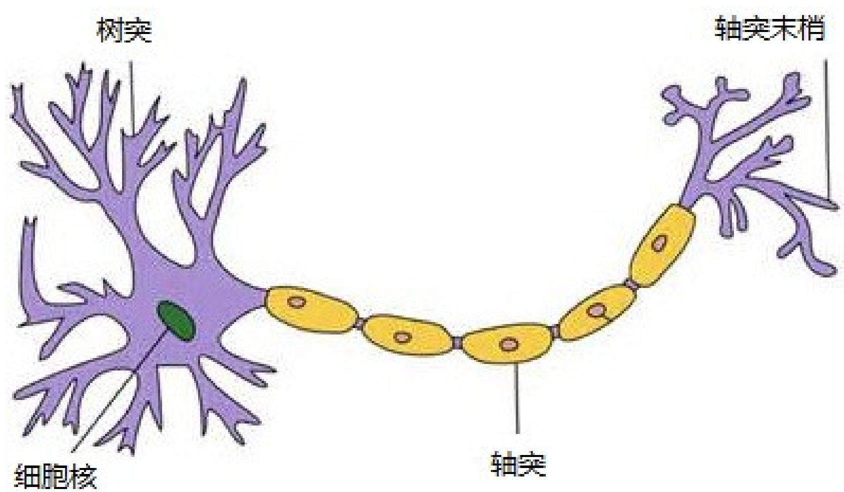
人工智能：机器学习 III

饶洋辉
计算机学院,
中山大学
raoyangh@mail.sysu.edu.cn
http://cse.sysu.edu.cn/node/2471

课件来源：中山大学陈川副教授等

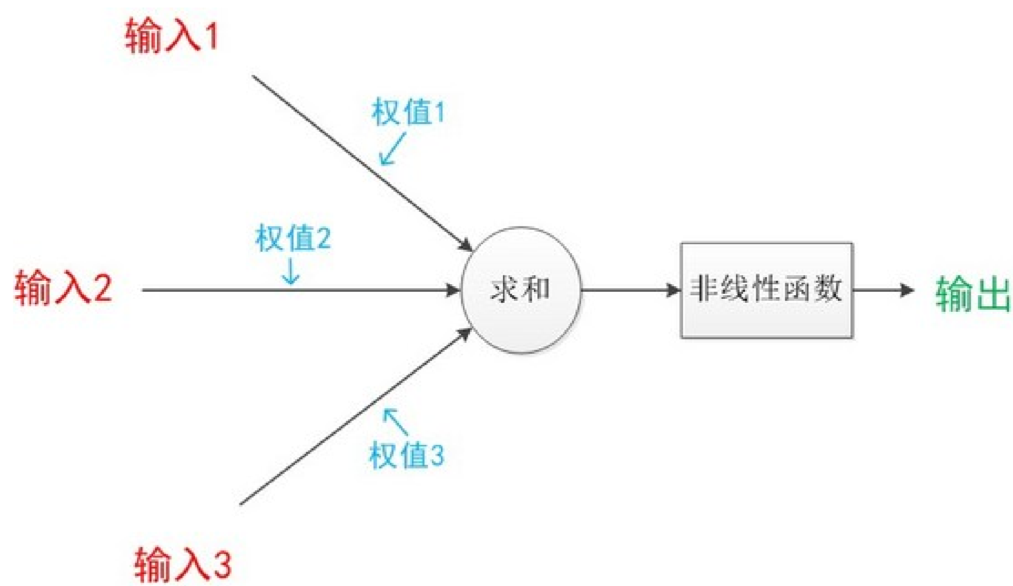
人工神经网络

- 人工神经网络（ANN）模型受到生物神经系统的启发。
- 人脑主要由称为神经元的神经细胞组成。1904年，生物学家已经知晓神经元的组成结构。
- 一个神经元通常具有多个树突，主要用来接受传入信息；而轴突只有一条，轴突尾端有许多轴突末梢可以给其他多个神经元传递信息。轴突末梢跟其他神经元的树突产生连接，从而传递信号。这个连接的位置在生物学上叫做“突触”。



人工神经网络

- 类比生物结构：输入可以类比为神经元的树突，而输出可以类比为神经元的轴突，计算则可以类比为细胞核。神经元模型是一个包含输入，输出与计算功能的模型。
- 下图是一个典型的人工神经网络模型，它包含3个输入，1个输出，以及2个计算功能。
- 中间的箭头线被称为“连接”，它们附有一个“权值”。



