卷积神经网络的反向传递函数

一、背景介绍

随着机器学习的迅速发展，图像识别以及图像分类的应用和研究越来越多的受到关注，卷积神经网络的正向传递过程被大量的研究，以降低正向运算时间，并提升图像分类的准确率，但是卷积神经网络的反向传递部分却并未被很好的分析研究，具体的传递过程在网络上难以找到讲解详细的内容，本文将通过具体分析，对于卷积神经网络的反向传递过程进行分析推导，并通过 matlab 来实现其相应为算法，以验证文中的公式推导，同时加深对于卷积神经网络反向传导的理解。

二、算法推导

1、softmax 函数以及 softmax-cross-entropy 函数

Softmax 函数为：

$$softmax(x)_j = \sigma(x)_j = \frac{\exp(x_j)}{\sum_i^n \exp(x_i)}$$

于是可以推导：

$$\frac{\partial \sigma(x)_j}{\partial x_i} = \begin{cases} \dfrac{\exp(x_i)\left(\sum_k^n \exp(x_k)\right) - \exp(x_i)\exp(x_i)}{\left(\sum_k^n \exp(x_k)\right)^2} & i = j \\ \dfrac{-\exp(x_j)\exp(x_i)}{\left(\sum_k^n \exp(x_k)\right)^2} & i \neq j \end{cases}$$

$$= \begin{cases} \sigma(x)_i - \sigma(x)_i{}^2 & i = j \\ -\sigma(x)_i \sigma(x)_j & i \neq j \end{cases}$$

一般来说当采用 softmax 作为分类函数时，对应的损失函数采用：

$$\mathrm{L(x)} = \frac{1}{2}\sum_i^n (label_i - \sigma(x)_i)^2$$

可以从上式得知：

$$\frac{\partial L(x)}{x_i} = (\sigma(x)_i - label_i) * \sigma(x)_i - \sigma(x)_i * \sum_j^n \sigma(x)_j * (\sigma(x)_j - label_j)$$

Softmax-cross-entropy 函数对应的损失函数为：

$$\mathrm{L(x)} = -\frac{1}{n}\sum_i^n label_i * \log(\delta(x_i))$$

其相对应的：

$$\frac{\partial L}{\partial x_i} = \frac{1}{n}\sum_j^n label_j * \delta(x_i) - \frac{1}{n} label_i$$

那么：

$$\frac{\partial L}{\partial x_i}=\begin{cases}\frac{1}{n}(\delta(x_i)-label_i) & label_i=1\\ \frac{1}{n}(\delta(x_i)-label_i) & label_i=0\end{cases}$$

在这一部分需要特别注意，优于 loss 函数往往是求和函数，所以从 loss 向前倒推的过程中，softmax 函数求导这部分尤其容易只考虑i = j部分，而忽略了i ≠ j部分，我在最开始的公式推导过程中犯过类似的错误。

2、全连接层反向传导

全连接层网络的正向函数为：

$$Y = WX + b$$

于是对应的反向传导过程为：

$$\frac{\partial Y}{\partial W}=X^T$$

$$\frac{\partial Y}{\partial b}=1$$

$$\frac{\partial Y}{\partial X}=W$$

3、卷积层反向传导

另从上一次传回来的导数为δ，也就是说$\frac{\partial L}{\partial y}=\delta$，其中 L 代表 loss 函数，y 表示当前卷积层的输出。那么根据卷积公式：

$$y_j=\sum_i^n x_i * k_{ij}+b_j$$

其中 j 代表输出的特征图相应的通道，i 代表出入图像相应的通道。在这里我在初始学习的过程中，阅读了大量的论文，但始终弄不明白输入特征图和输出特征图是如何变换使得其最终的通道数发生改变，大量的论文中在写到卷积核的尺寸的时候往往会默认通道数与输入特征图通道数相同，因而只介绍卷积核的长宽，使我最开始误以为卷积核是二维的。

从上面的公式可以看出：

$$\frac{\partial L}{\partial b_j}=\frac{\partial L}{\partial y_j}\frac{\partial y_j}{\partial b_j}=\sum_{u,v}(\delta_j)_{uv}$$

