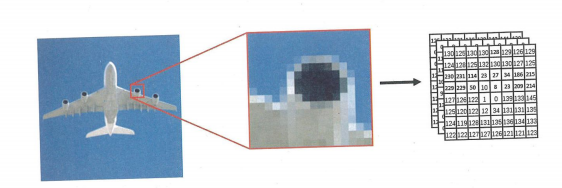
# 图像识别

## 计算机眼中的图像



在计算机中，对于图像存储是通过矩阵来存储的。

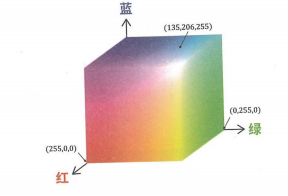
将一副图像放大，我们可以看到他是由一个个的小格子组成的，每个小格子都是一个色块。如果我们用不同的数字来表示不同的颜色，图像就可以表示为个有数字组成的矩阵（matrix）,这里的小格子我们称之为像素（pixel）,而格子的行数与列数，统称为分辨率（resolution）

反过来，如果给出一个数字组成的矩阵，我们将矩阵中的每个值转换为 对应的颜色，并在电脑上显示出来，就可以复现这张图像。

照片分为黑白和彩色。在图像里我们相应的有灰度图和彩色图。

对于灰度图像，由于只有明暗的区别，因此只需要一个数字就可以表示出不同的灰度。通常用0表示最暗的黑色，255表示最亮的白色，介于0和255之间的整数则表示不同明暗程度的灰色。

对于彩色图像，我们用（R,G,B）三个数字来表示一个颜色，他们表示用红（R）、绿（G）、蓝（B）三种基本颜色叠加后的颜色。对于每种基本颜色，我们用0到255之间的整数表示这个颜色分量的明暗程度。



三个数字中对应的某种基本颜色的数字越大，表示该基本颜色的比例越大，例如（255,0,0）表示纯红色，（0,255,0）表示纯绿色，（135,206,255）表示天蓝色。

一张彩色图片我们可以用一个由整数组成的立方体阵列来表示。我们称这样的立方体排列的数字阵列为三阶张量（tensor）。这个三阶张量的长度与宽度就是图片的分辨率，高度为3.对于数字图像而言，三阶张量的高度也成为通道（channel）数，因此我们说彩色图像有3个通道。矩阵可以看成是高度为1的三阶张量。

张量是数学、物理风学科的一个基本概念，标量属于零阶张量，向量属于一阶张量，矩阵 属于二阶张量。

## 图像特征概述

在深度学习出现之前，图像特征的设计一直是计算机视觉领域中一个重要的研究课题，在这个领域发展初期，人们手工设计了各种图像特征，这些特征可以描述图像的颜色、边缘、纹理等性质，结合机器学习技术，能解决物体识别和物体检测等实际问题。

既然图像在计算机中可以表示成三阶张量，那么从图像中提取特征便是对这个三阶张量进行运算的过程。其中非常重要的一种运算就是卷积。

## 卷积运算

### 3.1卷积定义

卷积是两个变量在某范围内相乘后求和的结果。

### 3.2一维卷积

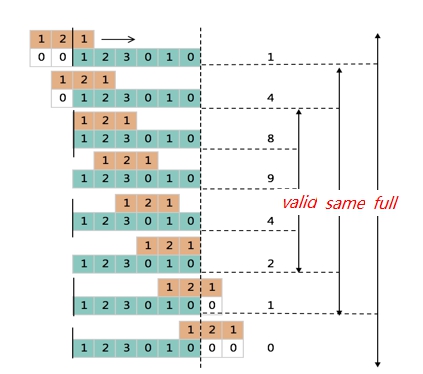
卷积运算的物理意义：一个函数（如：单位响应）在另一个函数（如:输入信号）上的加权叠加

有两个离散信号

待卷积信号 X=[1,2,3,0,1,0]，

卷积核 H=[1,2,1]

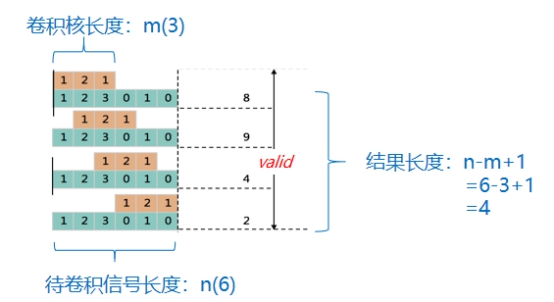
卷积运算 Y = X \* H



#### valid

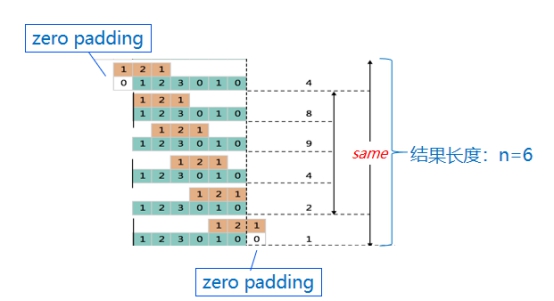
自始至终卷积核都在“信号内”

