卷积神经网络背后的数学原理

卫然

1. Preliminaries

假设读者已经对CNN、微积分和线性代数有了基本的了解，本文将省略大部分背景介绍而着重于CNN中的数学推导。

1.1. Notations. 我们用加粗的字母，如$\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$来表示Tensor。Tensor可以被理解为“矩阵”这一概念的延伸，它可以是i维的向量（order 1），可以是$i \times j$维的矩阵（order 2），还可以是$i \times j \times k$维的矩阵堆叠（order 3），甚至还能具备更高的order——比如feed到CNN的一个图像batch，它包含batch size、像素的(x, y)坐标、channel这四个信息，因此是一个order 4的Tensor。

我们用不加粗的小写字母，如x, y, z来表示标量；用不加粗的大写字母，如A, B, C来表示矩阵。如果标量z是标量x, y的函数，则$z = z(x, y)$对x和y的偏导数可以被记为$\partial z/\partial x$和$\partial z/\partial y$。为了随后的演算方便，我们引入如下的向量偏导数记号：假设z是一个标量，$\boldsymbol{x} \in \mathbb{R}^n$是一个$n$维列向量，则

$$\frac{\partial z}{\partial \boldsymbol{x}} := \left(\frac{\partial z}{\partial x_1}, \cdots, \frac{\partial z}{\partial x_n}\right)^T \tag{1.1}$$

这里T表示转置。如果$\boldsymbol{z} \in \mathbb{R}^m, \boldsymbol{x} \in \mathbb{R}^n$ 分别表示m维和n维的列向量，则

$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}^T} := \begin{pmatrix} \frac{\partial z_1}{\partial x_1} & \frac{\partial z_1}{\partial x_2} & \cdots & \frac{\partial z_1}{\partial x_n} \\ \frac{\partial z_2}{\partial x_1} & \frac{\partial z_2}{\partial x_2} & \cdots & \frac{\partial z_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial z_m}{\partial x_1} & \frac{\partial z_m}{\partial x_2} & \cdots & \frac{\partial z_m}{\partial x_n} \end{pmatrix} \tag{1.2}$$

1.2. Gradients in CNN. 对于任意的神经网络，我们用L^k表示网络的第k 层，用$\boldsymbol{x}^k$表示第k层的输入，以及用$\boldsymbol{w}^k$表示第k层的参数。那么一个n层CNN的输入输出流可以表示为

$$\begin{array}{ccccccccccccccccc} & & \boldsymbol{w}^1 & & & & & & \boldsymbol{w}^k & & & & & & \boldsymbol{w}^n & & \\ & & \downarrow & & & & & & \downarrow & & & & & & \downarrow & & \\ \boldsymbol{x}^1 & \rightarrow & L^1 & \rightarrow & \boldsymbol{x}^2 & \rightarrow & \cdots & \rightarrow & \boldsymbol{x}^k & \rightarrow L^k & \rightarrow & \boldsymbol{x}^{k+1} & \rightarrow & \cdots & \rightarrow \boldsymbol{x}^n & \rightarrow L^n & \rightarrow z \end{array}$$

一般情况下，$\boldsymbol{x}^1$是需要处理的图像，而z 是损失（loss）。注意不要将这里的上标与幂次混淆。

训练网络的过程，其实就是通过梯度下降法去优化网络中所有参数的过程。第t次训练对第k层参数的更新可以表示为

$$(\boldsymbol{w}^k)_{t+1} \leftarrow (\boldsymbol{w}^k)_t - \alpha \frac{\partial z}{\partial (\boldsymbol{w}^k)_t}$$

这里α表示学习速率（learning rate）。因此，在每次训练中，我们都需要计算$\partial z/\partial \boldsymbol{w}^1, \cdots, \partial z/\partial \boldsymbol{w}^n$。

让我们先从网络的最后一层开始，因为这一层相对特殊，也比较简单。通常，L^n会是$\boldsymbol{x}^n$与真实值的平方误差（MSE）或是交叉熵（cross entropy），它实际上不含任何参数，于是我们可以直接求得$\partial z/\partial \boldsymbol{x}^n$。假设$\boldsymbol{x}^n$包含$l$个元素，由链式法则，$z$对$\boldsymbol{w}^{n-1}$中第$i$个元素的偏导数可以表示为

$$\frac{\partial z}{\partial w_i^{n-1}} = \sum_{j=1}^{l} \frac{\partial z}{\partial x_j^n} \cdot \frac{\partial x_j^n}{\partial w_i^{n-1}} \tag{1.3}$$

