Batch Normalization（BN）的动机

一般来说，如果模型的输入特征不相关且满足标准正态分布时，模型的表现一般较好。在训练神经网络模型时，我们可以事先将特征去相关并使得它们满足一个比较好的分布，这样，模型的第一层网络一般都会有一个比较好的输入特征，但是随着模型的层数加深，网络的非线性变换使得每一层的结果变得相关了，且不再满足分布。更糟糕的是，可能这些隐藏层的特征分布已经发生了偏移。
论文的作者认为上面的问题会使得神经网络的训练变得困难，为了解决这个问题，他们提出在层与层之间加入Batch Normalization层。训练时，BN层利用隐藏层输出结果的均值与方差来标准化每一层特征的分布，并且维护所有mini-batch数据的均值与方差，最后利用样本的均值与方差的无偏估计量用于测试时使用。

鉴于在某些情况下非标准化分布的层特征可能是最优的，标准化每一层的输出特征反而会使得网络的表达能力变得不好，作者为BN层加上了两个可学习的缩放参数和偏移参数来允许模型自适应地去调整层特征分布。

BN层的作用

贴出论文中的两张图，就可以说明BN层的作用

1. 使得模型训练收敛的速度更快
2. 模型隐藏输出特征的分布更稳定，更利于模型的学习

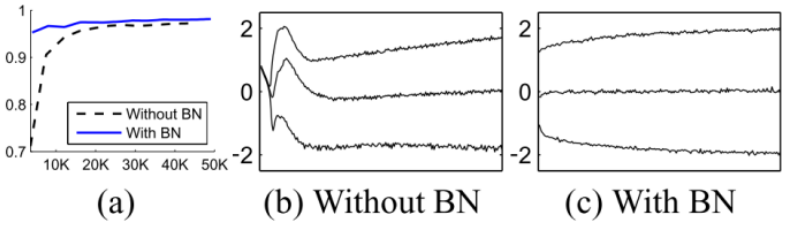

Figure 1: (a) *The test accuracy of the MNIST network trained with and without Batch Normalization, vs. the number of training steps. Batch Normalization helps the network train faster and achieve higher accuracy.* (b, c) *The evolution of input distributions to a typical sigmoid, over the course of training, shown as* $\{15, 50, 85\}$*th percentiles. Batch Normalization makes the distribution more stable and reduces the internal covariate shift.*

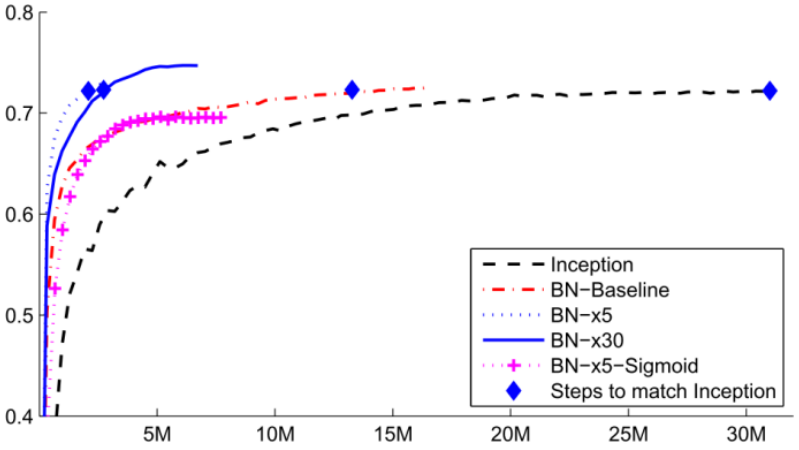

Figure 2: *Single crop validation accuracy of Inception and its batch-normalized variants, vs. the number of training steps.*

BN的推导过程

1. 前向算法

Batch Normalization层的实现很简单，主要的过程由如下算法给出

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

首先计算出mini-batch的均值与方差，接着对mini-batch中的每个特征做标准化处理
，最后利用缩放参数与偏移参数对特征进行后处理得到输出。
当模型进行训练的时候，论文中提出记录每一个mini-batch的均值与方差，在预测时利用均值与方差的无偏估计量来进行BN操作，也就是