最后得到的结果的长度会小于卷积信号的长度

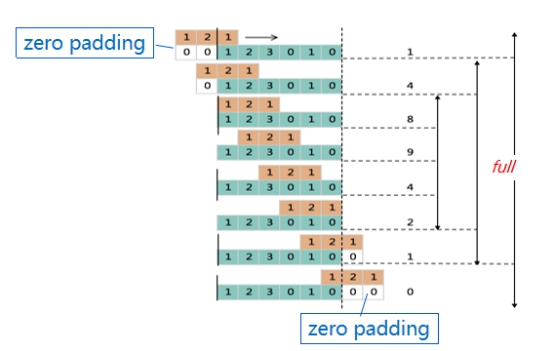


#### same

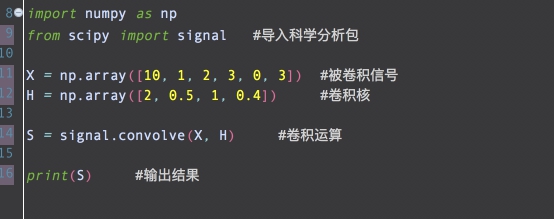
卷积核的中心刚好是从待卷积信号的第一个元素“划”到最后一个元素卷积结果的长度和待卷积信号长度一样



#### full

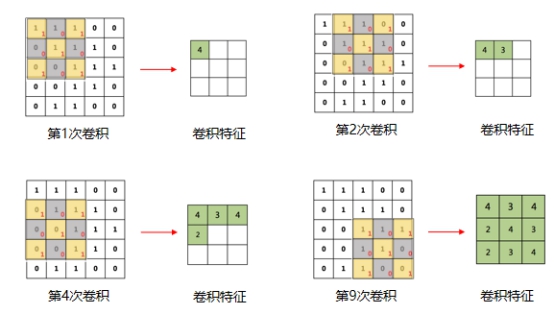
从卷积核的最后一个元素开始，直到第一个元素到与待卷积信号第一个元素对齐卷积结果的长度是n+m-1

#### 实现代码

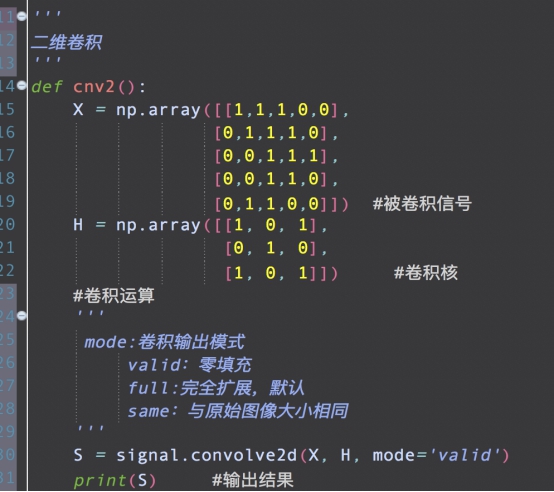


### 二维卷积

图像数据是5x5的二维矩阵，使用一个3x3的卷积核，从左到右从上到下滑动。滑动的过程称为stride，一个卷积层有两个stride，分别从上到下，从左到右，步长一般设定为1或2。



