神经网络

蔡少伟

中国科学院大学

2016

概述

神经网络(Neural networks)

- 使用“网络”一词是因为他们一般是由不同函数组合起来的。
- 使用“神经”一词是因为这些网络在某种程度上受神经科学的启发。
- 不过，现在的神经网络研究是由许多数学和工程原则指引的，神经网络研究的目标也不是为了模拟大脑。

神经网络几起几落

- 20世纪40年代M-P神经元模型，Hebb学习律出现后，50年代出现了以感知机，Adaline为代表的一系列成果。（第一个高潮期）
- 1969年，MIT计算机科学奠基人Marvin Minsky出版了《感知机》一书，指出单层神经网络无法解决非线性问题，而多层网络的训练算法还看不到希望。导致神经网络的研究进入“冰河期”。
- 1983年，加州理工学院的物理学家John Hopfield利用神经网络，在旅行商问题的求解上取得了当时最好结果，引起轰动。继Hopfield工作之后，Rumelhart等人重新发明了BP算法，由于正处在Hopfield带来的兴奋之中，BP算法迅速走红。（第二次高潮期）
- 20世纪90年代中期，随着统计学习理论和支持向量机的兴起，神经网络学习的理论不够清楚，在使用中充斥大量的人为经验的弱点更加明显，于是神经网络又进入低谷。
- 2010年前后，随着计算能力的迅速提升和大数据的涌现，神经网络在“深度学习”的名义下重新崛起。（第三次高潮期）

前馈神经网络概述

本课主要介绍前馈神经网络(feedforward neural networks)

- Feedforward neural networks, also called multilayer perceptrons, are the quintessential deep learning models. (from ”Deep Learning”)
- 目标是近似一个函数f^*,可以看成是一系列函数的嵌套，如$f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$
- 深度指的就是嵌套的深度，“深度学习”这个概念也是从这个意义上讲。

主要参考书

- ”Deep Learning”, Ian Goodfellow, Yoshua Bengio and Aaron Courville, MIT Press. (online version: http://www.deeplearningbook.org/)
- 《机器学习》by 周志华，清华大学出版社

