BatchNorm的原理及代码实现

Batch Normalization

机器学习领域有个很重要的假设：**IID独立同分布假设，就是假设训练数据和测试数据是满足相同分布的，**这是通过训练数据获得的模型能够在测试集获得好的效果的一个基本保障。？BatchNorm就是在深度神经网络训练过程中使得每一层神经网络的输入保持相同分布的。

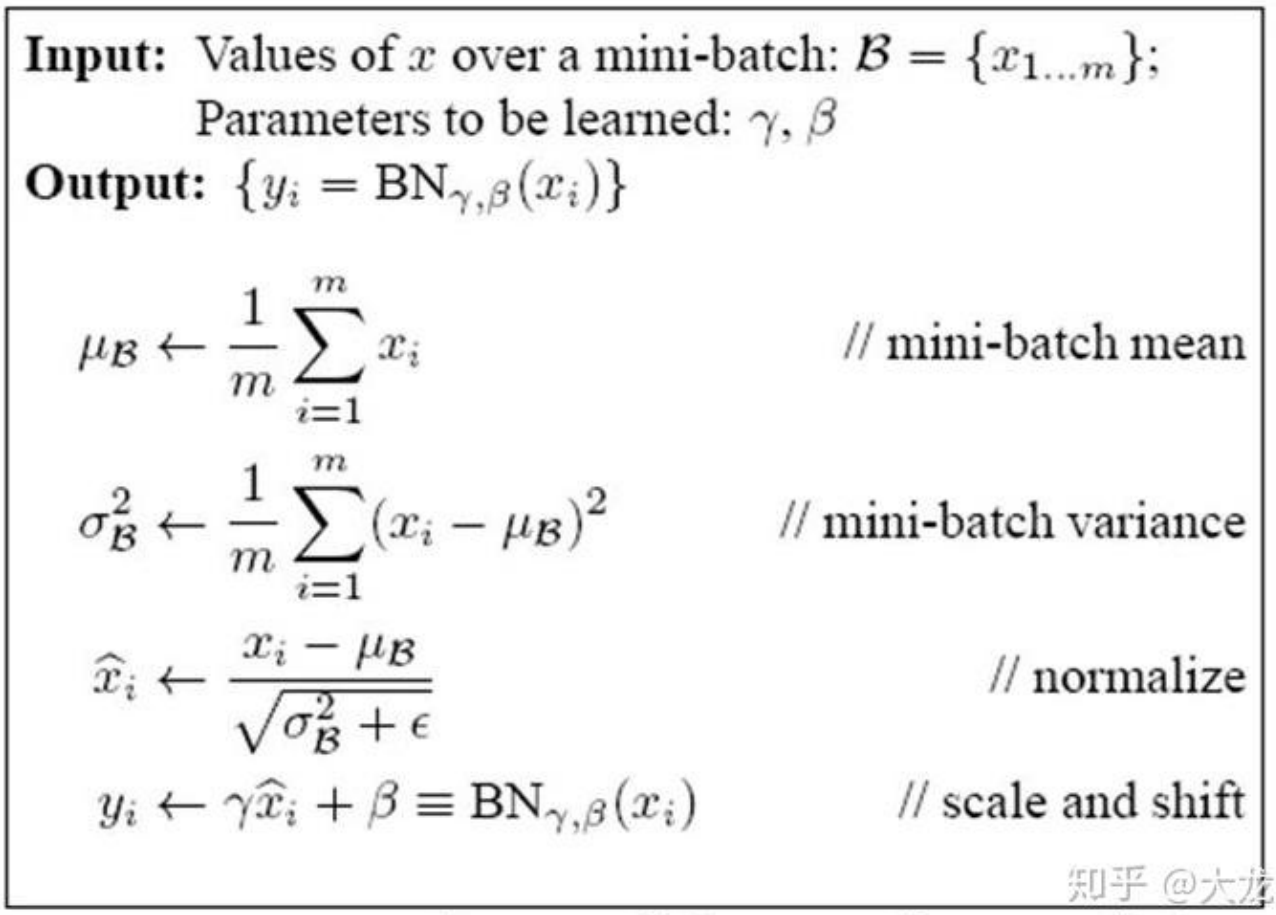
对于深度学习这种包含很多隐层的网络结构，在训练过程中，因为各层参数老在变，所以每个隐层都会面临covariate shift的问题，也就是在训练过程中，隐层的输入分布老是变来变去，这就是所谓的“Internal Covariate Shift”，Internal指的是深层网络的隐层，是发生在网络内部的事情，而不是covariate shift问题只发生在输入层。