#### 实现代码

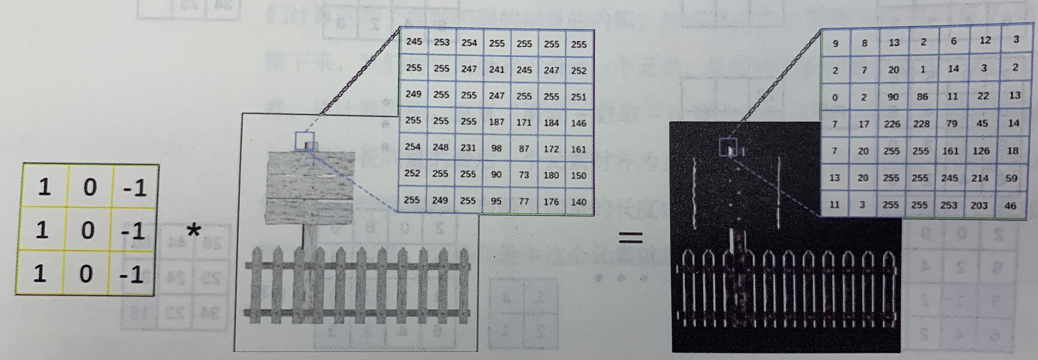


## 利用卷积提取图像特征

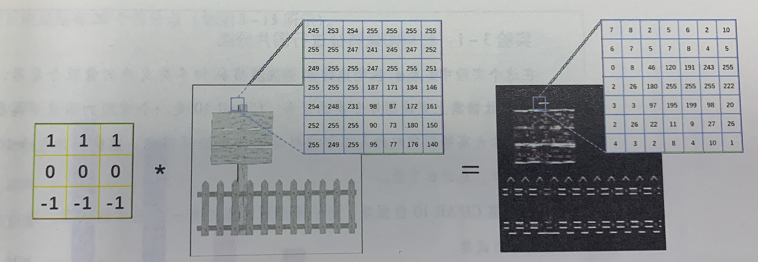
卷积运算在图像处理中应用十分广泛，许多图像特征提取方法都会用到卷积。以灰度图为例，，我们知道在计算机中，一个灰度图像被表示为一个整数矩阵，如果我们用一个形状较小的矩阵和这个图像矩阵做卷积运算，就可以得到一个新的矩阵，这个新的矩阵可以看作是一副新的图像，换句话说，通过卷积运算，我们可以将原图像变换为一副新的图像。这幅新图像比原图像更清楚地表示了某些性质，我们就可以把它看做原图像的一个特征。

这里用到的小矩阵就称为卷积核（convolution lernel），通常，图像矩阵中的元素都是介于0到255的整数，但卷积核中的元素可以是任意实数。

通过卷积，我们可以从图像中提取边缘特征，在没有边缘的比较平坦的区域，图像的像素值的变化较小，而横向边缘上下两侧的像素值 差异明显，竖向边缘左右两侧的像素也会有较大差别。



如上图，我们用1、0、-1 组成的卷积核与原图像进行卷积运算，可以从图像中提取出竖向边缘。



如上图，我们用三行1,0，-1组成的卷积核，从图中提取出了横向边缘。

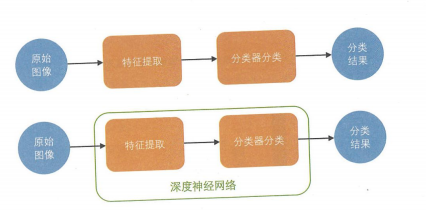
事实上，这两个卷积核分别计算了原图像上每个3\*3区域内左右像素或者上下像素的差值（为了将运算结果以图像的形式显示出来，我们对运算结果去了绝对值），通过这样的运算，我们就可以从图像上提取不同的边缘特征。

## 从特征设计到特征学习

上面所讲的利用人工设计的图像特征，由于很难直接表达一些高层次的抽象概念，图像分类的准确率并不令人满意，且遇到瓶颈，很难进一步进行优化。

深度神经网络可以实现自动从图像中学习有效的特征，将人工设计的图像特征无法解决的一些抽象的特征成为可能。在计算机视觉的各个领域，深度神经网络学习的特征也逐渐替代了手工设计的特征，人工智能变得更加“智能”。

另一方面，深度神经网络的出现也降低了人工智能的复杂度，在传统的模式分类中，特征提取与分类是两个独立的步骤，而深度神经网络将二者集成在了一起。我们只需要将一张图片输入给神经网络，就可以直接得出对图片类别的预测，不在需要分步完成特征提取与分类。从这个角度看，深度神经网络并不是对传统模式分类系统的颠覆，而是对传统系统的改进与增强。