相当于是将与第 j 张特征图相对应的δ_j各元素求和。

而对于卷积核k_{ij}的求导就相应复杂一些，具体来说就是对于$(k_{ij})_{uv}$，与输入第 i 张特征图相卷积，最终参与构成第 j 张特征图的二维卷积核，对应坐标(u,v)的元素，其对应的导数与和该元素相乘过的输入特征图上的元素有关：

$$\frac{\partial L}{\partial k_{ij}}=\sum_{u,v}(\delta_j)_{uv}p_{uv}$$

其中上式中的 P 对应着和k_{ij}中相应元素做过乘法计算的矩阵。具体化来说，假设卷积核的尺寸为 5*5，卷积采用了‘same’的方式，也就是在周围每边补上了宽度为 2 的 0，步长为 1 时，当输入特征图为 14*14 时：

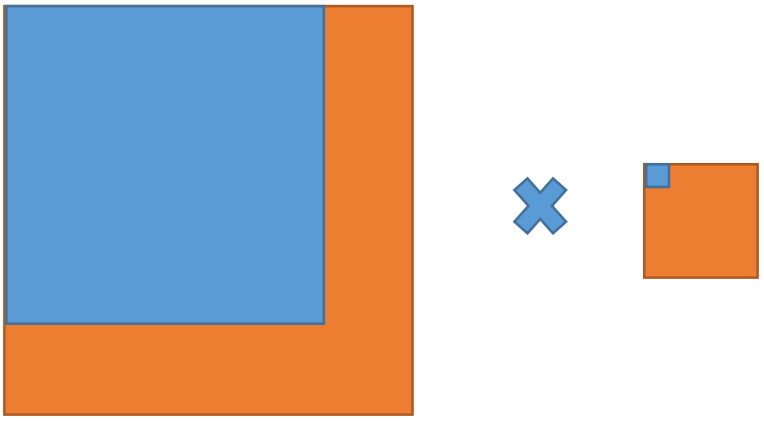


Figure 1

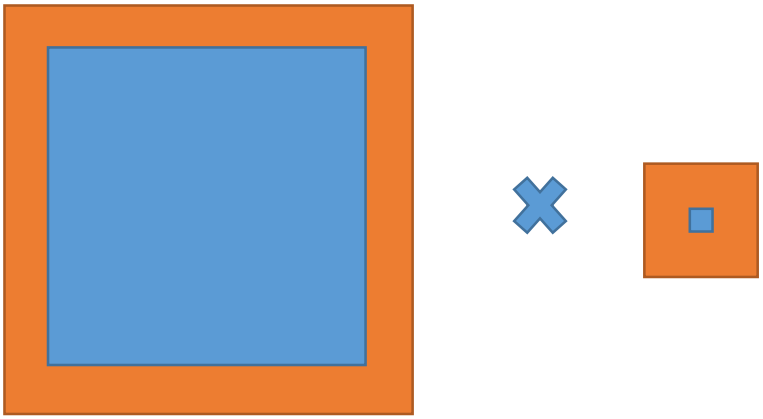


Figure 2

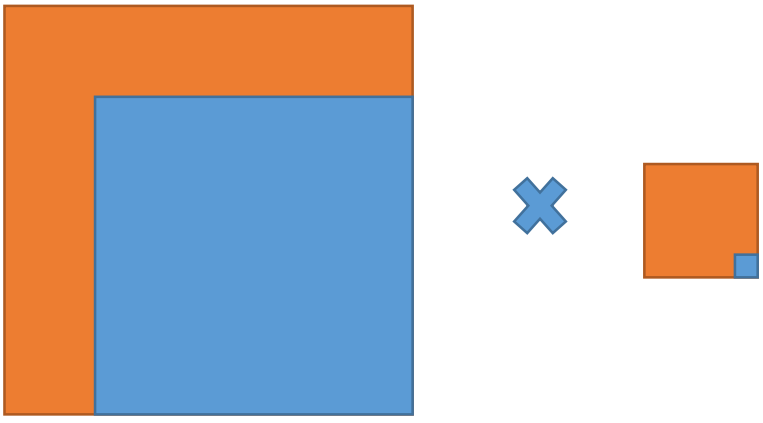


Figure 3

由于卷积计算要先对卷积核进行 180 度翻转，因为上图中对应的橘色小方块代表已经经过翻转后的卷积核，蓝色小方格代表相应的元素，蓝色大方格代表与之进行计算的 P 的范围，尺寸为 14*14，与输出特征图的整张图相对应，橘色大方格代表已经补 0 后的特征图，尺寸为 18*18，因此在反向传导过程中，相