人工神经网络

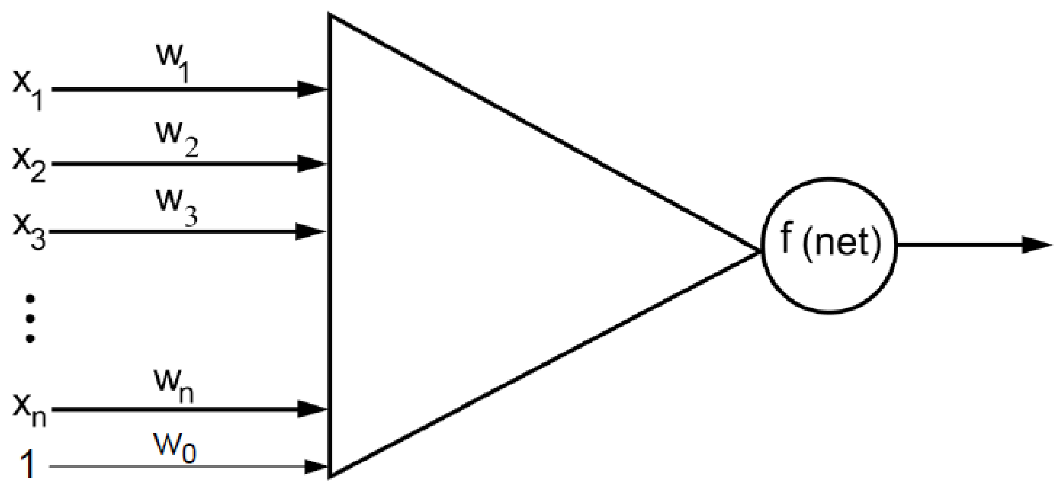
- 人工神经网络中的计算单位是人工神经元

- 人工神经元的组成部分为：
  - 输入信号 x_i. 这些信号代表来自环境的数据或其他神经元的激活(activation)
  - 一组实值(real-value)权重 w_i. 这些权重的值代表连接强度
  - 一个激活层 $\sum_i w_i x_i$. 神经元的激活层由加权输入的总和确定
  - 一个阈值函数 f. 该函数通过确定激活是低于还是高于阈值来计算最终输出

单层神经网络

- 给定一个激活值 $net = \sum_i w_i x_i$，该神经元的输出为：

$$f(net) = \begin{cases} +1 & if \sum_i w_i x_i > 0 \\ 0 & if \sum_i w_i x_i = 0 \\ -1 & if \sum_i w_i x_i < 0 \end{cases}$$



示例

- 可以使用单层神经网络来计算逻辑“与”功能
  - 神经元有三个输入
  - x_1 和 x_2 表示原始输入
  - 第三个是具有固定值+1的偏置输入
  - 输入数据和偏差分别具有+1，+1和-1.5的权重
- 那么逻辑或功能呢?

