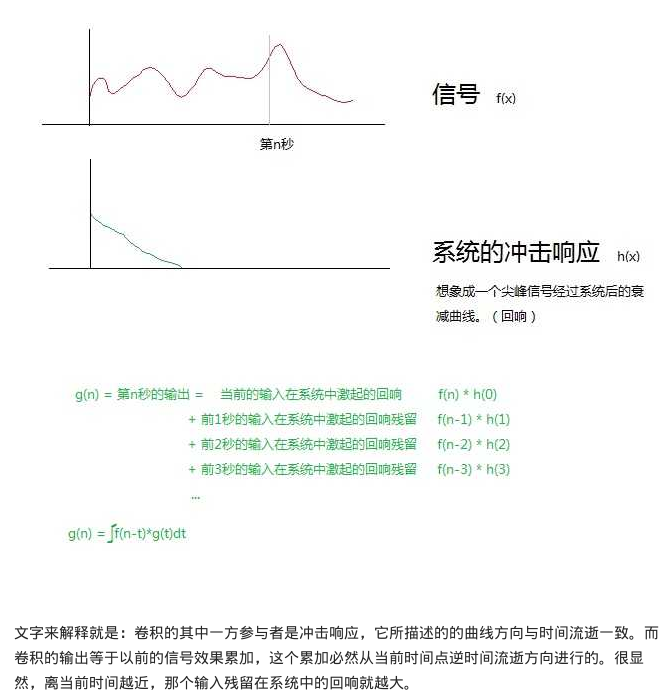
卷积

卷积翻转？

参考网址：<https://www.zhihu.com/question/20500497>



时间差f（x）和信号强度h（y）。以当前时间n为初始点，此时的信号强度为h（0），然后去观察n-1时刻的信号强度，相当于站在现在看过去的未来，过去是n-1，它此时的强度是最强的，也可以理解成h（0），那么未来的强度会比它弱（因为是衰减函数），也就是h（1）。

更为抽象一点的解释：两个函数，翻转其中一个，再滑动求积分，叫卷积（convultion）；不翻转就滑动求积分，叫做互相关（cross-correlation）。如果其中之一是偶函数，那么卷积和互相关效果相同。从定义上看，翻转这个操作就是一步操作而已，具体的物理意义只能在应用中找到。

以信号处理为例，卷积意味着把输入信号在时间轴上翻转，然后跟信号处理系统的描述方程（冲激响应）叠加积分。为什么要翻转？因为这样才符合现实：输入信号的 0 秒先跟冲激响应的 0 秒叠加，然后输入信号的 1 秒和冲激响应的 1 秒叠加，以此类推。当你把这两个函数分别画出来上下并列的时候，它们就好象合并的拉链，0 点处在同一侧，而卷积实际上是要把它们画在同一个轴上滑动，同时却必须保证输入信号的 0 点先遇到冲激响应函数的 0 点——怎么办呢？就好像拉链被拉开了：翻转一下。

MXnet是亚马逊开发的深度学习框架

CNN学习

对图像来说，总思路是将图片不断进行压缩，直到输出单一的值。

C语言实现CNN

<http://blog.csdn.net/tostq/article/details/51786265>

基于Tensorflow框架的CNN--python实现

<http://blog.csdn.net/xukaiwen_2016/article/details/70880694> <https://zhuanlan.zhihu.com/p/26645110?refer=carefree0910-pyml>

基于theano框架的CNN--Python实现

<http://blog.csdn.net/u014365862/article/details/54865609>

C++实现的深度学习框架--->tiny-cnn

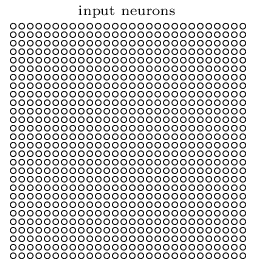
<http://blog.csdn.net/column/details/tiny-cnn.html>

**1、卷积神经网络CNN**

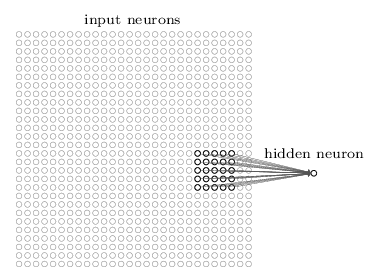
卷积神经网络有3个基本的idea：局部感知域(Local Recpetive Field)，权值共享和池化(Pooling)。下面我们来一个一个的介绍它们。