神经元模型

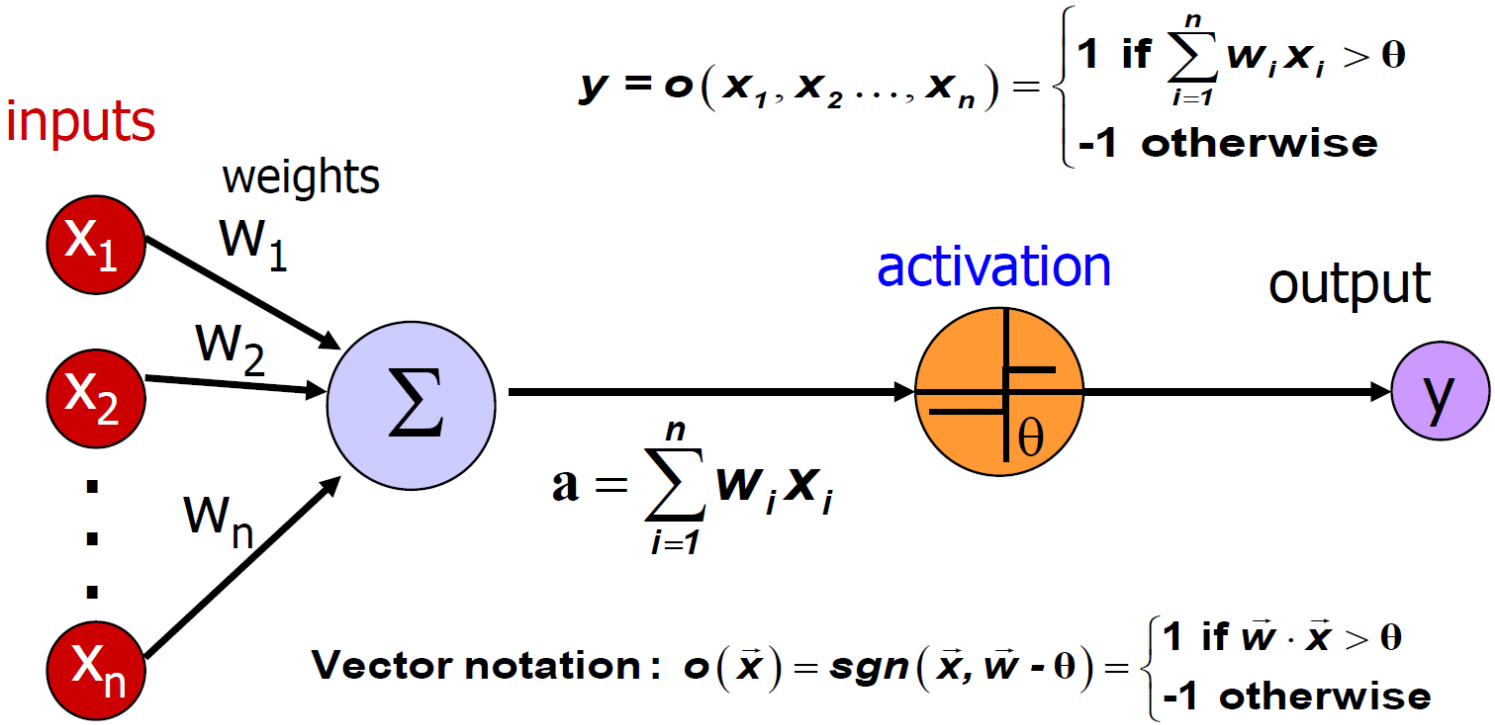


Figure: 神经元模型（阈值逻辑单元）

神经元模型

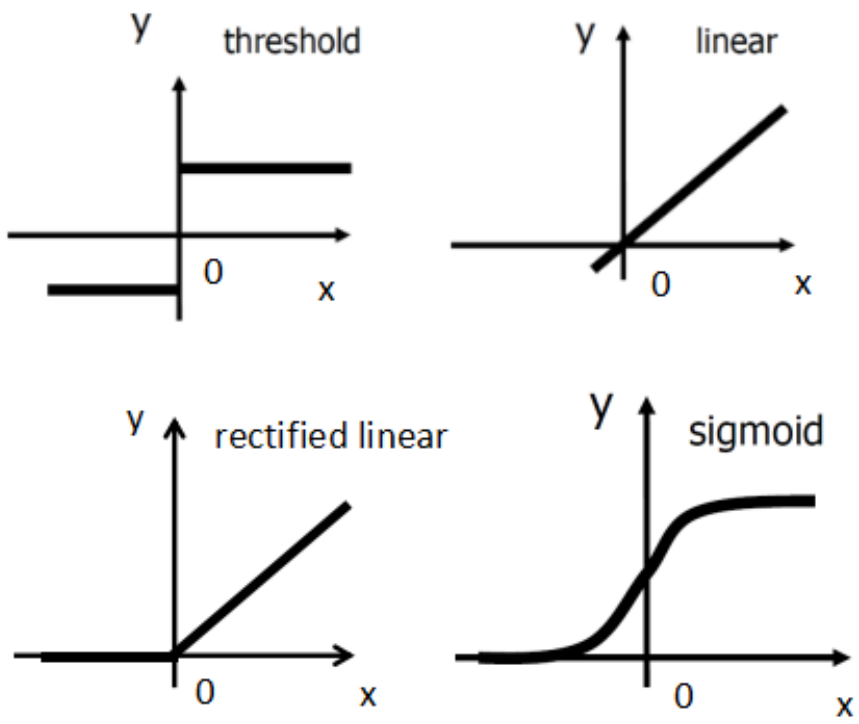


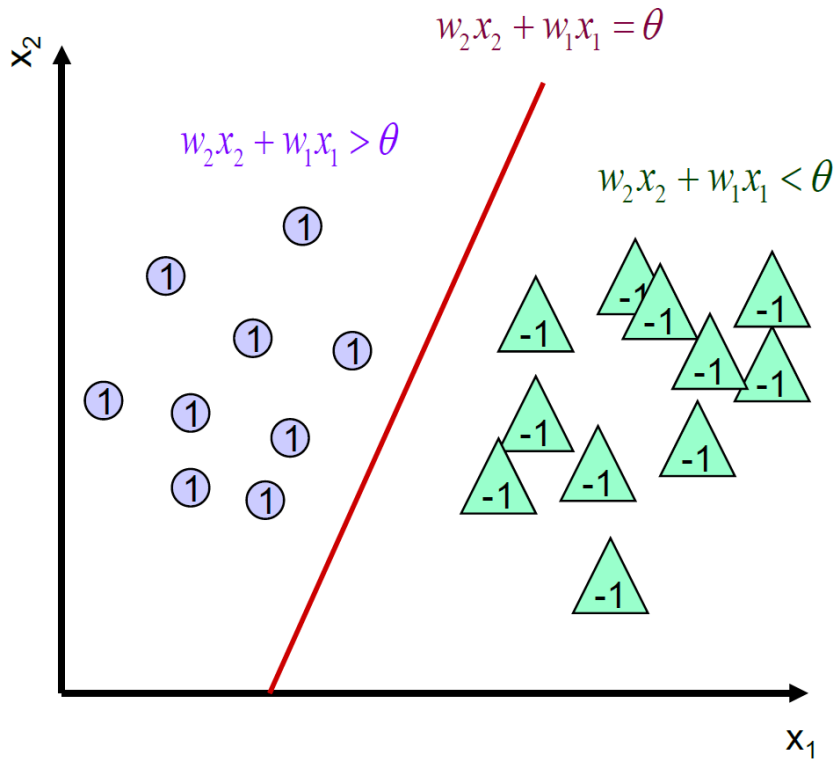
Figure: 激活函数

$$sgn(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}, \; linear(x) = x$$

$$rtflinear(x) = max\{0, x\}, \; sigmoid(x) = \frac{1}{1+e^{-x}},$$

感知机

感知机是最早的监督式训练算法，是神经网络构建的基础。
感知机其实就是一个神经元，输入信号传递给阈值逻辑单元产生输出。如果把输入层也看为一层神经元，感知机有两层神经元。



- **Line** equation:

$$w_2x_2 + w_1x_1 = \theta$$

- Classifier:
- If $w_2x_2 + w_1x_1 \geq \theta$
  - Output 1
- Else
  - Output -1

