图解 CNN 反向传播

■ Dehong Gao（Hangzhou）, gaodehong_polyu@163.com

1. Introduction

近年来，深度学习风靡整个学术界和工业界。有些媒体认为它将带来一次新的技术革命。深度学习在某些领域确实显著提高了算法识别准确率，例如，声音，图像，视频等方向研究。这其中卷积神经网络 Convolution neuron network（CNN）发挥了很大作用，尤其是在图像识别领域。

目前网上大部分资料停留在对于 CNN 网络结构，以及前向传导的介绍。但 CNN 的后向传播方法讨论甚少，能够搜集到的资料，部分高屋建瓴，部分语焉不详。各个资料数学符号自成体系，浪费了不少读者时间。因此，本文意在全面的解析 CNN，内容不仅包括 CNN 前馈，同时深入讨论 CNN 后向传播方法及一些实现技巧。希望读者可以通过本文的阅读，自己动手编写 CNN。

本文将从浅层神经网络谈起，逐步介绍 CNN 网络，以及实现方法。本文将重点介绍 CNN 中残差反向传导过程，以及相应参数更新实现。

第一节从浅层神经网络开始介绍；

想了解 BP 算法读者，可以从第 2.2 节开始；

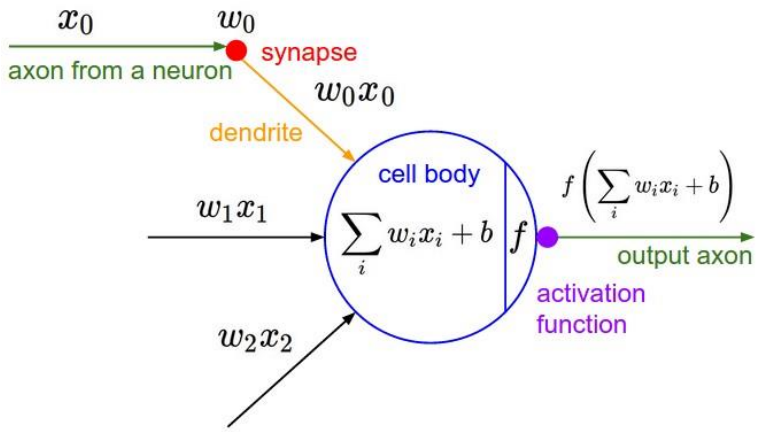
想了解 CNN 的读者，可以从第 3 节开始；

仅是想了解 CNN 实现 BP 过程，可以从第 3.3 节开始；

如果文章中有什么纰漏或者错误，请大侠不吝指正。另外，本文部分内容引用了网上博客，已经将重要的链接附上。如有内容觉得侵犯版权，请与我联系（题目下方邮箱）我会尽快删除。

2.Start from Shallow Neuron Network (NN)

神经元：多个输入（包括一个 bias）线性加和，然后通过非线性函数（也称激活函数，例如 sigmod 或者 tanh），输出，也称激活值。



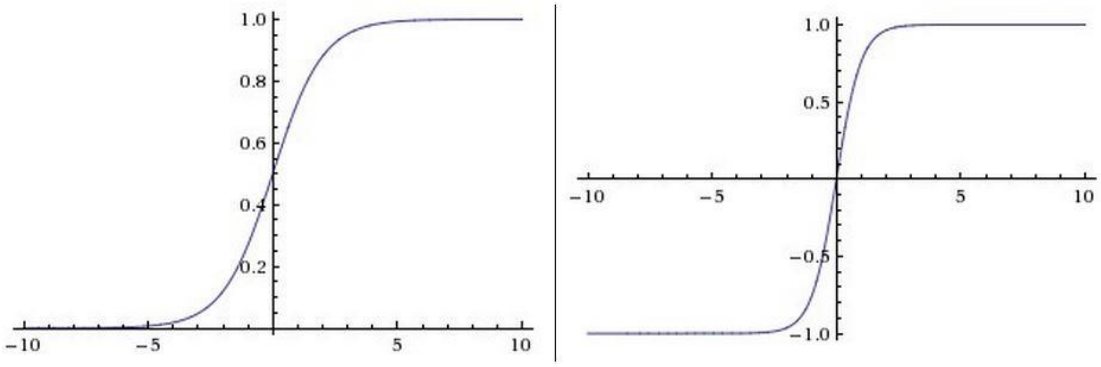
变量定义:

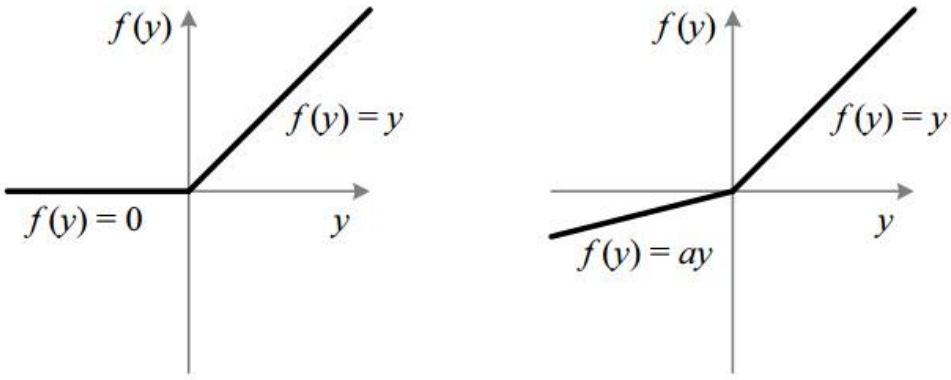
X： 网络输入

W: 网络链接权重

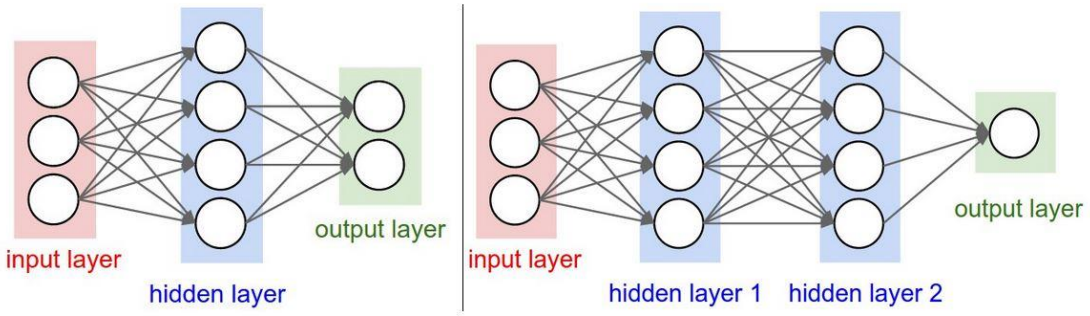
Z: 神经元的输入，Z=WX+B

A: 激活值 A = f(Z), f 是激活函数，可以是 sigmod，tanh，(Leaky) ReLu(Rectified Linear Units), Maxout. 关于激活函数优缺点：http://cs231n.github.io/neural-networks-1/



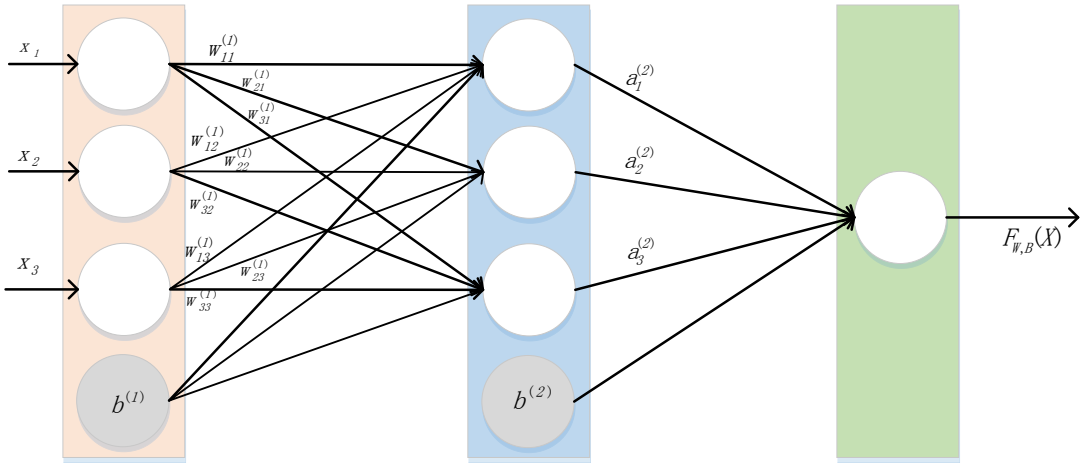


神经网络（浅层神经网络）：多个神经元，层次化组合在一起。如下图所示



这里面包含了输入层，输出层以及隐藏层。输出层可以单个神经元也可以是多个神经元，要根据实际问题而定。神经网络中包含两个重要的操作：前向传播(forward propagation)，以及后向传播(backward propagation)。

2.1 Forward (FW) Propagation in Shallow NN



网络前向传播数学表达

$$a_1^{(2)} = f\left(Z_1^{(2)}\right) = f\left(w_{11}^{(2)}x_1 + w_{12}^{(2)}x_2 + w_{13}^{(2)}x_3 + b_1^{(2)}\right)$$

$$a_2^{(2)} = f\left(Z_2^{(2)}\right) = f\left(w_{21}^{(2)}x_1 + w_{22}^{(2)}x_2 + w_{23}^{(2)}x_3 + b_2^{(2)}\right)$$

$$a_3^{(2)} = f\left(Z_2^{(2)}\right) = f(w_{31}^{(2)}x_1 + w_{32}^{(2)}x_2 + w_{33}^{(2)}x_3 + b_3^{(2)})$$

$$F_{W,B}(x) = a_1^{(3)} = f\left(Z_1^{(3)}\right) = f\left(w_{11}^{(3)}a_1^{(2)} + w_{12}^{(3)}a_2^{(2)} + w_{13}^{(3)}a_3^{(2)} + b_1^{(3)}\right)$$