**（1）局部感知域**

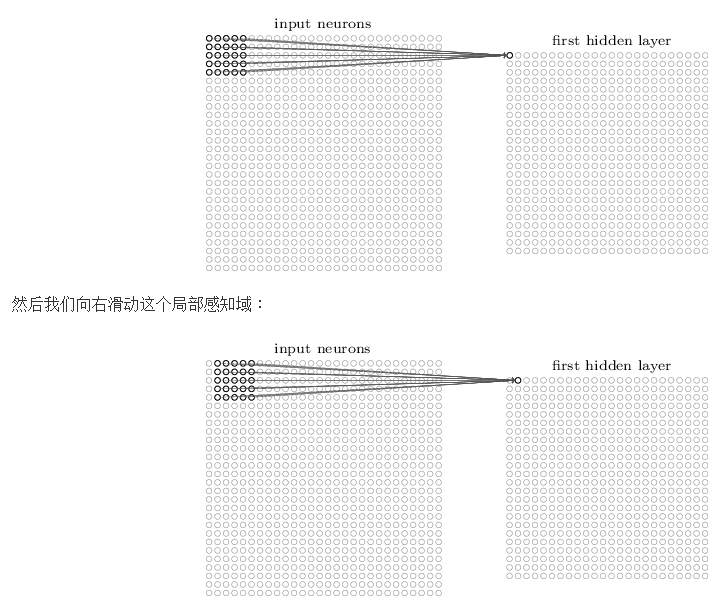
在前面图示的全连接的层里，输入是被描述成一列神经元。而在卷积网络里，我们把输入看成28×28方格的二维神经元，它的每一个神经元对应于图片在这个像素点的强度（灰度值），如下图所示：



和往常一样，我们把输入像素连接到隐藏层的神经元。但是我们这里不再把输入的每一个像素都连接到隐藏层的每一个神经元。与之不同，我们把很小的相临近的区域内的输入连接在一起。更加具体的来讲，隐藏层的每一个神经元都会与输入层一个很小的区域（比如一个5×5的区域，也就是25个像素点）相连接。隐藏对于隐藏层的某一个神经元，连接如下图所示：



输入图像的这个区域叫做那个隐藏层神经元的局部感知域。这是输入像素的一个小窗口。每个连接都有一个可以学习的权重，此外还有一个bias。你可以把那个神经元想象成用来分析这个局部感知域的。我们然后在整个输入图像上滑动这个局部感知域。对于每一个局部感知域，都有一个隐藏层的神经元与之对应。为了具体一点的展示，我们首先从最左上角的局部感知域开始：



以此类推，我们可以构建出第一个隐藏层。注意，如果我们的输入是28×28，并且使用5×5的局部关注域，那么隐藏层是24×24。因为我们只能向右和向下移动23个像素，再往下移动就会移出图像的边界了。【说明，后面我们会介绍padding和striding，从而让图像在经过这样一次卷积处理后尺寸可以不变小】

这里我们展示了一次向右/下移动一个像素。事实上，我们也可以使用一次移动不止一个像素【这个移动的值叫stride】。比如，我们可以一次向右/下移动两个像素。在这篇文章里，我们只使用stride为1来实验，但是请读者知道其他人可能会用不同的stride值。

**（2）共享权值**

之前提到过每一个隐藏层的神经元有一个5×5的权值。这24×24个隐藏层对应的权值是相同的。也就是说，对于隐藏层的第j,k个神经元，输出如下：

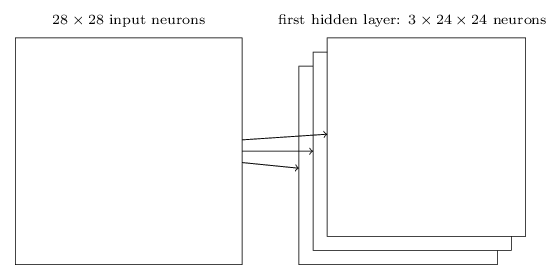
这里，σ是激活函数，可以是我们之前提到的sigmoid函数。b是共享的bias，Wl,m 是5×5的共享权值。ax,y 是输入在x,y的激活。

【从这个公式可以看出，权值是5×5的矩阵，不同的局部感知域使用这一个参数矩阵和bias】

这意味着这一个隐藏层的所有神经元都是检测同一个特征，只不过它们位于图片的不同位置而已。比如这组weights和bias是某个局部感知域学到的用来识别一个垂直的边。那么预测的时候不管这条边在哪个位置，它都会被某个对于的局部感知域检测到。更抽象一点，**卷积网络能很好的适应图片的位置变化**：把图片中的猫稍微移动一下位置，它仍然知道这是一只猫。

