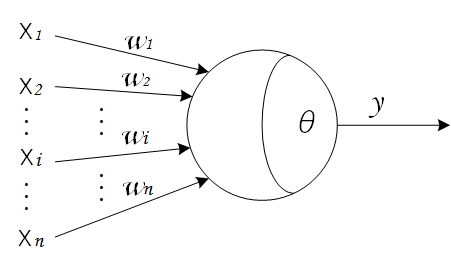
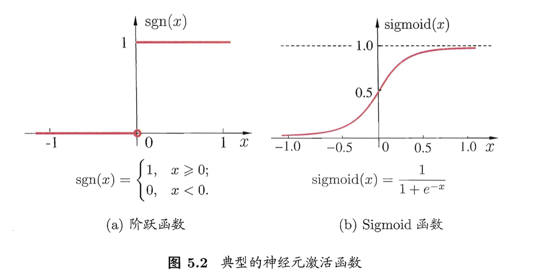
**神经网络通常不是一个凸优化的问题，他处处充满了局部最优解。**



**神经元接收来自 n 个其他神经元传递过来的输入信号 x，这些输入信号通过带权重 w 的连接(connection)进行传递，神经元接收到的总输入值 ∑ni=1wixi 将与神经元的阈值 θ 进行比较，然后通过“激活函数”(activation function) f 处理产生神经元的输出 y=f(∑ni=1wixi−θ)。**

**感知机由两层神经网络组成，输入层接收外界输入信号后传递给输出层，输出层是 M-P 神经元，亦称“阈值逻辑单元”(threshold logic unit)。**

**需要注意，感知机只有输出层神经元进行激活函数处理，即只拥有一层功能神经元(functional neuron)，其学习能力非常有限。如果遇到非线性可分问题，感知机就无法表示了，例如异或这样简单的非线性可分问题。如果要解决非线性可分问题，需要使用多层功能神经元，例如使用两层的感知机就能解决异或问题：**



神经科学家进行大脑能量消耗的研究中发现，神经元编码的工作方式具有稀疏性，推测大脑同时被激活的神经元只有1%-4%，神经元只会对输入信号有少部分的选择性响应，大量不相关信号被屏蔽，这样可以更高效的提取重要特征。

**理想的激活函数如图a所示的进阶函数，将输入映射为0和1，分别代表抑制和兴奋，然而进阶函数具有不连续、不光滑等不太好的特性，因此常用其他函数代替。**传统的sigmoid函数有近半神经元被激活，不符合神经科学的研究。softplus虽有单侧抑制，却没有稀疏激活性。因而ReLU成了最符合实际神经元的模型。

输入层一般用ReLU，输出层一般使用sigmoid,因为接近概率输出

dropout是指在深度学习网络的训练过程中，对于神经网络单元，按照一定的概率将其暂时从网络中丢弃。注意是暂时，对于随机梯度下降来说，由于是随机丢弃，故而每一个mini-batch都在训练不同的网络。

SGD本身也不是一个比较稳定的算法，结果可能在最优解附近波动，不同的学习速率可能导致神经网络落入截然不同的局部最优置空。

Adagrad,Adam,Adadelta等自适应方法可以减轻调参负担，是一种自适应的学习率，如前期学习率大，后期学习率小。

信号在超过某个阈值时，神经元才会进入兴奋和激活状态，平时则处于抑制状态。ReLu = max(0,x)可以很好的传递梯度，经过多层的反向传播，梯度依旧不会大幅缩小，非常适合训练深层神经网络，从正面解决了梯度弥散问题。

sigmoid函数在反向传播中梯度值会逐渐减小，经过多层传递后呈指数级急剧减小。

ReLU:单侧抑制，相对宽阔的兴奋边界，稀疏激活性。

softplus:单侧抑制，无稀疏激活性 log(1+ex)

梯度弥散，梯度越来越小，甚至消失,信息消失

梯度爆炸，梯度越来越大，信息发散

如果不用激励函数（其实相当于激励函数是f(x) = x），在这种情况下你每一层输出都是上层输入的线性函数，很容易验证，无论你神经网络有多少层，输出都是输入的线性组合，与没有隐藏层效果相当，这种情况就是最原始的感知机（Perceptron）了。 正因为上面的原因，我们决定引入非线性函数作为激励函数，这样深层神经网络就有意义了（不再是输入的线性组合，可以逼近任意函数）。最早的想法是sigmoid函数或者tanh函数，输出有界，很容易充当下一层输入（以及一些人的生物解释balabala）。