这里注意$w_{ij}^{(l)}, b_i^{(l)}$下标的含义。$w_{ij}^{(l)}$表示第l-1 层第j个神经元到第l层的第i个神经元连接的权重。$b_i^{(l)}$表示连接到第l层的第i个神经元偏置。

前向传播的矩阵表达

$$Z^{(l+1)} = W^{(l+1)}A^{(l)} + B^{(l+1)}$$
$$\mathrm{A}^{(l+1)} = f(Z^{(l+1)})$$

2.2 Backward (BW) Propagation in Shallow NN

Backward propagation 的目的只要是将误差回传，然后计算各个 weight 的权重。最后用于特征权重的更新。

2.2.1 Cost Function

代价函数是为了衡量我们的模型离目标还有多大距离。代价函数也有很多种类，目前比较流行代价函数 Quadratic cost， Cross-entropy cost

(1) Quadratic cost

$$C(W,B)=\frac{1}{2n}\sum_{x}\|y(x)-a(x)\|^2$$

x表示任意一个训练样本，$y(x)$表示样本的 Label。$a(x)$表示 NN Forward 的输出。

(2) Cross-entropy cost

$$C(W,B)=-\frac{1}{n}\sum_{x}[y(x)\ln a(x)+(1-y(x))\ln(1-a(x))]$$

代价函数选择：一般多为 Cross-entropy + Softmax，原因是可以解决神经元 saturate 问题。后面的公式推导可以展示为什么 Cross-entropy 会解决神经元 saturate 问题。

2.2.2 Backward Propagation (BP) in Shallow NN

从 SGD 说起

Stochastic Gradient Descent

Loop until convergence{

For $\boldsymbol{i=1,\ldots.,m}$

$$\boldsymbol{W^{(t+1)}=W^{(t)}-\eta^{(t)}\frac{\partial C}{\partial W}}$$

done

}

对于任意$w_{ij}^{(l)},b_i^{(l)}$，我们需要计算$\frac{\partial C}{\partial w_{ij}^{(l)}},\frac{\partial C}{\partial b_i^{(l)}}$，用于$w_{ij}^{(l)},b_i^{(l)}$的更新。这里注意$w_{ij}^{(l)},b_i^{(l)}$的下标，。

为了计算$\frac{\partial C}{\partial w_{ij}^{(l)}}$，这里引入残差δ的概念。残差用于定义为 cost function 对神经元的输入的导数

$$\delta^{(l)}=\frac{\partial C}{\partial Z^{(l)}}$$

那么对于任意$\frac{\partial C}{\partial w_{ij}^{(l)}}$的推导

$$\frac{\partial C}{\partial w_{ij}^{(l)}}=\frac{\partial C}{\partial z_i^{(l)}}\frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l)}}=\delta_i^{(l)}\frac{\partial\left(\sum_k w_{ik}^{(l)}a_k^{(l-1)}+b_i^{(l)}\right)}{\partial w_{ij}^{(l)}}=\delta_i^{(l)}a_j^{(l-1)}$$	公式 1

$$\frac{\partial C}{\partial b_i^{(l)}} = \frac{\partial C}{\partial z_i^{(l)}}\frac{\partial z_i^{(l)}}{\partial b_i^{(l)}} = \delta_i^{(l)}\frac{\partial\left(\sum_k w_{ik}^{(l)}a_k^{(l-1)} + b_i^{(l)}\right)}{\partial b_i^{(l)}} = \delta_i^{(l)}$$	公式 2

注意上面公式中的下标。

神经网络中第 L 层的残差，

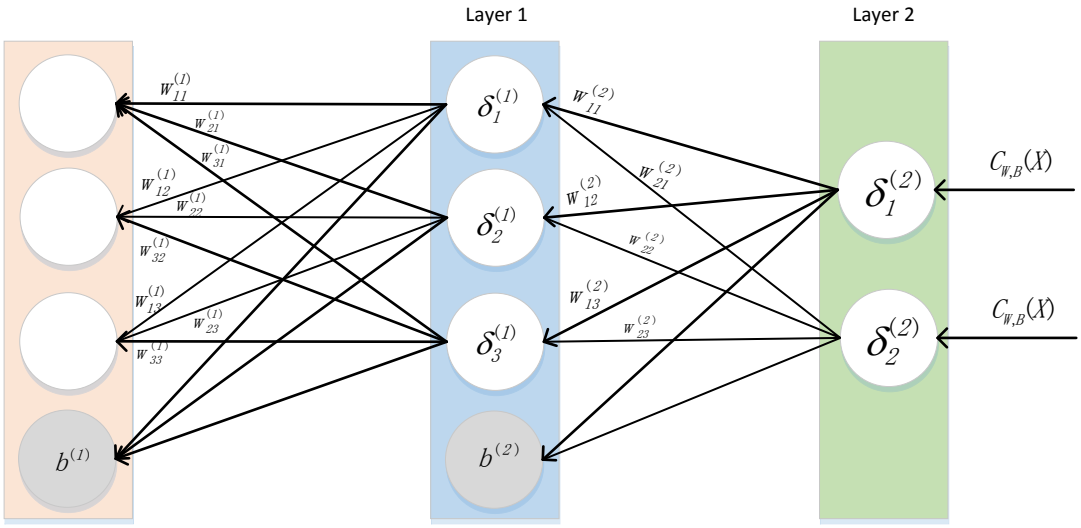
以 Quadratic Cost function 为例 $$\delta^{(L)} = \frac{\partial C}{\partial A^{(L)}}\frac{\partial A^{(L)}}{\partial Z^{(L)}} = \frac{\partial\frac{1}{2}\left\|Y - A^{(\mathrm{L})}\right\|^2}{\partial A^{(L)}}\frac{\partial f(Z^{(L)})}{\partial Z^{(L)}} = \left(A^{(\mathrm{L})} - Y\right)\cdot f`(Z^{(L)})$$	公式 3
以 Cross-entropy cost function 为例 $$\delta^{(L)} = \frac{\partial C}{\partial A^{(L)}}\frac{\partial A^{(L)}}{\partial Z^{(L)}} = \frac{\partial[-Y\ln A^{(L)} - (1-Y)\ln\left(1-A^{(L)}\right)]}{\partial A^{(L)}}\frac{\partial f(Z^{(L)})}{\partial Z^{(L)}}$$ $$= \left(-\frac{Y}{A^{(L)}} - \frac{(1-Y)}{(1-A^{(L)})}\right)\cdot f`\left(Z^{(L)}\right) = \frac{A^{(L)} - Y}{A^{(L)}(1-A^{(L)})}f`\left(Z^{(L)}\right)$$ $$= \frac{f(Z^{(L)}) - Y}{f(Z^{(L)})(1-f(Z^{(L)}))}f`\left(Z^{(L)}\right)$$ 假设激活函数是 sigmod 函数时 $f(x) = \mathrm{sigmod}(x)$,那么$f`(x) = f(x)(1-f(x))$ $$f`\left(Z^{(L)}\right) = f(Z^{(L)})\left(1-f(Z^{(L)})\right)$$ 将上面的公式带入后， $$\delta^{(L)} = \frac{f(Z^{(L)}) - Y}{f(Z^{(L)})(1-f(Z^{(L)}))}f`\left(Z^{(L)}\right) = f\left(Z^{(L)}\right) - Y = \mathrm{A}^{(\mathrm{L})} - Y$$	公式 4

在 sigmod 函数中，如果$x \in \pm\infty$，$f`(x)$趋于 0。这样当使用 Quadratic cost function 时候会出现$\delta^{(L)} \approx 0$。这样就不会有残差会传。而使用 Cross-entropy cost function 就不会出现神经元 saturate 的问题，因为$\delta^{(L)} = \mathrm{A}^{(\mathrm{L})} - Y$。所以当使用 Cross-entropy cost function，以及 sigmod 函数时候，神经元 saturate 的问题就基本解决。

残差后向传导

$$\delta^{(l)} = \frac{\partial C}{\partial Z^{(l)}} = \frac{\partial C}{\partial Z^{(l+1)}}\cdot\frac{\partial Z^{(l+1)}}{\partial Z^{(l)}} = \delta^{(l+1)}\frac{\partial\left(\mathrm{W}^{(l+1)}\mathrm{A}^{(l)} + B^{(l+1)}\right)}{\partial Z^{(l)}}$$ $$= \delta^{(l+1)}\frac{\partial\left(\mathrm{W}^{(l+1)}f\left(\mathrm{Z}^{(l)}\right) + B^{(l+1)}\right)}{\partial Z^{(l)}} = \delta^{(l+1)}\mathrm{W}^{(l+1)}f`\left(\mathrm{Z}^{(l)}\right)$$	公式 5

反向传播示例



假设 Quadratic cost function + sigmod 函数

Layer 2 层：残差计算

$$\delta_1^{(2)} = \left(a_1^{(2)} - y\right) \cdot f`(z_1^{(2)})$$

$$\delta_2^{(2)} = \left(a_2^{(2)} - y\right) \cdot f`(z_2^{(2)})$$

Layer 1 层：残差计算

$$\delta_1^{(1)} = \left(w_{11}^{(2)}\delta_1^{(2)} + w_{21}^{(2)}\delta_2^{(2)}\right) \cdot f`(z_1^{(1)})$$

$$\delta_2^{(1)} = \left(w_{12}^{(2)}\delta_1^{(2)} + w_{22}^{(2)}\delta_2^{(2)}\right) \cdot f`(z_2^{(1)})$$

$$\delta_3^{(1)} = \left(w_{13}^{(2)}\delta_1^{(2)} + w_{23}^{(2)}\delta_2^{(2)}\right) \cdot f`(z_3^{(1)})$$