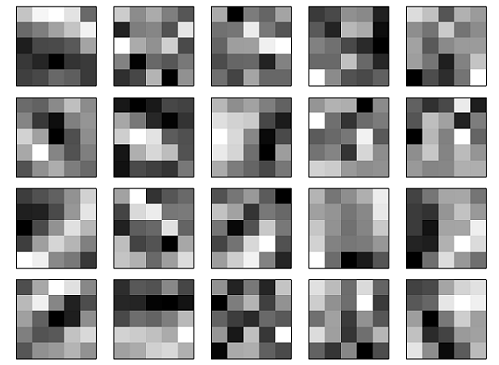
因为这个原因，我们有时把输入层到隐藏层的映射叫做特征映射(feature map)。我们把定义特征映射的权重叫做共享的权重(shared weights)，bias叫做共享的bias(shared bais)。这组weights和bias定义了一个kernel或者filter。

上面描述的网络结构只能检测一种局部的特征。为了识别图片，我们需要更多的特征映射。隐藏一个完整的卷积神经网络会有很多不同的特征映射：



在上面的例子里，我们有3个特征映射。每个映射由一个5×5的weights和一个biase确定。因此这个网络能检测3种特征，不管这3个特征出现在图像的哪个局部感知域里。

为了简化，上面之展示了3个特征映射。在实际使用的卷积神经网络中我们会使用非常多的特征映射。早期的一个卷积神经网络——LeNet-5，使用了6个特征映射，每一个都是5×5的局部感知域，来识别MNIST数字。因此上面的例子和LeNet-5很接近。后面我们开发的卷积层将使用20和40个特征映射。下面我们先看看模型学习到的一些特征：



这20个图片对应了20个不同的特征映射。每个映射是一个5×5的图像，对应于局部感知域的5×5个权重。颜色越白(浅)说明权值越小(一般都是负的)，因此对应像素对于识别这个特征越不重要。颜色越深(黑)说明权值越大，对应的像素越重要。

那么我们可以从这些特征映射里得出什么结论呢？很显然这里包含了非随机的空间结构。这说明我们的网络学到了一些空间结构。但是，也很难说它具体学到了哪些特征。我们学到的不是一个 Gabor滤波器 的。事实上有很多研究工作尝试理解机器到底学到了什么样的特征。如果你感兴趣，可以参考Matthew Zeiler 和 Rob Fergus在2013年的论文 Visualizing and Understanding Convolutional Networks。

共享权重和bias的一大好处是它极大的减少了网络的参数数量。对于每一个特征映射，我们只需要 25=5×5 个权重，再加一个bias。因此一个特征映射只有26个参数。如果我们有20个特征映射，那么只有20×26=520个参数。如果我们使用全连接的神经网络结构，假设隐藏层有30个神经元(这并不算很多)，那么就有784\*30个权重参数，再加上30个bias，总共有23,550个参数。换句话说，全连接的网络比卷积网络的参数多了40倍。

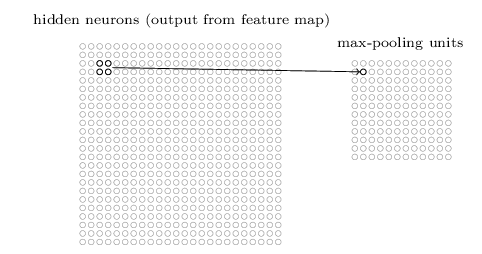
当然，我们不能直接比较两种网络的参数，因为这两种模型有本质的区别。但是，凭直觉，由于卷积网络有平移不变的特性，为了达到相同的效果，它也可能使用更少的参数。由于参数变少，卷积网络的训练速度也更快，从而相同的计算资源我们可以训练更深的网络。

“卷积”神经网络是因为公式(1)里的运算叫做“卷积运算”。更加具体一点，我们可以把公式(1)里的求和写成卷积：$a^1 = \sigma(b + w \* a^0)$。\*在这里不是乘法，而是卷积运算。这里不会讨论卷积的细节，所以读者如果不懂也不要担心，这里只不过是为了解释卷积神经网络这个名字的由来。【建议感兴趣的读者参考colah的博客文章 《Understanding Convolutions》】