从数学上来看，Sigmoid函数对中央区的信号增益较大，对两侧区的信号增益小，在信号的特征空间映射上，有很好的效果。

　　从神经科学上来看，中央区酷似神经元的兴奋态，两侧区酷似神经元的抑制态，因而在神经网络学习方面，可以将重点特征推向中央区，

TanHyperbolic(tanh)函数又称作双曲正切函数，数学表达式为y=(ex − e−x)/(ex + e− x),其函数曲线与Sigmoid函数相似

第一，采用sigmoid等函数，算激活函数时（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。

第二，对于深层网络，sigmoid函数反向传播时，很容易就会出现梯度消失的情况（在sigmoid接近饱和区时，变换太缓慢，导数趋于0，

这种情况会造成信息丢失），从而无法完成深层网络的训练。

第三，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（以

及一些人的生物解释balabala）。

以前的神经网络都是采取全连接的方式，认为下一层的输出与上一层所有输入都有关，但是这样很容易导致overfitting。

卷积神经网络出自感受野，科学家通过对猫的视觉皮层细胞研究发现，每一个视觉神经元只会处理一小块区域的视觉图像，即感受野，神经认知机可算卷积网络最初的实现原型。

神经认知机包含两类神经元，用于提取特征的S-cells对应于卷积神经网络的卷积核滤波操作，用来抗形变的C-cells对应激活函数、最大池化等操作。

Cnn理论上具有对图像缩放、平移和旋转的不变性。

一般卷积神经网络由多个卷积层构成，每个卷积层中通常会进行如下几个操作：

1. 图像通过多个不同的卷积核的滤波，并加偏置，提取出局部特征，每一个卷积核会映射出一个新的2D图像
2. 将前面卷积核的滤波输出结果，进行非线性的激活函数处理，如ReLU。
3. 对激活函数的结果再进行池化操作，即降维采样，目前一般采用最大池化，保留最显著特征，提升模型的畸变容忍能力。

全连接的核心操作就是矩阵向量乘积 Y = Wx

本质就是由一个特征空间线性变换到另一个特征空间。目标空间的任一维——也就是隐层的一个 cell——都认为会受到源空间的每一维的影响。不考虑严谨，可以说，目标向量是源向量的加权和。

在 CNN 中，全连接常出现在最后几层，用于对前面设计的特征做加权和。比如 mnist，前面的卷积和池化相当于做特征工程，后面的全连接相当于做特征加权，即分类器的作用。（卷积相当于全连接的有意弱化，按照局部视野的启发，把局部之外的弱影响直接抹为零影响；还做了一点强制，不同的局部所使用的参数居然一致。弱化使参数变少，节省计算量，又专攻局部不贪多求全；强制进一步减少参数。少即是多）

卷积神经网络（Convolutional Neural Network,CNN）,可用来解决图像识别，视频处理，时间序列信号处理，如音频信号、文本数据等。

图像特征不易人工提取，深度学习出现之前是借助SIFT，HoG等算法提取具有良好区分性的特征，再结合SVM等机器学习算法进行图片识别。卷积神经网络不需要将特征提取和分类训练两个过程分开，它在训练时就自动提取了最有效的特征。

CNN的最大特点在于卷积的权值共享结构，可大幅减少神经网络的参数量。卷积神经网络的要点就是局部连接（一个神经元连接一块儿区域的像素点，即卷积核的尺寸）、权值共享和池化层中的降采样（比如2 X 2图片降为1 X 1，目前一般采用最大池化，保留最显著特征，提升模型的畸变容忍能力）。局部连接和权值共享降低了参数量，使训练复杂度大大下降，减轻了过拟合。同时权值共享还赋予了卷积网络对平移的容忍性，而池化层降采样进一步降低了输出参数量，并赋予模型对轻度形变的容忍性，提高了模型的泛化能力。

全连接网络因参数过多及梯度弥散问题，早期很难顺利进行多层训练。

普通的神经网络仅能对向量进行操作，常见的图像、音频等高维数据都需要展开成向量才能输入给神经网络，这大大破坏了数据在空间上的位置信息。

卷积与反卷积操作使得神经网络可以处理二维以上的数据。它最大的好处在于：1.保持了数据在空间上的位置信息；

2.权重共享使得计算复杂度大大降低。

## **CNN的目的**

简单来说，CNN的目的是以一定的模型对事物进行特征提取，而后根据特征对该事物进行分类、识别、预测或决策等。在这个过程里，最重要的步骤在于特征提取，即如何提取到能最大程度区分事物的特征。如果提取的特征无法将不同的事物进行划分，那么该特征提取步骤将毫无意义。而实现这个伟大的模型的，是对CNN进行迭代训练。

### **特征**

在图像中（举个例子），目标事物的特征主要体现在像素与像素之间的关系。比如说，我们能区分一张图片中有一条直线，是因为直线上的像素与直线外邻像素的区别足够大（或直线两边的像素区别足够大），以至于这“直线”能被识别出来：除了直线外，其他特征也同理。在CNN中，大部分特征提取依赖于卷积运算。

## **卷积与特征提取**

卷积在此其实就是内积，步骤很简单，就是根据多个一定的权重（即卷积核），对一个块的像素进行内积运算，其输出就是提取的特征之一：

### **选用卷积的原因**

#### **局部感知**

简单来说，卷积核的大小一般小于输入图像的大小（如果等于则是全连接），因此卷积提取出的特征会更多地关注局部 —— 这很符合日常我们接触到的图像处理。而每个神经元其实没有必要对全局图像进行感知，只需要对局部进行感知，然后在更高层将局部的信息综合起来就得到了全局的信息。

