Author：Liu Jian

Time：2021-03-19

Batch Normalization 个人总结

多层神经网络的各层参数之间具有很强的非线性，即以参数为自变量 (样本和标记固定)，即使在很小的范围内，都无法用线性函数近似损失函数，参数 (自变量) 间都会存在明显的高阶交互。这就会使梯度下降算法的前提假设不成立，即损失函数对各参数的微小变化的响应并非线性，各参数的优化效果不会简单地线性叠加而会产生复杂的高阶交互，可能会使优化效果会相互抵消，以致难以进行，即出现所谓的内部协变量偏移 (Internal Covariate Shift)。Batch Normalization 在仿射变换后，激活函数前加入 BN 层以解决上述问题。假设我们对第l 层进行 BN 操作 (l 层的所有 n 个神经元都会被进行独立的 BN 处理，下面以其中的一个神经元 i 为例进行说明)，min batch 所含的样本容量为 m ，则 BN 层的操作可概括为以下两步：

1. 标准化：并行地将 m 个数据输入网络，直到被 $l-1$ 层处理完毕并按神经元 $i\ (i=1,2,\cdots,n)$ 的当前的权重向量作仿射变换，得到 m 个输出；对这 m 个输出进行标准化，使得它们的样本均值为 0，样本方差为 1；
2. 缩放和偏移：对标准化后的输出进行缩放和偏移 (缩放的倍数和偏移的量为待优化参数)，作为神经元 $i\ (i=1,2,\cdots,n)$ 激活函数的输入。

我们强调：

1. Batch Normalization 时，normalize 的是单个神经元的一个min-batch的样本 (我们会将这一层的所有神经元都进行normalize，但需要强调的是，这些神经元的normalize是相互独立的)，而不对是单个样本在网络某一层的多个输出作 normalize (这其实就是层归一化，Layer Normalization，用于 RNN，参见邱锡鹏《神经网络与深度学习》7.5.2)；normalize的目的是为了使当前层的输入分布稳定，避免底层参数的变化对当前层的输入分布产生过大的影响，从而使优化当前层参数时可以在一定程度上屏蔽掉底层的影响，缓解内部协变量偏移；
2. 为了不降低模型的表达能力，我们随后又进行了缩放和偏移，注意，先标准化后又进行缩放和偏移这看似矛盾且多次一举的操作 (若取缩放倍数和偏移量为样本标准差和均值，则又会将标准化的量还原) 实际上并不矛盾，而是一种重参数化技巧。加入 BN 层后，将“原先底层复杂作用产生的输出”替换为“zero mean unit variance的输出+缩放和偏移两个控制参数”，相当于对原先底层的输出进行建模，其表达能力不再由底层网络复杂的参数关系给定而是打包出来，由抽象提炼出的缩放参数和偏移参数控制，重参数化既降低了训练难度 (zero mean unit variance的稳定输出)，又兼顾了表达能力 (缩放和偏移)；

3. 含有 BN 的神经网络训练时必须要多个样本一起训练，也就是说，各样本点的训练并不是相互独立的，而是相互联系的，训练是考虑了样本的分布的 (这会起到正则化的效果，因为训练时，神经网络对一个样本的预测不仅和该样本自身相关，也和同一批次中的其他样本相关，使得神经网络不会过拟合到某个特定样本，从而提高网络的泛化能力)；训练算法依然是梯度下降法，只不过神经网络的结构是包含 BN 层的，反向传播也要经过 BN 层，使用链式求导法则计算即可；
4. 测试时，缩放倍数和偏移量已经在训练阶段确定，无需担心，但训练时每次normalize所用的均值和方差是基于min batch计算得到的，那么预测时对单个待预测样本，如何 normalize 呢？答案是我们可以基于整个数据集计算均值和方差；
5. min batch 应还有足够多的样本；
6. 因为批量归一化本身具有平移变换，所以仿射变换不再需要偏置。

> 下文的内容主要为转载，在原文的基础上作了一定修正，写的很清晰，实际上主要参考文献就是 arxiv-Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift。

