Batch Normalization原理与实战（上）

mp.weixin.qq.com/s/gsY2j14c8UpWShW9CRRy4A

”，重磅干货，第一时间送达！

链接 | https://zhuanlan.zhihu.com/p/34879333

前言

本文主要从理论与实战视角对深度学习中的Batch Normalization的思路进行讲解、归纳和总结，并辅以代码让小伙伴儿们对Batch Normalization的作用有更加直观的了解。
本文主要分为两大部分，由于篇幅过长，分为上下两篇。第一部分是理论板块，主要从背景、算法、效果等角度对Batch Normalization进行详解；第二部分是实战板块，主要以MNIST数据集作为整个代码测试的数据，通过比较加入Batch Normalization前后网络的性能来让大家对Batch Normalization的作用与效果有更加直观的感知。

一、理论板块

理论板块将从以下四个方面对Batch Normalization进行详解：

- 提出背景
- BN算法思想
- 测试阶段如何使用BN
- BN的优势

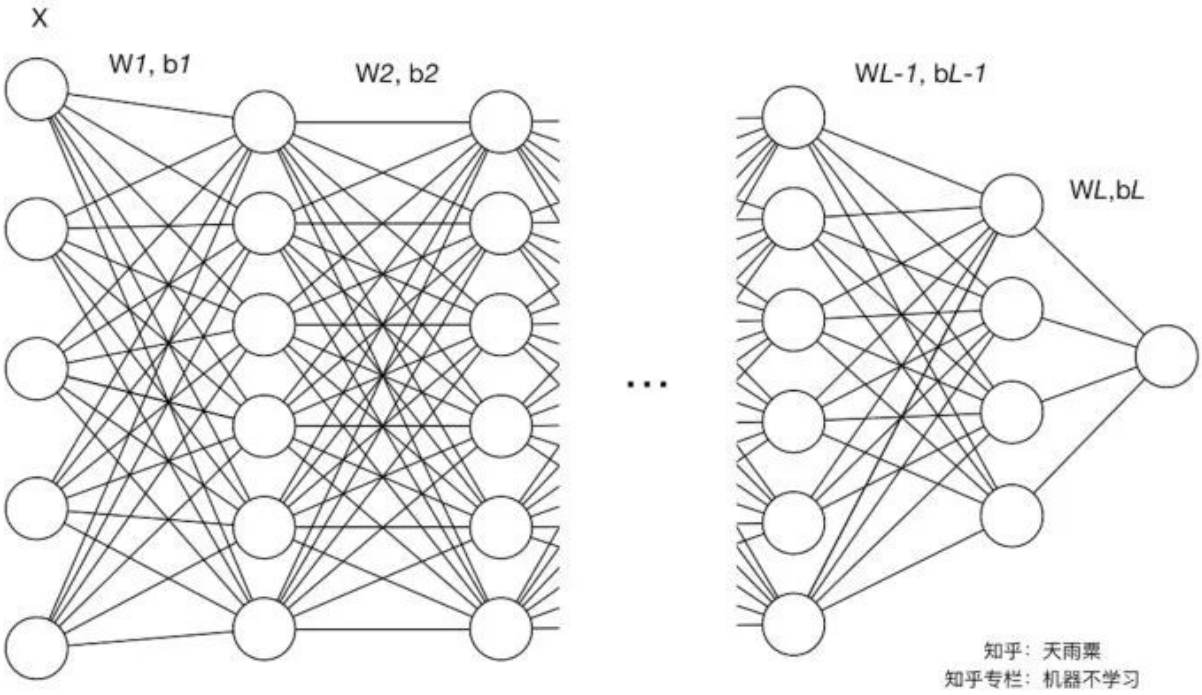
理论部分主要参考2015年Google的Sergey Ioffe与Christian Szegedy的论文内容，并辅以吴恩达Coursera课程与其它博主的资料。所有参考内容链接均见于文章最后参考链接部分。

1、提出背景

1.1 炼丹的困扰

在深度学习中，由于问题的复杂性，我们往往会使用较深层数的网络进行训练，相信很

多炼丹的朋友都对调参的困难有所体会，尤其是对深层神经网络的训练调参更是困难且复杂。在这个过程中，我们需要去尝试不同的学习率、初始化参数方法（例如Xavier初始化）等方式来帮助我们的模型加速收敛。深度神经网络之所以如此难训练，其中一个重要原因就是网络中层与层之间存在高度的关联性与耦合性。下图是一个多层的神经网络，层与层之间采用全连接的方式进行连接。