BN的基本思想其实相当直观：因为深层神经网络在做非线性变换前的激活输入值（就是那个x=WU+B，U是输入）随着网络深度加深或者在训练过程中，其分布逐渐发生偏移或者变动，之所以训练收敛慢，一般是整体分布逐渐往非线性函数的取值区间的上下限两端靠近（对于Sigmoid函数来说，意味着激活输入值WU+B是大的负值或正值），所以这导致后向传播时低层神经网络的梯度消失，这是训练深层神经网络收敛越来越慢的本质原因，而BN就是通过一定的规范化手段，把每层神经网络任意神经元这个输入值的分布强行拉回到均值为0方差为1的标准正太分布而不是正态分布，其实就是把越来越偏的分布强制拉回比较标准的分布，这样使得激活输入值落在非线性函数对输入比较敏感的区域，这样输入的小变化就会导致损失函数较大的变化，意思是这样让梯度变大，避免梯度消失问题产生，而且梯度变大意味着学习收敛速度快，能大大加快训练速度。

但是很明显，稍微了解神经网络的读者一般会提出一个疑问：如果都通过BN，那么不就跟把非线性函数替换成线性函数效果相同了？这意味着什么？我们知道，如果是多层的线性函数变换其实这个深层是没有意义的，因为多层线性网络跟一层线性网络是等价的。这意味着网络的表达能力下降了，这也意味着深度的意义就没有了。所以BN为了保证非线性的获得，对变换后的满足均值为0方差为1的x又进行了scale加上shift操作(y=scale * x + shift)，每个神经元增加了两个参数scale和shift参数，这两个参数是通过训练学习到的，意思是通过scale和shift把这个值从标准正态分布左移或者由移一点并长胖一点或者变瘦一点，每个实例挪动的程度不一样，这样等价于非线性函数的值从正中心周围的线性区往非线性区动了动。核心思想应该是想找到一个线性和非线性的较好平衡点，既能享受非线性的较强表达能力的好处，又避免太靠非线性区两头使得网络收敛速度太慢。

上文说过经过这个变换后某个神经元的激活x形成了均值为0，方差为1的正态分布，目的是把值往后续要进行的非线性变换的线性区拉动，增大导数值，增强反向传播信息流动性，加快训练收敛速度。但是这样会导致网络表达能力下降，为了防止这一点，每个神经元增加两个调节参数（scale和shift），这两个参数是通过训练来学习到的，用来对变换后的激活反变换，使得网络表达能力增强，即对变换后的激活进行如下的scale和shift操作，对比BN的过程会发现这其实是变换的反操作：

BN其具体操作流程



BN的优点

1. BN将Hidden Layer的输入分布从饱和区拉到了非饱和区，减小了梯度弥散，提升了训练速度，收敛过程大大加快，还能增加分类效果。
2. Batchnorm本身上也是一种正则的方式（主要缓解了梯度消失），可以代替其他正则方式如dropout等。
3. 调参过程也简单多了，对于初始化要求没那么高，而且可以使用大的学习率等。