#### **参数共享**

参数共享最大的作用莫过于很大限度地减少运算量了。

#### **多核**

一般我们都不会只用一个卷积核对输入图像进行过滤，因为一个核的参数是固定的，其提取的特征也会单一化。这就有点像是我们平时如何客观看待事物，必须要从多个角度分析事物，这样才能尽可能地避免对该事物产生偏见。我们也需要多个卷积核对输入图像进行卷积。

#### **Down-Pooling**

卷积后再接上一个池化层（Pooling）简直就是绝配，能很好的聚合特征、降维来减少运算量。

#### **多层卷积**

层数越高，提取到的特征就越全局化。

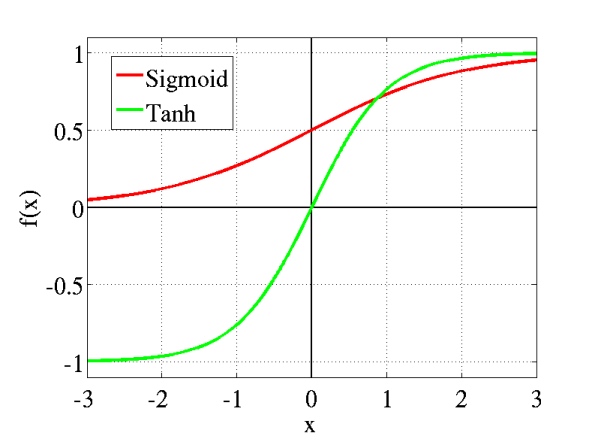
## **池化**

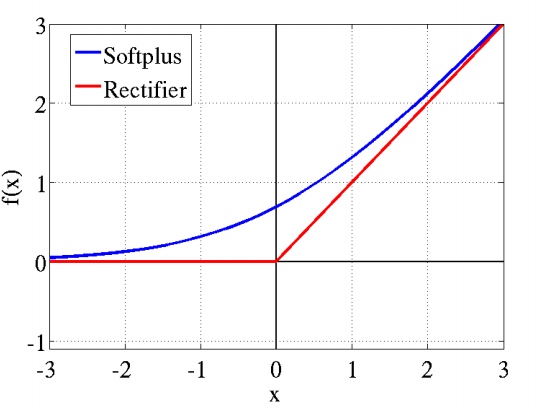
池化，即对一块数据进行抽样或聚合，例如选择该区域的最大值（或平均值）取代该区域：

上图的池化例子，将10 \* 10的区域池化层1 \* 1的区域，这样使数据的敏感度大大降低，同时也在保留数据信息的基础上降低了数据的计算复杂度。

## **激活函数的意义**

在数学上，激活函数的作用就是将输入数据映射到0到1上（tanh是映射-1到+1上）。至于映射的原因，除了对数据进行正则化外，大概是控制数据，使其只在一定的范围内。当然也有另外细节作用，例如Sigmoid（tanh）中，能在激活的时候，更关注数据在零（或中心点）前后的细小变化，而忽略数据在极端时的变化，例如ReLU还有避免梯度消失的作用。通常，Sigmoid（tanh）多用于全连接层，而ReLU多用于卷积层。





或者我们换一个卷积核（换一种角度）来看待这个激活函数，如果我们把每一次激活动作当成一次分类，即对输入数据分成两类（0或1），那么激活函数得到的输出是在0到1的值，它可以代表着这次“分类”的归属度。如果我们把0规定为未激活，1表示激活，那么输出0.44就表示激活了44%。

而激活函数的使用却有可能带来一定的负面影响（对于训练的负面影响），激活函数可能会使我们得输入数据都激活了大半，对此我们有另外的对策 —— LRN。

## **LRN的催化与抑制**

LRN，局部响应归一化。在神经学科中，有一个叫横向抑制（lateral inhibition）的概念，这种抑制的作用就是阻止兴奋神经元向邻近神经元传播其动作趋势，从而减少兴奋神经元的邻近神经元的激活程度。借鉴了这一生物现象（其实我们全部东西都是借鉴生物的，不是？），我们使用LRN层来对激活函数的输出数据进行横向抑制，在为激活函数收拾残局的同时，LRN还凸显了该区域的一个峰值 —— 这个峰值特征就是我们想要的特征。

特别是ReLU，它无限制的激活使得我们更需要LRN来到数据进行归一化。在大规模的数据中，我们大多情况下更看重被凸显的高频特征。因此，用LRN对数据的峰值进行催化而对其周围进行抑制，何乐不为。

## **Dropout的舍弃**

舍弃一直是一个伟大的哲学，生物的进化上例子比比皆是。Dropout的任务就是在训练时，通过设置阈值并与某些隐含层节点的权重对比，让特定的权重不工作 —— 在该层舍弃这些权重。Dropout的作用也很明显，除了加速运算外，就是防止过拟合了。