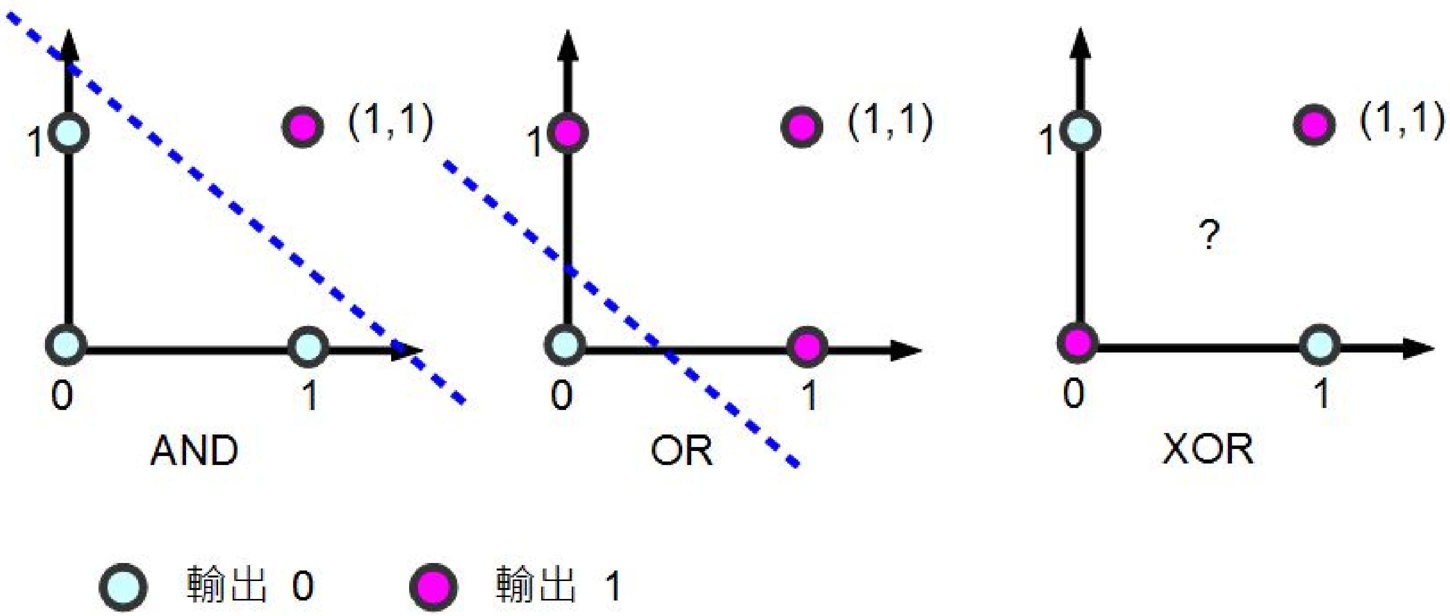
单层神经网络

- 感知机学习算法（PLA）可用于调整人工神经元的权重
- 调整权重，直到神经元的输出与训练样例的真实输出一致
- 使用以下规则

$$\mathbf{w}_{(t+1)} \leftarrow \mathbf{w}_{(t)} + y_{n(t)}\mathbf{x}_{n(t)}$$

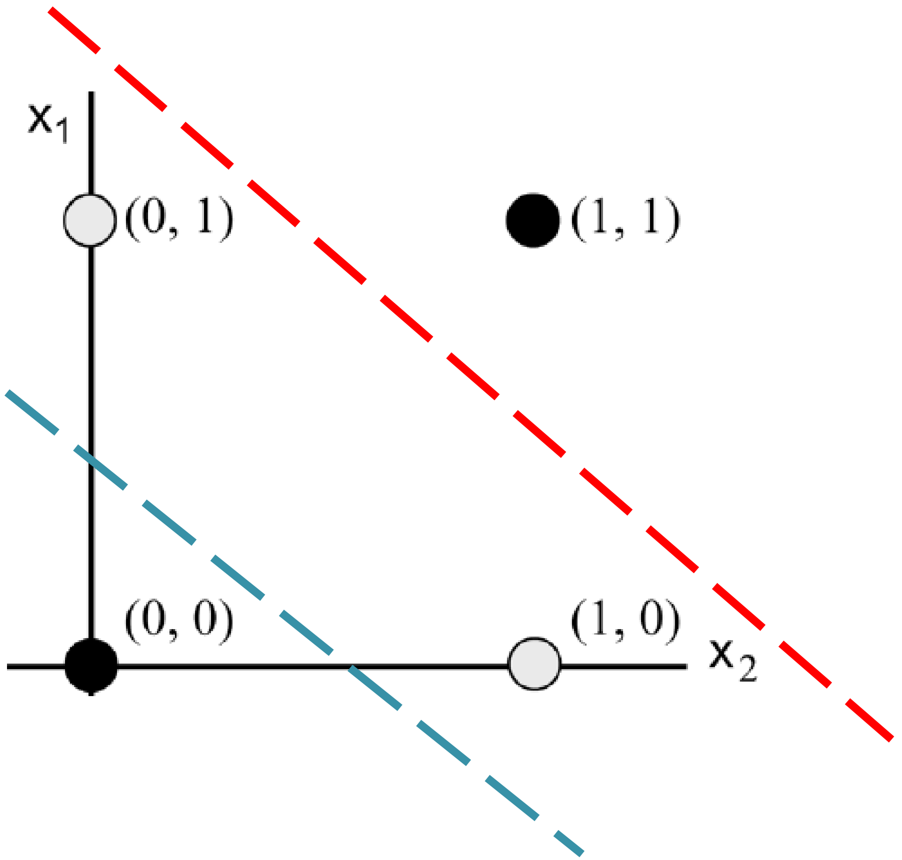
单层神经网络

- PLA只能做简单的线性分类任务。但是当时的人们热情太过于高涨，并没有人清醒的认识到这点
- Minsky在1969年出版了《Perceptron》这本书，里面用详细的数学证明了PLA的弱点，尤其是它对XOR（异或）这样的简单分类任务都无法解决



示例

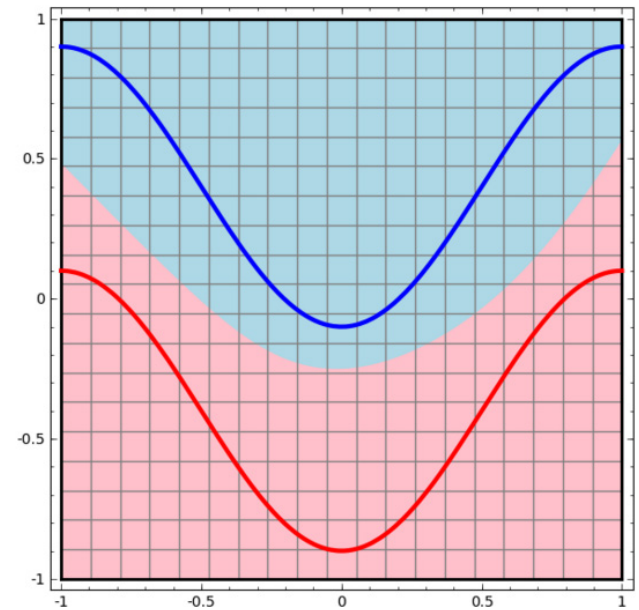
x_1	x_2	Output
1	1	-1
1	0	1
0	1	1
0	0	-1



多层神经网络

- 理论证明，两层神经网络可以无限逼近任意连续函数。面对复杂的非线性分类任务，带一个隐藏层的两层神经网络可以分类的很好。

- 示例：
  - 红色的线与蓝色的线代表数据
  - 红色区域和蓝色区域代表由神经网络划开的区域，两者的分界线就是决策分界

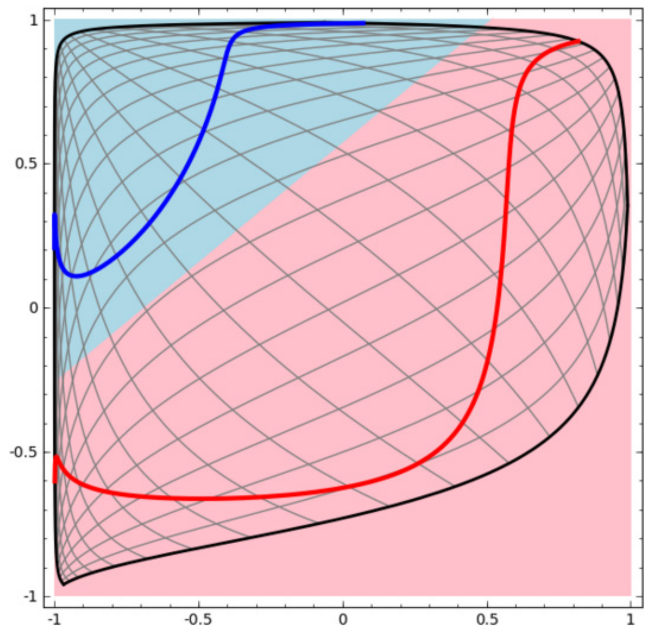


两层神经网络（决策分界）

- 这个两层神经网络的决策分界是非常平滑的曲线，而且分类的很好。
- Q：单层网络只能进行线性分类。而两层神经网络中的后一层也是线性分类层，应该只能做线性分类任务。为什么两个线性分类任务结合就可以做非线性分类任务？