同时，z对$\boldsymbol{x}^{n-1}$中第m个元素的偏导数可以表示为

$$\frac{\partial z}{\partial x_m^{n-1}} = \sum_{j=1}^{l} \frac{\partial z}{\partial x_j^n} \cdot \frac{\partial x_j^n}{\partial x_m^{n-1}} \tag{1.4}$$

这样，我们便得到了$\partial z/\partial \boldsymbol{w}^{n-1}$和$\partial z/\partial \boldsymbol{x}^{n-1}$。从$\partial z/\partial \boldsymbol{x}^{n-1}$出发，重复(1.3)-(1.4)中的过程，我们便可通过链式法则求出z对任意参数的偏导数。

在实际应用中，我们肯定不会像(1.3)-(1.4)那样去一个一个地计算偏导数，那样不仅效率低下而且难以实现。我们会运用一些巧妙的向量、矩阵运算技巧，去得到$\partial z/\partial \boldsymbol{w}^k$和$\partial z/\partial \boldsymbol{x}^k$的漂亮表达式。下面，我们将分别给出convolutional layer（卷积层）、fully connected layer（全连接层）、pooling layer（池化层）和Relu（线性整流单元）的梯度计算公式及其推导。

2. Convolutional layer

2.1. Introduction. 卷积层无疑是CNN最核心也最复杂的部分，对卷积层的论述将会占用本文大部分的篇幅。在卷积层中，卷积核（kernel）通过扫描特征面以进一步提取特征。卷积核与特征面的相互作用是通过矩阵卷积来实现的。矩阵卷积的计算过程可以如下表述：假设矩阵A是一张特征面，矩阵K是一个卷积核，

$$A = \begin{pmatrix} 1 & 3 & 1 & 4 \\ 2 & 8 & 5 & 1 \\ 3 & 2 & 0 & 3 \end{pmatrix} \qquad K = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix}$$

在padding=valid，stride=$(1,1)$的条件下，我们首先将K放置在A的左上角，使得K与$[[1,3],[2,8]]$对齐，计算对应位置乘积之和

$$1 \times 1 + 0 \times 3 + (-1) \times 2 + 1 \times 8 = 7$$

接着将K右移stride[0]个位置，使得K与$[[3,1],[8,5]]$对齐，计算对应位置乘积之和

$$1 \times 3 + 0 \times 0 + (-1) \times 8 + 1 \times 5 = 0$$

继续将K右移并计算对应位置乘积之和如上，直至K 的右端与A的右端重合；此时令K回到A的最左端，然后先下移stride[1]个位置，再向右对A扫描直至最右。重复上述过程直到K运动到A的右下角。最终的矩阵卷积结果，可以根据每次运算时K相对于A 的位置写为一个矩阵：

$$A * K = \begin{pmatrix} 7 & 0 & -3 \\ 1 & 6 & 8 \end{pmatrix} \tag{2.1}$$

一般来说，卷积核的尺寸越小，卷积层所能提取的特征就越"细"，同时也可能越抽象。比较流行的核尺寸大小是3×3。而卷积核的参数决定了卷积层能学习到什么样的特征。比如著名的Sobel operator：

$$G = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

它可以用来检测物体或图像之间的水平分界线，而G^T则可以用来检测垂直分界线。将不同的卷积核配合使用，可以学习到多种多样的特征。例如在一个判断图像是否是动物的CNN中，有的卷积核负责检测"头部"，有的卷积核负责检测"四肢"，还有的卷积核负责检测"毛发"。

接下来，我们要改写(2.1)式，为之后的求导做准备。对于(2.1)中的矩阵A，如果运行一下matlab或
octave中的im2col$(A,[2,2])$语句，我们会得到另一个矩阵

$$B = \begin{pmatrix} 1 & 2 & 3 & 8 & 1 & 5 \\ 2 & 3 & 8 & 2 & 5 & 0 \\ 3 & 8 & 1 & 5 & 4 & 1 \\ 8 & 2 & 5 & 0 & 1 & 3 \end{pmatrix}$$

矩阵B的每一列，实际上都由"矩阵A与一个2×2的矩阵做矩阵卷积时，每次参与相乘求和运算"的四个数组成。只不过与矩阵卷积所不同的是，在im2col中，这个2×2矩阵的扫描

