**反卷积函数介绍**

这是tensorflow里实现反卷积的函数

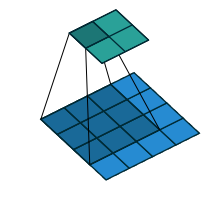
**tf.nn.conv2d\_transpose(value, filter, output\_shape, strides,**

**padding='SAME', name=None)**

1. value是上一层的feature map，
2. filter是卷积核[kernel\_size, kernel\_size, output\_channel, input\_channel ]，
3. output\_shape定义输出的尺寸[batch\_size, height, width, channel]，
4. padding是边界打补丁的算法。

这里需要特别说明的是，output\_shape和strides里的参数是相互耦合的，我们可以根据输入和输出确定strides参数（正整数），也可以根据输入和strides确定输出尺寸。

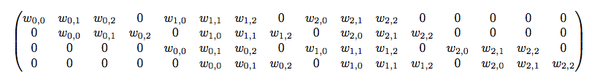
一句话解释：逆卷积相对于卷积在神经网络结构的正向和反向传播中做相反的运算。逆卷积(Deconvolution)比较容易引起误会，转置卷积(Transposed Convolution)是一个更为合适的叫法.  
举个栗子：

4x4的输入，卷积Kernel为3x3, 没有Padding / Stride, 则输出为2x2。  


输入矩阵可展开为16维向量，记作x,

输出矩阵可展开为4维向量，记作y,

卷积运算可表示为 y=Cx

不难想象C其实就是如下的稀疏阵:   


平时神经网络中的正向传播就是转换成了如上矩阵运算。

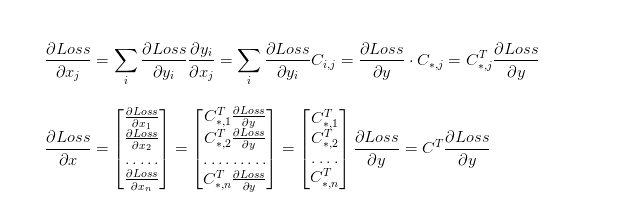
C是一个4\*16的矩阵，C的第一行与x分别相乘再相加，就是上图画的第一个绿框的值。

我们看C的元素，不参与运算对应的像素在C中的值为0。

因此C第一行对应输入图像左上角的3\*3的9个像素，第二行对应输入图像右上角的3\*3的9个像素，第三行对应输入图像右下角的3\*3的9个像素，第四行对应输入图像右下角的3\*3的9个像素。这样以来，C的每一行和x分别相乘再相加，就对应着输入图像左上角的9个元素和3\*3卷积核的卷积，右上角的9个元素和3\*3卷积核的卷积，左下角的9个元素和3\*3卷积核的卷积，右下角的9个元素和3\*3卷积核的卷积。

那么当反向传播时又会如何呢？

首先我们已经有从更深层的网络中得到的

所谓逆卷积其实就是正向时左乘C^T，而反向时左乘(C^T)^T，即C的运算。

逆卷积的一个很有趣的应用是GAN。

Deconv本身其实有标准定义指conv的逆运算，一般通过Fourier Transform求解。深度学习里的deconv其实用错了，不是指逆运算，而是指transposed convolution, backward convolution, fractally strided convolution, upsampling convolution.

第一篇用deconv表示transposed convolution的是ZF的Visualizing and Understanding Convolutional Neural Networks，论文提出了ZFNet，而且是ImageNet 2013年的winner，比较有影响力，之后大家就错误的用下去了。之后有些researchers提出应该叫transposed convolution, backward convolution (见cs231n的课件)，但似乎并没有改过来，比较新的深度学习框架应该都是有Deconvolution和TransposedConvolution两个名字的，一般也会注明他们是等价的。

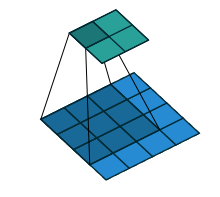
Transposed convolution 等价于 backward convolution，个人感觉这个结论是比较beautiful的，这个结果做下简单是数学推导就好了。