多层神经网络

- 我们可以把输出层的决策分界单独拿出来看一下。就是下图：

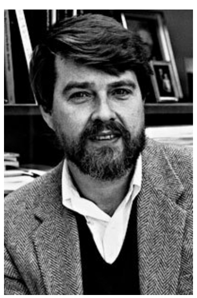


两层神经网络（空间变换）

- 输出层的决策分界仍然是直线。关键是，从输入层到隐藏层时，数据发生了空间变换。也就是说，两层神经网络中，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

多层神经网络

- 单层神经网络无法解决异或问题，但是当增加一个隐藏层之后，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。不过多层神经网络的计算是一个问题，没有一个较好的解法。
- 1986年，Rumelhar和Hinton等人提出了反向传播（Backpropagation，BP）算法，解决了多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。
- 这时候的Hinton还很年轻，30年以后，正是他重新定义了神经网络，带来了神经网络复苏。



David Rumelhart

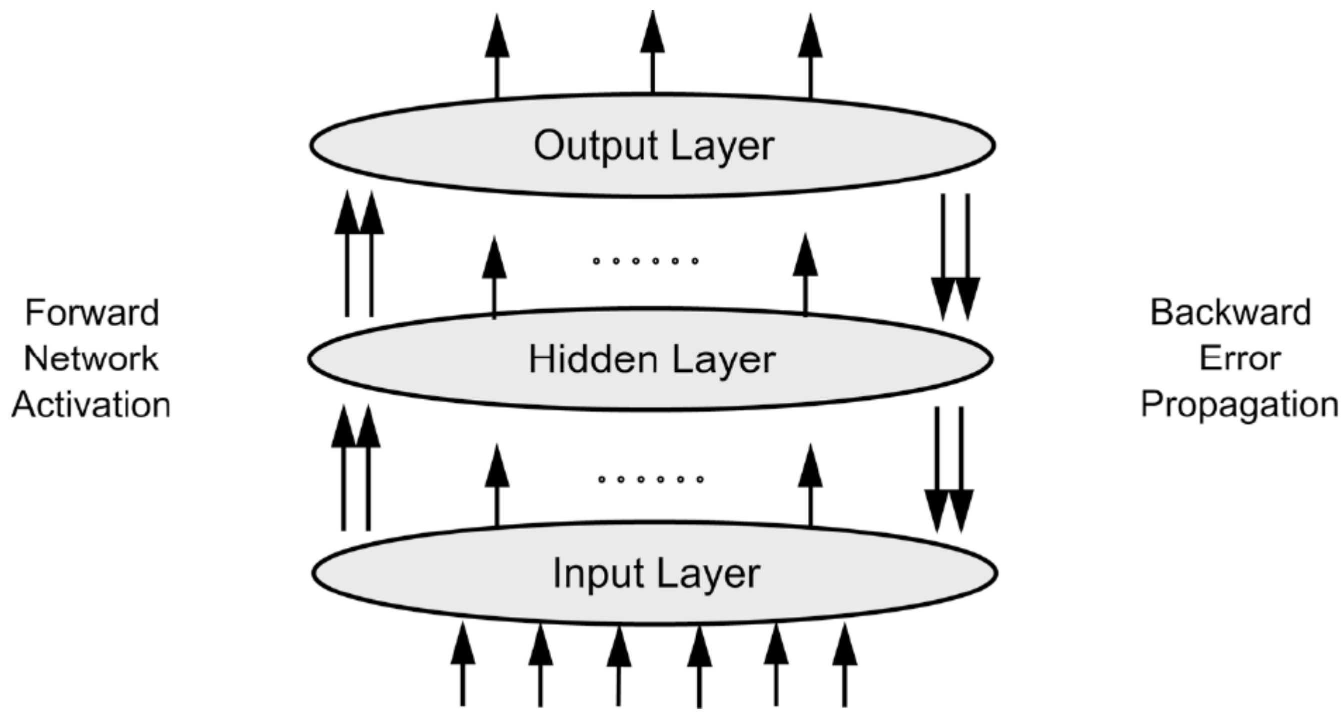
Geoffery Hinton

多层神经网络

- 输入和输出节点之间的附加层称为隐藏层
- 嵌入在这些层中的节点被称为隐藏节点
- 一个前馈神经网络(feedforward neural networks)，其中一层中的节点仅连接到下一层中的节点
- 反向传播学习算法是专门为多层神经网络设计的

多层神经网络

- 训练算法的每次迭代中有两个阶段
  - 前向传递阶段(forward phase)
  - 反向传递阶段(backward phase)