顺序是先垂直后水平的。如果再把(2.1)式中矩阵K的所有列向量合并为向量$[1,-1,0,1]^T$，我们有

$$B^T \times [1,-1,0,1] = [7,1,0,6,-3,8]^T \tag{2.2}$$

可以发现，(2.2)的右端已经和(2.1)的右端很接近了。更进一步，我们引入矩阵的线性化算子vec。$vec(X)$表示将矩阵X的所有列向量按顺序堆叠为一个很长的列向量。于是，对比(2.1) 式和(2.2)式，我们得到

$$\text{im2col}(A,[2,2])^T \times (vec(K))^T = vec(A*K) \tag{2.3}$$

im2col和(2.3)式将会在随后的推导中起到关键作用。不过，在继续数学之旅前，让我们先再次回到卷积层本身。

我们上述的讨论事实上只考虑了channel为1的情况。一般地，如果输入$\boldsymbol{x}$的维度是$D \times H \times W$，卷积核$\boldsymbol{\kappa}$的维度是$D' \times H' \times W'$，那么必须满足$D = D'$，才能定义矩阵卷积。记$\boldsymbol{x} = ((A_1),\cdots,(A_D))$，$\boldsymbol{\kappa} = ((K_1),\cdots,(K_D))$。这里$(A_i)_{i=1}^D$ 是D个$H \times W$ 的矩阵，$(K_i)_{i=1}^D$是D个$H' \times W'$的矩阵。为避免与分块矩阵的记号混淆，我们用给矩阵加括号的方式来表示矩阵的堆叠。在padding=valid的条件下，$\boldsymbol{x}$ 与$\boldsymbol{\kappa}$ 的卷积为

$$\boldsymbol{x} * \boldsymbol{\kappa} := \sum_{i=1}^{D} A_i * K_i \tag{2.4}$$

注意这里$\boldsymbol{x} * \boldsymbol{\kappa}$的结果是一个$(H-H'+1)\times(W-W'+1)$的矩阵，即padding=valid时，特征面通过一个卷积层后会缩小。如果希望特征面的长和宽在卷积作用后保持不变，常用的技巧是先将原特征面扩张成$(H+H'-1)\times(W+W'-1)$的矩阵，多出来的位置都以0填充。为方便起见，本文余下的论述总是假设padding=valid，stride=$(1,1)$。

当第k层的输入$\boldsymbol{x}^k = ((A_1),\cdots,(A_{D_k}))$的维度是$D_k \times H_k \times W_k$，卷积核$\boldsymbol{\kappa} = ((K_1),\cdots,(K_{D_k}))$ 的维度是$D_k \times H'_k \times W'_k$ 时，可以类似定义

$$\text{im2col}(\boldsymbol{x}^k,[H'_k,W'_k]) := \begin{pmatrix} \text{im2col}(A_1,[H'_k,W'_k]) \\ \text{im2col}(A_2,[H'_k,W'_k]) \\ \vdots \\ \text{im2col}(A_{D_k},[H'_k,W'_k]) \end{pmatrix} \tag{2.5}$$

即将每一channel中相同子块经vec转化得到的列向量，再堆叠到新矩阵的同一列中。记

$$\phi(\boldsymbol{x}^k) = \left(\text{im2col}(\boldsymbol{x}^k,[H'_k,W'_k])\right)^T \tag{2.6}$$

以简化符号。

如果在第k层中，有D_{k+1}个形如$\boldsymbol{\kappa}$ 的卷积核$\boldsymbol{\kappa}_1^k,\cdots,\boldsymbol{\kappa}_{D_{k+1}}^k$，那么第$k$层的输出（同时也是第$k+1$ 层的输入）$\boldsymbol{y}^k = \boldsymbol{x}^{k+1}$的维度为$D_{k+1}\times(H_k - H'_k + 1)\times(W_k - W'_k + 1)$。我们也可以将矩阵线性化算子$vec$ 的适用范围拓展到任意order$\geq$ 2的Tensor上。假设$\boldsymbol{y}$是一