********残差从哪里来，回哪里去，多个残差加和********

$\delta^{(l)} = \delta^{(l+1)}\mathrm{W}^{(l+1)}f`\left(\mathrm{Z}^{(l)}\right)$想表达的含义一定要记住，后面 CNN BP 时也要用到

Layer 2 层：参数更新

$$\frac{\partial C}{\partial w_{12}^{(2)}} = \delta_1^{(2)} a_2^{(1)}$$

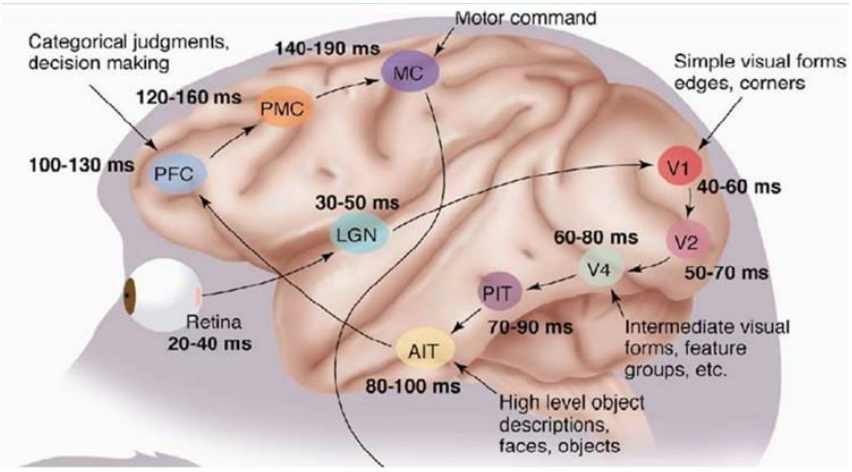
********权重的导数=神经元的残差乘以连接激活值********

$\frac{\partial C}{\partial w_{ij}^{(l)}} = \delta_i^{(l)} a_j^{(l-1)}$想表达的含义一定要记住，后面 CNN BP 时也要用到

3.Convolution Neuron Network

3.1 Introduction

神经科学证明人脑抽象能力如下图。下图中图片从眼睛视网膜沿着箭头一直到 Motor command，可以看到人脑在对图片的一层层的抽象过程。感兴趣的读者可以自行 google 相关文章。



卷积神经网络定义：

*"**Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.**"* – from "Deep Learning", Ian Goodfellow, Yoshua Bengio, Aaron Courville.

卷积定义：对于两个函数$x(a), w(a)$，那么这两个函数的卷积为

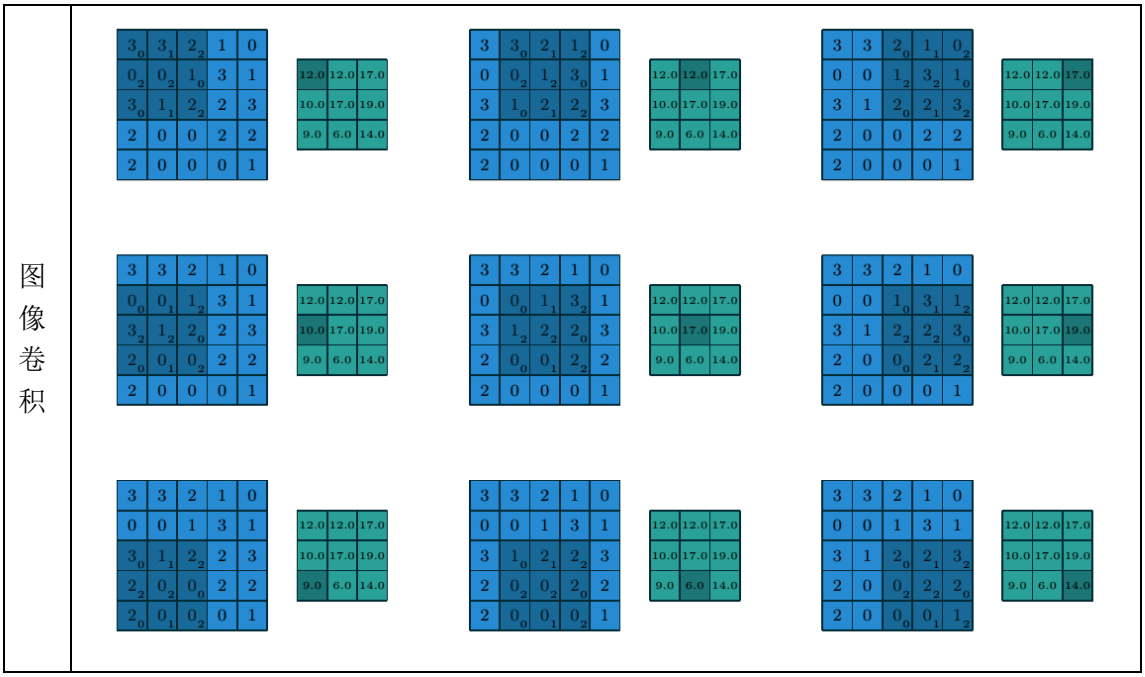
$$s(t) = \int x(a)w(t-a)da$$

离散形式为：

$$s(t) = \sum_{a=-\infty}^{a=\infty} x(a)w(t-a)$$

卷积基本操作示例：

卷积核	0 1 2 2 2 0 0 1 2

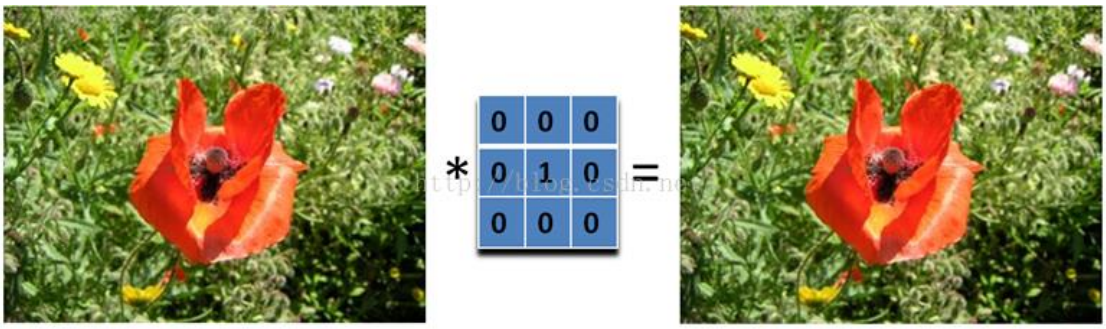


3.2 CNN in Detail

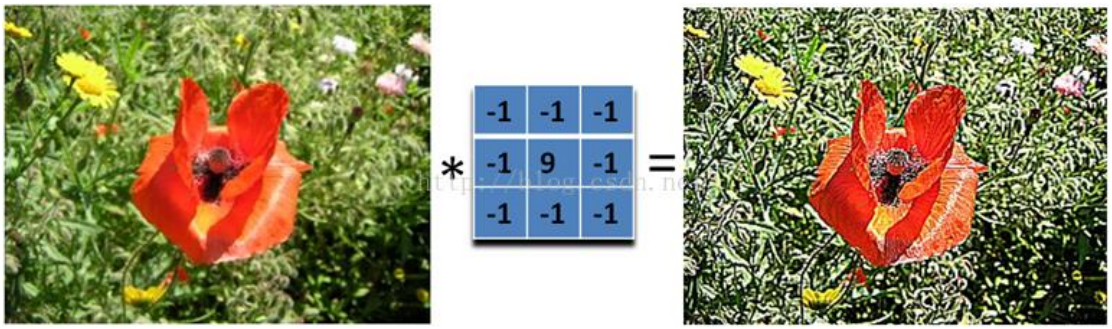
卷积在图像处理上的应用。线性滤波在图像处理中是一些非常基本方法，允许对图像进行处理，产生不同效果。一般情况要构造一个滤波器矩阵（也称是卷积核）。然后，对于图像的每一个像素点，计算它的邻域像素和滤波器矩阵的对应元素的乘积，然后加起来，作为该像素位置的值。这样就完成了滤波过程。

卷积核能做哪些图像处理。这部分内容大部分拷贝自博客
（http://blog.csdn.net/zouxy09/article/details/49080029**），这个博客的原文在**
（http://lodev.org/cgtutor/filtering.html）。里面有大量的源码可以试试。