## 深度神经网络的结构

一个深度神经网络由多个顺序连接的层组成。

第一层一般以图像为输入，通过特定的运算从图像中提取特征，接下来的每一层以前一层提取出的特征为输入，对其进行特定形式的变换，便可以得到更复杂一些特征。

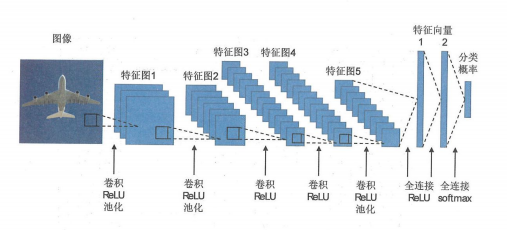
这种层次化特征提取过程可以累加，赋予神经网络强大的特征提取能力。

神经网络包含三层：输入层，隐含层，输出层

当一个神经网络以卷积层为主体（隐含层）时，称之为卷积神经网络。

### 6.1卷积神经网络基本结构

Alex Net 神经网络



* 上图为Alex Net 神经网络的主体部分，主体部分有5个卷积层和3个全连接层组成
* 5个卷积层位于网络的最前端，依次**对图像进行变换以提取特征**；
* 每个卷积层之后都有一个**ReLU**非线性激活层**完成非线性变换；**
* 第一、二、五个卷积层之后连接有最大池化 层，用以**降低特征图的分辨率**。
* 经过5个卷积层以及相连的非线性激活层与池化层之后，特征图被转换为4096维特征向量，在经过两次全连接层和ReLU层变换之后，成为最终的特征向量，在经过一个全连接层和一个softmax归一化指数层之后，就得到了对图片所属类型的预测。

### 6.2 卷积层

神经网络中的卷积层就是用卷积运算对原始图像或者上一层的特征进行变换的层。在前边的学习中，我们学习了边缘特征的提取，知道一种特定的卷积核可以对图像进行一种特定的变换，从而提取出某种特定的特征，如横向边缘或者竖向边缘。

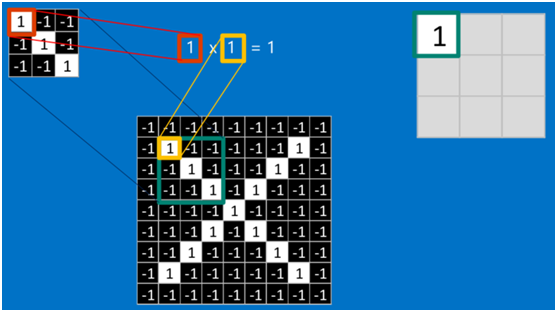
在一个卷积层中，为了从图像中提取多种形式的特征，我们通常使用多个卷积核对输入的图像进行不同的卷积操作。一个卷积核可以得到一个通道为1的三阶张量，，多个卷积核就可以得到多个通道为1的三阶张量结果。我们把这些结果作为不同的通道组合起来，就可以得到一个新的三阶张量，这个三阶张量的通道数就等于我们使用的卷积核的个数。由于每一个通道都是从原图像中提取的一种特征，我们也将这个三阶张量称为特征图（feature map）。这个特征图就是卷积层的最终输出。

特征图与彩色图像都是三阶张量，也都有若干个通道。因此卷积层不仅可以作用于图像，也可以作用于其他输出的特征图。通常，一个深度神经网络的第一个卷积层会以图像作为输入，而之后的卷积层会以前面的特征图为输入。