个order$= n$的Tensor，当$n=2$时，vec的定义已在上文给出；当$n\geq 3$时，假设$\boldsymbol{y}$的维度为$i_1\times\cdots\times i_n$，则可以将$\boldsymbol{y}$表示为$(\boldsymbol{y}_1,\cdots,\boldsymbol{y}_{i_1})$，同时递归地将$vec$定义为：

$$vec(\boldsymbol{y}) := \begin{pmatrix} vec(\boldsymbol{y}_1) \\ vec(\boldsymbol{y}_2) \\ \vdots \\ vec(\boldsymbol{y}_{i_1}) \end{pmatrix} \tag{2.7}$$

因此$vec(\boldsymbol{y})$的结果仍然是一个列向量。定义

$$C^k := \Big(vec(\boldsymbol{\kappa}_1^k), vec(\boldsymbol{\kappa}_2^k), \cdots, vec(\boldsymbol{\kappa}_{D_{k+1}}^k)\Big) \tag{2.8}$$

再结合(2.3)，(2.4)，(2.6)，(2.8)四式，我们神奇地得到了如下简洁而又漂亮的forward propagation公式：

$$vec(\phi(\boldsymbol{x}^k)\times C^k) = vec(\boldsymbol{y}^k) = vec(\boldsymbol{x}^{k+1}) \tag{2.9}$$

如果记$H_{k+1}=H_k-H_k'+1, W_{k+1}=W_k-W_k'+1$，则$\phi(\boldsymbol{x}^k)$的维度为$H_{k+1}W_{k+1}\times H_k'W_k'D_k$，$C^k$的维度为$H_k'W_k'D_k\times D_{k+1}$。

2.2. **Some properties.** 为了计算损失z相对于卷积层参数的偏导数，还需要一些数学准备工作。

首先我们引入Kronecker product。两个矩阵$A_{m\times n}, B_{k\times l}$的Kronecker product 被定义为：

$$A\otimes B := \begin{pmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{pmatrix}$$

能够看出，Kronecker product对参与运算的矩阵没有维度要求。

可以验证，

$$(A\otimes B)^T = A^T\otimes B^T \tag{2.10}$$

$$vec(AXB) = (B^T\otimes A)vec(X) \tag{2.11}$$

因此，我们可以将(2.9)式改写为

$$vec(\boldsymbol{x}^{k+1}) = vec(\boldsymbol{y}^k) = vec(\phi(\boldsymbol{x}^k)C^kI) = (I\otimes\phi(\boldsymbol{x}^k))vec(C^k) \tag{2.12}$$

$$vec(\boldsymbol{x}^{k+1}) = vec(\boldsymbol{y}^k) = vec(I\phi(\boldsymbol{x}^k)C^k) = ((C^k)^T\otimes I)vec(\phi(\boldsymbol{x}^k)) \tag{2.13}$$

这里I表示单位矩阵。

让我们再深入探讨一下$\phi(\boldsymbol{x}^k)$。我们用(p,q)表示$\phi(\boldsymbol{x}^k)$中元素的坐标，用(d_k,i_k,j_k)表示$\boldsymbol{x}^k$中元素的坐标，用$(d_{k+1},i_{k+1},j_{k+1})$表示$\boldsymbol{x}^{k+1}$中元素的坐标；同时注意到卷积核$\boldsymbol{w}^k =$

$((\boldsymbol{\kappa}_1^k),\cdots,(\boldsymbol{\kappa}_{D_{k+1}}^k))$ 是一个order 4 Tensor，因此我们用(d',d,i,j)表示$\boldsymbol{w}^k$中元素的坐标；并且为了与计算机科学中的记号保持一致，坐标的初始值设为0。

注意到$\phi(\boldsymbol{x}^k)\times C^k$是一个$H_{k+1}W_{k+1}\times D_{k+1}$的矩阵，这个矩阵的第$d_{k+1}$列其实就是由$\boldsymbol{x}^{k+1}$的第$d_{k+1}$ 个channel进行矩阵向量化后得到的，因此

$$p = i_{k+1} + H_{k+1} j_{k+1} \tag{2.14}$$

再来看q。注意到卷积核的每个channel都包含$H_k' W_k'$个元素，所以q除以$H_k' W_k'$ 的商，实际上是$\phi(\boldsymbol{x}^k)$中坐标为(p,q) 的元素在$\boldsymbol{x}^k$中所属channel的序号d_k，因此

$$q = H_k' W_k' d_k + H_k' j + i \tag{2.15}$$

最后，由矩阵卷积的计算过程，不难发现：

$$i_k = i + i_{k+1} \tag{2.16}$$

$$j_k = j + j_{k+1} \tag{2.17}$$

由(2.14)-(2.17)式，$\phi(\boldsymbol{x}^k)$中的坐标(p,q)唯一决定了一个元素在$\boldsymbol{x}^k$的坐标(d_k,i_k,j_k)；反之则不然，因为只要卷积核的尺寸大于1×1，$\boldsymbol{x}^k$中的元素就可能在计算卷积时被多次使用。