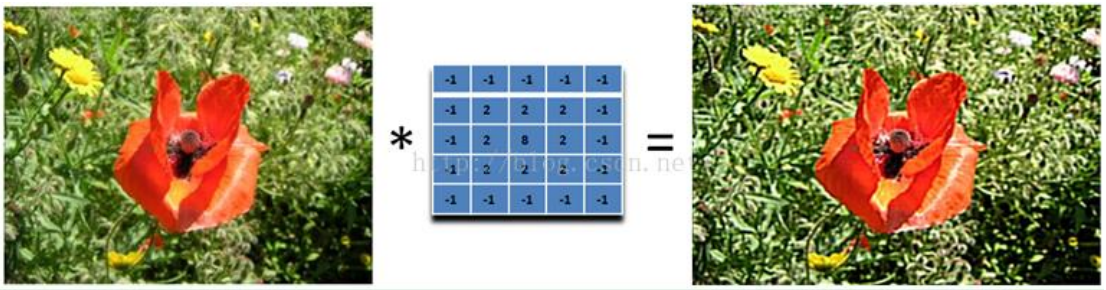
(1) 不进行处理



(2) 图像锐化

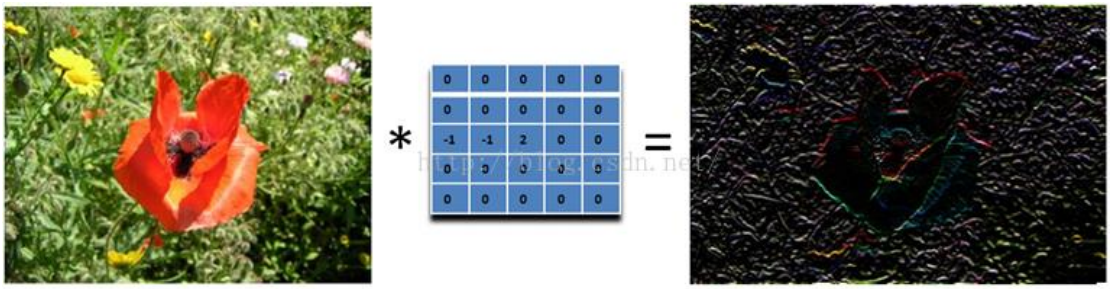


可以加大核区域，得到更加细腻的锐化效果

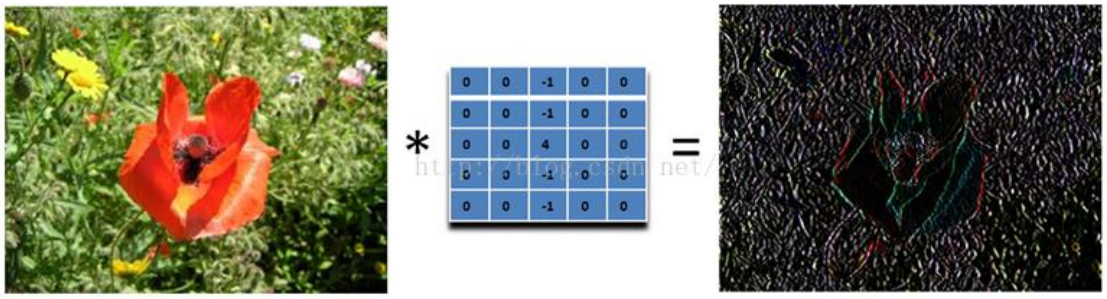


(3) 边缘检测

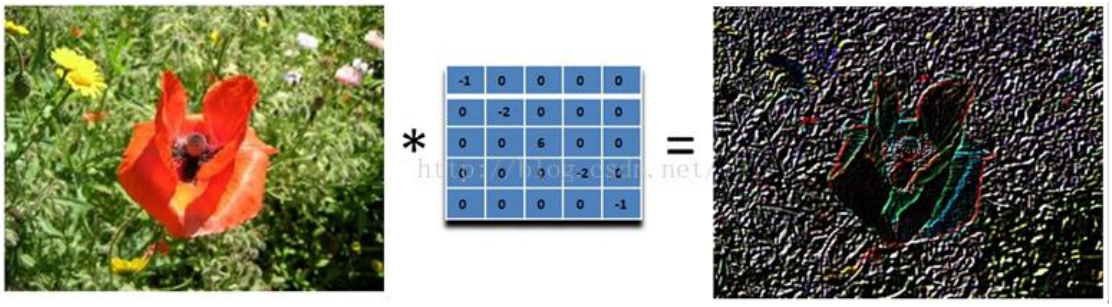
水平边缘检测



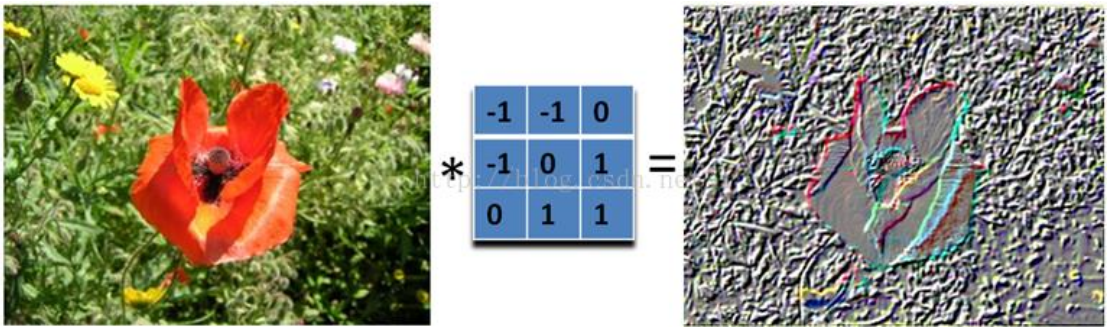
竖直边缘检测



45 度边缘检测

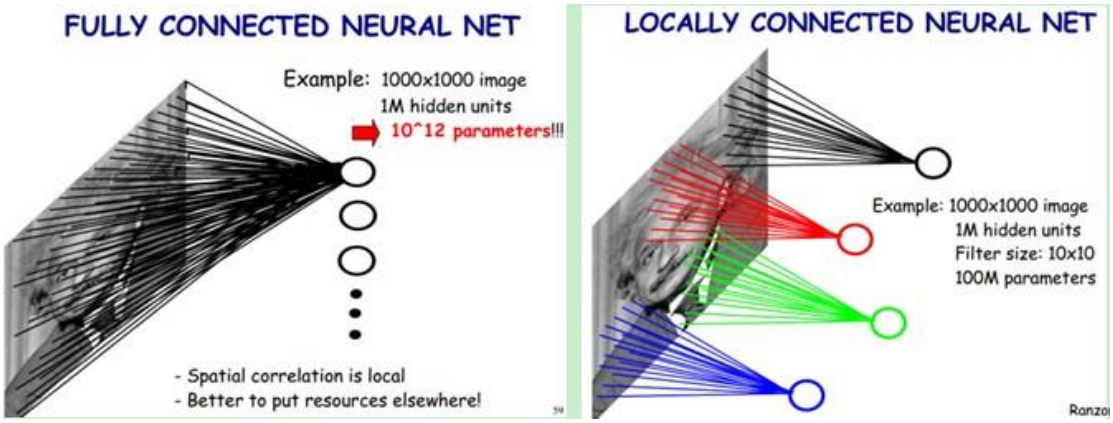


(4) 浮雕效果



3.2.1 Convolutional Layer

(1) 全连接(full connected) v.s. 局部感知(local receptive field)



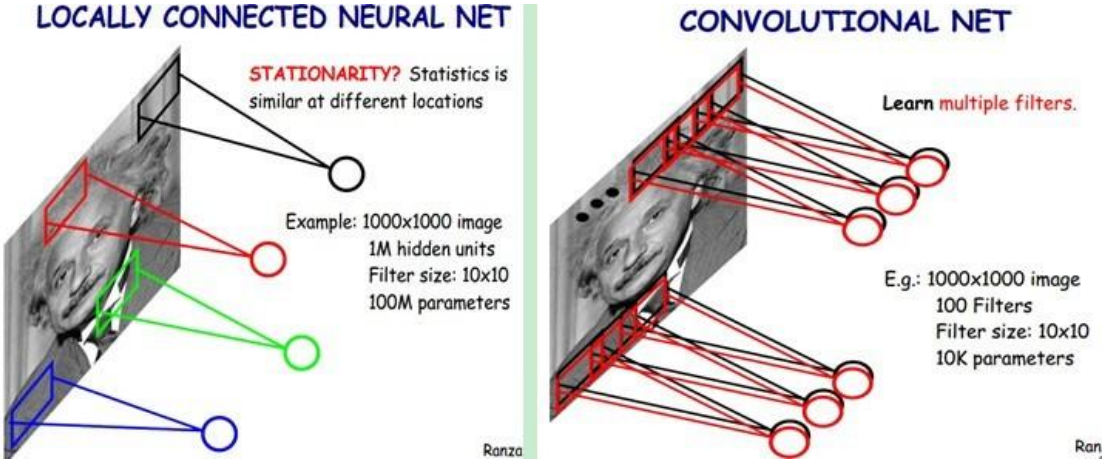
一般认为人对外界的认知是从局部到全局的，而图像的空间联系也是局部的像素联系较为紧密，而距离较远的像素相关性则较弱。因而，每个神经元其实没有必要对全局图像进行感知，只需要对局部进行感知，然后在更高层将局部的信息综合起来就得到了全局的信息。网络部分连通的思想，也是受启发于生物学里面的视觉系统结构。视觉皮层的神经元就是局部接受信息的（即这些神经元只响应某些特定区域的刺激）

上图中，原始图像是 1000*1000，隐藏层假设有 10^6 个神经元。全连接个数 1000*1000*10^6=10^12。而假设卷积核（滤波器）的大小 10*10，局部感知链接 10*10*10^6=10^8

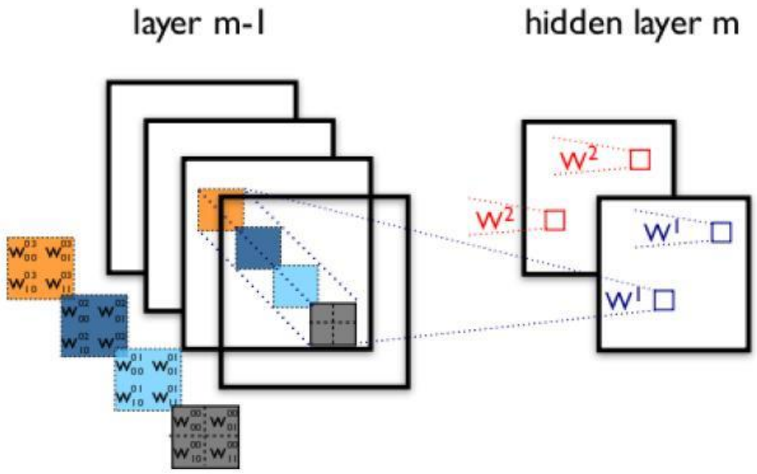
(2) 权重共享(weight sharing or parameter sharing)