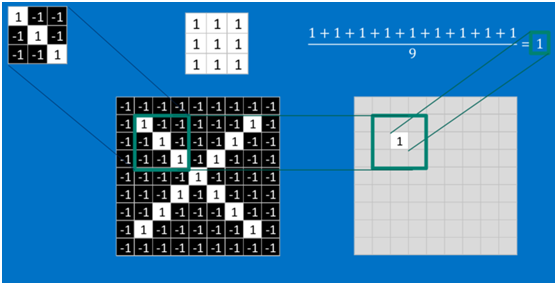
#### 实例：提取X的特征

计算一个feature（特征）和其在原图上对应的某一小块的结果

1. 将两个小块内对应位置的像素值进行乘法运算

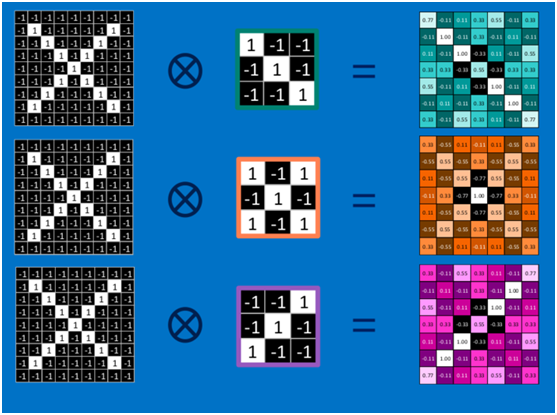


1. 将整个小块内乘法运算的结果累加起来
2. 除以小块内像素点总个数即可（注：也可不除以总个数的）



三个feature（特征）分别进行卷积操作，得到新的二维数组，称之为feature map

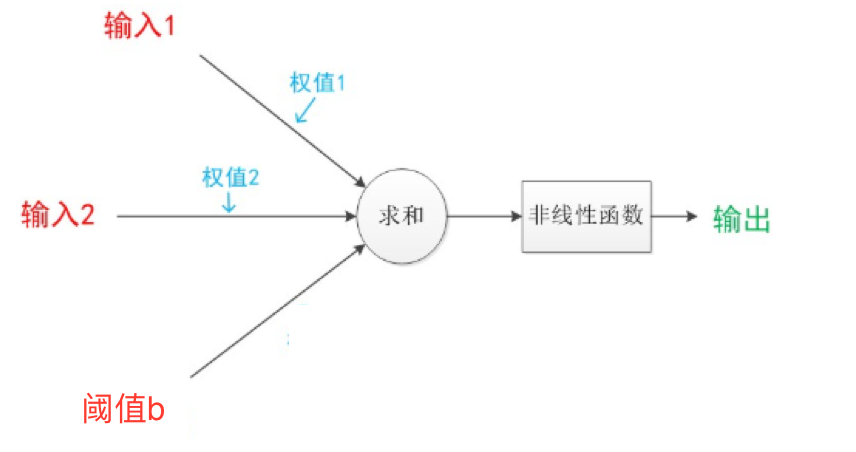
* 值越接近1表示对应位置和feature的匹配越完整
* 值越是接近-1，表示对应位置和feature的反面匹配越完整
* 值接近0的表示对应位置没有任何匹配或者说没有什么关联



### 6.3 全连接层

在图片分类中，输入图片在经过若干卷积层之后，会将得到的特征图转换为特征向量。如果需要对这个特征向量进行变换，经常用到的是全连接层（fully-connected layer）。

在全连接层中，我们会使用若干维数相同的向量与输入向量做内积操作，并将所有结果拼接成一个向量作为输出。具体来说，如果一个全连接层以向量X作为输入，我们会用总共K个维数相同的参数向量和X做內积运算，再将每个结果加上一个标量，即完成的运算。最后，我们将K个标量结果组成的Y作为这一层的输出。



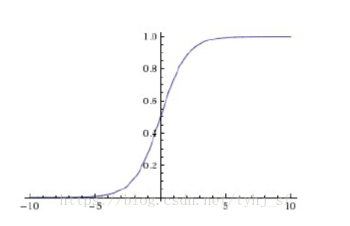
### 6.4 非线性激活层

通常我们需要在每个卷积层和全连接层后面都连接一个非线性激活层（non-linear activation layer）。为什么呢？其实不管是卷积运算还是全连接层中的运算，他们都是自变量的一次函数，即所谓的线性函数（linear function）。线性函数有一个性质：若干线性计算的符合仍然是线性的。换句话说，如果我们只是将卷积层和全连接层直接堆叠起来，，那么它们对输入图片产生的效果就可以被一个全连接层替代。这样一来，虽然我们堆叠了很多层，但对每一层的变换效果实际上被合并到了一起。而如果我们在每次线性运算后，再进行一次非线性运算，那么每次变换的效果就可以保留。非线性激活层的形式与很多种，它们的基本形式是先选定某种非线性函数，然后对输入特征图或者特征向量的每一个元素应用这种非线性函数，得到输出。

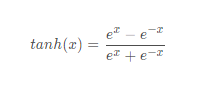
常见的非线性函数有：

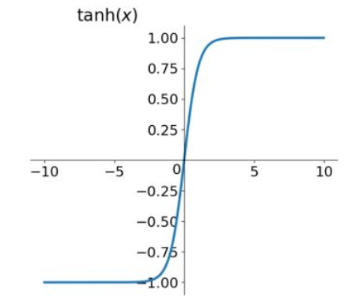
* 逻辑函数（logistic function）sigmoid



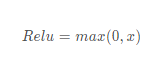


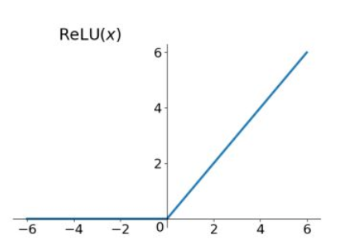
* 双曲正切函数（hyperbolic tangent function）





* 线性整流函数（rectified linear function）

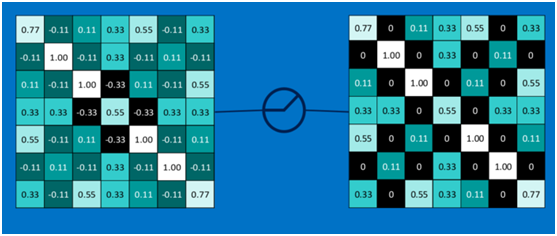




前两者sigmoid/tanh比较常见于全连接层，后者ReLU常见于卷积层。

激活函数是用来加入非线性因素的，使得神经网络可以任意逼近任何非线性函数，提高经网络对模型的表达能力，解决线性模型所不能解决的问题，这样神经网络就可以应用到众多的非线性模型中。