本小节总结的公式，将在下一小节的梯度推导中起到关键作用。

2.3. **Back propagation.** 假设我们已经求得了$\partial z/\partial \boldsymbol{x}^{k+1}$，由链式法则，$z$对第$k$ 层参数$\boldsymbol{w}^k$的偏导数有如下关系式：

$$\frac{\partial z}{\partial (vec(\boldsymbol{w}^k))^T} = \frac{\partial z}{\partial (vec(\boldsymbol{x}^{k+1}))^T}\frac{\partial vec(\boldsymbol{x}^{k+1})}{\partial (vec(\boldsymbol{w}^k))^T} \tag{2.18}$$

由(2.8)式，$vec(C^k) = vec(\boldsymbol{w}^k)$；再由(2.12) 式

$$\frac{\partial vec(\boldsymbol{x}^{k+1})}{\partial (vec(\boldsymbol{w}^k))^T} = \frac{\partial (I\otimes \phi(\boldsymbol{x}^k)) vec(\boldsymbol{w}^k)}{\partial (vec(\boldsymbol{w}^k))^T} = I\otimes\phi(\boldsymbol{x}^k) \tag{2.19}$$

因此，

$$\begin{aligned}
\frac{\partial z}{\partial (vec(C^k))^T} &= \frac{\partial z}{\partial (vec(\boldsymbol{x}^{k+1}))^T}(I\otimes\phi(\boldsymbol{x}^k)) \\
&= \left(vec\left(\frac{\partial z}{\partial \boldsymbol{x}^{k+1}}\right)\right)^T (I\otimes\phi(\boldsymbol{x}^k)) \\
&\overset{vec(\boldsymbol{x}^{k+1})=vec(\phi(\boldsymbol{x}^k)C^k)}{=} \left((I\otimes\phi(\boldsymbol{x}^k)^T) vec\left(\frac{\partial z}{\partial(\phi(\boldsymbol{x}^k)C^k)}\right)\right)^T \\
&\overset{(2.12)}{=} \left(vec\left(\phi(\boldsymbol{x}^k)^T\left(\frac{\partial z}{\partial(\phi(\boldsymbol{x}^k)C^k)}\right)\right)\right)^T
\end{aligned} \tag{2.20}$$

由于(2.20)式两端vec作用的矩阵具有相同的维度，因此去掉vec并不改变元素对应关系，于是我们得到

$$\frac{\partial z}{\partial C^k}=\phi(\boldsymbol{x}^k)^T\frac{\partial z}{\partial(\phi(\boldsymbol{x}^k)C^k)} \tag{2.21}$$

这里$C^k,\phi(\boldsymbol{x}^k)C^k$可以分别由$\boldsymbol{w}^k,\boldsymbol{x}^{k+1}$ resize得到，所以z 对$\boldsymbol{w}^k$的偏导数表达式惊人地简洁。

要继续back propagation过程，求z对$\boldsymbol{w}^{k-1}$的偏导数，显然我们必须知道$\partial z/\partial\boldsymbol{x}^k$。由链式法则

$$\frac{\partial z}{\partial(vec(\boldsymbol{x}^k))^T}=\frac{\partial z}{\partial(vec(\boldsymbol{x}^{k+1}))^T}\frac{\partial vec(\boldsymbol{x}^{k+1})}{\partial(vec(\boldsymbol{x}^k))^T} \tag{2.22}$$

欲简化(2.22)式，我们还需要一些准备知识。回忆一下，$\phi(\boldsymbol{x}^k)$包含了$H_{k+1}W_{k+1}H_k'W_k'D_k$个元素，$\boldsymbol{x}^k$包含了$H_kW_kD_k$个元素。我们可以引入一个维度为$H_{k+1}W_{k+1}H_k'W_k'D_k\times H_kW_kD_k$的大矩阵$M$来将$\phi(\boldsymbol{x}^k)$ 和$\boldsymbol{x}^k$联系起来。M 的每一个行指标与$\phi(\boldsymbol{x}^k)$的坐标(p,q) 一一对应，每一个列指标与$\boldsymbol{x}^k$的坐标(d_k,i_k,j_k) 一一对应。再引入一个从(p,q)到(d_k,i_k,j_k)的映射m，由(2.14)-(2.17)式，m是一个单射，但它的逆映射m^{-1}不是单射。M的具体表达式为：

$$M((p,q),(d_k,i_k,j_k))=\begin{cases}1, & \text{如果}m(p,q)=(d_k,i_k,j_k),\\ 0, & \text{其它}\end{cases} \tag{2.23}$$