上面的局部链接中，如果隐藏层的每个神经元都对应 100 个参数（因为感受区域大小为 10*10），那么参数的个数为 100*10^6=10^8。权重共享的含义是这 100 个参数都是一样的，那么这一个隐藏层就只有 100 个参数。

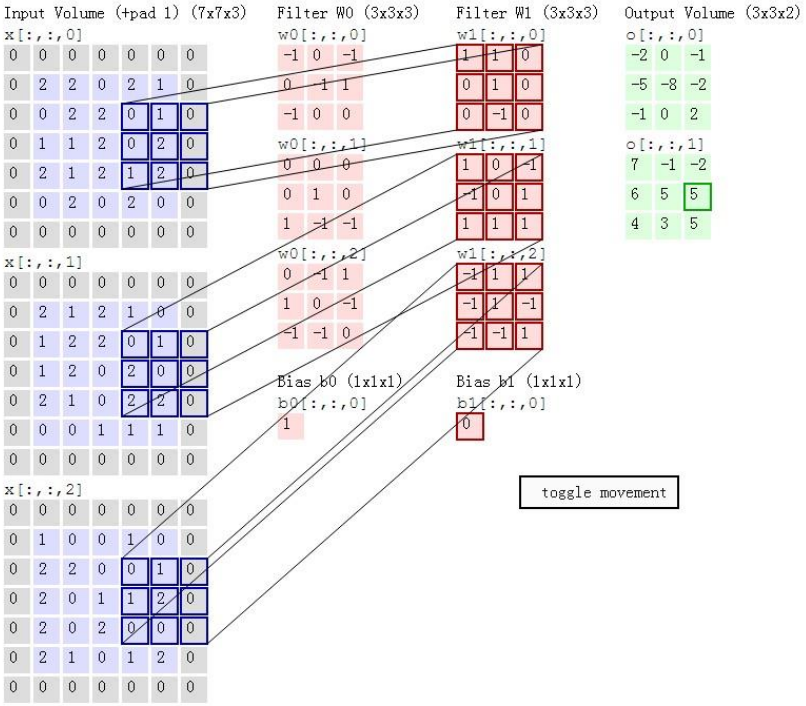
(3) 多卷积核



(4) 多通道多卷积核



上面这段博客大部分摘自(http://blog.csdn.net/stdcoutzyx/article/details/41596663)
多通道卷积计算的动画可以参看如下链接：http://cs231n.github.io/convolutional-networks/



(5) Stride(步长)，Zero-padding(零补全)

Stride(s): 滑动窗口每次滑动的步长。s越大，那么卷积次数越少，效率越高。但特征也更稀疏，某些图片特征可能不能被正常检测到。一般情况，在开始的 CNN 结构中$s = 1$，后面可以逐渐加大s值。

Zero-padding(p): 在图片边缘增加零，以保证边缘特征可以获得。如果假设输入图像 size 为i，卷积核大小为k，做步长为s，padding 为p的卷积输出图像 size 大小为o。它们之间的关系为 $o = \frac{i-k+2p}{s} + 1$

卷积操作示意图：

A. 任意 Padding 后卷积

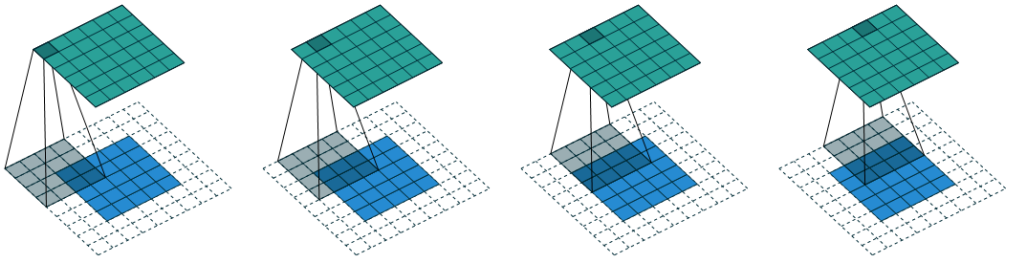


Figure 2.2: (Arbitrary padding, no strides) Convolving a 4×4 kernel over a 5×5 input padded with a 2×2 border of zeros using unit strides (i.e., $i = 5$, $k = 4$, $s = 1$ and $p = 2$).

B．Half (same) padding 后卷积：输入和输出的 size 一致

当$s = 1$，如果$p = \frac{k-1}{2}$时，那么$o = \frac{i-k+2p}{s} + 1 = i - k + 2\frac{k-1}{2} + 1 = i$

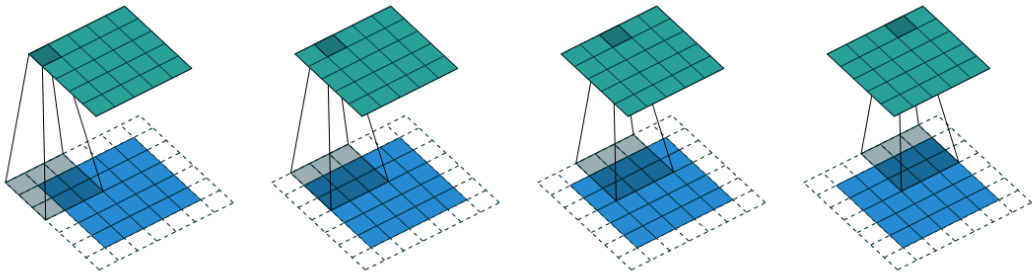


Figure 2.3: (Half padding, no strides) Convolving a 3×3 kernel over a 5×5 input using half padding and unit strides (i.e., $i = 5$, $k = 3$, $s = 1$ and $p = 1$).

C．Full padding 后卷积：$o = i + (k - 1)$

当$p = k - 1$时，$o = i + (k - 1)$

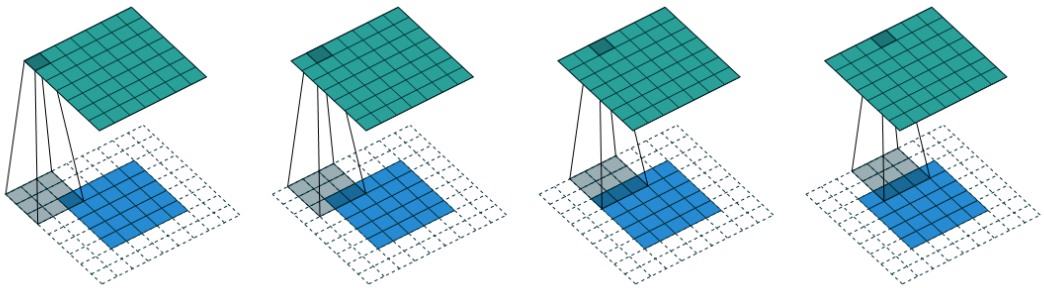
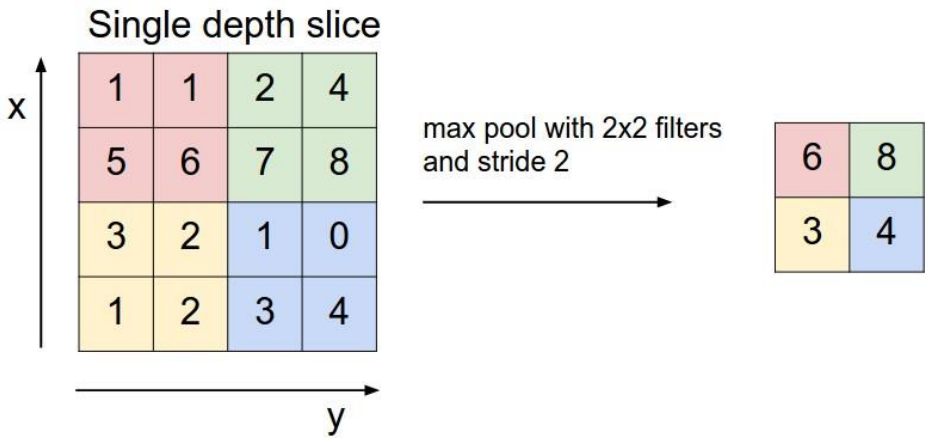


Figure 2.4: (Full padding, no strides) Convolving a 3×3 kernel over a 5×5 input using full padding and unit strides (i.e., $i = 5$, $k = 3$, $s = 1$ and $p = 2$).