BN的缺陷

batch normalization依赖于batch的大小，当batch值很小时，计算的均值和方差不稳定。

BN的推导

参考BN推导

python代码

训练过程

```
def Batchnorm_simple_for_train(x, gamma, beta, bn_param):
"""
param:x    : 输入数据，设shape(B,L)
param:gama : 缩放因子  γ
param:beta : 平移因子  β
param:bn_param   : batchnorm所需要的一些参数
    eps      : 接近0的数，防止分母出现0
    momentum : 动量参数，一般为0.9， 0.99， 0.999
    running_mean ：滑动平均的方式计算新的均值，训练时计算，为测试数据做准备
    running_var  : 滑动平均的方式计算新的方差，训练时计算，为测试数据做准备
"""
    running_mean = bn_param['running_mean']  #shape = [B]
    running_var = bn_param['running_var']    #shape = [B]
    momentun = bn_param['momentun']    #shape = [B]
    results = 0. # 建立一个新的变量

    x_mean=x.mean(axis=0)  # 计算x的均值
    x_var=x.var(axis=0)    # 计算方差

    running_mean = momentum * running_mean + (1 - momentum) * x_mean
    running_var = momentum * running_var + (1 - momentum) * x_var

    x_normalized=(x - running_mean)/np.sqrt(running_var + eps)       # 归一化
    results = gamma * x_normalized + beta             # 缩放平移


    #记录新的值
    bn_param['running_mean'] = running_mean
    bn_param['running_var'] = running_var

    return results , bn_param
```

理解滑动平均(exponential moving average)

首先计算均值和方差，然后归一化，然后缩放和平移，完事！但是这是在训练中完成的任务，每次训练给一个批量，然后计算批量的均值方差，但是在测试的时候可不是这样，测试的时候每次只输入一张图片，这怎么计算批量的均值和方差，于是，就有了代码中下面两行，在训练的时候实现计算好 mean 和 var 测试的时候直接拿来用就可以了，不用计算均值和方差。

测试过程

```
def Batchnorm_simple_for_test(x, gamma, beta, bn_param):
"""
param:x    : 输入数据，设shape(B,L)
param:gama : 缩放因子  γ
param:beta : 平移因子  β
param:bn_param   : batchnorm所需要的一些参数
    eps      : 接近0的数，防止分母出现0
    momentum : 动量参数，一般为0.9， 0.99， 0.999
    running_mean ：滑动平均的方式计算新的均值，训练时计算，为测试数据做准备
    running_var  : 滑动平均的方式计算新的方差，训练时计算，为测试数据做准备
"""
```

```
    running_mean = bn_param['running_mean']  #shape = [B]
    running_var = bn_param['running_var']    #shape = [B]
    results = 0. # 建立一个新的变量

    x_normalized=(x-running_mean )/np.sqrt(running_var +eps)       # 归一化
    results = gamma * x_normalized + beta            # 缩放平移

    return results , bn_param
```

当 β 越大时，滑动平均得到的值越和 θ 的历史值相关。如果 β=0.9，则大致等于过去 10 个 θ 值的平均；如果 β=0.99，则大致等于过去 100 个 θ 值的平均。