易见M的每一行只有一个元素为1，其它元素都为0，这是一个稀疏矩阵；同时

$$vec(\phi(\boldsymbol{x}^k))=M\times vec(\boldsymbol{x}^k) \tag{2.24}$$

我们有

$$\begin{aligned}\frac{\partial vec(\boldsymbol{x}^{k+1})}{\partial(vec(\boldsymbol{x}^k))^T}&\overset{(2.13)}{=}\frac{((C^k)^T\otimes I)vec(\phi(\boldsymbol{x}^k))}{\partial(vec(\boldsymbol{x}^k))^T}\\&\overset{(2.24)}{=}((C^k)^T\otimes I)M\end{aligned} \tag{2.25}$$

所以

$$\frac{\partial z}{\partial(vec(\boldsymbol{x}^k))^T}=\frac{\partial z}{\partial(vec(\phi(\boldsymbol{x}^k)C^k))^T}((C^k)^T\otimes I)M \tag{2.26}$$

其中

$$\begin{aligned}\frac{\partial z}{\partial(vec(\phi(\boldsymbol{x}^k)C^k))^T}((C^k)^T\otimes I)&=\left((C^k\otimes I)vec\left(\frac{\partial z}{\partial(\phi(\boldsymbol{x}^k)C^k)}\right)\right)^T\\&\overset{(2.11)}{=}\left(vec\left(\frac{\partial z}{\partial(\phi(\boldsymbol{x}^k)C^k)}(C^k)^T\right)\right)^T\end{aligned} \tag{2.27}$$

将(2.27)带入(2.26)，我们得到

$$\frac{\partial z}{\partial(vec(\boldsymbol{x}^k))}=M^Tvec\left(\frac{\partial z}{\partial(\phi(\boldsymbol{x}^k)C^k)}(C^k)^T\right) \tag{2.28}$$

$\partial z/\partial(vec(\boldsymbol{x}^k))$中的元素可以由坐标$(d_k,i_k,j_k)$唯一确定；对应到右边的$M^T$，我们需要知道它被$(d_k,i_k,j_k)$
标注的那一行中有哪些元素为1。由$m:(p,q)\to(d_k,i_k,j_k)$的定义可知，在(d_k,i_k,j_k)行中，有且仅有那些列编号为$(p,q)\in m^{-1}(d_k,i_k,j_k)$的元素唯一。用数学式表达，并注意

$$\frac{\partial z}{\partial(\phi(\boldsymbol{x}^k)C^k)}(C^k)^T \tag{2.29}$$

到是一个$H_{k+1}W_{k+1}\times H_k'W_k'D_k$ 的矩阵，我们有

$$\left(\frac{\partial z}{\partial \boldsymbol{x}^k}\right)_{(d_k,i_k,j_k)}=\sum_{(p,q)\in m^{-1}(d_k,i_k,j_k)}\left(\frac{\partial z}{\partial(\phi(\boldsymbol{x}^k)C^k)}(C^k)^T\right)_{(p,q)} \tag{2.30}$$

尽管(2.30)式不如(2.21)式漂亮，但由于有(2.14)-(2.17)的帮助，我们并不需要像(2.28)式那样去做大矩阵乘法，而只需要根据(d_k,i_k,j_k)去求取少数几组(p,q) 值，所以(2.30)式仍然是简单而高效的。

至此，我们成功给出了卷积层的梯度计算公式。接下来对全连接层、池化层和Relu的介绍将会轻松许多。

3. Fully connected layer

全连接层（FC）其实可以视为一种特殊的卷积层，只不过它的卷积核的长宽与输入的特征面的长宽相同，因此我们无须对FC的导数做专门的推导。假设输入的特征面的维度为$D_k\times H_k\times W_k$，FC的卷积核维度为$D_{k+1}\times D_k\times H_k\times W_k$，则该FC的输出是一个$D_{k+1}$维的向量。