Figure: 感知机

感知机的训练

将阈值θ看作一个固定输入，这样权重和阈值的学习就可以统一为权重的学习。

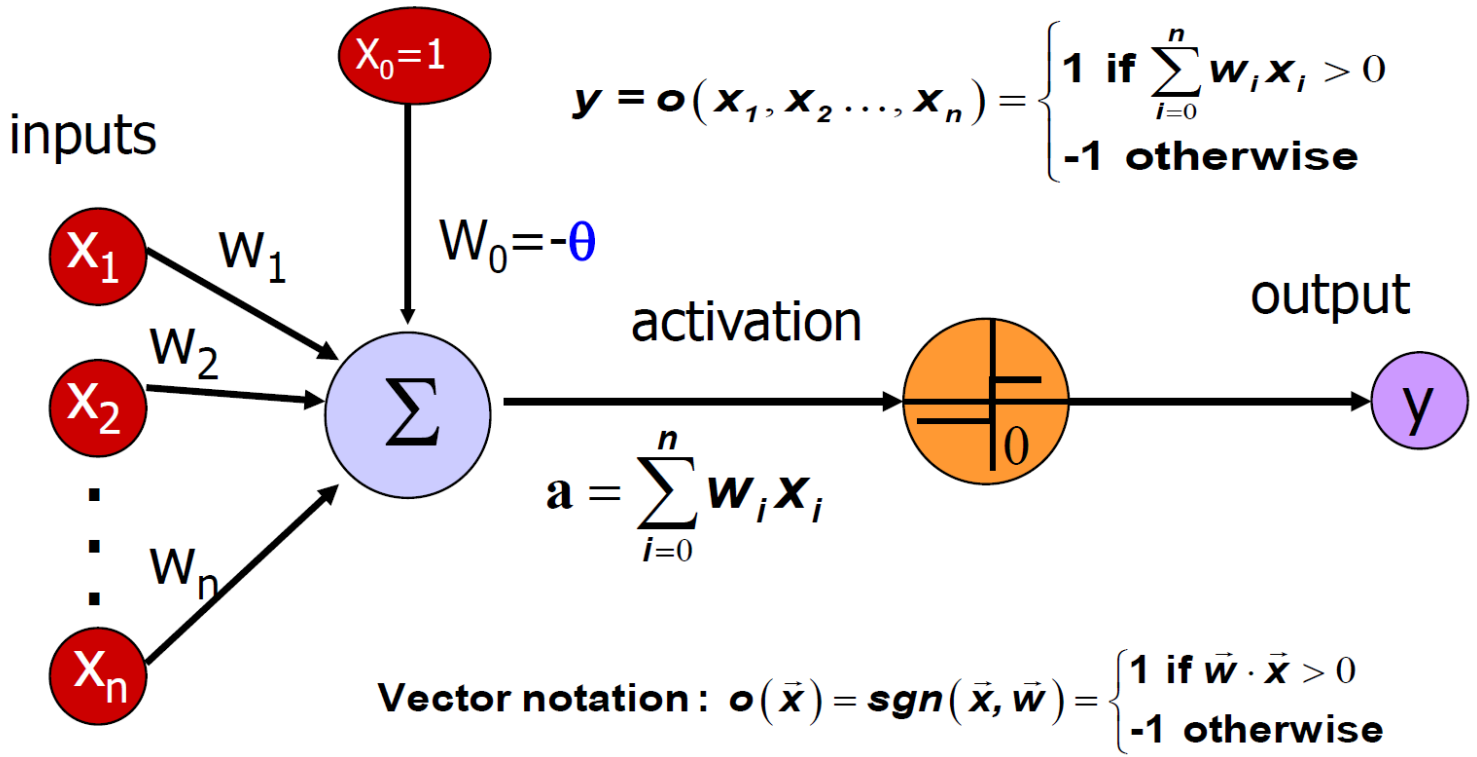


Figure: 感知机的训练

感知机的训练

感知机的学习规则非常简单，对训练样例$(\mathbf{x}, y)$，如果当前感知机的输出为$\hat{y}$,则感知机权重这样调整：

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

$$\Delta w_i = \eta(y - \hat{y})x_i$$

其中$\eta \in (0, 1)$称为学习率。可以看出，如果预测正确，则感知机不发生变化，否则根据错误的方向和错误的程度进行权重的调节。

感知机的缺陷

感知机只有一层功能神经元，学习能力非常有限。
可以证明，若两类模式是线性可分的，则感知机的学习过程一定收敛；否则感知机学习过程会发生震荡而不能求得合适解。[Minsky and Papert, 1969]
也就是说，感知机只能学习线性可分函数。