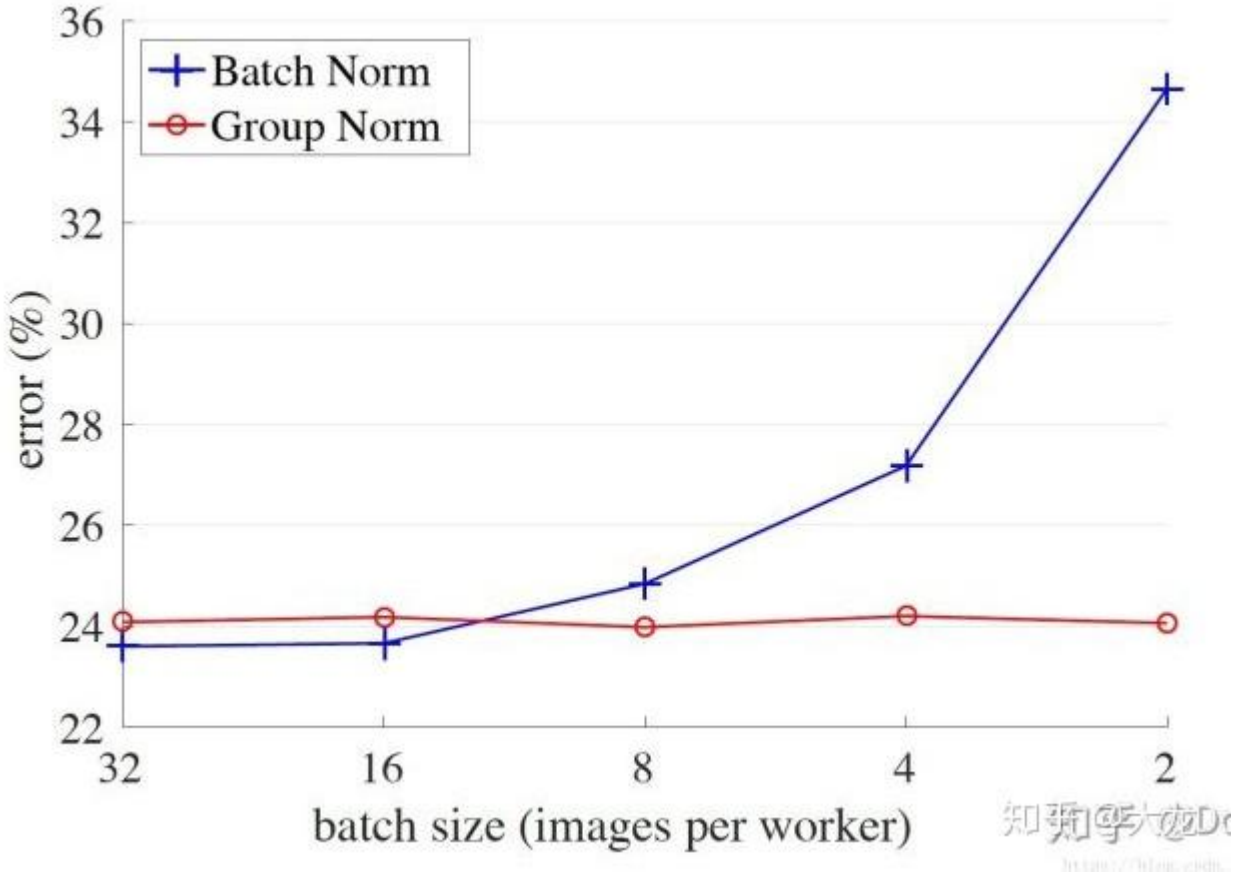
滑动平均的好处：

占内存少，不需要保存过去10个或者100个历史 θ 值，就能够估计其均值。（当然，滑动平均不如将历史值全保存下来计算均值准确，但后者占用更多内存和计算成本更高）

Group Normalization

What's wrong with BN

BN全名是Batch Normalization，见名知意，其是一种归一化方式，而且是以batch的维度做归一化，那么问题就来了，此归一化方式对batch是independent的，过小的batch size会导致其性能下降，一般来说每GPU上batch设为32最合适，但是对于一些其他深度学习任务batch size往往只有1-2，比如目标检测，图像分割，视频分类上，输入的图像数据很大，较大的batchsize显存吃不消。那么，对于较小的batch size，其performance是什么样的呢？如下图：



另外，Batch Normalization是在batch这个维度上Normalization，但是这个维度并不是固定不变的，比如训练和测试时一般不一样，一般都是训练的时候在训练集上通过滑动平均预先计算好平均-mean，和方差-variance参数，在测试的时候，不在计算这些值，而是直接调用这些预计算好的来用，但是，当训练数据和测试数据分布有差别是时，训练机上预计算好的数据并不能代表测试数据，这就导致在训练，验证，测试这三个阶段存在inconsistency。

Why GN work?