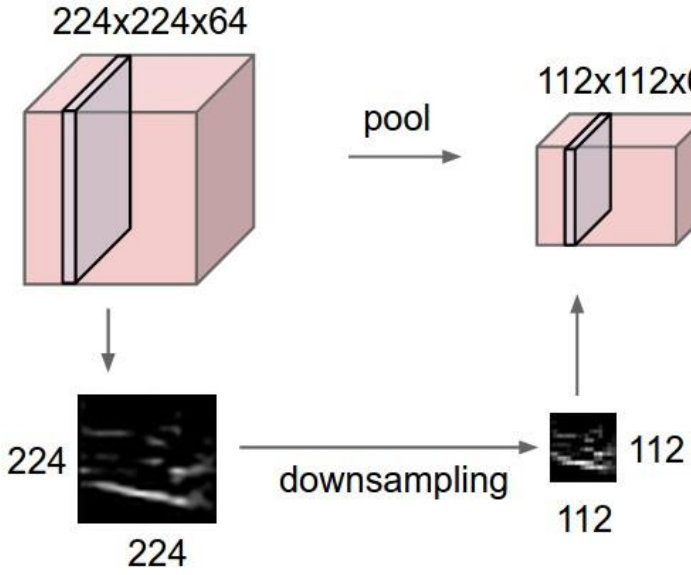
3.2.2 Pooling Layer

下采样即是对原图像的区域进行局部采样。采样方式有很多种，例如 Max-pooling，Mean-pooling，Square-pooling 等等。下面基本上以 max-pooling 为例：

采样的数学操作如下：

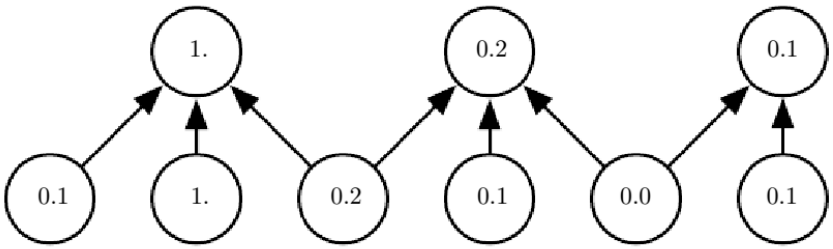


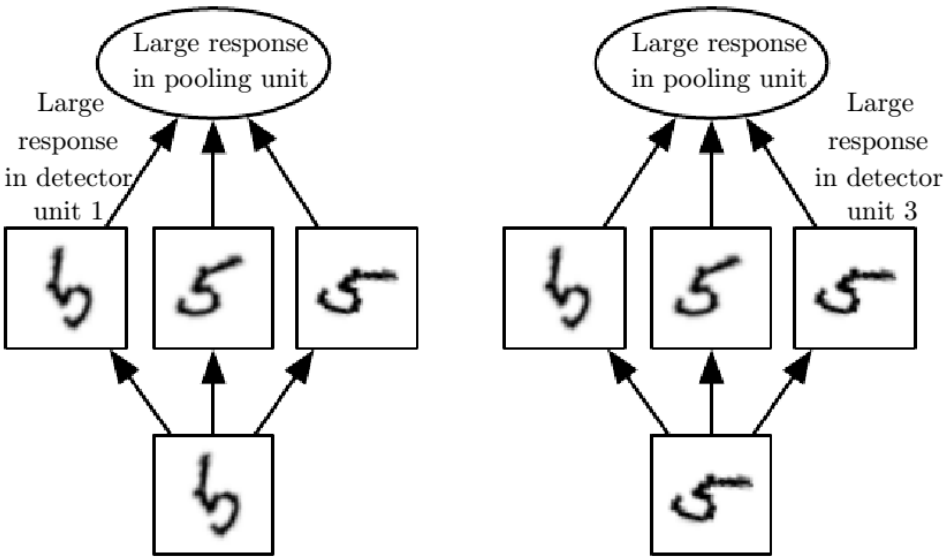
下采样在图片采样示例：



Downsampling Layer 作用

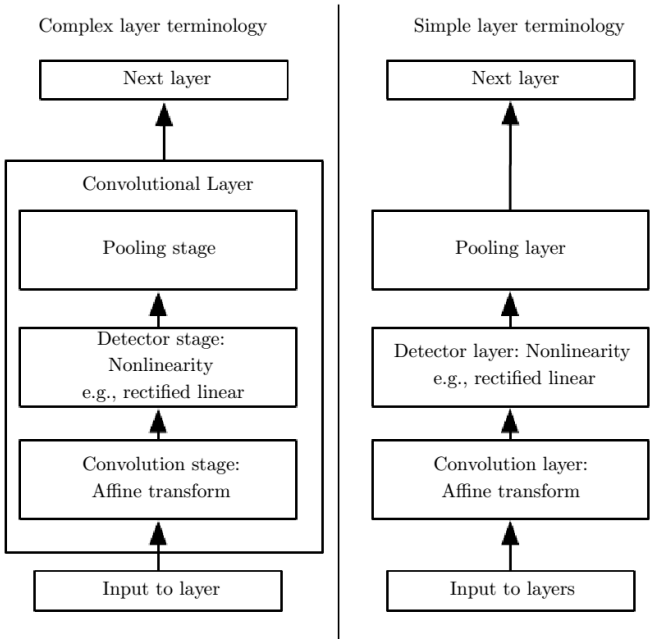
1. 减少网络参数，加快训练速度；
2. 增加识别鲁棒性，对图片中物体位置，角度改变不敏感；以 maxpooling 为例



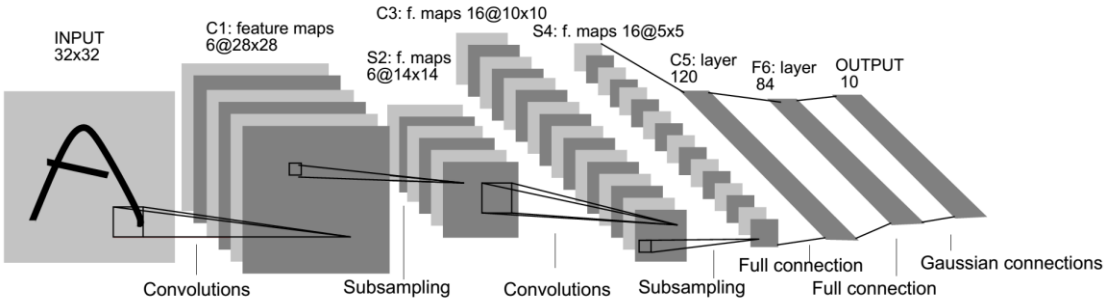


3.3 LeNet in Action

CNN 网络结构基本框架如下



下面以 LeNet 为例介绍 CNN 网络具体结构以及训练方法



“*Gradient-Based Learning Applied to Document Recognition*”-Yann LeCun98

计算一下各层之间的参数个数，以及连接数

A. Input ←→ C1: feature map: 6, kernel_size: 5*5

   参数数：(5*5+1)*6=156 连接数：(5*5+1)*6*28*28=122,304

B. C1 ←→ S2: 2*2 的 subsampling

   参数数：(1+1)*6 = 12 连接数：(2*2+1)*14*14*6=5880

C. S2 ←→ C3:C3 层中有 16 个 feature map，kernel_size:5*5。C3 层的 16 个卷积核和 S2 层的链接方式如下

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

   参数个数：

$$1516\begin{cases}(5*5*3+1)*6\\(5*5*4+1)*9\\(5*5*6+1)*1\end{cases}$$

   链接数：

$$151600\begin{cases}(5*5*3+1)*6*10*10\\(5*5*4+1)*9*10*10\\(5*5*6+1)*1*10*10\end{cases}$$

D. C3 ←→ S4:参数个数 12，链接数(2*2+1)*16*5*5=2000

E. S4 ←→ C5: 全连接 C5 层（120 个 neuron）

   参数个数：5*5*16(S4 层输入)*120 + 120*1(C5 层的 bias 数目)=48120

   链接数：48120(全连接)

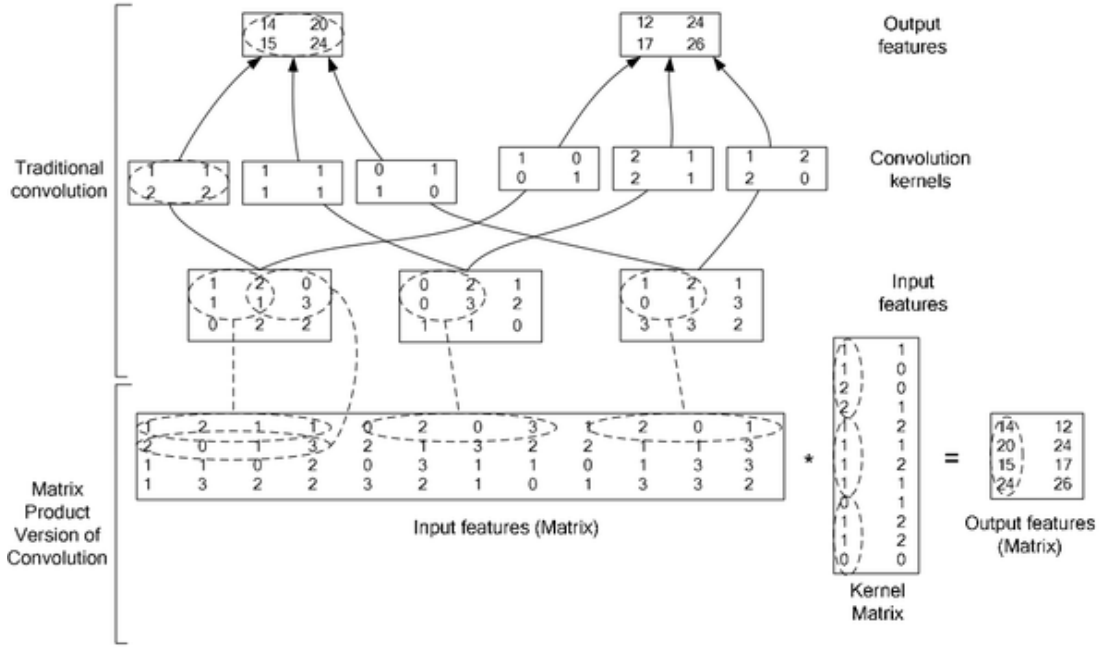
F. C5 ←→ F6: 全连接(120+1)*84

3.3.1 CNN Forward Implementation

CNN 前馈神经网络重点关键性问题是：如何快速实现卷积操作。假如假设 Input 维度为 C*W*H，卷积的 Kernel 为 C*K*K，卷积 Kernel 的个数为 M 个。那么卷积的伪代码如下

```
for w in 1..W
  for h in 1..H
    for x in 1..K
      for y in 1..K
        for m in 1..M
          for c in 1..C
            output(w, h, m) += input(w+x, h+y, c) * filter(m, x, y, c)
```