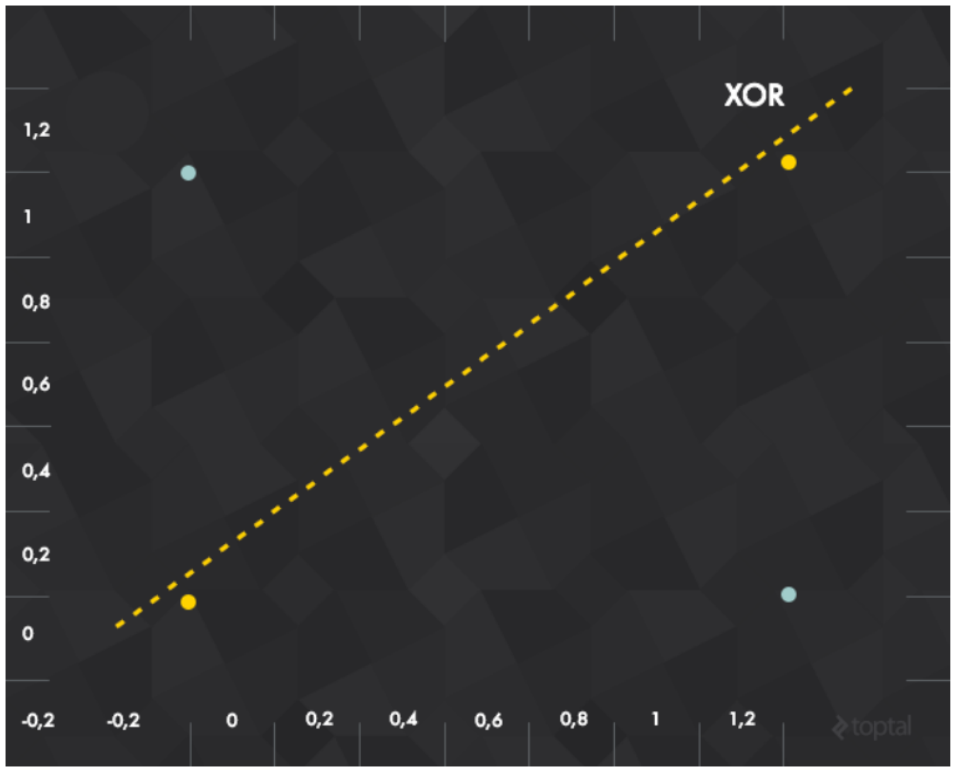


Figure: 感知机不能学习XOR函数

前馈神经网络

要解决非线性可分问题，需要考虑使用多层神经元。

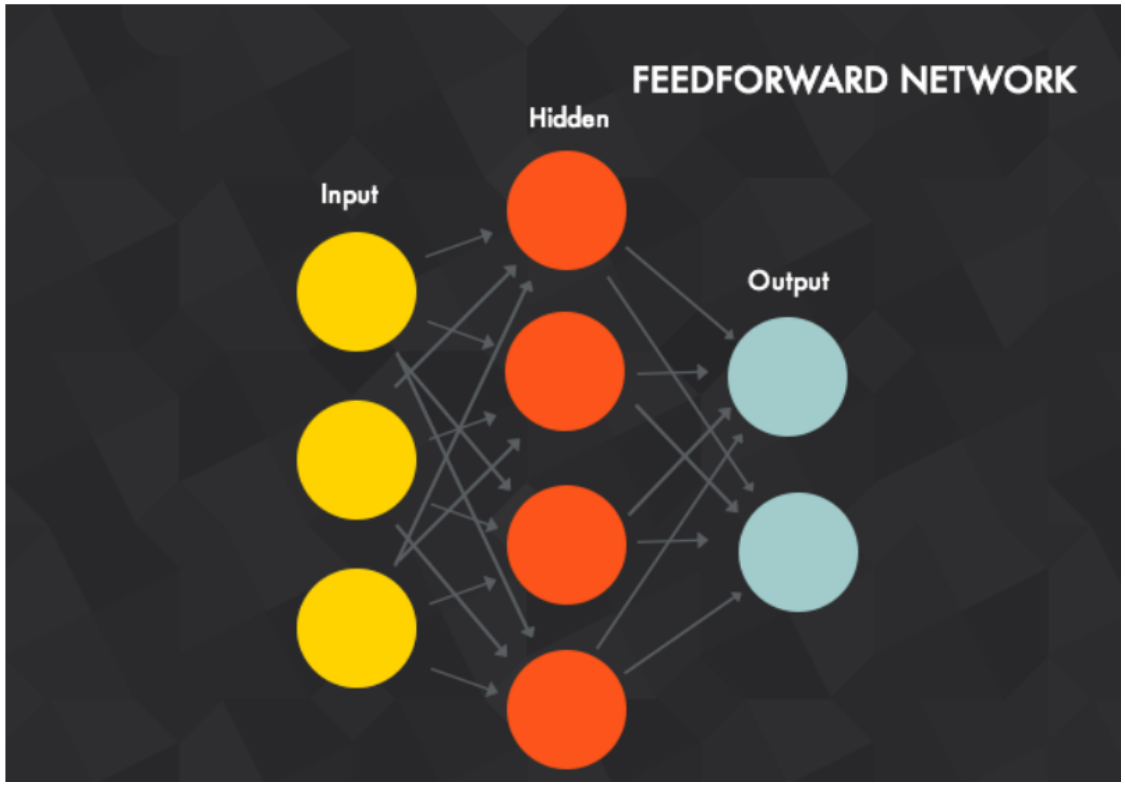


Figure: 前馈神经网络（多层感知机）

前馈神经网络

前馈神经网络，又称为多层感知机，具有以下属性：

- 一个输入层，一个输出层，一个或多个隐含层。
- 每一个神经元都是一个感知机。
- 输入层的神经元作为隐含层的输入，同时隐含层的神经元也是输出层神经元的输入。
- 每条神经元之间的连接都有一个权重w (与感知机中提到的权重类似)。
- 在t 层的每个神经元通常与前一层(t - 1层)中的每个神经元都有连接（但可以通过将这条连接的权重设为0来断开这条连接）。神经元之间不存在同层连接和跨层连接。

为了处理输入数据，将输入向量赋到输入层中,这些值将被传播到隐含层，通过加权传递函数传给每一个隐含层神经元（这就是前向传播），隐含层神经元再计算输出（激活函数）。输出层和隐含层一样进行计算，输出层的计算结果就是整个神经网络的输出。