传统角度来讲，在深度学习没有火起来之前，提取特征通常是使用SIFT，HOG和GIST特征，这些特征有一个共性，都具有按group表示的特性，每一个group由相同种类直方图的构建而成，这些特征通常是对在每个直方图（histogram）或每个方向（orientation）上进行组归一化（group-wise norm）而得到。而更高维的特征比如VLAD和Fisher Vectors(FV)也可以看作是group-wise feature，此处的group可以被认为是每个聚类（cluster）下的子向量sub-vector。

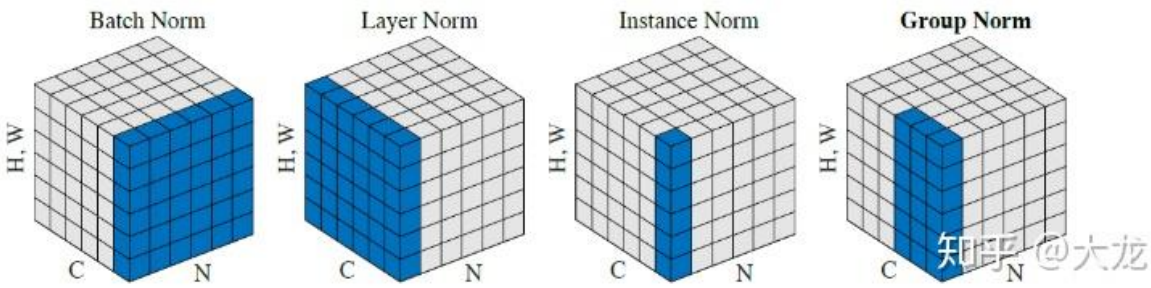
从深度学习上来讲，完全可以认为卷积提取的特征是一种非结构化的特征或者向量，拿网络的第一层卷积为例，卷积层中的的卷积核filter1和此卷积核的其他经过transform过的版本filter2（transform可以是horizontal flipping等），在同一张图像上学习到的特征应该是具有相同的分布，那么，具有相同的特征可以被分到同一个group中，按照个人理解，每一层有很多的卷积核，这些核学习到的特征并不完全是独立的，某些特征具有相同的分布，因此可以被group。

导致分组（group）的因素有很多，比如频率、形状、亮度和纹理等，HOG特征根据orientation分组，而对神经网络来讲，其提取特征的机制更加复杂，也更加难以描述，变得不那么直观。

另在神经科学领域，一种被广泛接受的计算模型是对cell的响应做归一化，此现象存在于浅层视觉皮层和整个视觉系统。

How GN work

- batchNorm是在batch上，对NHW做归一化，对小batchsize效果不好；
- layerNorm在通道方向上，对CHW归一化，主要对RNN作用明显；
- instanceNorm在图像像素上，对HW做归一化，用在风格化迁移；
- GroupNorm将channel分组，然后再做归一化；
- SwitchableNorm是将BN、LN、IN结合，赋予权重，让网络自己去学习归一化层应该使用什么方法



BatchNorm（NWH）假设batchsize为32，某一层的输出是64个通道，则是对32个batch的第一个通道的特征图进行归一化

LayerNorm（CWH）仅对某一个batch上的所有通道归一化

GroupNorm 相当于layernorm中的通道分组归一化

参考

全面解读Group Normalization-（吴育昕-何恺明）

Batch Normalization导读

基础 | batchnorm原理及代码详解

深度学习中 Batch Normalization为什么效果好？ - 魏秀参的回答 - 知乎

深度学习中 Batch Normalization为什么效果好？ - 龙鹏-言有三的回答 - 知乎

理解滑动平均(exponential moving average)

BatchNormalization、LayerNormalization、InstanceNorm、GroupNorm、SwitchableNorm总结

GitHub

个人主页