我们规定左侧为神经网络的底层，右侧为神经网络的上层。那么网络中层与层之间的关联性会导致如下的状况：随着训练的进行，网络中的参数也随着梯度下降在不停更新。一方面，当底层网络中参数发生微弱变化时，由于每一层中的线性变换与非线性激活映射，这些微弱变化随着网络层数的加深而被放大（类似蝴蝶效应）；另一方面，参数的变化导致每一层的输入分布会发生改变，进而上层的网络需要不停地去适应这些分布变化，使得我们的模型训练变得困难。上述这一现象叫做Internal Covariate Shift。

1.2 什么是Internal Covariate Shift

Batch Normalization的原论文作者给了Internal Covariate Shift一个较规范的定义：在深层网络训练的过程中，由于网络中参数变化而引起内部结点数据分布发生变化的这一过程被称作Internal Covariate Shift。

这句话该怎么理解呢？我们同样以1.1中的图为例，我们定义每一层的线性变换为 ，其中 l 代表层数；非线性变换为 ，其中 $g^{[l]}(\cdot)$ 为第 l 层的激活函数。

$$Z^{[l]} = W^{[l]} \times inputb^{[l]}$$

随着梯度下降的进行，每一层的参数 $W^{[l]}$ 与 $b^{[l]}$ 都会被更新，那么 $Z^{[l]}$ 的分布也就发生了改变，进而 $A^{[l]}$ 也同样出现分布的改变。而 $A^{[l]}$ 作为第 $l1$ 层的输入，意味着 $l1$ 层就需要去不停适应这种数据分布的变化，这一过程就被叫做Internal Covariate Shift。

$$A^{[l]} = g^{[l]}(Z^{[l]})$$

1.3 Internal Covariate Shift会带来什么问题？

(1) 上层网络需要不停调整来适应输入数据分布的变化，导致网络学习速度的降低

我们在上面提到了梯度下降的过程会让每一层的参数 $W^{[l]}$ 和 $b^{[l]}$ 发生变化，进而使得每一层的线性与非线性计算结果分布产生变化。后层网络就要不停地去适应这种分布变化，这个时候就会使得整个网络的学习速率过慢。

(2) 网络的训练过程容易陷入梯度饱和区，减缓网络收敛速度

当我们在神经网络中采用饱和激活函数（saturated activation function）时，例如sigmoid，tanh激活函数，很容易使得模型训练陷入梯度饱和区（saturated regime）。随着模型训练的进行，我们的参数 $W^{[l]}$ 会逐渐更新并变大，此时 就会随之变大，并且 $Z^{[l]}$ 还受到更底层网络参数 的影响，随着网络层数的加深，$Z^{[l]}$ 很容易陷入梯度饱和区，此时梯度会变得很小甚至接近于0，参数的更新速度就会减慢，进而就会放慢网络的收敛速度。

$$Z^{[l]} = W^{[l]} A^{[l-1]} b^{[l]}$$

对于激活函数梯度饱和问题，有两种解决思路。第一种就是更为非饱和性激活函数，例如线性整流函数ReLU可以在一定程度上解决训练进入梯度饱和区的问题。另一种思路是，我们可以让激活函数的输入分布保持在一个稳定状态来尽可能避免它们陷入梯度饱和区，这也就是Normalization的思路。

$$W^{[1]}, W^{[2]}, \ldots, W^{[l-1]}$$

1.4 我们如何减缓Internal Covariate Shift？

要缓解ICS的问题，就要明白它产生的原因。ICS产生的原因是由于参数更新带来的网络中每一层输入值分布的改变，并且随着网络层数的加深而变得更加严重，因此我们可以通过固定每一层网络输入值的分布来对减缓ICS问题。

(1) 白化（Whitening)

白化（Whitening）是机器学习里面常用的一种规范化数据分布的方法，主要是PCA白化与ZCA白化。白化是对输入数据分布进行变换，进而达到以下两个目的：

- 使得输入特征分布具有相同的均值与方差。其中PCA白化保证了所有特征分布均值为0，方差为1；而ZCA白化则保证了所有特征分布均值为0，方差相同；
- 去除特征之间的相关性。

通过白化操作，我们可以减缓ICS的问题，进而固定了每一层网络输入分布，加速网络训练过程的收敛（LeCun et al.,1998b；Wiesler&Ney,2011）。