FC通常会被设置在CNN的末端。当输入的图像经过多个卷积层、池化层和Relu处理后，如果我们想一次使用当前特征面的所有特征，那么FC就可以派上用场。不过，近年来FC的使用在逐渐减少，一是因为它的计算量很大，二是因为它的效果并不比较小的卷积核更好。

4. Pooling layer

池化层（PL）的作用方式，是将特征面划分为许多小的子块，然后提取每个子块上的最重要特征（max pooling）或平均特征（average pooling）以构建新的特征面。PL的意义在于，它可以显著地缩小特征面的尺寸，从而减轻计算量，同时又不至于损失过多信息。实践证明，池化已经是CNN中一种相当有效的技术手段。现今最常用的池化方式是max pooling，本小节就以max pooling为例，来对PL的梯度进行推导。

假设网络的第k层是一个max pooling layer，输入$\boldsymbol{x}^k$的维度为$D_k\times H_k\times W_k$，池化使用的filter的尺寸为$H\times W$。为了方便论述，不妨再假设stride=(H,W)，同时H整除H_k，

W整除W_k，则该层的输出$\boldsymbol{x}^{k+1}$的维度满足

$$D_{k+1} = D_k, \quad H_{k+1} = \frac{H_k}{H}, \quad W_{k+1} = \frac{W_k}{W} \tag{4.1}$$

显然，$\boldsymbol{x}^k$和$\boldsymbol{x}^{k+1}$中的元素有如下对应关系

$$\boldsymbol{x}^{k+1}_{d_{k+1},i_{k+1},j_{k+1}} = \max_{0\le i<H, 0\le j<W} \boldsymbol{x}^k_{d_{k+1},i_{k+1}H+i,j_{k+1}W+j} \tag{4.2}$$

注意池化层是不含参数的，因此我们只需要计算

$$\frac{\partial z}{\partial (vec(\boldsymbol{x}^k))^T} = \frac{\partial z}{\partial (vec(\boldsymbol{x}^{k+1}))^T} \frac{\partial vec(\boldsymbol{x}^{k+1})}{\partial (vec(\boldsymbol{x}^k))^T} \tag{4.3}$$

为了求$\partial vec(\boldsymbol{x}^{k+1})/\partial(vec(\partial \boldsymbol{x}^k))^T$，我们引入一个维度为$H_{k+1}W_{k+1}D_{k+1} \times H_kW_kD_k$的矩阵$S$。$\boldsymbol{x}^{k+1}$中的坐标$(d_{k+1}, i_{k+1}, j_{k+1})$与$S$中的行一一对应，$\boldsymbol{x}^k$中的坐标$(d_k, i_k, j_k)$与$S$中的列一一对应。对于$\boldsymbol{x}^{k+1}$中的任一元素$\boldsymbol{x}^{k+1}_{d_{k+1},i_{k+1},j_{k+1}}$，我们通过(4.2)式找到它在$\boldsymbol{x}^k$中对应的元素$\boldsymbol{x}^k_{d_{k+1},i_k,j_k}$——如果(4.2)式右端的最大值被多个元素取到，则首先选出这些元素中列指标j_k最小的子集，再在这个子集上选出行指标i_k最小的元素。令S中所有对应于坐标对$((d_{k+1}, i_{k+1}, j_{k+1}), (d_{k+1}, i_k, j_k))$的元素为1，其它元素为0。不难发现，$S$是一个每行恰有一个非0元素的稀疏矩阵。

利用矩阵S，我们得到

$$vec(\boldsymbol{x}^{k+1}) = S \times vec(\boldsymbol{x}^k) \Rightarrow \frac{\partial vec(\boldsymbol{x}^{k+1})}{\partial (vec(\boldsymbol{x}^k))^T} = S \tag{4.4}$$

于是

$$\frac{\partial z}{\partial vec(\boldsymbol{x}^k)} = S^T \frac{\partial z}{\partial vec(\boldsymbol{x}^{k+1})} \tag{4.5}$$