当于用 180 度翻转后的δ_j作为卷积核，与输入特征图进行卷积运算，得到 180 度翻转后的卷积核[1] ：

$$\mathrm{rot180}\left(\frac{\partial L}{\partial k_{ij}}\right) = \mathrm{conv2}(x_i, \mathrm{rot180}(\delta_j), 'same')$$

padding()是补零函数。

在对于输入特征图进行求导的过程中，则对于边缘的分析相对复杂，但与之前卷积核求导方法类似，就是要求与相应元素做过乘法运算的部分卷积核矩阵：

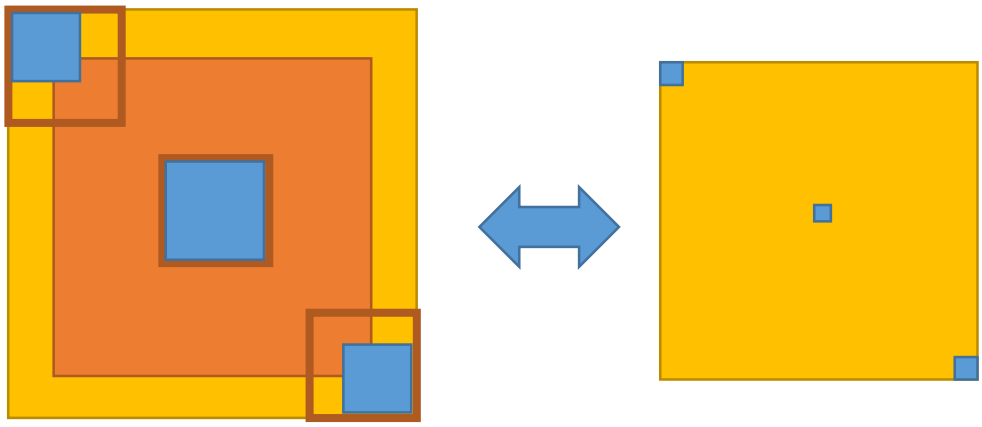


Figure 4

上图 Figure 4 中左侧黄色代表补零后的特征图，橘色区域代表补零前的特征图，右边黄色方框代表输入特征图，左图蓝色方格和棕色边框分别为参与和相应元素相乘的部分以及卷积核，这里的卷积核是经过了 180 度旋转的卷积核，可以通过上图的分析看出的是，对输入特征图求导相当于[1] ：

$$\frac{\partial L}{\partial x_i} = \sum_j conv2(\delta_j, rot180(k_{ij}), 'same')$$

以上求和是输出通道数个卷积结果求和[1] 。

4、激活函数

1）ReLU 算法：

$$\mathrm{relu(x)} = \max(x, 0)$$

对应的求导运算：(其中 y 代表 y = relu(x))

$$\mathrm{relu\_derive(y)} = \begin{cases} 1 & y > 0 \\ 0 & y \le 0 \end{cases}$$

2）sigmoid 算法：

$$\mathrm{sigmoid(x)} = \frac{1}{1 + \exp(-x)}$$

对应的求导运算：（其中 y 代表 y=sigmoid(x)）

$$\mathrm{sigmoid\_derive(y)} = \frac{\exp(-x)}{(1 + \exp(-x))^2}$$

$$= \left(1 - \frac{1}{1+\exp(-x)}\right)\frac{1}{1+\exp(-x)} = y(1-y)$$

3）tanh 算法：

$$\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$

对应的求导运算：（其中 y 代表 y=tanh(x)）

$$\tanh\_\text{derive}(y) = 1 - y^2$$

5、池化层

1）maxpooling 算法：（以 2*2 池化为例）

$$y_{ij} = \text{maxpooling}\left(x_{2i-1:2i,2j-1:2j}\right) = \max(x_{2i-1,2j-1}, x_{2i,2j-1}, x_{2i-1,2j}, x_{2i,2j})$$