多层神经网络

- 在前向传递阶段，从前一次迭代获得的权重被用来计算每个神经元的输出值

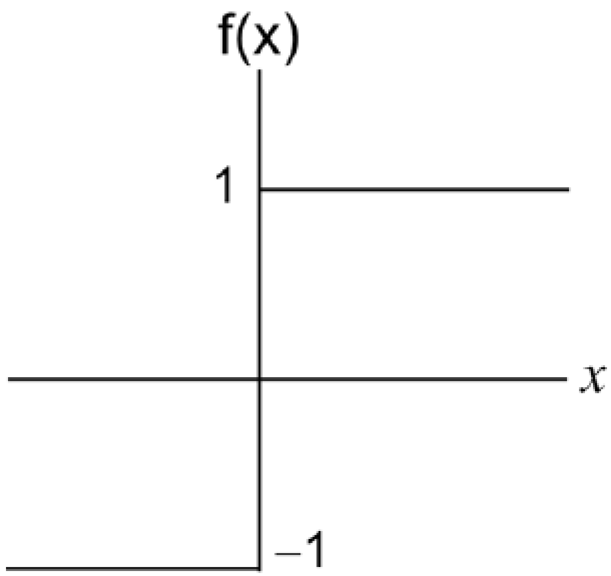
- 在计算第(l+1)层的输出之前计算第(l)层处的神经元的输出

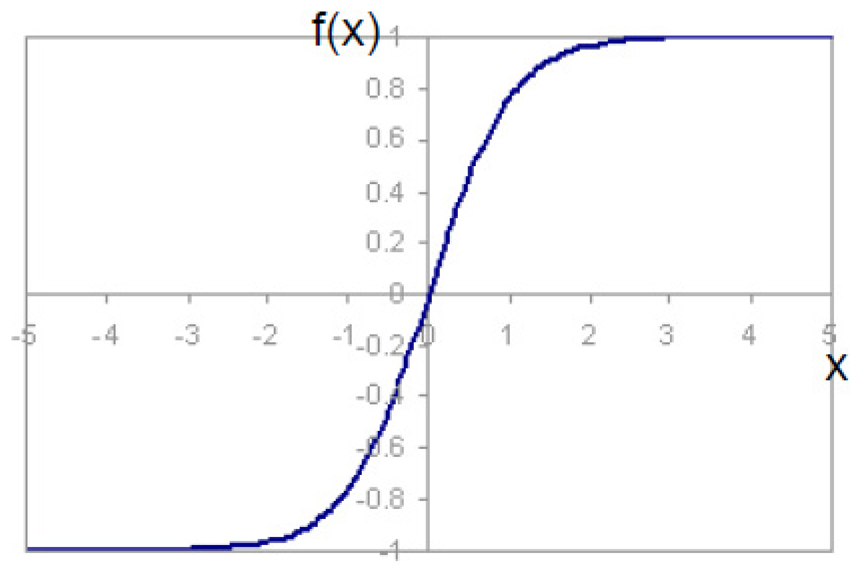
多层神经网络

- 在反向传递阶段，权重更新方程应用于相反的方向
- 换句话说，第$(l+1)$层的权重在更新第(l)层的权重之前被更新
- 这种学习算法允许我们使用第$(l+1)$层神经元的误差来估计第(l)层神经元的误差

多层神经网络

- 对于这种类型的网络，不采用PLA的阈值函数，而是使用另一个激活函数(activation function)





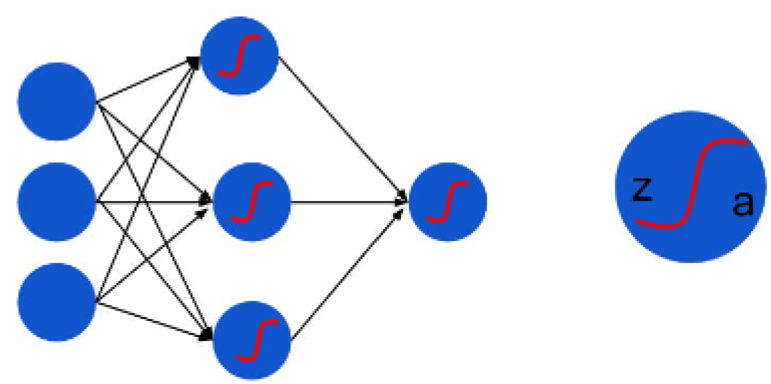
多层神经网络

- 常见的激活函数是双曲正切函数(tanh)