Batch Normalization详解

> 乍一看到某个问题，你会觉得很简单，其实你并没有理解其复杂性。当你把问题搞清楚之后，又会发现真的很复杂，于是你就拿出一套复杂的方案来。实际上，你的工作只做了一半，大多数人也都会到此为止……但是，真正伟大的人还会继续向前，直至找到问题的关键和深层次原因，然后再拿出一个优雅的、堪称完美的有效方案。—— 乔布斯

作者博客：blog.shinelee.me | 博客园 | CSDN

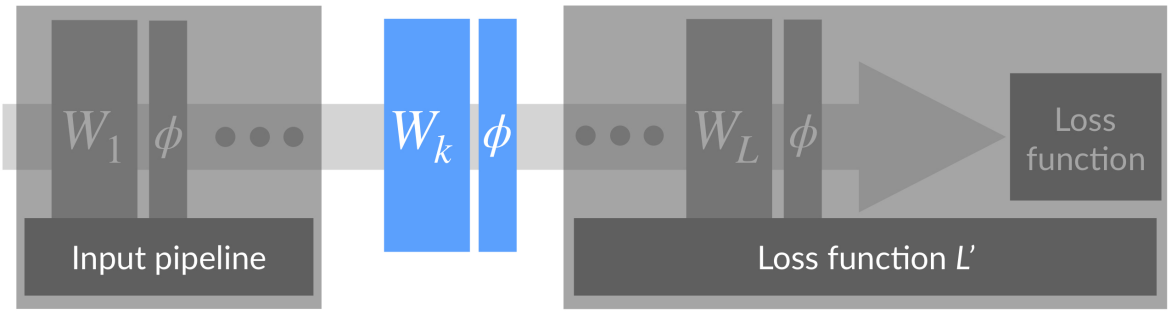
动机

在博文《**为什么要做特征归一化/标准化？** 博客园 | csdn | blog》中，我们介绍了对输入进行Standardization后，梯度下降算法更容易选择到合适的（较大的）学习率，下降过程会更加稳定。

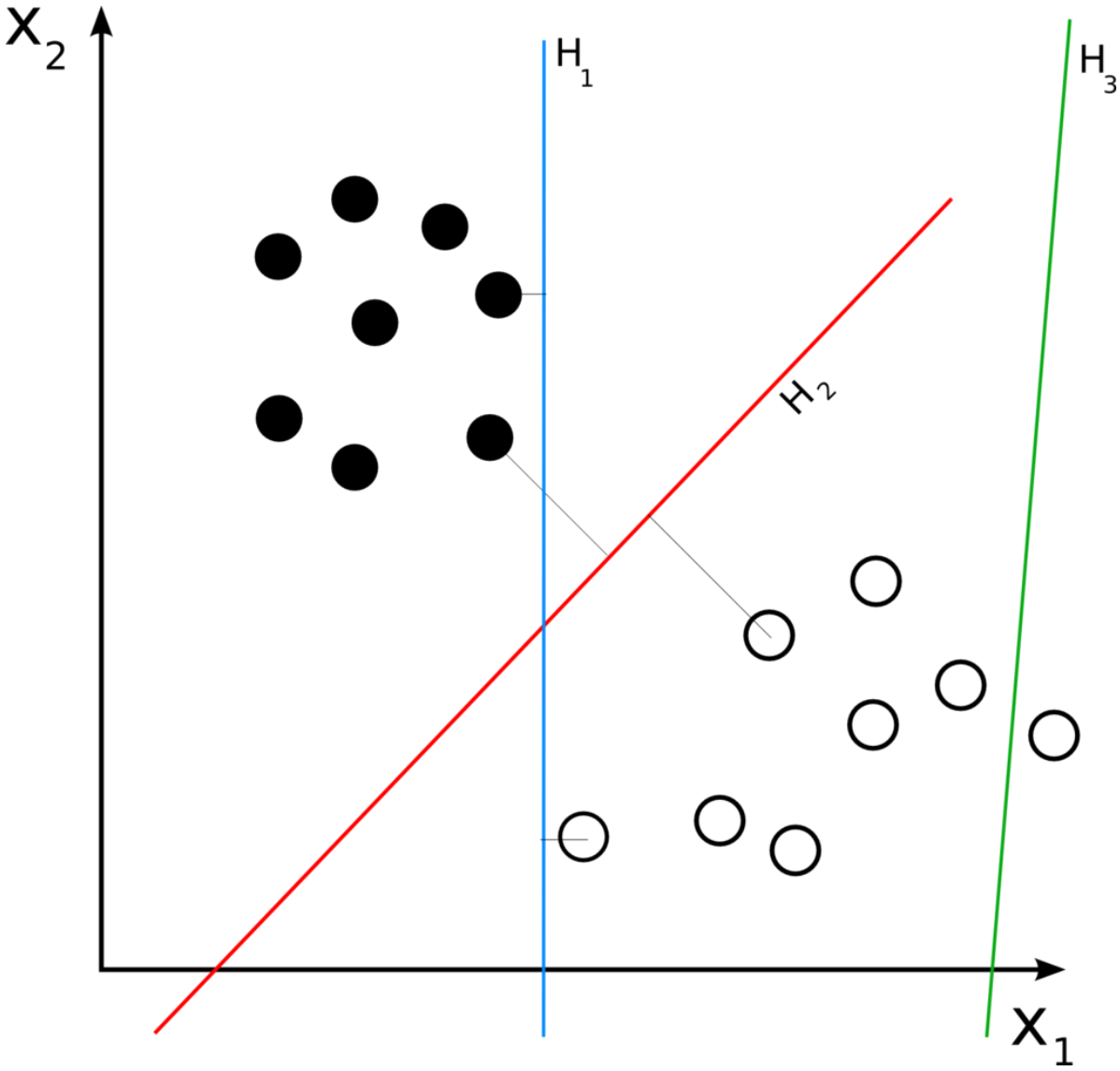
在博文《**网络权重初始化方法总结（下）：Lecun、Xavier与He Kaiming** 博客园 | csdn | blog》中，我们介绍了如何通过权重初始化让网络在训练之初保持激活层的输出（输入）为zero mean unit variance分布，以减轻梯度消失和梯度爆炸。