一种思路是通过某种运算将滑动的卷积操作变成矩阵操作，利用已有的矩阵运算库(如 Blas，Numpy 等)加速运算。im2col 方法，基本想法是多维图片，以及卷积核转化成矩阵，然后进行矩阵乘积运算进行加速。使用 im2col 如何进行卷积示意图如下。



“*Im2col: High Performance Convolutional Neural Networks for Document Processing*”

优点：速度快，可以利用已有矩阵运算 lib 加速运算；

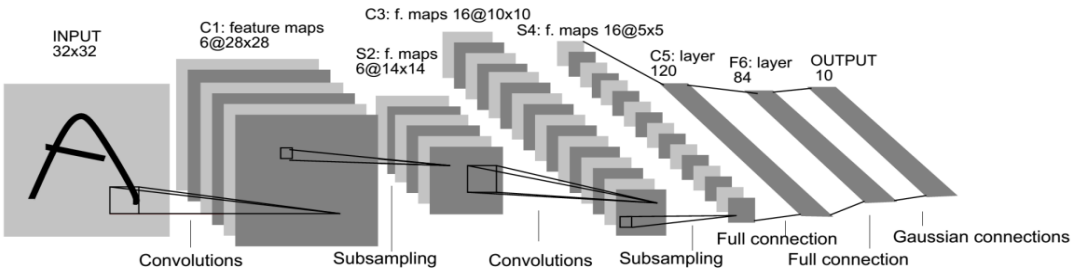
缺点：占用内存变大，内存换速度方法。

更加快速的卷积计算方法

Alex Krizhevsky - Cuda-convnet2： 原理使用 FFT。时域的卷积和频域的乘法是等价的，同时时域的乘法和频域的卷积也是等价的。将图像和 Kernel 都先变换到频域，然后在频域中相乘，即可以得到时域的卷积。实际计算时候 Kernel 只要一次变换即可。

Python 卷积 http://blog.5long.me/2016/algorithms-on-cuda-fft-convolution/

3.3.2 CNN Backward Implementation



问题 1. 残差从 C5 层向 S4 层传播 $\delta^{(l)} = \delta^{(l+1)}\mathrm{W}^{(l+1)}f`\left(\mathrm{Z}^{(l)}\right)$；

问题 2. 残差从 S4 层向 C3 层传播 $\delta_j^{(l)} = \beta_j^{(l+1)}(f`(z_j^{(l)})\odot\mathrm{up}(\delta_j^{(l+1)}))$；

问题 3. 残差从 C3 层向 S2 层传播 $\delta_j^{(l)} = f`(z_j^{(l)})\odot\mathrm{conv}(\delta_j^{(l+1)}, \mathrm{rot180}\left(w_j^{(l+1)}\right), \mathrm{'full'})$；

问题 4. 在 Convolution，以及 Pooling 层中，参数更新 $\frac{\partial C}{\partial w_{ij}^{(l)}} = \mathbf{conv}(\mathrm{A}_i^{l-1}, \delta_j^{(l)}, \mathrm{'valid'})$。

问题 1. BW from the full connected layer to the pooling layer

由于 S4 和 C5 是全连接的，所以使用标准 BP 算法直接将残差从 C5 层传到 S4 层。

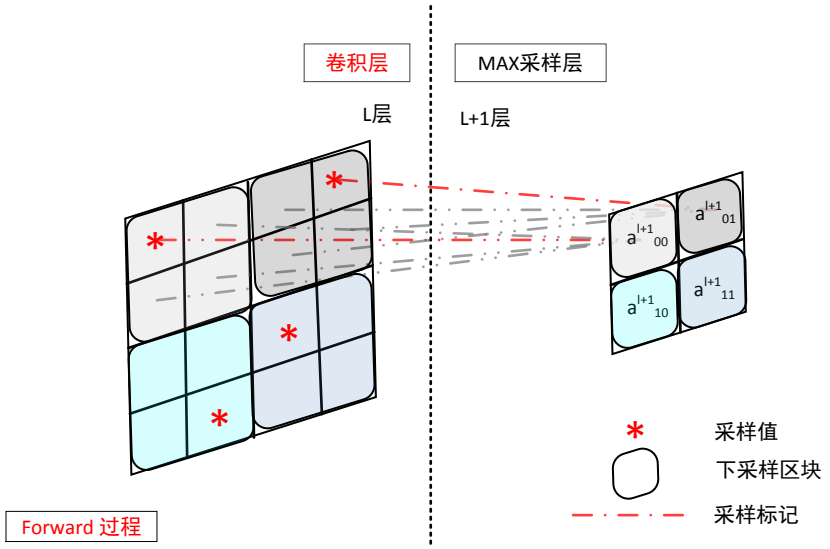
问题 2. BW from the Pooling layer (S4) to the Convolutional layer (C3)

重要的公式：$\delta^{(l)} = \delta^{(l+1)} W^{(l+1)} f`(Z^{(l)})$

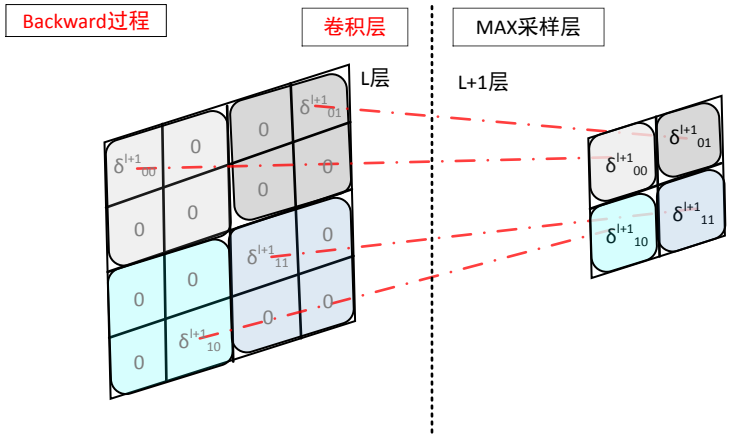
*********残差从哪里来，残差回哪里去，多个残差加和*********

如何将下采样层的残差向后传导

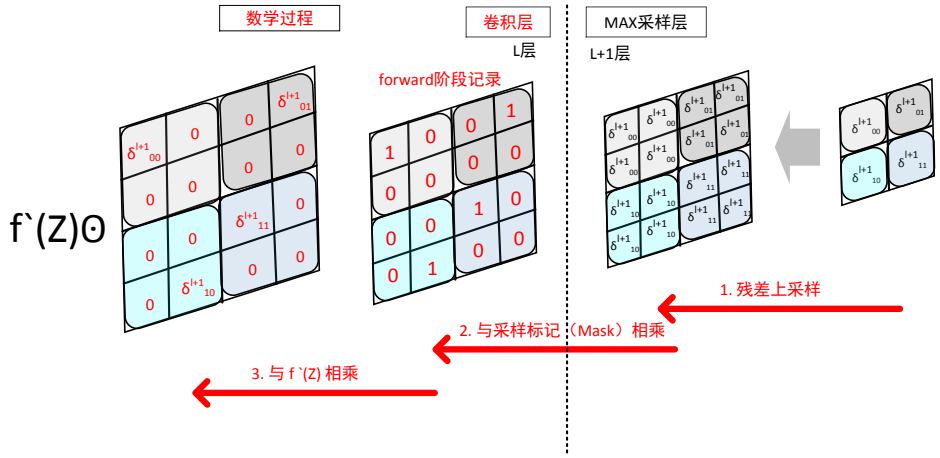
假设前向传播时候，特征采样如下



由$\delta^{(l)} = \delta^{(l+1)} W^{(l+1)} f`(Z^{(l)})$以及残差从哪里来，残差回哪里去，多个残差加和，我们期望将残差传导到相应的位置上。因为 FW 时误差只会从采样标记位置 FW 到 L+1 层，那么 BW 时，残差也只回传到 L 层的采样位置。



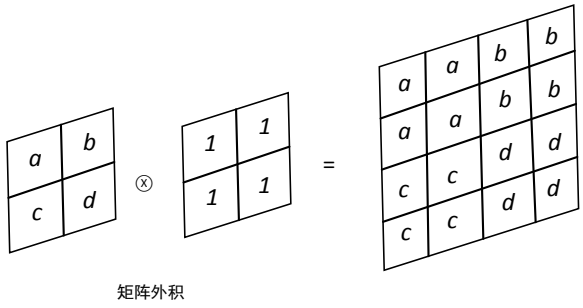
上面的过程可以转换成下面的数学过程



数学表达式：$\delta_j^{(l)} = \beta_j^{(l+1)}(f`(z_j^{(l)}) \odot \mathrm{up}(\delta_j^{(l+1)}))$

⊙ element-wise product; up(.) 上采样操作;

如何实现上采样操作：矩阵外积



矩阵外积

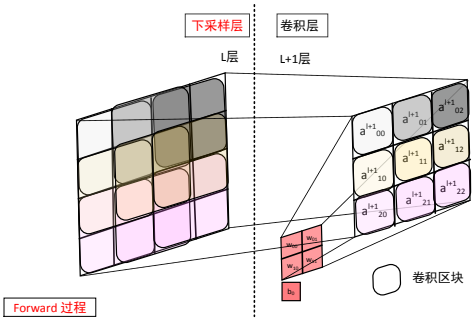
问题 3. BW from the Convolution layer to Pooling layer