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

- 这个函数的一个重要属性是可求导的

$$f'(x)=1-f(x)^2$$



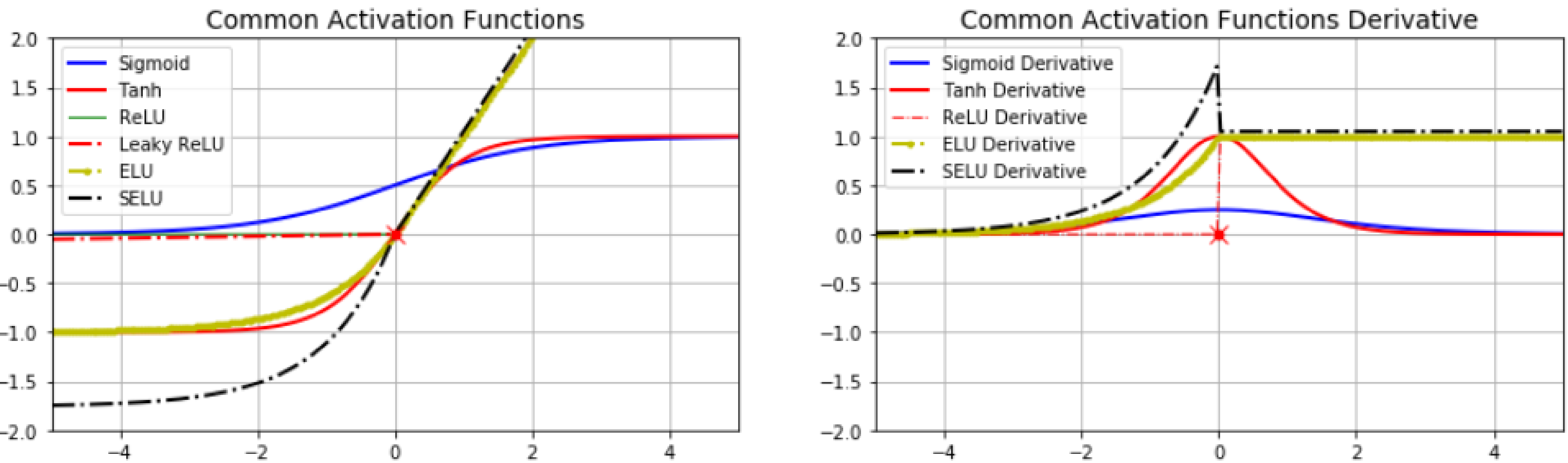
多层神经网络

- 另一个常见的激活函数是sigmoid函数

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

- 这个函数也具备可求导的属性

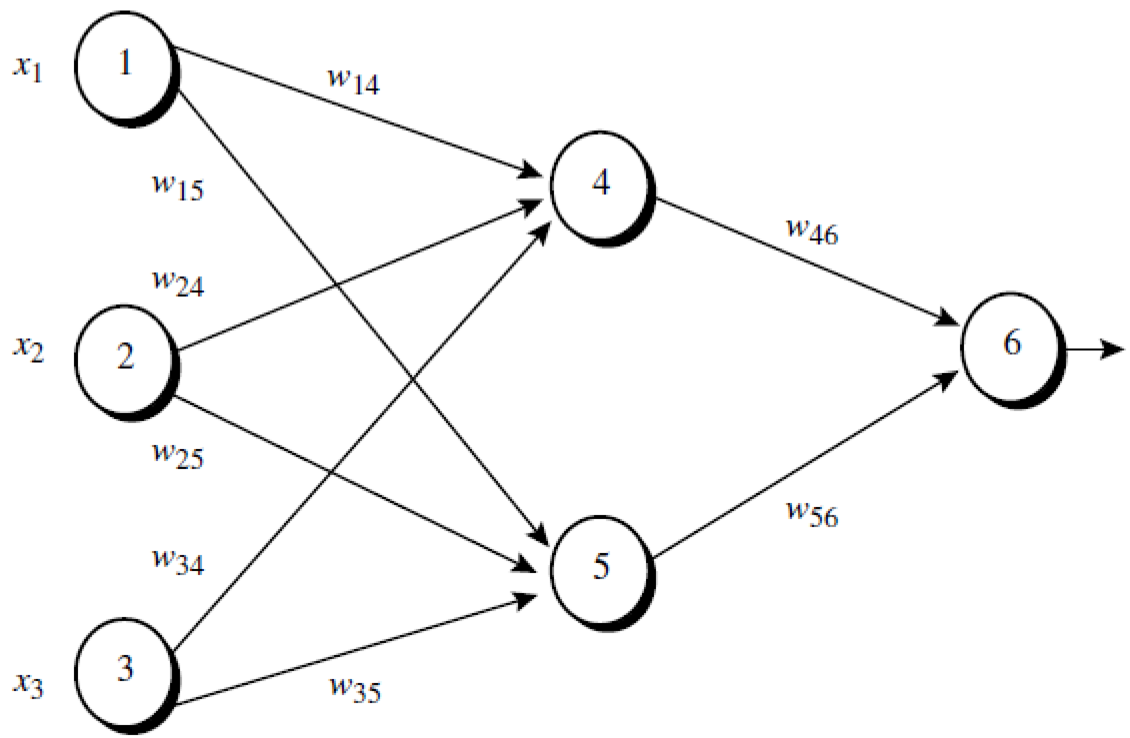
$$f'(x)=f(x)(1-f(x))$$



常用的激活函数及其求导

示例

- 一个训练元组X=(1,0,1), 标签为1



J. Han, M. Kamber, J. Pei著; 范明等译. *数据挖掘：概念与技术（第3版）*. 机械工业出版社, 2012.

示例

Initial input, weight, and bias values.

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	−0.3	0.4	0.1	−0.5	0.2	−0.3	−0.2	−0.4	0.2	0.1

The net input and output calculations.

Unit j	Net input, I_j	Output, O_j
4	$0.2+0-0.5-0.4=-0.7$	$1/(1+e^{0.7})=0.332$
5	$-0.3+0+0.2+0.2=0.1$	$1/(1+e^{-0.1})=0.525$
6	$(-0.3)(0.332)-(0.2)(0.525)+0.1=-0.105$	$1/(1+e^{0.105})=0.474$

示例

Calculation of the error at each node.

Unit j	Err_j
6	$(0.474)(1-0.474)(1-0.474) = 0.1311$
5	$(0.525)(1-0.525)(0.1311)(-0.2) = -0.0065$
4	$(0.332)(1-0.332)(0.1311)(-0.3) = -0.0087$

Calculations for weight and bias updating.

Weight or bias	*New value*
w_{46}	$-0.3+(0.9)(0.1311)(0.332) = -0.261$
w_{56}	$-0.2+(0.9)(0.1311)(0.525) = -0.138$
w_{14}	$0.2+(0.9)(-0.0087)(1) = 0.192$
w_{15}	$-0.3+(0.9)(-0.0065)(1) = -0.306$
w_{24}	$0.4+(0.9)(-0.0087)(0) = 0.4$
w_{25}	$0.1+(0.9)(-0.0065)(0) = 0.1$
w_{34}	$-0.5+(0.9)(-0.0087)(1) = -0.508$
w_{35}	$0.2+(0.9)(-0.0065)(1) = 0.194$
θ_6	$0.1+(0.9)(0.1311) = 0.218$
θ_5	$0.2+(0.9)(-0.0065) = 0.194$
θ_4	$-0.4+(0.9)(-0.0087) = -0.408$