但在训练过程中，权重在不断更新，导致激活层输出(输入)的分布会一直变化，可能无法一直保持zero mean unit variance分布，还是有梯度消失和梯度爆炸的可能，直觉上感到，这可能是个问题。下面具体分析。

单层视角



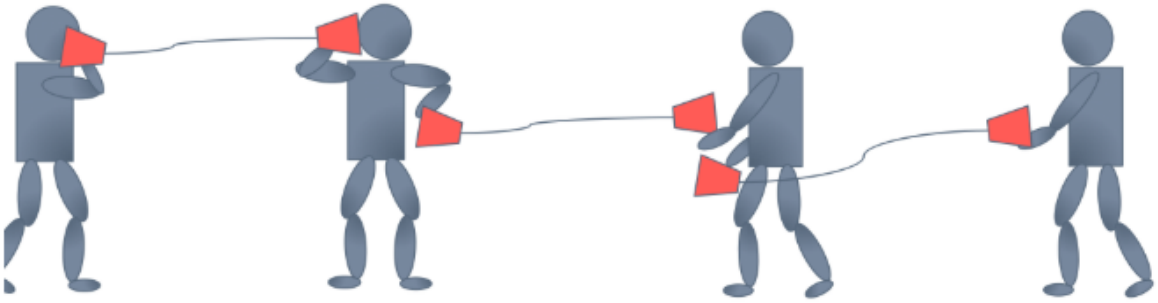
神经网络可以看成是上图形式，对于中间的某一层，其前面的层可以看成是对输入的处理，后面的层可以看成是损失函数。一次反向传播过程会同时更新所有层的权重$W_1, W_2, \ldots, W_L$，前面层权重的更新会改变当前层输入的分布，**而跟据反向传播的计算方式，我们知道，对W_k的更新是在假定其输入不变的情况下进行的。**如果假定第k层的输入节点只有2个，对第k层的某个输出节点而言，相当于一个线性模型$y = w_1x_1 + w_2x_2 + b$，如下图所示，

