Batch Normalization详解

By jiangqh on May.02-2017

Batch Normalization广为人知应该是在15年，当时微软亚研院的何恺明提出ResNet在各项视觉比赛中获得冠军并得到了当年的Cvpr best paper。ResNet除了使用跳过式连接还大量使用了Bath Normalization，网络大获成功的同时也证明了BN在深度神经网络训练中的巨大威力。

本文尝试从原理到推导详细的梳理Batch Normalization。下文中为了方便将Batch Normalization简称为BN。

什么是BN？

什么是BN呢？如名字所示，BN所做十分简单，即将某一层输出归一化，使得其均值为0方差为1。值得注意的是BN是在channel维度做的，即将每个channel都进行归一化，如果有n个channel那么便会有n个归一化操作。具体来说如果某个层的输出为$x = (x^0, x^1, \dots x^n)$，那么BN做的便是：

$$x^k = \frac{x^k - E[x^k]}{\sqrt{Var[x^k]}}$$

而在卷积神经网络中我们有$C \times H \times W$的输出，遵照卷积层的特性BN在每个$H \times W$上做归一化，也就是做C个归一化操作。

为什么要做BN？

在神经网络训练中我们使用基于梯度的优化方法(gradient-based optimization)，最常用的便是结合了Momentum的随机梯度下降。在使用梯度下降法时我们的一般步骤为：1、计算各个weight相对于损失函数loss的梯度。2、假设其它weight不变，对某个weight进行更新—但是在实际更新中我们其实是同时对weight进行了更新。回忆上一篇blog浅谈优化方法与泰勒展开梯度下降法实际上利用的是函数的泰勒一次展开近似，实际上它是会受到高阶影响的，泰勒展开实际为：

$$f(\overrightarrow{w}+\overrightarrow{s})=f(\overrightarrow{w})+g(\overrightarrow{w})^T\overrightarrow{s}+\frac{1}{2}\overrightarrow{s}^TH(\overrightarrow{w})\overrightarrow{s}+\ldots$$

而我们做了如下近似：

$$f(\overrightarrow{w}+\overrightarrow{s})\approx f(\overrightarrow{w})+g(\overrightarrow{w})^T\overrightarrow{s}$$

考虑一个简单情况：神经网络中不包含激活函数，我们的预测是关于x的线性函数：

$$y=w_1w_2w_3\ldots w_mx$$

在这种情况下y是关于x的线性函数，但不是w的线性函数，进一步假设loss关于y的梯度为1。在梯度下降法中我们想要降低loss，有如下更新步骤：

$$w_i=w_i-\epsilon g_i$$

其中g_i为损失函数关于w_i的梯度。那么此时的y变为了：

$$y=(w_1-\epsilon g_1)(w_2-\epsilon g_2)\ldots(w_m-\epsilon g_m)x$$

我们期待此时y会有所减小，因为y与loss成正比(梯度为1)。但实际情况真是如此吗？考察上式将前两项展开可以看到其中一项有：

$$+\epsilon^2 g_1 g_2 \prod_{i=3}^{m} w_i$$

即其它w对整个网络的实际更新效果是有影响的，式子$\prod_{i=3}^{m} w_i$中如果w从3到m如果都小于1那么影响便可忽略不计，但是如果它们都大于1那么该整体值就会变成一个大数，甚至超过梯度下降带来的下降效应，导致y不降反升甚至溢出。这种效应使得神经网络的训练中学习速率ϵ的选择变得尤其困难，需要考虑大量的weight的影响，所以往往只能使用很小的学习效率和精心设计的初始化操作。

一种解决方案是直接利用采用多阶信息的优化方法，对线性的我们可以考虑二阶方法比如牛顿法。但是在实际中神经网络包含激活函数并不是线性的，所以其泰勒展开将包含多阶的信息，同时神经网络参数十分巨大，这使得使用多阶信息的优化方法根本不可行。

BN的提出便是为了解决网络训练weight更新中的多层相互耦合问题，通过BN操作我们使得某一层的分布始终为方差为1均值为0的标准分布。在我们的例子中$y = w_m h_{l-1}$，如果x服从高斯分布那么h_{l-1}也将服从高斯分布只是其不再是单位标准分布。当我们对h_{l-1}进行BN后它将重新满足均值为0方差为1的标准高斯分布，对于低于$l-1$的层的大多数情况，不管它们的weight怎么改变(少数情况除外)h_{l-1}都将是一个稳定的方差为1均值为0的高斯分布。于是整个学习过程变成了简单的学习$y = w_m h_{l-1}$，这使得学习变得容易起来，而如果没有进行BN操作的话，低层的每一次更新都会给h_{l-1}层带来改变。当然在这个线性的例子中所有的低层作用不管正负都被抹去了(一阶和二阶信息)，但是在实际的神经网络中由于有更高阶的影响，所以它们仍然有用。总结来说即—BN会使某个单元的均值为0方差为1，去除一阶和二阶信息影响，但是保留了高阶的信息；这使得整个网络的学习变得容易起来，并且网络的非线性变换能力得到了保留。