重要的公式：$\delta^{(l)} = \delta^{(l+1)} \mathrm{W}^{(l+1)} f`(\mathrm{Z}^{(l)})$

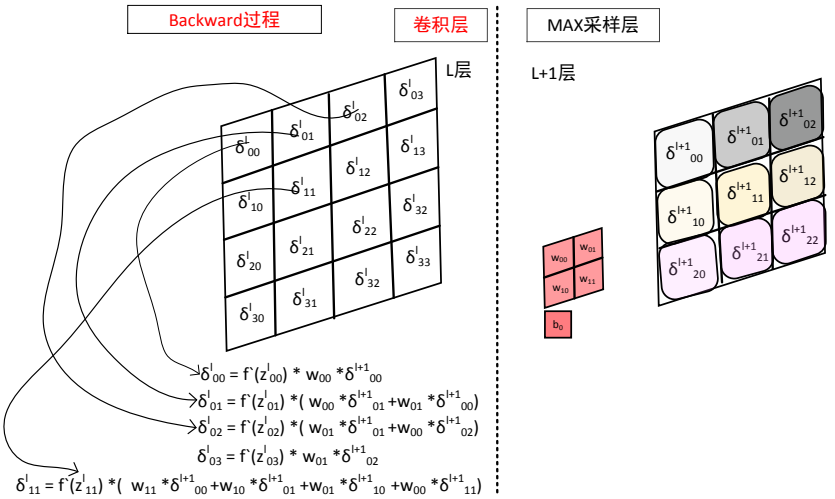
********残差从哪里来，残差回哪里去，多个残差加和********

如何将卷积层的残差向后传导

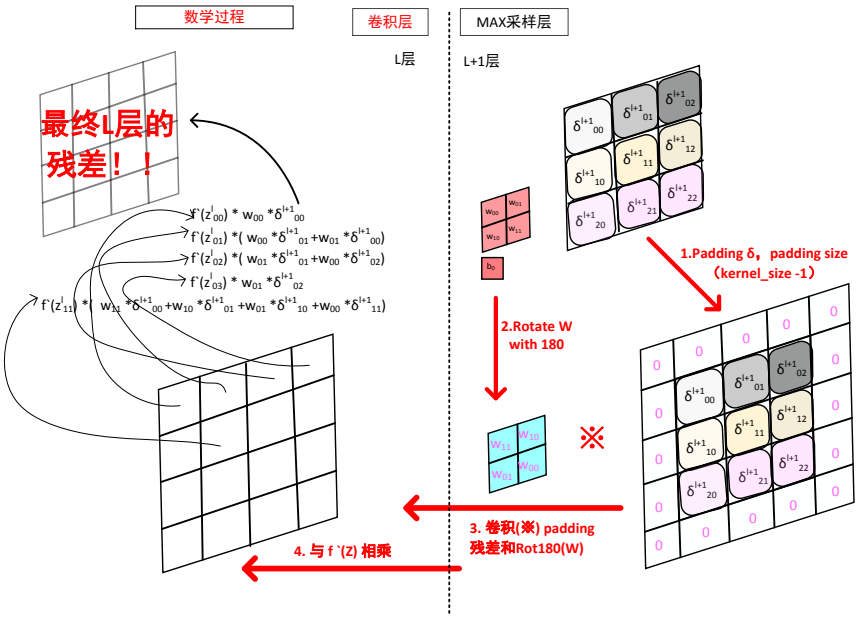
假设前向网络如下



由$\delta^{(l)}=\delta^{(l+1)}\mathrm{W}^{(l+1)}f`\left(\mathrm{Z}^{(l)}\right)$以及残差从哪里来，残差回哪里去，多个残差加和，我们期望将残差传导到相应的位置上。因为 FW 时误差只会从采样标记位置 FW 到 L+1 层，那么 BW 时，残差也只回传到 L 层的采样位置。



数学表达



数学公式：$\delta_j^{(l)}=f`(z_j^{(l)})\odot\mathrm{conv}(\delta_j^{(l+1)},\mathrm{rot180}\left(w_j^{(l+1)}\right),'\mathrm{full}')$

对比经典 BP：$\delta^{(l)}=\delta^{(l+1)}\mathrm{W}^{(l+1)}f`\left(\mathrm{Z}^{(l)}\right)$

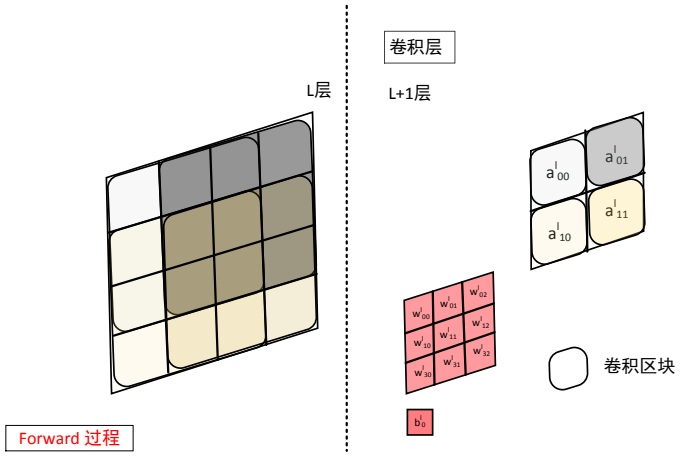
注意使用：full-padding

问 题 4. Parameter update in the Pooling (S2,S4) and Convolution(C1,C3) layer

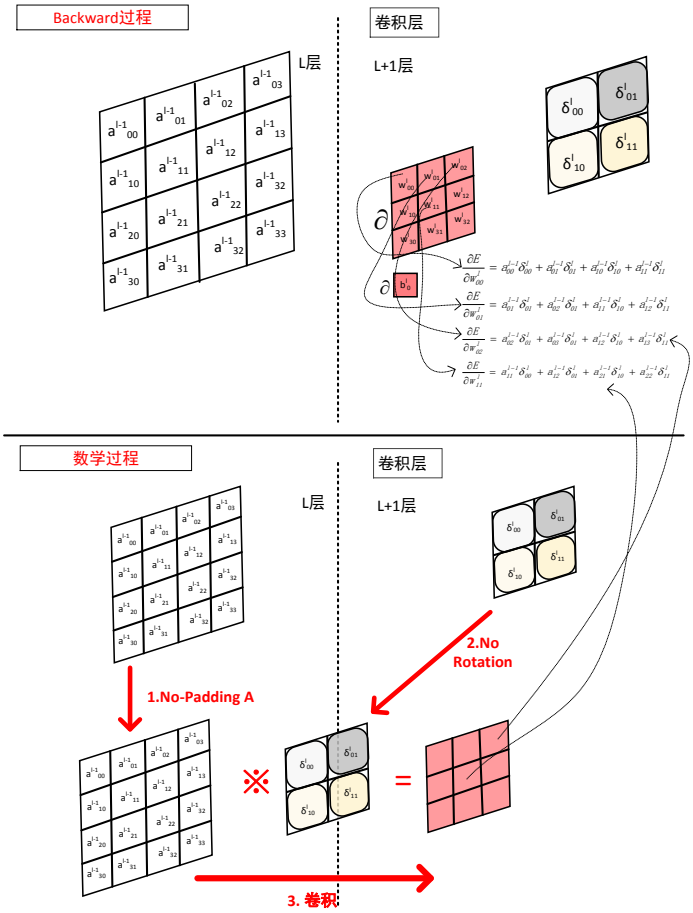
重要公式：$\frac{\partial C}{\partial w_{ij}^{(l)}}=\delta_i^{(l)}a_j^{(l-1)}$　权重的导数=神经元的残差乘以连接激活值

如何更新卷积层的参数

假设前向传播



BW 过程和数学过程



数学公式：$\frac{\partial C}{\partial w_{ij}^{(l)}} = \mathbf{conv}(\mathrm{A}_i^{l-1}, \delta_j^{(l)}, \text{'valid'})$

对比经典 BP：$\frac{\partial C}{\partial w_{ij}^{(l)}} = \delta_i^{(l)} a_j^{(l-1)}$

注意：不使用 padding

4. Conclusion

CNN 优点：

1. 通过 local receptive field，weight sharing，pooling 减小了网络参数，学习效率高；
2. CNN 天然带有防止 overfitting 的效果，可以不使用 Dropout 技术；
3. 不同 kernel 之间一般是独立的，所以适合进行并行，或者分布式计算；
4. 使用 Rectified Linear Unit 代替 sigmod 函数提高训练速度；
5. 参数初始化要进行优化，防止 slowdown

有些观点认为 DNN 使得我们的学习模式从 “how to learn” 到 “what to learn”。早期的时候都是算法人员去设计特征，告诉模型如何通过这些特征去学习样本，类似与告诉模型怎么取学习，带有人工的经验在里面。而 DNN 其实摒弃了这个过程，我只是告诉模型我想要学习到什么。而如何去学习有模型自己完成。

5. References

Books and articles：

1. UFLDL http://deeplearning.stanford.edu/wiki/index.php/Neural_Networks
2. Michael Nielsen, Neural network and deep learning: *http://neuralnetworksanddeeplearning.com/*
3. FeiFei Li, CS231: Convolutional Neural network for visual recognition：*http://cs231n.github.io/convolutional-networks/*
4. Ian Goodfellow, Yoshua Bengio, Aaron Couville, *Deep learning*
5. Vincent Dumoulin and Francesco Visin, Univercity of Montreal, *A guide to convolution arithmetic for deep learning*
6. Jake Bouvrie, Massachusetts Institue of technology, *Notes on Convolutional Neuron Networks*
7. Matthew D. Zeiler and Rob Fergus. *Visualizing and Understanding Convolutional Networks*
8. *Im2col: High Performance Convolutional Neural Networks for Document Processing*

其他博客：

1. http://blog.csdn.net/stdcoutzyx/article/details/41596663
2. http://blog.5long.me/2016/algorithms-on-cuda-fft-convolution/
3. http://blog.csdn.net/zouxy09/article/details/49080029
4. https://github.com/Yangqing/caffe/wiki/Convolution-in-Caffe:-a-memo