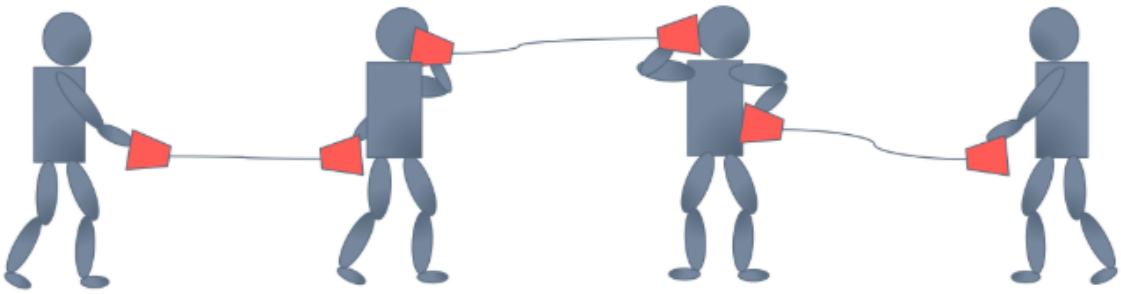


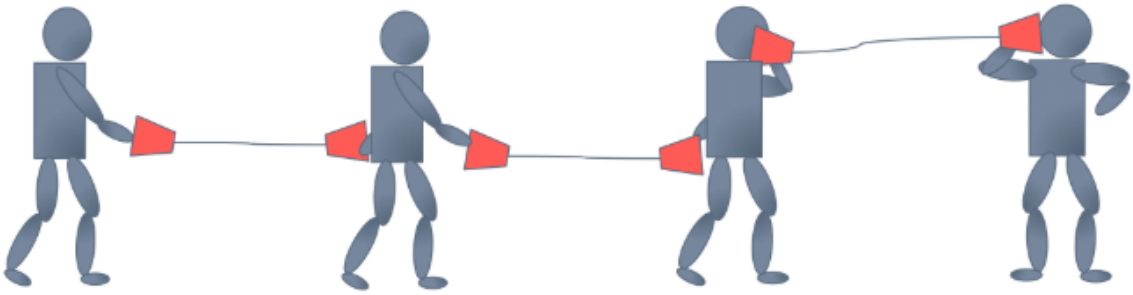
假定当前输入x_1和x_2的分布如图中圆点所示，本次更新的方向是将直线H_1更新成H_2，本以为切分得不错，但是当前面层的权重更新完毕，当前层输入的分布换成了另外一番样子，直线相对输入分布的位置可能变成了H_3，下一次更新又要根据新的分布重新调整。**直线调整了位置，输入分布又在发生变化，直线再调整位置，就像是直线和分布之间的"追逐游戏"。对于浅层模型，比如SVM，输入特征的分布是固定的，即使拆分成不同的batch，每个batch的统计特性也是相近的，因此只需调整直线位置来适应输入分布，显然要容易得多。而深层模型，每层输入的分布和权重在同时变化，训练相对困难。**

多层视角

上面是从网络中单拿出一层分析，下面看一下多层的情况。在反向传播过程中，每层权重的更新是在假定其他权重不变的情况下，向损失函数降低的方向调整自己。问题在于，在一次反向传播过程中，所有的权重会同时更新，导致层间配合"缺乏默契"，每层都在进行上节所说的"追逐游戏"，而且层数越多，相互配合越困难，文中把这个现象称之为 **Internal Covariate Shift**，示意图如下。为了避免过于震荡，学习率不得不设置得足够小，足够小就意味着学习缓慢。





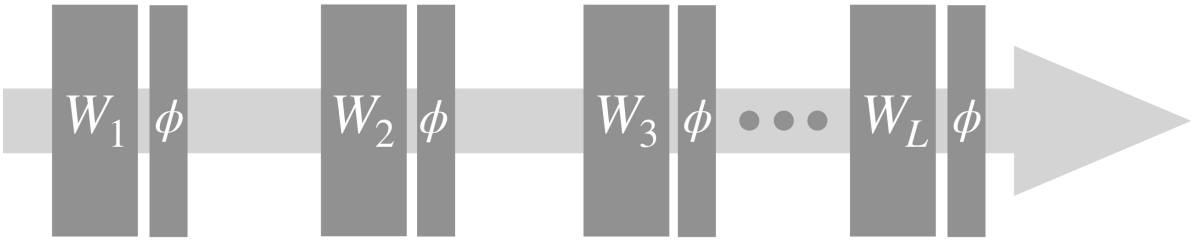


为此，希望对每层输入的分布有所控制，于是就有了**Batch Normalization**，其出发点是对每层的输入做Normalization，只有一个数据是谈不上Normalization的，所以是对一个batch的数据进行Normalization。