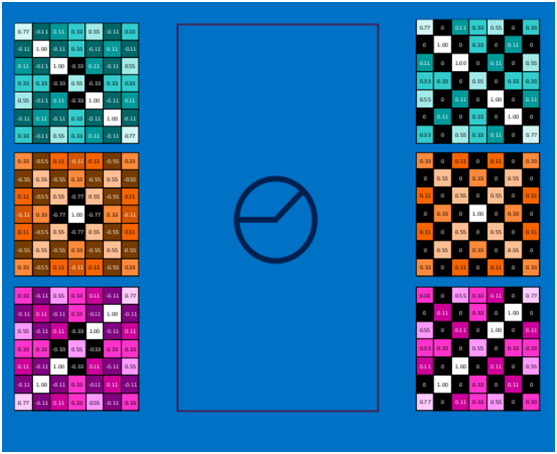
以线性整流函数构成的非线性激活层（简称ReLU层）为例，对于输入的特征向量或特征图，他会将其中小于零的元素变成零，而其他元素的值保持不变，就得到了输出。



因为ReLU的计算非常简单，所以它的计算速度往往比其他非线性激活层快很多，价值其在实际应用中的效果也很好，因此在深度神经网络中被广泛地使用。

#### 实例：

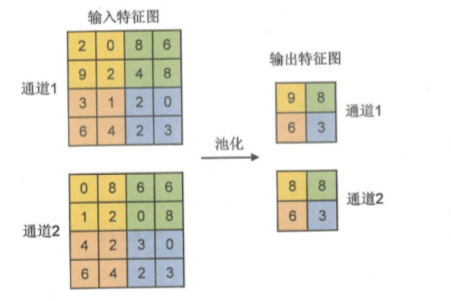
对所有的feature map执行ReLU激活函数操作，结果如下：



### 6.5 池化层

在计算卷积时，我们会用卷积核滑过图像或者特征图的每一个像素。如果图像或者特征图的分辨率很多，那么卷积的计算量就会很大。为了解决 这个问题，我们通常在几个卷积层之后插入池化层（pooling layer），已降低特征图的分辨率。

池化层的基本操作步骤如下。首先，我们将特征图按通道分开，得到若干个矩阵。对于每个矩阵，我们将其切割成若干大小相等的正方形小块。如下图，我们将一个4\*4的矩阵分割成4个正方形区块，每个区块的大小为2\*2.接下来，我们对每一个区块取最大值或者平均值，并将结果组成一个新的矩阵。最后，我们将所有通道的结果矩阵按原顺序堆叠起来形成一个三阶张量，这个三阶张量就是池化层的输出。**对于每一个区块取最大值的池化层，我们称之为最大池化层（max pooling），而取平均值的池化层成为平均池化层（average pooling layer）。**



在经过池化后，特征图的长和宽都会减小到原来的1/2，特征图中的元素数目减小到原来的1/4。通常我们会在卷积层之后增加池化层。这样，在经过若干卷积、池化层的组合之后，在不考虑通道数的情况下，特征图的分辨率就会远小于输入图像的分辨率，大大减小了对计算量和参数数量的需求。