超线性

关于前馈神经网络的超线性：
如果每一个感知机都使用一个线性激活函数会怎么样？

Remark 1
线性函数的组合还是线性函数。

如果只使用线性激活函数，则

- 整个网络的最终输出是将输入数据通过一些线性函数计算过一遍，结果仍然是输入的线性函数。
- 前馈神经网络仍然只能学习线性可分函数。

大多数神经网络都会使用非线性激活函数，如对数函数、双曲正切函数、阶跃函数、修正线性函数等。

BP算法（概述）

多层感知机的学习能力比单层感知机强很多。要训练多层感知机，简单感知机学习规则显然不够，需要更强大的学习算法。

反向传播(BackPropagation,简称BP)算法就是最杰出的代表，是迄今最成功的神经网络学习算法。

用BP算法训练的多层前馈神经网络也称为BP网络。

BP算法概述

BP算法是一个迭代算法，在每一轮迭代中使用一个样本进行训练：

1. 将训练样本通过神经网络进行前向传播计算。
2. 计算输出误差，常用均方差。
3. 基于梯度下降策略，以目标的负梯度方向对参数进行调整。

BP算法（记号）

训练集$D = \{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), ..., (\mathbf{x}_m, \mathbf{y}_m)\}$, $\mathbf{x}_i \in \mathbb{R}^d$, $\mathbf{y}_i \in \mathbb{R}^l$. 即，输入层有$d$个神经元，输出成有$l$个神经元。
假设有一层隐含层，包含的神经元个数为q，隐层和输出层神经元都使用Sigmoid函数。

- θ_j: 表示第j个输出神经元的阈值(模型参数)；
- γ_h: 表示隐层第h个神经元的阈值(模型参数)；
- b_h: 表示隐层第h个神经元的输出；
- v_{ih}: 表示输入层第i个神经元与隐层第h个神经元的连接权(模型参数)；
- w_{hj}: 表示隐层第h个神经元与输出层第j个神经元的连接权(模型参数)。

则

- 隐含层第h个神经元收到的输入为: $\alpha_h = \sum_{i=1}^{d} v_{ih} x_i$;
- 输出层第j个神经元收到的输入为: $\beta_j = \sum_{h=1}^{q} w_{hj} b_h$.

BP算法（规则）

对训练样例$(\mathbf{x}_k, \mathbf{y}_k)$，记神经网络的输出为$\hat{y}_k = (\hat{y}_1^k, \hat{y}_2^k, ..., \hat{y}_l^k)$，也即

$$\hat{y}_j^k = f(\beta_j - \theta_j)$$

则网络在样例$(\mathbf{x}_k, \mathbf{y}_k)$上的均方差为（下面式子中的$\frac{1}{2}$只是为了后续求导方便）

$$E_k = \frac{1}{2}\sum_{j=1}^{l}(\hat{y}_j^k - y_j^k)^2$$

网络中需要确定的参数有：输入层到隐层的$d \times q$个权值，隐层到输出层的$q \times l$个权值，以及隐层q个神经元的阈值，输出层l个神经元的阈值。

在每一轮迭代中，采用广义的感知机学习规则对参数进行更新。
即，任何参数的更新式为

$$v \leftarrow v + \Delta v$$

BP算法基于梯度下降策略，以目标的负梯度方向对参数进行调整。

梯度下降的优化原理

- 梯度的几何意义：函数在某点的梯度是这样一个向量，沿着梯度的方向函数的值增长最快。所以，沿着负梯度方向使得函数的值减小最快。
- 梯度下降法：一种常用的一阶优化方法（即只使用目标函数的一阶导数而不用高阶导数），是求解无约束优化问题最简单最经典的方法之一。

理解梯度下降法：
考虑无约束最小化问题$min_{\mathbf{x}} f(\mathbf{x})$，其中$f(\mathbf{x})$是连续可微函数。
如果我们能构造一个序列$\mathbf{x}^0, \mathbf{x}^1, \mathbf{x}^2, ...$,满足

$$f(\mathbf{x}^{t+1}) < f(\mathbf{x}^t), t = 0, 1, 2, ...$$

则不断执行该过程可以收敛到局部极小值。

梯度下降的优化原理

多元函数$f(\mathbf{x})$在某点$\mathbf{x}^t$的泰勒展开式(设$\mathbf{x}$为n维向量)：

$$f(\mathbf{x}) = f(\mathbf{x}^t) + \sum_{i=1}^{n} \frac{\partial f}{\partial x_i}(x_i - x_i^t) + o$$

所以

$$f(\mathbf{x}^t + \Delta\mathbf{x}) = f(\mathbf{x}^t) + \sum_{i=1}^{n} \frac{\partial f}{\partial x_i}\Delta x_i + o = f(\mathbf{x}^t) + \Delta\mathbf{x}^T\Delta f(\mathbf{x}) + o$$

要使得

$$f(\mathbf{x}^t + \Delta\mathbf{x}) < f(\mathbf{x}^t)$$

可以令

$$\Delta\mathbf{x} = -\gamma\Delta f(\mathbf{x})$$

即

$$\Delta\mathbf{x}_i = -\gamma\frac{\partial f}{\partial x_i}, i = 1, 2, ..., n$$