(2) Batch Normalization提出

既然白化可以解决这个问题，为什么我们还要提出别的解决办法？当然是现有的方法具有一定的缺陷，白化主要有以下两个问题：

- 白化过程计算成本太高，并且在每一轮训练中的每一层我们都需要做如此高成本计算的白化操作；
- 白化过程由于改变了网络每一层的分布，因而改变了网络层中本身数据的表达能力。底层网络学习到的参数信息会被白化操作丢失掉。

既然有了上面两个问题，那我们的解决思路就很简单，一方面，我们提出的normalization方法要能够简化计算过程；另一方面又需要经过规范化处理后让数据尽可能保留原始的表达能力。于是就有了简化+改进版的白化——Batch Normalization。

2、Batch Normalization

2.1 思路

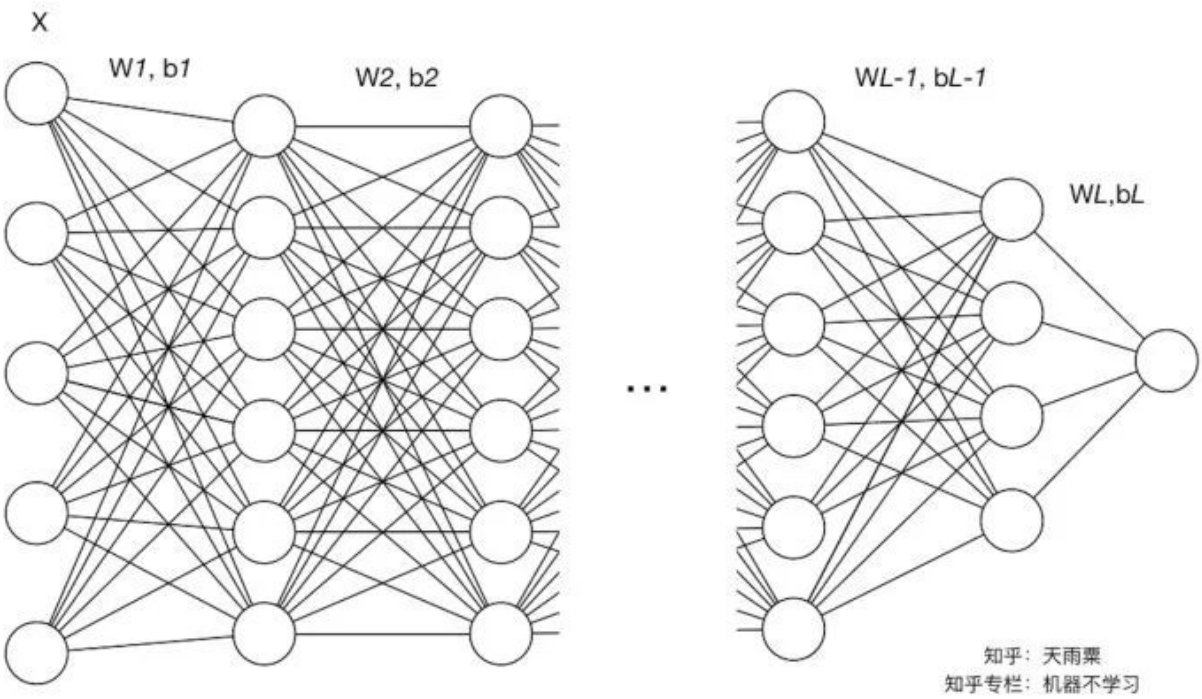
既然白化计算过程比较复杂，那我们就简化一点，比如我们可以尝试单独对每个特征进行normalizaiton就可以了，让每个特征都有均值为0，方差为1的分布就OK。另一个问题，既然白化操作减弱了网络中每一层输入数据表达能力，那我就再加个线性变换操作，让这些数据再能够尽可能恢复本身的表达能力就好了。因此，基于上面两个解决问题的思路，作者提出了Batch Normalization，下一部分来具体讲解这个算法步骤。

2.2 算法

在深度学习中，由于采用full batch的训练方式对内存要求较大，且每一轮训练时间过长；我们一般都会采用对数据做划分，用mini-batch对网络进行训练。因此，Batch Normalization也就在mini-batch的基础上进行计算。