## 神经网络的训练-反向传播

深度神经网络也需要经过训练才能学习出有效的图像特征。

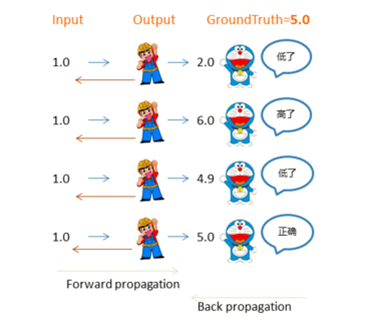
训练的本质是寻找最佳参数的过程。

在神经网络中，卷积层中所欲卷积核的元素值、全连接层中所有內积运算的系数都是参数，都需要经过训练来达到最优。

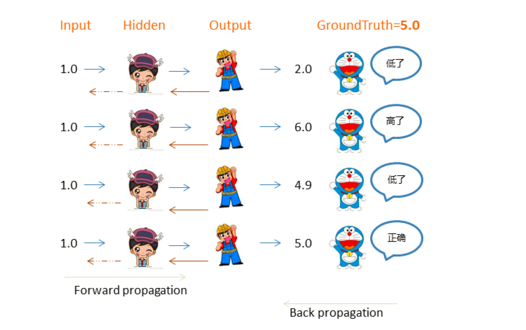
训练神经网络最有效的方法之一是反向传播（backpropagation）算法。

每次我们将一副训练图像输入网络中，经过逐层计算，最终得到预测的属于每一类的概率，我们将预测结果与正确答案进行对比，如果发现预测结果不够好，那么会从最后一层开始，逐层调整神经网络的参数，使得网络对这个训练样本能够做出更好的预测，我们将这种**从后往前调整参数的方法称为反向传播算法。**

### 7.1 算法介绍



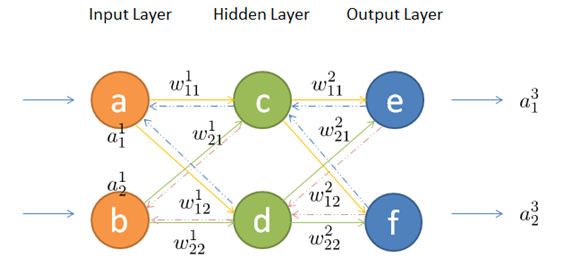
这一过程类比没有隐层的神经网络，比如逻辑回归，其中小黄帽代表输出层节点，左侧接受输入信号，右侧产生输出结果，小蓝猫则代表了误差，指导参数往更优的方向调整。由于小蓝猫可以直接将误差反馈给小黄帽，同时只有一个参数矩阵和小黄帽直接相连，所以可以直接通过误差进行参数优化（实纵线），迭代几轮，误差会降低到最小。

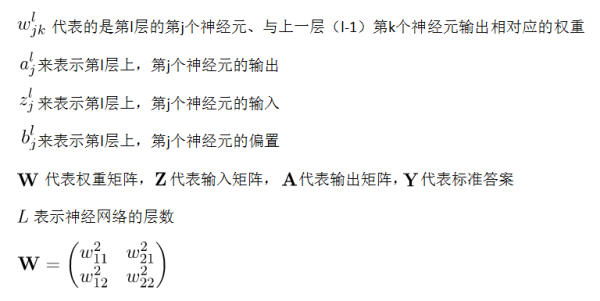


### 7.2 完整过程

上边的栗子从直观角度了解了反向传播，接下来就详细的介绍其中两个流程前向传播与反向传播，在介绍之前先统一一下标记。

#### 7.2.1 数学标记





#### 7.2.2 前向传播

如何将输入层的信号传输至隐藏层呢，以隐藏层节点c为例，站在节点c上往后看（输入层的方向），可以看到有两个箭头指向节点c，因此a，b节点的信息将传递给c，同时每个箭头有一定的权重，因此对于c节点来说，输入信号为：



同理，节点d的输入信号为：



由于计算机善于做带有循环的任务，因此我们可以用矩阵相乘来表示：



所以，隐藏层节点经过非线性变换后的输出表示如下：



同理，输出层的输入信号表示为权重矩阵乘以上一层的输出：



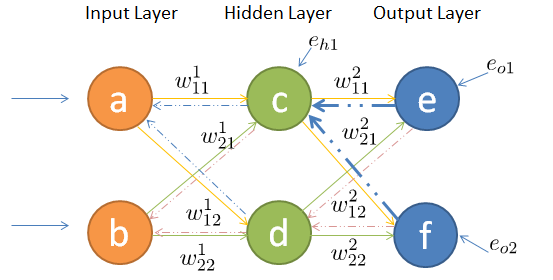
同样，输出层节点经过非线性映射后的最终输出表示为：



输入信号在权重矩阵们的帮助下，得到每一层的输出，最终到达输出层。可见，权重矩阵在前向传播信号的过程中扮演着运输兵的作用，起到承上启下的功能。

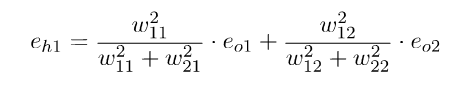
#### 7.2.3 反向传播

既然梯度下降需要每一层都有明确的误差才能更新参数，所以接下来的重点是如何将输出层的误差反向传播给隐藏层。

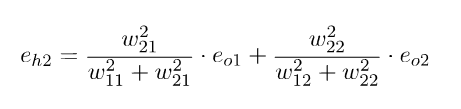


其中输出层、隐藏层节点的误差如图所示，输出层误差已知，接下来对隐藏层第一个节点c作误差分析。还是站在节点c上，不同的是这次是往前看（输出层的方向），可以看到指向c节点的两个蓝色粗箭头是从节点e和节点f开始的，因此对于节点c的误差肯定是和输出层的节点e和f有关。

