Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

1. 简介

在训练深度神经网络的时候经常会出现内部的协同变量变化（internal covariate shift）的问题，也就是说当训练神经网络的时候，隐藏层中每一层的参数都会由于上一层的参数变化而变化。这样会大大减小网络的训练速度，而且还要求很小的学习速率和很严格的训练流程。这篇论文主要提出了batch normalization的方法，就是在每一层都对输入进行规范化，也就是通过减去一个小批量（mini-batch)内的m个输入的均值再除以这个小批量内的方差。

2. 原理

- Internal Covariate Shift

在深度神经网络的训练中，我们要求优化损失函数L，如下面函数：

> $$\theta = argmin(\sum_{i=1}^{N} \ell(xi, \theta))$$

但是当层数不断增多，下一层的损失函数由上一层函数决定：

> $$L = F2\ (\ F1\ (\ u, \theta 1), \theta 2\ )$$

这样迭代下去，每一层的参数变化都会影响到之后层内的参数变化，这样大大增加了计算量，也增加了训练的时间。而且这样层层迭代容易使输入的很多维度移动到非线性饱和状态，最终导致收敛缓慢。

- Whitening

首先可以想到白化来减少Internal Covariate Shift的发生，对于每层的输出都对其进行归一化，也就是减去激活值的均值，但是这样只会使bias无限增大而损失函数不变。而且白化是针对每一层要对全层进行计算，计算量很大。

- Batch Normalization

因为白化的操作并不理想，作者提出了batch normalization的想法，这个原理就是在激活函数之前，对于输入值的每一个维度进行标准化，这样保留了不同维度之间的相关性。

> $$x^k = \frac{x^k - E[x^k]}{\sqrt{(var(x^k))}}$$

> $$y^k = \gamma x^k + \beta$$

当我们对x^k进行规范化的时候，参数γ和β也需要进行学习，换句话说就是再反向传播的时候BN是可求导的，γ和β也会影响到损失函数。

$$\frac{\partial \ell}{\partial \widehat{x}_i} = \frac{\partial \ell}{\partial y_i} \cdot \gamma$$

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

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

$$\frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{2(x_i - \mu_{\mathcal{B}})}{m} + \frac{\partial \ell}{\partial \mu_{\mathcal{B}}} \cdot \frac{1}{m}$$

$$\frac{\partial \ell}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i} \cdot \widehat{x}_i$$

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

并且batch normalization之后的网络反向传播时，增加学习率γ不会影响梯度。因为 $\frac{\partial BN(aWu)}{\partial au} = \frac{1}{a}\frac{\partial BN(aWu)}{\partial u}$

$\frac{\partial BN(aWu)}{\partial u} = \frac{\partial BN(Wu)}{\partial u}$

所以即使使用很大学习率，神经网络反向传播的时候也不会出现梯度爆炸和梯度消失。 这样加入了batch normalization的网络可以把学习率提高，这样可以加快训练的速度。

3. 神经网络中的实现

在神经网络中，在激活函数之前插入BN，也就是激活函数之前的输入是x^k， 现在是 BN(x^k).

在训练的时候用小批量来进行规范化，但预测的时候并不需要用小批量，因为预测的时候数据量不会像训练的时候那么大。 这时候的μ 和 σ 是根据之前训练的时候每个批量存下来的μ 和 σ 来计算。

$\mu = Etrain[\mu]$

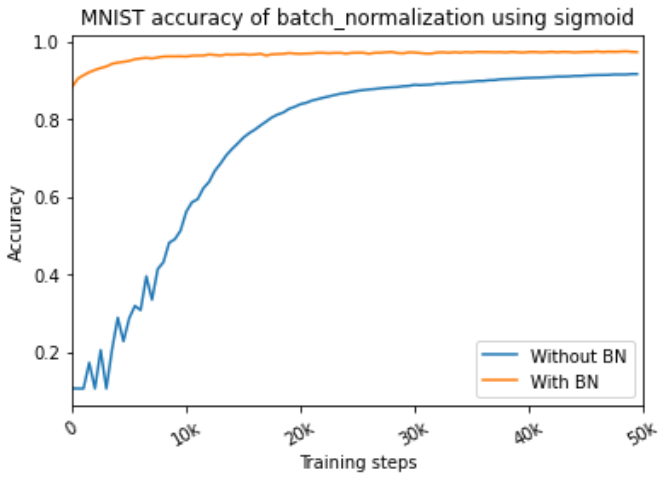
$Var[x] = \frac{m}{m-1} Etrain[\sigma^2]$

$$ytest = Bn = \gamma \frac{x-\mu}{\sqrt{Var[x]^2+\varepsilon}} + \beta$$

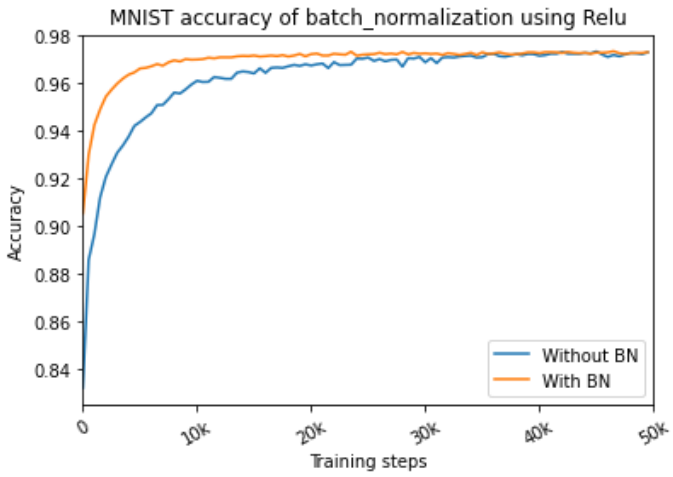
$$= \frac{\gamma}{\sqrt{Var[x]^2+\varepsilon}} x + \beta - \frac{\gamma\mu}{\sqrt{Var[x]^2+\varepsilon}}$$

4. 测试结果

在MNIST数据及上进行数字分类预测的实验，用简单的网络，有三层隐藏层，每层有100个神经元，激活函数是Sigmoid. 对网络进行50000个time steps. 得到的准确率随着时间的变化而变化的结果，如下图。



因为Sigmoid会使输入值移入非线性饱和，所以改变了激活函数，用reLU作为实验的激活函数，得到的结果如图。



5. 总结

通过加入了batch normalization， 深度神经网络的训练比没有加入batch norlization的网络收敛速度加快了很多，而且对激活函数的要求不再那么严格。同时因为避免学习率增大造成的梯度爆炸，可以使用高的学习率加快对网络的训练速度。