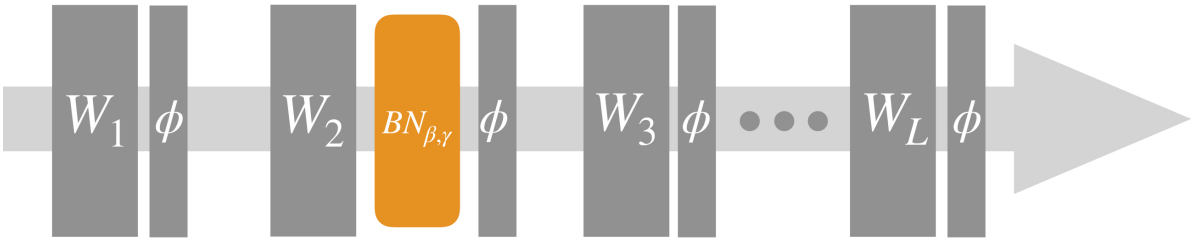
什么是Batch Normalization

Batch Normalization，简称BatchNorm或BN，翻译为“批归一化”，是神经网络中一种特殊的层，如今已是各种流行网络的标配。**在原paper中，BN被建议插入在（每个）ReLU激活层前面**，如下所示，

Standard Network



Adding a BatchNorm layer (between weights and activation function)



如果batch size为 m，则在前向传播过程中，网络中每个神经元都有 m个输出，所谓的Batch Normalization，就是对该层每个神经元的这 m个输出进行归一化再输出，具体计算方式如下，

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

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

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

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

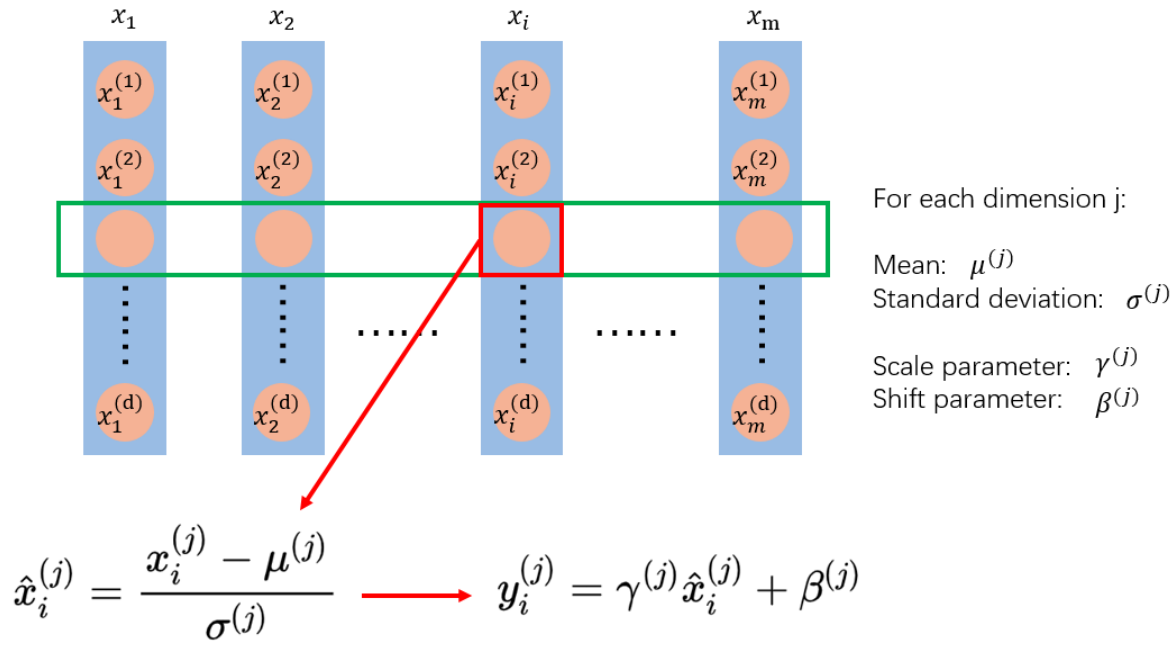
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