换个角度理解我们知道反向传播的时候梯度每次是要乘以w向后传播的：

$h_l = w_l h_{l-1}$
$\frac{\partial l}{\partial hl-1} = \frac{\partial l}{\partial hl} \times \frac{\partial hl}{\partial hl-1} = \frac{\partial l}{\partial hl} \times w_l$

每次梯度更新后都会改变w_l的值，如果变得过大或者过小都可能会发生**梯度弥散或者梯度爆炸问题。**

考虑当梯度从l层传到k层，此时的梯度为：

$$\frac{\partial l}{\partial h_k} = \frac{\partial l}{\partial h_l} \times \prod_{i=k}^{l} w_i$$

即由w连乘所引入的问题。那么当我们做了BN以后很明显由：

$$BN(wu) = BN((\alpha w)u)$$

即与w的尺度无关，那么在做反向传播的时候有：

$$\frac{\partial BN((\alpha w)u)}{\partial u} = \frac{\partial BN(wu)}{\partial u}$$

可以看到反向传播时候的倒数变得与w的scale无关了！也就是说我们的梯度消失和梯度爆炸问题得到了解决，我们不会再因为w的尺度而受影响。更进一步有：

$$\frac{\partial BN((\alpha w)u)}{\partial (\alpha w)} = \frac{1}{\alpha} \frac{\partial BN(wu)}{\partial w}$$

即**较大的w将获得较小的梯度，这意味着weight的更新更加稳健了，较大的weight更新较少，较小的weight更新较大。**

总结起来便是，**BN解决了反向传播中的梯度弥散和爆炸问题，同时使得weight的更新更加稳健**，从而使网络的学习更加容易，减少了对weight初始化的依赖和可以使用更大的学习速率。

怎么做BN

说完了为什么我们来看看具体怎么做。根据定义，我们只需要对每个channel求解其均值和方差，然后进行操作即可。假设某个batch内共有m个数据，那么对某一个channel有：

$$u = \frac{1}{m} \sum_{i=1}^{m} x_i$$

$$var = \frac{1}{m}\sum_{i=1}^{m}(x_i - u)^2$$

$$\hat{x_i} = \frac{x_i - u}{\sqrt{var + \epsilon}}$$

$$y_i = \gamma\hat{x_i} + \beta$$

在上式中前两项为求取均值和方差，第三项分布中ϵ是为了防止数值问题而加的一个小数，比如10^{-6}。最后一项中γ和β是可以学习的参数，通过这两个参数我们使BN保持了更强的学习能力可以自己的分布，那么我们为什么在进行了归一化操作后还要加上这两个参数呢？这是因为加上这两个参数后现在的分布族便包含了原来BN前的分布，但是原来的分布方差和均值由下面层的各种参数weight耦合控制，而现在仅由γ和β控制，这样在保留BN层足够的学习能力的同时，我们使得其学习更加容易。

利用链式求导法则我们有：

$$\frac{\partial l}{\partial \gamma} = \sum_{i=1}^{m}\frac{\partial l}{\partial y_i}\hat{x_i}$$

$$\frac{\partial l}{\partial \beta} = \sum_{i=1}^{m}\frac{\partial l}{\partial y_i}$$

$$\frac{\partial l}{\partial \hat{x_i}} = \frac{\partial l}{\partial y_i}\gamma$$

$$\frac{\partial l}{\partial var} = \sum_{i=1}^{m}\frac{\partial l}{\hat{x_i}}\frac{\hat{x_i}}{\partial var} = \sum_{i=1}^{m}\frac{\partial l}{\partial y_i}\gamma(x_i - u)\frac{-1}{2}(var + \epsilon)^{\frac{-3}{2}}$$

$$\frac{\partial l}{\partial u} = (\sum_{i=1}^{m}\frac{\partial l}{\partial \hat{x_i}}\frac{-1}{\sqrt{var + \epsilon}}) + \frac{\partial l}{\partial var}\frac{\sum_{i=1}^{m} -2(x_i - u)}{m}$$

$$\frac{\partial l}{x_i} = \frac{\partial l}{\hat{x}_i}\frac{1}{\sqrt{var+\epsilon}} + \frac{\partial l}{\partial var}\frac{2(x_i-u)}{m} + \frac{\partial l}{\partial u}\frac{1}{m}$$

至此我们完整的梳理了BN的由来和它解决的问题以及详细推导过程，具体实现可以参考caffe或者TensorFlow里相应的代码。值得注意的是在做test的时候为了对一个sample也可以用BN，此时的u, var往往采用做training时候的一个统计平均，同时方差采样的是无差的平均统计，即做test时有：

$$u_{test} = E[u]$$

$$var_{test} = \frac{m}{m-1}E[var]$$

同时注意在卷积层做BN时也是按照channel来的，即进行channel个BN操作。