实际上，这也是使得函数$f(\mathbf{x})$在点$\mathbf{x}^t$下降最快的方向。这就是梯度下降法。

BP算法(更新参数)

对于样例$(\mathbf{x}_k, \mathbf{y}_k)$，BP算法要使得神经网络在该样例上的均方差最小化。

$$E_k = \frac{1}{2}\sum_{j=1}^{l}(\hat{y}_j^k - y_j^k)^2$$

影响该函数的变量，也即模型参数有

- θ_j: 表示第j个输出神经元的阈值；
- γ_h: 表示隐层第h个神经元的阈值；
- v_{ih}: 表示输入层第i个神经元与隐层第h个神经元的连接权；
- w_{hj}: 表示隐层第h个神经元与输出层第j个神经元的连接权。

我们要更新这些参数，使得E_k最小化。以w_{hj}的更新为例。
根据梯度下降法，有

$$\Delta w_{hj} = -\eta\frac{\partial E_k}{\partial w_{hj}}$$

注意到w_{hj}先影响第j个输出层神经元的输入值β_j，进而影响到其输出值$\hat{y}_j^k$，所以影响到E_k。所以，

$$\frac{\partial E_k}{\partial w_{hj}} = \frac{\partial E_k}{\partial \hat{y}_j^k}\cdot\frac{\partial \hat{y}_j^k}{\partial \beta_j}\cdot\frac{\partial \beta_j}{\partial w_{hj}}$$

BP算法(更新参数)

根据β_j的定义，有

$$\frac{\partial \beta_j}{\partial w_{hj}} = b_h$$

于是，

$$\Delta w_{hj} = -\eta \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot b_h$$

记$g_j = -\frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j}$, 其中$\frac{\partial E_k}{\partial \hat{y}_j^k} = (\hat{y}_j^k - y_j^k)$.

Sigmoid函数的性质：$f'(x) = f(x)(1 - f(x))$，所以

$$\begin{aligned} g_j &= -\frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \\ &= -(\hat{y}_j^k - y_j^k) f'(\beta_j - \theta_j) \\ &= \hat{y}_j^k (1 - \hat{y}_j^k)(y_j^k - \hat{y}_j^k) \end{aligned}$$

则

$$\Delta w_{hj} = \eta g_j b_h$$

BP算法(更新参数)

类似地，可以计算得到

$$\begin{aligned} \Delta\theta_j &= -\eta g_j \\ \Delta v_{ih} &= \eta e_h x_i \\ \Delta\gamma_h &= -\eta e_h \end{aligned}$$

其中，

$$\begin{aligned} e_h &= -\frac{\partial E_k}{\partial b_h} \cdot \frac{\partial b_h}{\partial \alpha_h} \\ &= -\sum_{j=1}^{l} \frac{\partial E_k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial b_h} f'(\alpha_h - \gamma_h) \\ &= \sum_{j=1}^{l} w_{hj} g_j f'(\alpha_h - \gamma_h) \\ &= b_h(1 - b_h) \sum_{j=1}^{l} w_{hj} g_j \end{aligned}$$

学习率$\eta \in (0, 1)$控制着算法每一轮迭代的更新步长，如果太大容易震荡，太小则收敛速度会过慢。（常设$\eta = 0.1$，可细调）

BP算法(描述)

Input: 训练集$D = \{(\mathbf{x}_k, \mathbf{y}_k)\}_{k=1}^{m}$
Input: 学习率η
Output: 连接权与阈值
在$(0,1)$范围内随机初始化网络中所有连接权和阈值;
while 不达到停止条件 **do**
    for $(\mathbf{x}_k, \mathbf{y}_k) \in D$ **do**
        在当前参数下的模型中计算当前样本的输出$\hat{\mathbf{y}}_k$;
        计算输出层神经元的梯度项g_j;
        计算隐含层神经元的梯度项e_h;
        更新连接权w_{hj}, v_{ih}和阈值θ_j, γ_h;

标准BP算法和累积BP算法

- 我们上面介绍的是标准BP算法，每次仅针对一个训练样例更新连接权和阈值，也即参数的更新是基于单个样例的误差最小化原则。
- 还可以基于所有样例的累计误差最小化规则来更新参数，就得到了累积BP算法。

标准BP和累积BP算法之比较

- 标准BP算法每次更新只针对单个样例，参数更新得非常频繁，往往需要更多的迭代次数；累积BP算法在读取训练集D一遍之后才对参数进行更新，更新频率低很多。
- 标准BP算法每次更新快，累积BP算法每次更新慢。在很多任务中，当累积误差下降到一定程度后，进一步下降会非常缓慢，而标准BP能更快地获得较好的解。这种情况在D非常大时更明显。
- 标准BP算法和累积BP算法的区别，类似于随机梯度下降和（最速）梯度下降的区别。

局部极小和全局最小

BP算法的过程是一个参数寻优过程，也即在参数空间中，寻找一组参数使得误差最小。这其实是一个在搜索空间中找到目标函数最小化的点。

- 梯度下降是朝着函数值下降最快的方向搜索。
- 若误差函数在当前梯度为0，则已经达到局部极小，这时参数的迭代更新就此停止。
- 这有一个问题：如果误差函数只有一个局部极小，那么找到的局部极小就是全局最小；如果误差函数有多个局部极小，则不能保证找到的解是全局最小，这种情形我们称搜索过程陷入了局部极小。
- 人们常采用一些策略来跳出局部极小【各种局部搜索技术】