其操作可以分成2步，

1. **Standardization**：首先对m个 x 进行 Standardization，得到 zero mean unit variance的分布$\hat{x}$。
2. **scale and shift**：然后再对 $\hat{x}$ 进行scale and shift，缩放并平移到新的分布 y，具有新的均值 β 方差 γ。

假设BN层有 d 个输入特征 (对应低层的 d 个神经元)，则 x 可构成 $d \times m$ 大小的矩阵 X，BN层相当于以行为操作单位 (每行的元素经历相同的操作)，将其映射为另一个 $d \times m$ 大小的矩阵 Y，如下所示，



将2个过程写在一个公式里如下，

$$y_i^{(j)} = \gamma^{(j)} \frac{x_i^{(j)} - \mu^{(j)}}{\sigma^{(j)}} + \beta^{(j)}$$

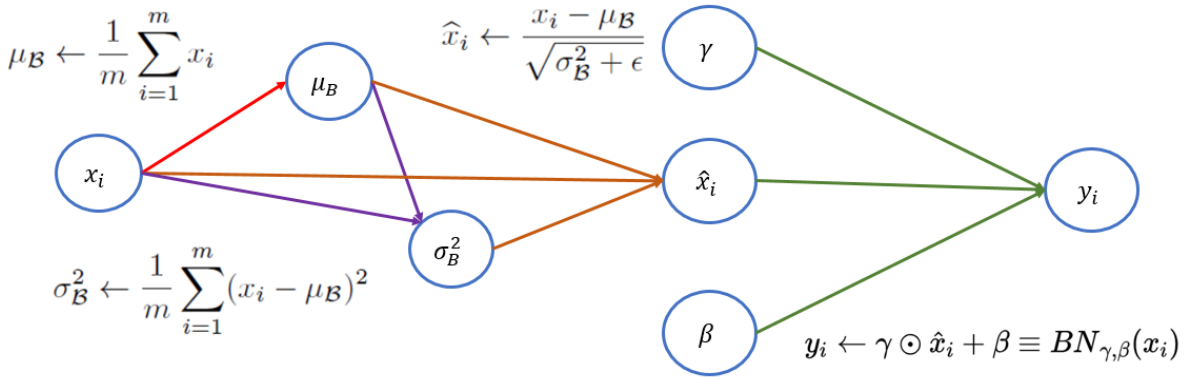
$$y_i = \gamma \odot \frac{x_i - \mu}{\sigma} + \beta \ \text{(element wise)}$$

其中，$x_i^{(j)}$表示输入当前batch的i-th样本时，该层j-th输入特征的值，$x^{(j)}$为$[x_1^{(j)}, x_2^{(j)}, \ldots, x_m^{(j)}]$构成的行向量，长度为batch size m，$\mu^{(j)}$和$\sigma^{(j)}$为该行的均值和标准差，ϵ为防止除零引入的极小量，$\gamma^{(j)}$和$\beta^{(j)}$为该行的scale和shift参数，均为标量，可知

- μ和σ为当前行的统计量，不可学习；
- γ和β为待学习的scale和shift参数向量，用于控制y_i的方差和均值；
- BN层中，X的任意两行$x^{(j)}$和$x^{(k)}$之间不存在信息交流$(j \neq k)$。
- 可见，**无论$x^{(j)}$原本的均值和方差是多少，通过BatchNorm后其均值和方差分别变为待学习的$\beta^{(j)}$和$\gamma^{(j)}$**。
- 论文中 Algorithm 1 最后一行的公式写得并不严谨，$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i)$会让人误认为$\gamma, \beta$是标量，邱锡鹏老师的《神经网络与深度学习》7.5.1 的写法
  $y_i = \gamma \odot \frac{x_i - \mu}{\sigma} + \beta \ \text{(element wise)}$ 是严谨的，$\odot$ 表示按元素相乘。