在反向求导过程中要用到 max 对应的位置坐标，其相应的位置导数为 1，其他位置导数为 0，当$\frac{\partial L}{\partial y_{ij}} = \delta_{ij}$时，假设 max 对应的坐标为(u,v)：

$$\frac{\partial L}{\partial x_{2i-1:2i,2j-1:2j}} = \begin{cases} \delta_{ij} & if\ (i,j) = (u,v) \\ 0 & else \end{cases}$$

2）meanpooling 算法：（以 2*2 池化为例）

$$y_{ij} = \text{meanpooling}\left(x_{2i-1:2i,2j-1:2j}\right) = \frac{x_{2i-1,2j-1} + x_{2i,2j-1} + x_{2i-1,2j} + x_{2i,2j}}{4}$$

反向求导：

$$\text{meanpooling\_derive}(x_{2i-1,2j-1}, x_{2i,2j-1}, x_{2i-1,2j}, x_{2i,2j}) = \frac{1}{4}$$

$$\frac{\partial L}{\partial x_{2i-1:2i,2j-1:2j}} = \frac{1}{4}\delta_{ij}$$

三、实验过程

1、单层全连接层

在这里为了循序渐进的研究相应算法的效果，先设置单层全连接层作为对比，采用两种分类方式 softmax 和 softmax-cross-entropy 进行精度对比，其中 weights 和 biases 的初始化都直接采用了 matlab 的内置 rand()函数，为了实验的简化和降低其他方面的影响，train_dataset 都只跑了一遍，并且将学习率固定在了 0.05，对应的实验结果精度见 Table1。

Table1. 分类方式对于单层全连接层的精度影响

分类方式	精度
Softmax	0.8875
Softmax-cross-entropy	0.8815

经过以上实验发现，对于单层的全连接层而言，softmax 函数和 softmax-cross-entropy 函数并不会使精度产生巨大的差别，且从精度上来看单层全连接层已经能够对于图像进行很好的分类了。

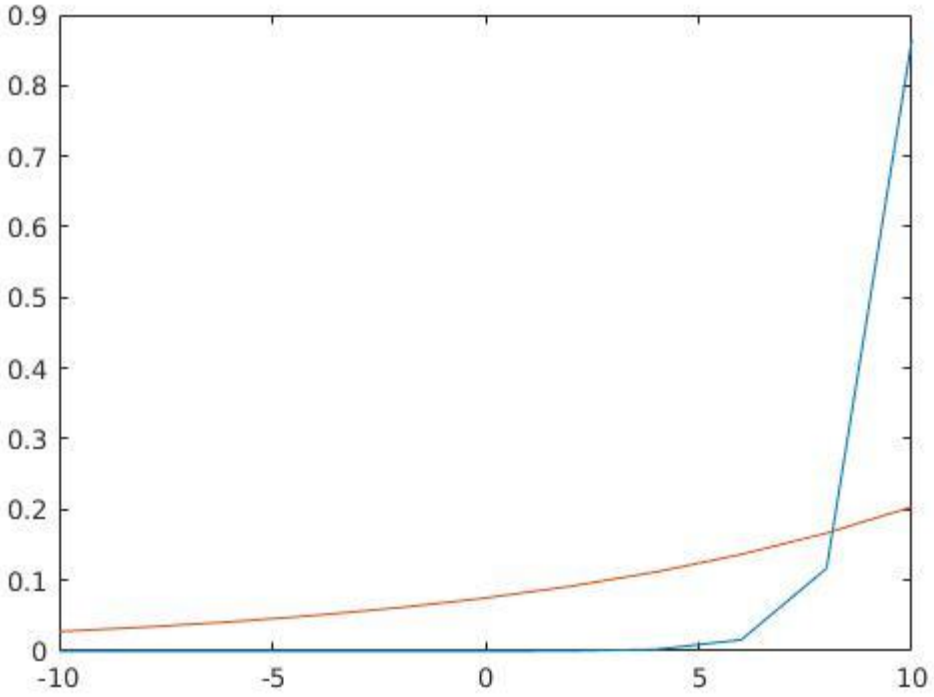
2、双层全连接层

不同于单层全连接层实验过程中的一帆风顺，双层全连接层在数据的量级上遇到了很大的麻烦，仅仅简单的将两个全连接层放在一起最后一层的输出数据位于 10^6 量级，接着要通过 softmax 或 softmax-cross-entropy 时，会因为指数运算而产生溢出，也因为如此，在最初的实验过程中，权重的反向更新过程中传递回来的数据都是 NaN。经过查阅资料，结合自己的想法，将 weights 和 biases 的初始化过程从 rand()函数产生，到 weights 初始化为标准差为 0.1 的矩阵，biases 初始化为 0.1，具体原因见卷积神经网络部分实验。