2.2.1 参数定义

我们依旧以下图这个神经网络为例。我们定义网络总共有 L 层（不包含输入层）并定义如下符号：



参数相关：

- l ：网络中的层标号
- L ：网络中的最后一层或总层数
- d_l ：第 l 层的维度，即神经元结点数

- $W^{[l]}$ ：第 l 层的权重矩阵，
- $b^{[l]}$ ：第 l 层的偏置向量， $b^{[l]} \in \mathbb{R}^{d_l \times 1}$
- $Z^{[l]}$ ：第 l 层的线性计算结果，
- $g^{[l]}(\cdot)$ ：第 l 层的激活函数
- $A^{[l]}$ ：第 l 层的非线性激活结果，

$$W^{[l]} \in \mathbb{R}^{d_l \times d_{l-1}}$$

$$Z^{[l]} = W^{[l]} \times inputb^{[l]}$$

$$A^{[l]} = g^{[l]}(Z^{[l]})$$

样本相关：

- M ：训练样本的数量
- N ：训练样本的特征数
- X ：训练样本集， （注意这里 X 的一列是一个样本）
- m ：batch size，即每个batch中样本的数量
- $\chi^{(i)}$ ：第 i 个mini-batch的训练数据， ，其中

$$X = \{x^{(1)}, x^{(2)}, \cdots, x^{(M)}\}, \ X \in \mathbb{R}^{N \times M}$$

$$X = \{\chi^{(1)}, \chi^{(2)}, \cdots, \chi^{(k)}\}$$

2.2.2 算法步骤

介绍算法思路沿袭前面BN提出的思路来讲。第一点，对每个特征进行独立的normalization。我们考虑一个batch的训练，传入m个训练样本，并关注网络中的某一层，忽略上标 l 。

$\chi^{(i)} \in \mathbb{R}^{N \times m}$

$Z \in \mathbb{R}^{d_l \times m}$

我们关注当前层的第 j 个维度，也就是第 j 个神经元结点，则有 $Z_j \in \mathbb{R}^{1 \times m}$ 。我们当前维度进行规范化：

其中 ϵ 是为了防止方差为0产生无效计算。

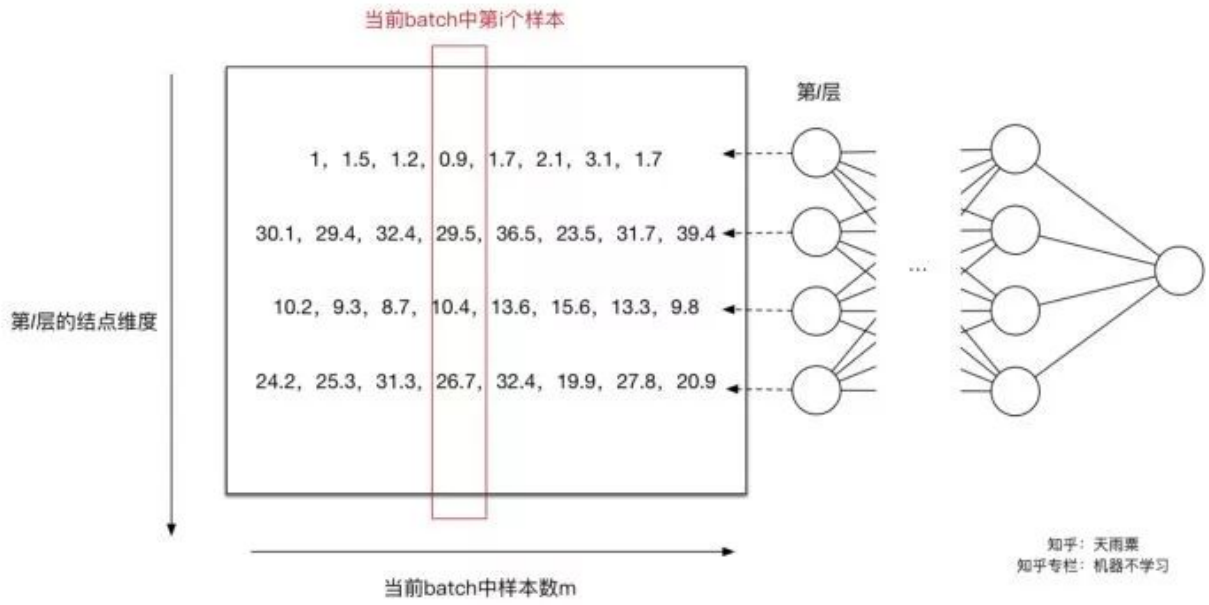
下面我们再来结合个具体的例子来进行计算。下图我们只关注第 l 层的计算结果，左边的矩阵是 线性计算结果，还未进行激活函数的非线性变换。此时每一列是一个样本，图中可以看到共有8列，代表当前训练样本的batch中共有8个样本，每一行代表当前 l 层神经元的一个节点，可以看到当前 l 层共有4个神经元结点，即第 l 层维度为4。我们可以看到，每行的数据分布都不同。