输出也就表示为

2. 反向传播

BN层的反向传播相比于普通层要略微复杂一些，首先给出论文中的公式，对其中省略的步骤在下面会给出细致的推导过程。

$$\frac{\partial \ell}{\partial \widehat{x}_i} = \frac{\partial \ell}{\partial y_i} \cdot \gamma$$
$$\frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot (x_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{2}(\sigma_{\mathcal{B}}^2 + \epsilon)^{-3/2}$$
$$\frac{\partial \ell}{\partial \mu_{\mathcal{B}}} = \left(\sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \right) + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{\sum_{i=1}^{m} -2(x_i - \mu_{\mathcal{B}})}{m}$$
$$\frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{2(x_i - \mu_{\mathcal{B}})}{m} + \frac{\partial \ell}{\partial \mu_{\mathcal{B}}} \cdot \frac{1}{m}$$
$$\frac{\partial \ell}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i} \cdot \widehat{x}_i$$
$$\frac{\partial \ell}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i}$$

对于上图中的式（2）（公式序号按图中次序），由下面的分解式给出

对于式（3），有

对于式（4），有

若仔细观察式（3）与式（4），我们令

则可以将式（3）与式（4）简化为

这样做一个简单的替换，在实现代码的时候，运算会简化很多。式（1）（5）（6）的证明都很显然，在此略过了。

BN的代码实现

下面给出BN层的前向算法和反向传播的Python实现。

前面说过，论文中采用的是维护所有mini-batch的均值与方差，最后利用无偏估计量进行预测。在这里我们实现另一种方案，利用一个动量参数维护一个动态均值与动态方差，这样更方便简洁，torch7采用的也是这种方法，具体公式如下

写代码的时候可以利用之前的文章中提到的快速计算方法，可以很方便的写出BN层前向算法和反向传播。

1. 前向算法

```
defbatchnorm_forward(x,gamma,beta,bn_param):""" Forward pass for batch normalization. Input: - x: Data of shape (N, D) - gamma: Scale parameter of shape (D,) - beta: Shift paremeter of shape (D,) - bn_param: Dictionary with the following keys: - mode: 'train' or 'test'; required - eps: Constant for numeric stability - momentum: Constant for running mean / variance. - running_mean: Array of shape (D,) giving running mean of features - running_var Array of shape (D,) giving running variance of features Returns a tuple of: - out: of shape (N, D) - cache: A tuple of values needed in the backward pass """mode=bn_param['mode']eps=bn_param.get('eps',1e-5)momentum=bn_param.get('momentum',0.9)N,D=x.shaperunning_mean=bn_param.get('running_mean',np.zeros(D,dtype=x.dtype))running_var=bn_param.get('running_var',np.zeros(D,dtype=x.dtype))out,cache=None,Noneifmode=='train':sample_mean=x.mean(axis=0)sample_var=x.var(axis=0)x_normalized=(x-sample_mean)/np.sqrt(sample_var+eps)running_mean=momentum*running_mean+(1-momentum)*sample_meanrunning_var=momentum*running_var+(1-momentum)*sample_varout=gamma*out_+betacache=(out_,x,sample_var,sample_mean,eps,gamma,beta)elifmode=='test':scale=gamma/np.sqrt(running_var+eps)out=x*scale+(beta-running_mean*scale)else:raiseValueError('Invalid forward batchnorm mode "%s"'%mode)# Store the updated running means back into bn_parambn_param['running_mean']=running_meanbn_param['running_var']=running_varreturnout,cache
```

2. 反向传播

```
defbatchnorm_backward(dout,cache):"""    Backward pass for batch normalization.    Inputs:    - dout: Upstream derivatives, of shape (N, D)    - cache: Variable of intermediates from batchnorm_forward.    Returns a tuple of:    - dx: Gradient with respect to inputs x, of shape (N, D)    - dgamma: Gradient with respect to scale parameter gamma, of shape (D,)    - dbeta: Gradient with respect to shift parameter beta, of shape (D,)
"""dx,dgamma,dbeta=None,None,Noneout_,x,sample_var,sample_mean,eps,gamma,beta=cacheN=x.shape[0]dout_=(x-sample_mean)*-0.5*(sample_var+eps)**-1.5,axis=0)dx_=1/np.sqrt(sample_var+eps)dvar_=2*(x-sample_mean)/N# intermediate for convenient calculationdi=dout_*dx_+dvar*dvar_dmean=-1*np.sum(di,axis=0)dmean_=np.ones_like(x)/Ndx=di+dmean*dmear
```

参考

- Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift
- CS231n Convolutional Neural Networks for Visual Recognition