多层神经网络

前向传播输入

- 给定隐藏层或输出层的单元 j, 它的净输入 I_j 为:

$$I_j = \sum_i w_{ij} O_i + \theta_j$$

w_{ij} 是从上一层单元 i 到单元 j 的连接权重; O_i 是上一层单元 i 的输出; θ_j 是单元 j 的偏置

- 给定单元 j 的净输入I_j，它的输出 O_j 的计算公式为:

$$O_j = \frac{1}{1+e^{-I_j}}$$

反向传播误差

- 对于输出层中的单元 k，临时变量 Err_k 的取值由下式计算:

$$Err_k = O_k(1-O_k)(T_k - O_k)$$

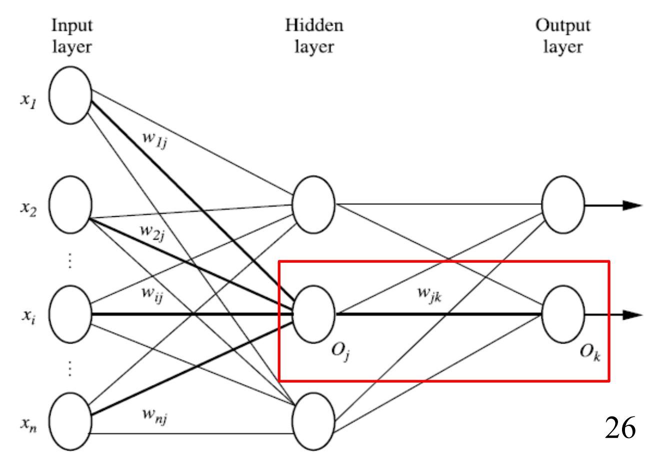
- 隐藏层单元 j 的临时变量取值为:

$$Err_j = O_j(1-O_j)\sum_k Err_k w_{jk}$$

- 权重的更新公式为

$$w_{jk} = w_{jk} + \eta Err_k O_j$$

$$\theta_k = \theta_k + \eta Err_k$$



多层神经网络

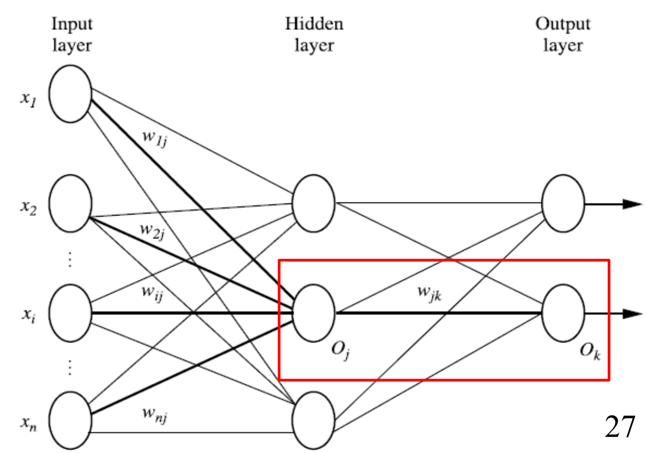
- 最小化节点O_k的误差
- 我们将其定义为 $E=\frac{1}{2}e^2=\frac{1}{2}(T-O)^2$
- 为了调整权重 w_{jk}, 我们首先计算在w_{jk}上误差E的偏导数

$$\frac{\partial E}{\partial w_{jk}}=\frac{\partial E}{\partial e}\times\frac{\partial e}{\partial O_k}\times\frac{\partial O_k}{\partial w_{jk}}$$

$$=-(e)\times\left(O_k(1-O_k)\right)\times\left(O_j\right)$$

$$=-(T_k-O_k)O_k(1-O_k)O_j$$

- 然后我们使用梯度下降法



多层神经网络

- 反向传播学习算法是基于错误曲面的思想
- 表面代表了作为网络权重函数的数据集上的累积误差
- 曲面表示数据集上的累积误差每个可能的网络权重配置由表面上的点表示为网络权重的函数

多层神经网络

- 学习算法的目标是确定一组最小化错误的权重
- 学习算法应该被设计成能够找到表面上最快速减少错误的方向
- 这可以通过在每个表面点处的梯度矢量的相反方向上移动来实现（即，通过采用梯度下降学习方法）

多层神经网络

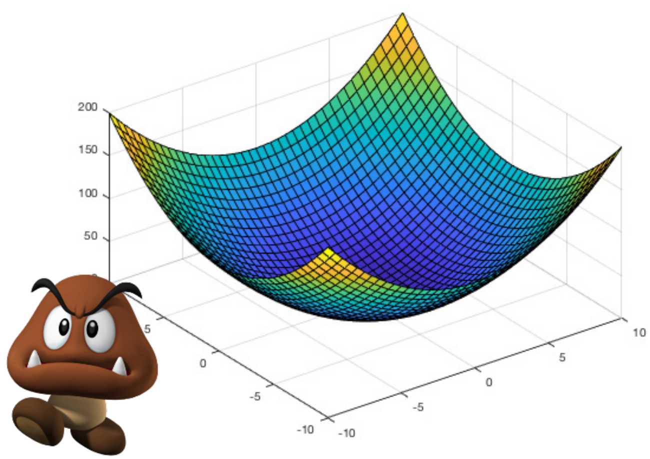
- **优点**
  - 高鲁棒性
  - 非常适合于连续值输入和输出
  - 在广泛的现实数据上取得成功