Batch Normalization的反向传播

对于目前的神经网络计算框架，**一个层要想加入到网络中，要保证其是可微的，即可以求梯度。**BatchNorm的梯度该如何求取呢？还是反向传播。



如上图所示，为简便起见，我们只画出了第i个样本的 $x_i \rightarrow y_i$ 的路径，事实上 Batch Normalization 同时优化一个batch的 m 个数据，即完整计算图包含的结点有 $\mu_{\mathcal{B}}, \sigma_{\mathcal{B}}^2, \gamma, \beta$ 和 $x_j, \hat{x}_j, y_j$ ($j = 1, 2, \cdots, i, \cdots m$)。依次进行反向传播，反向传播时若到达一个结点存在多条路径，将多条路径的结果相加即可，计算顺序如下：

1. 求取损失ℓ对BN层输出y_i的偏导$\frac{\partial \ell}{\partial y_i}$；
2. 求损失ℓ对可学习参数γ, β的偏导$\frac{\partial \ell}{\partial \gamma}$和$\frac{\partial \ell}{\partial \beta}$，用于对这两个参数的更新；求损失$\ell$对$\hat{x}_i$的偏导；
3. 求损失ℓ对$\sigma_{\mathcal{B}}^2$的偏导$\frac{\partial \ell}{\partial \sigma_B^2}$；
4. 求损失ℓ对$\mu_{\mathcal{B}}$的偏导$\frac{\partial \ell}{\partial \mu_{\mathcal{B}}}$，从已知偏导结果的结点到结点 $\mu_{\mathcal{B}}$ 存在两条路径：
    1. $\hat{x}_i \rightarrow \mu_{\mathcal{B}}$;
    2. $\sigma_{\mathcal{B}}^2 \rightarrow \mu_{\mathcal{B}}$;

    将这两条路径的结果相加即可；
5. 求损失ℓ对x_i的偏导$\frac{\partial \ell}{\partial x_i}$，从已知偏导结果的结点到结点 x_i 存在三条路径：
    1. $\hat{x}_i \rightarrow x_i$;
    2. $\sigma_{\mathcal{B}}^2 \rightarrow x_i$;
    3. $\mu_{\mathcal{B}} \rightarrow x_i$;

    将这三条路径的结果相加即可；

按上述计算顺序依次得如下公式：

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

在实际实现时，通常以矩阵或向量运算方式进行，比如逐元素相乘、沿某个axis求和、矩阵乘法等操作，具体可以参见Understanding the backward pass through Batch Normalization Layer和BatchNorm in Caffe。

Batch Normalization的预测阶段

在预测阶段，所有参数的取值是固定的，对BN层而言，意味着μ、σ、γ、β都是固定值。

γ和β比较好理解，随着训练结束，两者最终收敛，预测阶段使用训练结束时的值即可。

对于μ和σ，在训练阶段，它们为当前mini batch的统计量，随着输入batch的不同，μ和σ一直在变化。在预测阶段，输入数据可能只有1条，该使用哪个μ和σ，或者说，每个BN层的μ和σ该如何取值？**可以采用训练收敛最后几批mini batch的μ和σ的期望，作为预测阶段的μ和σ，**如下所示，

Input: Network N with trainable parameters Θ;
subset of activations $\{x^{(k)}\}_{k=1}^{K}$

Output: Batch-normalized network for inference, $N_{\text{BN}}^{\text{inf}}$

1: $N_{\text{BN}}^{\text{tr}} \leftarrow N$ // Training BN network
2: **for** $k = 1 \ldots K$ **do**
3: Add transformation $y^{(k)} = \text{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to $N_{\text{BN}}^{\text{tr}}$ (Alg. 1)
4: Modify each layer in $N_{\text{BN}}^{\text{tr}}$ with input $x^{(k)}$ to take $y^{(k)}$ instead
5: **end for**
6: Train $N_{\text{BN}}^{\text{tr}}$ to optimize the parameters $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$
7: $N_{\text{BN}}^{\text{inf}} \leftarrow N_{\text{BN}}^{\text{tr}}$ // Inference BN network with frozen // parameters
8: **for** $k = 1 \ldots K$ **do**
9: // For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.
10: Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them:

$$\text{E}[x] \leftarrow \text{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$
$$\text{Var}[x] \leftarrow \frac{m}{m-1}\text{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