$$\mu_j = \frac{1}{m} \sum_{i=1}^{m} Z_j^{(i)}$$

$$\sigma_j^2 = \frac{1}{m} \sum_{i=1}^{m} (Z_j^{(i)} - \mu_j)^2$$

$$\hat{Z}_j = \frac{Z_j - \mu_j}{\sqrt{\sigma_j^2 \epsilon}}$$

$$Z^{[l]} = W^{[l]} A^{[l-1]} b^{[l]}$$



对于第一个神经元，我们求得 $\mu_1 = 1.65$ ， $\sigma_1^2 = 0.44$ （其中 $\epsilon = 10^{-8}$ ），此时我们利用 μ_1, σ_1^2 对第一行数据（第一个维度）进行normalization得到新的值

$$[-0.98, -0.23, -0.68, -1.13, 0.08, 0.68, 2.19, 0.08]$$

。同理我们可以计算出其他输入维度归一化后的值。如下图：

通过上面的变换，我们解决了第一个问题，即用更加简化的方式来对数据进行规范化，使得第 l 层的输入每个特征的分布均值为0，方差为1。
如同上面提到的，Normalization操作我们虽然缓解了ICS问题，让每一层网络的输入数据分布都变得稳定，但却导致了数据表达能力的缺失。也就是我们通过变换操作改变了原有数据的信息表达（representation ability of the network），使得底层网络学习到的参数信息丢失。另一方面，通过让每一层的输入分布均值为0，方差为1，会使得输入在经过sigmoid或tanh激活函数时，容易陷入非线性激活函数的线性区域。
因此，BN又引入了两个可学习（learnable）的参数 γ 与 β 。这两个参数的引入是为了恢复数据本身的表达能力，对规范化后的数据进行线性变换，即 。特别地，当 时，可以实现等价变换（identity transform）并且保留了原始输入特征的分布信息。

$$\tilde{Z}_j = \gamma_j \hat{Z}_j \beta_j$$

$$\gamma^2 = \sigma^2, \beta = \mu$$

通过上面的步骤，我们就在一定程度上保证了输入数据的表达能力。
以上就是整个Batch Normalization在模型训练中的算法和思路。

补充：在进行normalization的过程中，由于我们的规范化操作会对减去均值，因此，偏置项 b 可以被忽略掉或可以被置为0，即

$$BN(Wub) = BN(Wu)$$

2.2.3 公式

对于神经网络中的第 l 层，我们有：

3、测试阶段如何使用Batch Normalization？

我们知道BN在每一层计算的 μ 与 σ^2 都是基于当前batch中的训练数据，但是这就带来了一个问题：我们在预测阶段，有可能只需要预测一个样本或很少的样本，没有像训练样本中那么多的数据，此时 μ 与 σ^2 的计算一定是有偏估计，这个时候我们该如何进行计算呢？

利用BN训练好模型后，我们保留了每组mini-batch训练数据在网络中每一层的 μ_{batch} 与 σ^2_{batch} 。此时我们使用整个样本的统计量来对Test数据进行归一化，具体来说使用均值与方差的无偏估计：

得到每个特征的均值与方差的无偏估计后，我们对test数据采用同样的normalization方法：

另外，除了采用整体样本的无偏估计外。吴恩达在Coursera上的Deep Learning课程指出可以对train阶段每个batch计算的mean/variance采用指数加权平均来得到test阶段mean/variance的估计。

4、Batch Normalization的优势

Batch Normalization在实际工程中被证明了能够缓解神经网络难以训练的问题，BN具有的有事可以总结为以下三点：

(1) BN使得网络中每层输入数据的分布相对稳定，加速模型学习速度

BN通过规范化与线性变换使得每一层网络的输入数据的均值与方差都在一定范围内，使得后一层网络不必不断去适应底层网络中输入的变化，从而实现了网络中层与层之间的解耦，允许每一层进行独立学习，有利于提高整个神经网络的学习速度。