至于fractally strided convolution这个名字，应该是stride>1的时候可以upsampling，因此也叫upsampling convolution，具体操作就是dialation(在相邻元素之间添0)+transposed conv，一般用于FCN和DCGAN。

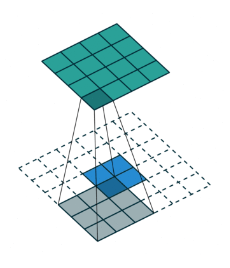
**【下面是卷积和反卷积的示例图】**

1. 卷积(convolution):卷积核为 3x3；**no padding , strides=1**

输入特征图为4\*4，经过3\*3的卷积核，输出的特征图为2\*2



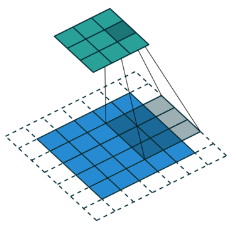
1. "反卷积"(the transpose of conv) 可以理解为upsample conv.  
   卷积核为:3x3; **no padding , strides=1**。



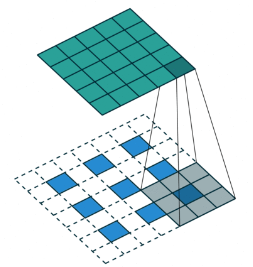
因为输出的特征图为4\*4，卷积核为3\*3，因此输入的特征图大小应该为6\*6。然而原来的特征图是2\*2，因此其他位置的像素补0，形成6\*6的特征图，再进行卷积，得到4\*4的特征图。

这样一来，就把原来2\*2的特征图，经过反卷积后，变成了4\*4的特征图。即upsample conv。

3.strides=2的时候的卷积:



4. strides=2的时候的反卷积:



因为输出的特征图为5\*5，卷积核为3\*3，因此输入的特征图大小应该为7\*7。然而原来的特征图是3\*3，因此**在相邻元素之间添0**，形成7\*7的特征图，再进行卷积，得到5\*5的特征图。

**【卷积和反卷积的例子】**

在实际计算过程中，卷积的计算都要转化为矩阵的乘积的形式，一个转化为Toeplitz matrix，一个reshape为列矩阵。

举个简单的例子：

**1.卷积**

比如 input= [4,4],Reshape之后，为A=[1,16]，B(可以理解为滤波器)=[16,4](Toeplitz matrix)，那么A\*B=C=[1,4]。Reshape C=[2，2]。所以，通过B 卷积，我们从shape=[3,3]变成了shape=[2,2]。

A: 4\*4

B：(16\*4)4对应滑动的4个小窗口，16代表输入的4\*4=16个数据。

C: 3\*3

关于B这个3\*3的卷积核，怎么生成16\*4的矩阵，很容易想明白。

假设3\*3的卷积核的各个参数表示如下：

那么，Toeplitz matrix的16\*4矩阵为

这样，原来4\*4的特征图，Reshape后变成1\*16的特征图，和16\*4的矩阵相乘后，得到1\*4的特征图，shape为2\*2。

**2.反卷积**

输入A=[2,2],reshape之后为[1,4] ，B的转置为[4,16]，那么A\*B=C=[1,16],reshape为[4,4]， 所以，通过B的转置 - "反卷积"，我们从shape=[2,2]得到了shape=[4,4]。

A: 2\*2

B的转置：(4\*16)4对应滑动的4个小窗口，16代表输出的4\*4=16个数据。

C: 4\*4

下面是B的转置（4\*16）：

**【总结】**

卷积：输入feature map A=[4,4]经过了卷积滤波B=[3,3] 输出为 [2,2] ,所以padding=0,stride=1。

反卷积：输入feature map A=[2,2],经过了反卷积滤波B=[3,3]。输出为[4,4]。padding=0,stride=1。

那么[3,3]的卷积核(滤波器)是怎么转化为[4,16]或者[16,4]的呢？这个上面的矩阵已经展示了。

**所以所谓的反卷积其实是转置卷积。**

**那为什么不能叫反卷积？**

**反卷积的数学含义，通过反卷积可以将通过卷积的输出信号，完全还原输入信号。而事实是，转置卷积只能还原shape大小，不能还原value.**