Table2. 激活函数对于双层全连接层的精度影响

第一层	第二层	分类方式	精度
Normal	Normal	softmax	0.6717
normal	-max	softmax	0.8959
relu	-max	softmax	0.9051
sigmoid	-max	softmax	0.9204
normal	normal	Softmax-cross-entropy	0.6186
Normal	-max	Softmax-cross-entropy	0.8848
Relu	-max	Softmax-cross-entropy	0.9228
sigmoid	-max	Softmax-cross-entropy	0.9152

从 Table2 中可以看出，不同于单层全连接层，双层全连接层需要更多的考虑到激活函数，且不同的激活函数对于最终的结果影响也是巨大的。表格中-max 的含义是：减去对应矩阵中的最大值；表格中的 normal 的含义是：矩阵除以矩阵的 norm，其中此处的 norm 求解用到了 matlab 内置函数 norm()。从理论上比较可知，矩阵 X-max(X)与原矩阵 X 在运用于 softmax 函数是，其对应的结果不考虑计算机内置计算的精度影响，应该是一样的，而采用 normal 来降低数据量级的方法存在的最大问题是：会很大程度上改变矩阵各元素之间的相对差距，以本例为例，fc2_out 的各元素通常处于 10^3 的量级，若所有元素都除以 10^3，在通过函数 softmax 函数，就会造成，大数被削弱小数被加强，具体用图像形象化就是：



本图中横坐标 x=-10:2:10，其中橘色的线条就是在蓝色线条的基础上对于输入除以 10，从上图可知，normal 的降低数据数量级的方式会造成 softmax 分类后的结果并不能很好的代表网络计算结果，因而也就会进一步影响，反向传递过程中 weights 和 biases 的权重的尺度大小，normal 会造成本不需要大幅度更新的 weights 和 biases 被大幅度改变，而本需要大幅度更新的数据反而得不到有效更新，这也就是为什么无论最终的分类方式是 softmax 还是 softmax-cross-entropy，在第一和第二层都采用了 normal 激活的网络，几遍经过长时间的训练依旧无法有效提高训练精度。

当然可以预见的是，采用-max 的方式来进行 softmax 前预处理也并非完全没有问题，当 fc2_out 的所有值都处于一个比较正常的范围时，对所有元素减去矩阵的最大值，有可能造成本来处于正常范围的数值在指数计算过程中下溢出，这样同样会造成某些本来有微小值的部分全部变为 0，在反向传递过程中得到 NaN。

从 Table2 同样可以看出，不论是单层全连接层，还是双层全连接层而言，在本次试验中 softmax-cross-entropy 的分类结果和 softmax 差不太多。理论上来说 softmax-cross-entropy 相当于是在 softmax 的基础上嫁接了一层多项式逻辑损失函数(multinomial logistic loss layer)[1]。以向量(0.87,0.01,0.12)为例，当对应的 label 为(1,0,0)时，2*softmax-loss =(1-0.87)^2 + (0-0.01)^2 + (0-0.12)^2，3*softmax-cross-entropy-loss = - 1 * log(0.87) - 0 * log(0.01) - 0 * log(0.12) - 0 * log(0.13) – 1 * log(0.99)-1 * log(0.88)，从这两个式子可以看出，两式都同时关注了 label=1 的部分和 label=0 的部分，这样可以有效的降低第二和第三项的值，并同时提升第一项的值，使得在下次计算 loss 时，能够有效降低其值。不同之处在于，softmax 对应得 loss function 直接采用了最小二乘法，而 softmax-cross-entropy 则采用了与求解信息熵类似的方式，将每一项的概率由 softmax 的似然概率，变为 softmax-cross-entropy 的最大似然概率。两种方法都是从不同的方式试图降低损失函数，提升