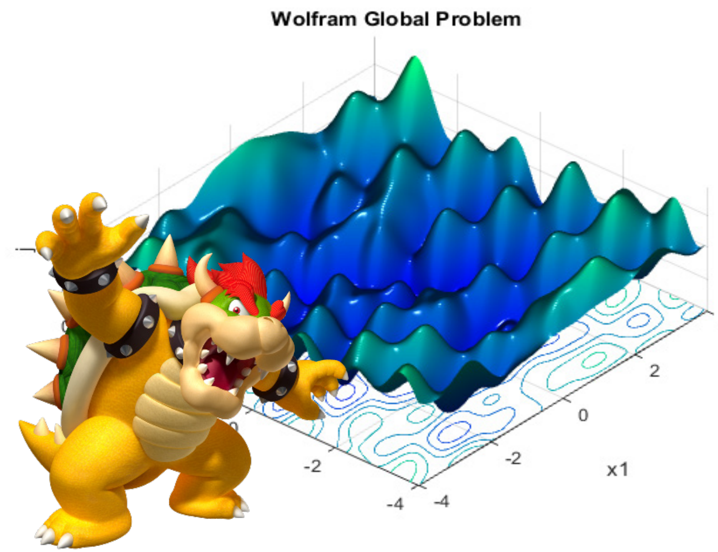
多层神经网络

- **缺点**
  - 训练时间长：尽管使用了BP算法，一次多层神经网络的训练仍然耗时太久
  - 局部最优解和梯度消失问题，这使得神经网络的优化较为困难
  - 通常需要经验最佳确定的多个参数，例如网络拓扑或“结构”
  - 可解释性差：很难解释网络中学习到的权重和“隐藏层神经元”背后的象征意义

非凸模型优化问题

- 非凸模型优化的一个问题就是局部最优解问题，这使得神经网络的优化较为困难。





- 优化效率慢也限制了感知器规模。

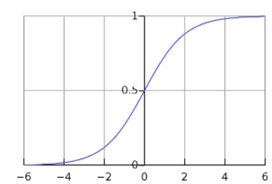
梯度消失问题

- 在深度网络中，不同的层的学习速度差异很大。使用sigmoid作为神经元的输入输出函数，在BP反向传播梯度时，每传递一层梯度衰减为原来的0.25。层数一多，梯度指数衰减后低层基本上接受不到有效的训练信号。

$$\frac{\partial E_{\text{total}}}{\partial w_1} = \boxed{\frac{\partial E_{\text{total}}}{\partial out_{h1}}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\boxed{\frac{\partial E_{\text{total}}}{\partial out_{h1}} = \boxed{\frac{\partial E_{o1}}{\partial out_{h1}}} + \frac{\partial E_{o2}}{\partial out_{h1}}}$$

$$\boxed{\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}}}$$



the sigmoid function as $\sigma(x) = \dfrac{1}{1+e^{-x}}$.

$$0 <= \frac{d}{dx}\sigma(x) = \sigma(x)(1-\sigma(x)). <= 0.25$$

- 梯度消失，使得多层网络等价于浅层网络。

冰河期

- 90 年代中期，由 Vapnik 等人发明的 SVM（Support Vector Machine，支持向量机）算法诞生，很快就在若干个方面体现出了优势：无需调参；高效；全局最优解。基于以上种种理由，SVM迅速打败了神经网络算法成为主流。

Vladimir Vapnik

- 神经网络的研究再次陷入了冰河期。当时，只要你的论文中包含神经网络相关的字眼，很可能被拒收，研究界那时对神经网络的不待见可想而知。

深度学习

- 在被人摒弃的10年中，有几个学者仍然在坚持研究，包括加拿大多伦多大学的Geoffery Hinton教授
- 2006年，Hinton在《Science》和相关期刊上发表了论文，首次提出了“深度信念网络”的概念
- 与传统的训练方式不同，“深度信念网络”有一个“预训练”（pre-training）的过程，这可以方便的让神经网络中的权值找到一个接近最优解的值，之后再使用“微调”(fine-tuning)技术来对整个网络进行优化训练。这两个技术的运用大幅度减少了训练多层神经网络的时间。他给多层神经网络相关的学习方法赋予了一个新名词--“深度学习”（深度学习是多个算法的总称）

深度学习

- 2006年Hinton提出的逐层预训练方法
- pre-training + fine-tunning…
- 使用Rectified Linear Unit (Relu)激活函数
- sigmoid等函数涉及指数运算，计算量大，反向传播求误差梯度时，求导涉及除法，计算量相对大，而采用Relu激活函数，整个过程的计算量节省很多
- sigmoid函数反向传播时，容易出现梯度消失
- Relu会使一部分神经元的输出为0，这样就造成了网络的稀疏性，并且减少了参数的相互依存关系，缓解了过拟合问题的发生
- Dropout：模型训练时随机让网络某些隐含层节点的权重不工作，防止过拟合

深度学习

- 参数越多的模型复杂度越高
  - 优点： “容量”(capacity)越大，能完成更复杂的任务
  - 缺点： 训练效率低，数据量少时易陷入过拟合
- 缺点解决方案
  - 云计算和大数据时代下，计算能力大幅提升，可缓解训练低效性
  - 训练数据大幅增加，降低过拟合风险
- 以深度学习为代表的复杂模型开始受人关注

深度学习

- 很快，深度学习在语音识别领域暂露头角。2012年，Hinton与他的学生在ImageNet竞赛中，用多层的卷积神经网络成功地对包含一千类别的一百万张图片进行了训练，取得了分类错误率15%的好成绩，这个成绩比第二名高了近11个百分点，充分证明了多层神经网络识别效果的优越性。
- 此后，关于深度神经网络的研究与应用不断涌现。

Geoffery Hinton
1986，BP算法

Geoffery Hinton
2018年度图灵奖获得者