(2) BN使得模型对网络中的参数不那么敏感，简化调参过程，使得网络学习更加稳定

在神经网络中，我们经常会谨慎地采用一些权重初始化方法（例如Xavier）或者合适的学习率来保证网络稳定训练。

当学习率设置太高时，会使得参数更新步伐过大，容易出现震荡和不收敛。但是使用BN的网络将不会受到参数数值大小的影响。例如，我们对参数 W 进行缩放得到 aW 。对于缩放前的值 Wu ，我们设其均值为 μ_1 ，方差为 σ_1^2 ；对于缩放值 aWu ，设其均值为 μ_2 ，方差为 σ_2^2 ，则我们有：

$\mu_2 = a\mu_1$ ， $\sigma_2^2 = a^2\sigma_1^2$

我们忽略 ϵ ，则有：

$$BN(aWu) = \gamma \cdot \frac{aWu - \mu_2}{\sqrt{\sigma_2^2}}\beta = \gamma \cdot \frac{aWu - a\mu_1}{\sqrt{a^2\sigma_1^2}}\beta = \gamma \cdot \frac{Wu - \mu_1}{\sqrt{\sigma_1^2}}\beta = BN(Wu)$$

$$Z^{[l]} = W^{[l]}A^{[l-1]}b^{[l]}$$

$$\mu = \frac{1}{m}\sum_{i=1}^{m} Z^{[l](i)}$$

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(Z^{[l](i)} - \mu)^2$$

$$\tilde{Z}^{[l]} = \gamma \cdot \frac{Z^{[l]} - \mu}{\sqrt{\sigma^2 \epsilon}}\beta$$

$$A^{[l]} = g^{[l]}(\tilde{Z}^{[l]})$$

$$\mu_{test} = \mathbb{E}(\mu_{batch})$$

$$\sigma^2_{test} = \frac{m}{m-1}\mathbb{E}(\sigma^2_{batch})$$

$$BN(X_{test}) = \gamma \cdot \frac{X_{test} - \mu_{test}}{\sqrt{\sigma^2_{test}\epsilon}}\beta$$

注：公式中的 u 是当前层的输入，也是前一层的输出；不是下标啊旁友们！

$$\frac{\partial BN((aW)u)}{\partial u} = \gamma \cdot \frac{aW}{\sqrt{\sigma_2^2}} = \gamma \cdot \frac{aW}{\sqrt{a^2\sigma_1^2}} = \frac{\partial BN(Wu)}{\partial u}$$

$$\frac{\partial BN((aW)u)}{\partial (aW)} = \gamma \cdot \frac{u}{\sqrt{\sigma_2^2}} = \gamma \cdot \frac{u}{a\sqrt{\sigma_1^2}} = \frac{1}{a} \cdot \frac{\partial BN(Wu)}{\partial W}$$

我们可以看到，经过BN操作以后，权重的缩放值会被“抹去”，因此保证了输入数据分布稳定在一定范围内。另外，权重的缩放并不会影响到对 u 的梯度计算；并且当权重越大时，即 a 越大，$\frac{1}{a}$ 越小，意味着权重 W 的梯度反而越小，这样BN就保证了梯度不会依赖于参数的scale，使得参数的更新处在更加稳定的状态。

因此，在使用Batch Normalization之后，抑制了参数微小变化随着网络层数加深被放大的问题，使得网络对参数大小的适应能力更强，此时我们可以设置较大的学习率而不用过于担心模型divergence的风险。

(3) BN允许网络使用饱和性激活函数（例如sigmoid，tanh等），缓解梯度消失问题

在不使用BN层的时候，由于网络的深度与复杂性，很容易使得底层网络变化累积到上层网络中，导致模型的训练很容易进入到激活函数的梯度饱和区；通过normalize操作可以让激活函数的输入数据落在梯度非饱和区，缓解梯度消失的问题；另外通过自适应学习 γ 与 β 又让数据保留更多的原始信息。

(4) BN具有一定的正则化效果

在Batch Normalization中，由于我们使用mini-batch的均值与方差作为对整体训练样本均值与方差的估计，尽管每一个batch中的数据都是从总体样本中抽样得到，但不同mini-batch的均值与方差会有所不同，这就为网络的学习过程中增加了随机噪音，与Dropout通过关闭神经元给网络训练带来噪音类似，在一定程度上对模型起到了正则化的效果。

另外，原作者通过也证明了网络加入BN后，可以丢弃Dropout，模型也同样具有很好的泛化效果。

理论部分到此结束了，下一篇让我们来看看Batch Normalization在实际应用中对网络有什么作用呢？

仓库地址共享：

https://github.com/yizhen20133868/NLP-Conferences-Code

END