[1] http://freemind.pluskid.org/machine-learning/softmax-vs-softmax-loss-numerical-stability/

期望分类对应的概率。

在双层神经网络和单层神经网络的训练过程中，都尝试过采用 batch 的训练方式，从训练过程来看，batch 的训练初始过程速度慢于单个样本的训练过程，且要达到与单个样本的训练精度所需要的训练时长是要大于单个样本的训练时长的，但是不考虑训练时间长度，batch 的最终结果某种程度上是优于单个样本训练精度的。

3、卷积网络

在这里卷积网络采用了 mnist 数据集分类网络，相应的网络结构为：

输入数据为[28,28,1];

第一层卷积层，32 个 5*5 的卷积核，对应 kernel = [5,5,1,32]，biases=[32]，后面接着 ReLU 和 maxpooling 层；

第二层卷积层，64 个 5*5 的卷积核，对应 kernel = [5,5,32,64]，biases=[64]，后面接着 ReLU 和 mxapooling 层；

第一层全连接层，1024 个神经元；

第二层全连接层，10 个神经元，采用 softmax-cross-entropy 分类方式。

在卷积神经网络中遇到了与双层神经网络里一样的问题，随着数据在网络中传递，当数据到达最后一级时，数据的规模已经变为 10^6 量级，也因此数据爆炸问题很大程度上困扰了训练过程，导致精度始终上不去，即便是将所有的 weights 和 biases 都初始化原来的 0.001 倍，也依旧无法降低最后一级的输出数据量级。数据爆炸问题很大程度上阻碍了神经网络的深度加深，因为可以理论上推导得到，网络深度越深，在运算过程中，数字相乘相加次数越多，也就因此带来的最终级输出数据爆炸越严重。经过大量实验和资料阅读，我将 weights 和 biases 的初始化方式从原来的 rand()，变为 weights 采用初始化为标准差为 0.1 的矩阵，而 biases 都初始化为 0.1。具体的思路得益于文章《Batch Normalization：Accelerating Deep Network Training by Reducing Internal Covariate Shift》的启发，这篇文章的优秀之处在于它很好的分析了网络数据前向传递过程中数据爆炸的内在原因，并且为改善这种现象提供了一个灵活的网络层，使得数据爆炸现象减弱甚至消除的同时还能够带来一定的可学习空间，当然这一定程度上会带来运算量的加大，并且使得训练过程中对于已被训练的数据造成依赖，但是从标准差和平均值的角度去分析网络数据流动过程，某种程度上给提升网络训练效果指了明路，并提供了相应的理论基础。

在训练阶段，为了使训练速度加快，当输入数据被白化的时候训练速度最快，也就是说其对应的平均值为 0，标准差为 1 时，训练最快[2] 。但是本次试验中，输入样本就是直接将图片进行输入的，每个像素点的值位于 0 和 255 之间，若假定输入图像标准差为 Var(x)，那么最终的输出标准差会经过各层的计算进行积累，从而变得很大，为了有效降低输出的标准差，可以将每一级的参数进行调整，降低各级 weights 的标准差，防止数据爆炸，从而降低反向传导过程中梯度消失的状况。从实验验证的情况来看，将 weights 的标准差从 1 变为 0.1 的确对于训练效果起到了很好的促进作用。

在本次试验中，比较了计算精度对于最终结果精度的影响情况，从 Table3 中的数据可以看出，在 matlab 中采用“format long”，也就是小数点后 15 位有效位的情况和“format long”小数点后 4 位有效位，对于最终训练精度几乎不会有太大影响，相反“format long”由于运算量更大，训练速度比“format short”的情况慢的

多。从训练结果来看同样是对于 train_dataset 只进行了一次训练，但是卷积网络的效果还是要明显由于单层和双层全连接层的。

Table3.运算精度对于训练精度的影响

计算精度	精度
Formal short	0.9358
Formal long	0.9383

参考论文：

[1] Bouvrie J. Notes on Convolutional Neural Networks[J]. Neural Nets, 2006.

[2] Ioffe S, Szegedy C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift[J]. Computer Science, 2015.