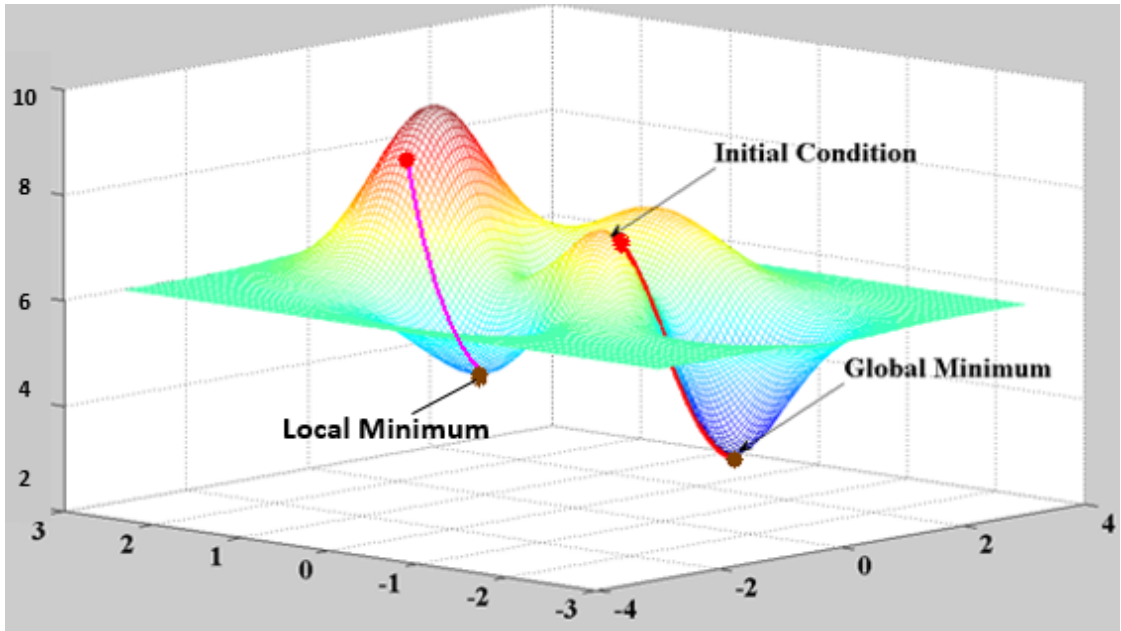


Figure: 局部极小和全局最小

深层神经网络

深度学习的背后原因

摘自《机器学习》by周志华

- 理论上，参数越多模型越复杂，则模型的复杂度越高，容量越高。
- 但是复杂模型的训练效率低，且容易陷入过拟合，因此难以收到人们青睐。
- 随着云计算和大数据时代的到来，计算能力的提升可以缓解训练低效性，而训练数据的大幅增加可以降低过拟合风险，因此以“深度学习”为代表的复杂模型开始受到人们的关注。
- 典型的深度学习模型就是很深层的神经网络，隐层多了，参数也更多，模型的学习能力也更强。

From ”Deep Learning” book

- The core ideas behind modern feedforward networks have not changed substantially since the 1980s. The same back-propagation algorithm and the same approaches to gradient descent are still in use.
- Most of the improvement in neural network performance from 1986 to 2015 can be attributed to two factors.
  - First, larger datasets have reduced the degree to which statistical generalization is a challenge for neural networks.
  - Second, neural networks have become much larger, due to more powerful computers, and better software infrastructure.

算法上的改进

- 深度学习，主要包括深层网络，他们的高性能主要归功于更强的计算能力和更大的训练集合。
- 不过，也有一些算法上的改进。

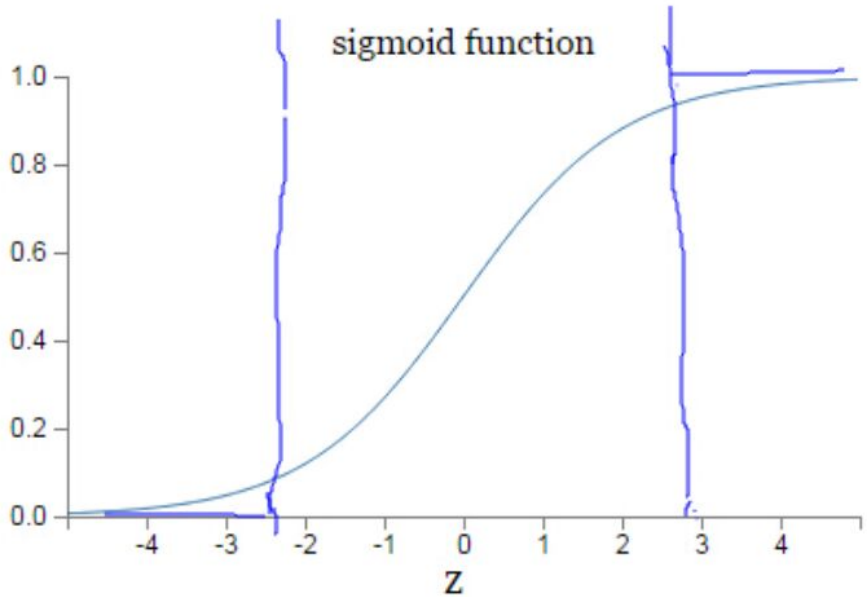
算法改进From "Deep Learning" book

- 当用sigmoid函数作为激活函数时，使用交叉熵代价函数来作为回归的代价函数（替代方差代价函数）；
- 用分段线性函数，比如修正线性函数，代替sigmoid函数作隐层神经元的激活函数；