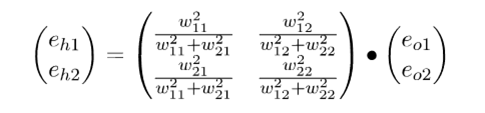
不难发现，输出层的节点e有箭头分别指向了隐藏层的节点c和d，因此对于隐藏节点e的误差不能被隐藏节点c霸为己有，而是要服从按劳分配的原则（按权重分配），同理节点f的误差也需服从这样的原则，因此对于隐藏层节点c的误差为：



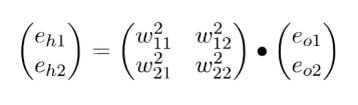
同理，对于隐藏层节点d的误差为：



为了减少工作量，我们还是乐意写成矩阵相乘的形式：



你会发现这个矩阵比较繁琐，如果能够简化到前向传播那样的形式就更好了。实际上我们可以这么来做，只要不破坏它们的比例就好，因此我们可以忽略掉分母部分，所以重新成矩阵形式为：



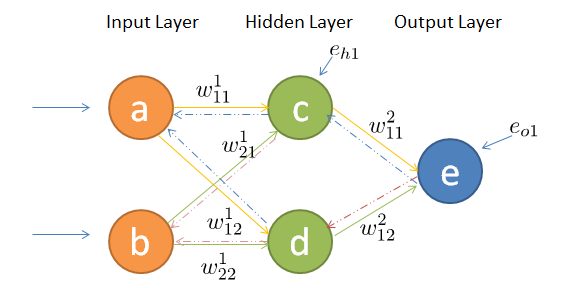
仔细观察，你会发现这个权重矩阵，其实是前向传播时权重矩阵w的转置，因此简写形式如下：



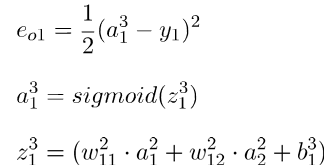
不难发现，输出层误差在转置权重矩阵的帮助下，传递到了隐藏层，这样我们就可以利用间接误差来更新与隐藏层相连的权重矩阵。可见，权重矩阵在反向传播的过程中同样扮演着运输兵的作用，只不过这次是搬运的输出误差，而不是输入信号。

### 7.3 链式求导

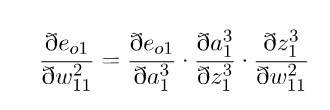
第三部分大致介绍了输入信息的前向传播与输出误差的后向传播，接下来就根据求得的误差来更新参数。



首先对隐藏层的w11进行参数更新，更新之前让我们从后往前推导，直到预见w11为止：



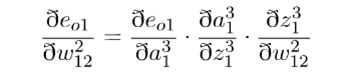
因此误差对w11求偏导如下：



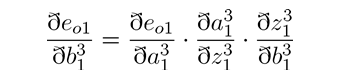
求导得如下公式（所有值已知）：



同理，误差对于w12的偏导如下：



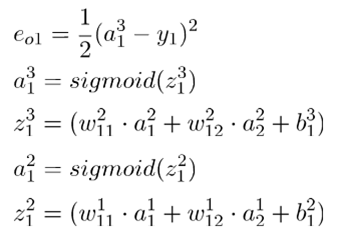
同样，求导得w12的求值公式：



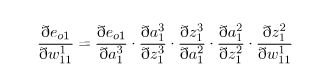
带入上述公式为：



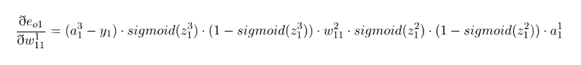
接着对输入层的w11进行参数更新，更新之前我们依然从后往前推导，直到预见第一层的w11为止（只不过这次需要往前推的更久一些）：



因此误差对输入层的w11求偏导如下：

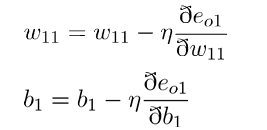


求导得如下公式：



同理，输入层的其他三个参数按照同样的方法即可求出各自的偏导，在这不再赘述。

在每个参数偏导数明确的情况下，带入梯度下降公式即可（不在重点介绍）：



至此，利用链式法则来对每层参数进行更新的任务已经完成。