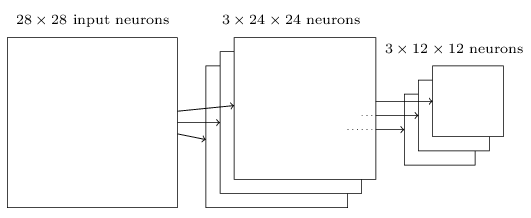
**（3）池化**

除了上面的卷积层，卷积神经网络也包括池化层(pooling layers)。池化层一般都直接放在卷积层后面池化层的目的是简化从卷积层输出的信息。

更具体一点，一个池化层把卷积层的输出作为其输入并且输出一个更紧凑(condensed)的特征映射。比如，池化层的每一个神经元都提取了之前那个卷积层的一个2×2区域的信息。更为具体的一个例子，一种非常常见的池化操作叫做Max-pooling。在Max-Pooling中，这个神经元选择2×2区域里激活值最大的值，如下图所示：



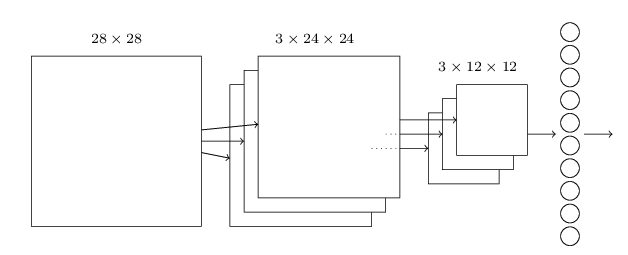
注意卷积层的输出是24×24的，而池化后是12×12的。

就像上面提到的，卷积层通常会有多个特征映射。我们会对每一个特征映射进行max-pooling操作。因此，如果一个卷积层有3个特征映射，那么卷积加max-pooling后就如下图所示：   


我们可以把max-pooling看成神经网络关心某个特征在这个区域里是否出现。它忽略了这个特征出现的具体位置。直觉上看，如果某个特征出现了，那么这个特征相对于其它特征的精确位置是不重要的【精确位置不重要，但是大致的位置是重要的，比如识别一个猫，两只眼睛和鼻子有一个大致的相对位置关系，但是在一个2×2的小区域里稍微移动一下眼睛，应该不太影响我们识别一只猫，而且它还能解决图像拍摄角度变化，扭曲等问题】。而且一个很大的好处是池化可以减少特征的个数【2×2的max-pooling让特征的大小变为原来的1/4】，因此减少了之后层的参数个数。

Max-pooling不是唯一的池化方法。另外一种常见的是L2 Pooling。这种方法不是取2×2区域的最大值，而是2×2区域的每个值平方然后求和然后取平方根。虽然细节有所不同，但思路和max-pooling是类似的：L2 Pooling也是从卷积层压缩信息的一种方法。在实践中，两种方法都被广泛使用。有时人们也使用其它的池化方法。如果你真的想尝试不同的方法来提供性能，那么你可以使用validation数据来尝试不同池化方法然后选择最合适的方法。但是这里我们不在讨论这些细节。【Max-Pooling是用的最多的，甚至也有人认为Pooling并没有什么卵用。深度学习一个问题就是很多经验的tricks由于没有太多理论依据，只是因为最早的人用了，而且看起来效果不错（但可能换一个数据集就不一定了），所以后面的人也跟着用。但是过了没多久又被认为这个trick其实没啥用】

**（4）合到一起**  
 现在我们可以把这3个idea放到一起来构建一个完整的卷积神经网络了。它和之前我们看到的结构类似，不过增加了一个有10个神经元的输出层，这个层的每个神经元对应于0-9直接的一个数字：



这个网络的输入的大小是28×28，每一个输入对于MNIST图像的一个像素。然后使用了3个特征映射，局部感知域的大小是5×5。这样得到3×24×24的输出。然后使用对每一个特征映射的输出应用2×2的max-pooling，得到3×12×12的输出。

最后一层是全连接的网络，3×12×12个神经元会连接到输出10个神经元中的每一个。这和之前介绍的全连接神经网络是一样的。

卷积结构和之前的全连接结构有很大的差别。但是整体的图景是类似的：一个神经网络有很多神经元，它们的行为有weights和biase确定。并且整体的目标也是类似的：使用训练数据来训练网络的weights和biases使得网络能够尽量好的识别图片。

和之前介绍的一样，这里我们仍然使用随机梯度下降来训练。不过反向传播算法有所不同。原因是之前bp算法的推导是基于全连接的神经网络。不过幸运的是求卷积和max-pooling的导数是非常简单的。