从方差代价函数说起

对于样例$(\mathbf{x}, \mathbf{y})$（方便讨论，这里不记为$(\mathbf{x}_k, \mathbf{y}_k)$），记神经网络的输出为$\hat{y}$。

- 代价函数经常用方差代价函数$E_k = \frac{1}{2}\sum_{j=1}^{l}(\hat{y}_j - y_j)^2$。
- 在训练神经网络过程中，我们通过梯度下降算法来更新参数，因此需要计算代价函数对各个参数的导数。
- 使用方差代价函数的BP算法，以w_{hj}的更新：$\Delta w_{hj} = \eta g_j b_h$，其中$g_j = -\frac{\partial E}{\partial \hat{y}_j} \cdot \frac{\partial \hat{y}_j}{\partial \beta_j}$。而$\frac{\partial \hat{y}_j}{\partial \beta_j}$就是sigmoid函数对该神经元输入的导数。
- 因为sigmoid函数的性质，导致其导数在函数变量的大部分取值下都很小（如下图标出来的两端，几近于平坦），这样会使得参数的更新非常慢，比如Δw_{hj}接近于0。



交叉熵代价函数

为了克服这个缺点，引入了交叉熵代价函数：

$$C = -\frac{1}{\ell}\sum_{j=1}^{\ell}[y_j \ln \hat{y}_j + (1-y_j)\ln(1-\hat{y}_j)]$$

与方差代价函数一样，交叉熵代价函数同样有两个性质：

- 非负性（我们的目标就是最小化代价函数）；
- 当真实输出$\hat{y}$与期望输出y接近的时候，代价函数接近于0.

另外，它可以克服方差代价函数更新权重过慢的问题。

- 根据交叉熵代价函数对参数的求导，权重的更新是受$(\hat{y}-y)$这一项影响，即受误差的影响。
- 当误差大的时候，权重更新就快，当误差小的时候，权重的更新就慢。这是一个很好的性质。

Note:在上面的讨论中，我们假定输出层是通过sigmoid函数，因此采用了交叉熵代价函数。而深度学习中更普遍的做法是将softmax作为最后一层，此时常用的是代价函数是log-likelihood（对数似然）函数。

修正线性激活函数

另一个显著提高前馈神经网络性能的重要算法改进，是使用修正线性函数来代替sigmoid函数做激活函数。

关于修正线性函数

- 修正函数在早期是比较常见的，但是80年代开始被sigmoid函数取代，主要是由于当时的实验表明sigmoid函数（在规模较小的神经网络上）性能更好。
- 到2000年左右，一般认为应该避免使用修正线性函数这类存在不可微分点的函数
- 直到2009年，Jarrett等人（ICCV09）发现"Using a rectifying nonlinearity is the single most important factor in improving the performance of a recognition system" among several factors of neural network architecture design.
- 2011年，Glorot等人（AISTATS'2011）从生物学的角度阐释了修正线性函数的motivations。所以，修正线性函数的重新兴起也说明了神经科学仍然对深度学习产生影响。

其他神经网络

我们只介绍了前馈神经网络（FNN）。神经网络模型很多，常见的下面这些神经网络

- RBF(Radial Basis Function, 径向基函数)网络是一种前馈神经网络，但使用径向基函数做激活函数，是某种沿径向对称的标量函数，一般定义为样本到数据中心的欧式距离的单调函数。
- ART(Adaptive Resonance Theory，自适应谐振理论)网络，是竞争型学习的代表，由比较层，识别层，识别阈值和重置模块构成。比较层负责接收输入样本并将其传递给识别层神经元，识别层每个神经元对应一个模式类。识别层神经元之间相互竞争以产生一个获胜的神经元。
- SOM(Self-Organizing Map, 自组织映射)网络，是一种竞争型学习的无监督神经网络，能将高维输入数据映射到低维空间。
- Cascade-Correaltion（级联相关）网络：一般神经网络模型假定网络结构是事先固定的，训练目的是确定合适的连接权和阈值等参数。级联相关网络是结构自适应网络的代表，训练过程会改变网络结构。
- Recurrent neural networks（递归神经网络）：允许网络出现环形结构，从而可以让一些神经元的输出反馈回来作为输入信号。Elman网络是最常用的递归神经网络之一。
- Boltzmann机：一种基于能量的模型，网络状态定义为能量，能量最小化时网络达到理想状态。

Why FNN

From ”Deep Learning” book

- When the modern resurgence of deep learning began in 2006, feedforward networks continued to have a bad reputation. From about 2006-2012, it was widely believed that feedforward networks would not perform well unless they were assisted by other models, such as probabilistic models.
- Today, it is now known that with the right resources and engineering practices, feedforward networks perform very well. Today, gradient-based learning in feedforward networks is used as a tool to develop probabilistic models.
- Rather than being viewed as an unreliable technology that must be supported by other techniques, gradient-based learning in feedforward networks has been viewed since 2012 as a powerful technology that may be applied to many other machine learning tasks.
- Feedforward networks continue to have unfulfilled potential. In the future, we expect they will be applied to many more tasks, and that advances in optimization algorithms and model design will improve their performance even further.