在stride和filter尺寸相同的假设下，(4.5)式的计算是非常简单的。因为在此条件下，$\boldsymbol{x}^k$中的每个元素至多只在池化中被选中一次，所以S^T的每一行至多只有一个元素非0。具体来说，对于与$(d_{k+1}, i_{k+1}, j_{k+1})$对应的$(d_{k+1}, i_k, j_k)$，$(\partial z/\partial \boldsymbol{x}^k)_{(d_{k+1},i_k,j_k)} = (\partial z/\partial \boldsymbol{x}^{k+1})_{(d_{k+1},i_{k+1},j_{k+1})}$；而对于其它的$(d_k, i_k, j_k)$，$(\partial z/\partial \boldsymbol{x}^k)_{(d_k,i_k,j_k)} = 0$。

当stride的尺寸比filter的尺寸小时，除了需要重新计算H_{k+1}和W_{k+1}以改写(4.1)式外，(4.2)-(4.5)式的推导仍然是有效的。为了计算(4.5)式，我们需要在forward propagation时记录每个$\boldsymbol{x}^k_{d_k,i_k,j_k}$在通过池化层后到达了$\boldsymbol{x}^{k+1}$的什么位置以构建矩阵$S$。在运算时我们仍然不需要做矩阵乘法，记$\boldsymbol{x}^k_{d_k,i_k,j_k}$传递到$\boldsymbol{x}^{k+1}$后的位置集为$A_{d_k,i_k,j_k}$，则偏导数公式为

$$\left(\frac{\partial z}{\partial \boldsymbol{x}^k}\right)_{(d_k,i_k,j_k)} = \sum_{(d_{k+1},i_{k+1},j_{k+1})\in A_{d_k,i_k,j_k}} \left(\frac{\partial z}{\partial \boldsymbol{x}^k}\right)_{(d_{k+1},i_{k+1},j_{k+1})} \tag{4.6}$$

5. Relu

Relu应该算是CNN中结构最简单的部分了，它的定义如下：

$$Relu(x) := \begin{cases} x, & \text{如果} x > 0 \\ 0, & \text{其它} \end{cases} \tag{5.1}$$

Relu的导数被定义为：

$$Relu'(x) := \begin{cases} 1, & \text{如果} x > 0 \\ 0, & \text{其它} \end{cases} \tag{5.2}$$

之所以将(5.2)式称为“定义”，是因为Relu在0处不可导。从实际使用看来，规定$Relu'(0) = 0$并不会产生问题。

由于Relu不含任何参数，因此对该层（假设为第k层）求梯度，只需要计算$\partial z/\partial \boldsymbol{x}^k$。显然有

$$\begin{aligned} \left(\frac{\partial z}{\partial \boldsymbol{x}^k}\right)_{(d_k,i_k,j_k)} &= \left(\frac{\partial z}{\partial \boldsymbol{x}^{k+1}}\right)_{(d_k,i_k,j_k)} \frac{\partial Relu(\boldsymbol{x}^k_{d_k,i_k,j_k})}{\partial \boldsymbol{x}^k_{d_k,i_k,j_k}} \\ &= \begin{cases} \left(\frac{\partial z}{\partial \boldsymbol{x}^{k+1}}\right)_{(d_k,i_k,j_k)}, & \text{如果} \boldsymbol{x}^k_{d_k,i_k,j_k} > 0 \\ 0, & \text{其它} \end{cases} \end{aligned} \tag{5.3}$$

Relu的作用在于增强网络的非线性性以获得更好的泛化性能。在Relu出现之前，通常被用于增强非线性性的函数是sigmoid函数$\sigma(x) := 1/(1+e^{-x})$。sigmoid函数和Relu相比有一个很大的缺点：它导数的最大值只有1/4，特别是当x的绝对值很大的时候，$\sigma'(x)$会非常小，这使得训练过程中参数更新非常缓慢。Relu的导数虽然可能为0，但是其前提条件是该处的特征小于0；在添加了适当的bias之后，可以认为小于0的特征都是不重要的，无需激活，在训练时也就不用更新与之对应的参数；而对于那些被激活的特征，它们在通过Relu时，无论是其自身还是导数都不衰减，训练时对应参数的优化就会比较充分。

6. Summary

以上便是CNN中各层梯度的计算方法，了解这些背后的数学原理有助于我们更好地理解CNN，同时为设计具体的网络提供强有力的理论支持。感谢来自朋友同事的帮助，笔者在与他们的交流中受益匪浅。由于笔者水平有限，本文不可避免地会存在一些谬误，欢迎大家批评指正。

Reference

[1] J. Wu, *Introduction to Convolutional Neural Networks*, https://cs.nju.edu.cn/wujx/paper/CNN.pdf