11: In $N_{\text{BN}}^{\text{inf}}$, replace the transform $y = \text{BN}_{\gamma,\beta}(x)$ with $y = \frac{\gamma}{\sqrt{\text{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\text{E}[x]}{\sqrt{\text{Var}[x]+\epsilon}}\right)$
12: **end for**

Algorithm 2: Training a Batch-Normalized Network

因为Standardization和scale and shift均为线性变换，在预测阶段所有参数均固定的情况下，参数可以合并成$y = k \odot x + b$的形式，如上图中行号11所示。

Batch Normalization的作用

使用Batch Normalization，可以获得如下好处，

- **可以使用更大的学习率**，训练过程更加稳定，极大提高了训练速度。

- **可以将bias置为0**，因为Batch Normalization的Standardization过程会移除直流分量，所以不再需要bias。
- **对权重初始化不再敏感**，通常权重采样自0均值某方差的高斯分布，以往对高斯分布的方差设置十分重要，有了Batch Normalization后，对与同一个输出节点相连的权重进行放缩，其标准差σ也会放缩同样的倍数，相除抵消。
- **对权重的尺度不再敏感**，理由同上，尺度统一由γ参数控制，在训练中决定。
- **深层网络可以使用sigmoid和tanh了**，理由同上，BN抑制了梯度消失。
- **Batch Normalization具有某种正则作用，不需要太依赖dropout，减少过拟合。**

几个问题

卷积层如何使用BatchNorm?

> For convolutional layers, we additionally want the normalization to obey the convolutional property – **so that different elements of the same feature map, at different locations, are normalized in the same way.** To achieve this, we jointly normalize all the activations in a mini-batch, over all locations.
>
> ...
>
> so for a mini-batch of size m and feature maps of size $p \times q$, we use the effective mini-batch of size $m\prime = |B| = m \cdot pq$. We learn a pair of parameters $\gamma^{(k)}$ and $\beta^{(k)}$ per feature map, rather than per activation.
>
> —— Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

1个卷积核产生1个feature map，1个feature map有1对γ和β参数，同一batch同channel的feature map共享同一对γ和β参数，若卷积层有n个卷积核，则有n对γ和β参数。

没有scale and shift过程可不可以?

BatchNorm有两个过程，Standardization和scale and shift，前者是机器学习常用的数据预处理技术，在浅层模型中，只需对数据进行Standardization即可，Batch Normalization可不可以只有Standardization呢?

答案是可以，但网络的表达能力会下降。

直觉上理解，**浅层模型中，只需要模型适应数据分布即可。**对深度神经网络，每层的输入分布和权重要相互协调，强制把分布限制在zero mean unit variance并不见得是最好的选择，加入参数γ和β，对输入进行scale and shift，**有利于分布与权重的相互协调**，特别地，令$\gamma = \mathbf{1}, \beta = \mathbf{0}$等价于只用Standardization，令$\gamma = \sigma, \beta = \mu$等价于没有BN层，scale and shift涵盖了这2种特殊情况，在训练过程中决定什么样的分布是适合的，所以使用scale and shift增强了网络的表达能力。

表达能力更强，在实践中性能就会更好吗？并不见得，就像曾经参数越多不见得性能越好一样。在caffenet-benchmark-batchnorm中，作者实验发现没有scale and shift性能可能还更好一些，图见下一小节。

BN层放在ReLU前面还是后面?

原paper建议将BN层放置在ReLU前，因为ReLU激活函数的输出非负，不能近似为高斯分布。

> The goal of Batch Normalization is to achieve a stable distribution of activation values throughout training, and in our experiments **we apply it before the nonlinearity since that is where matching the first and second moments is more likely to result in a stable distribution.**

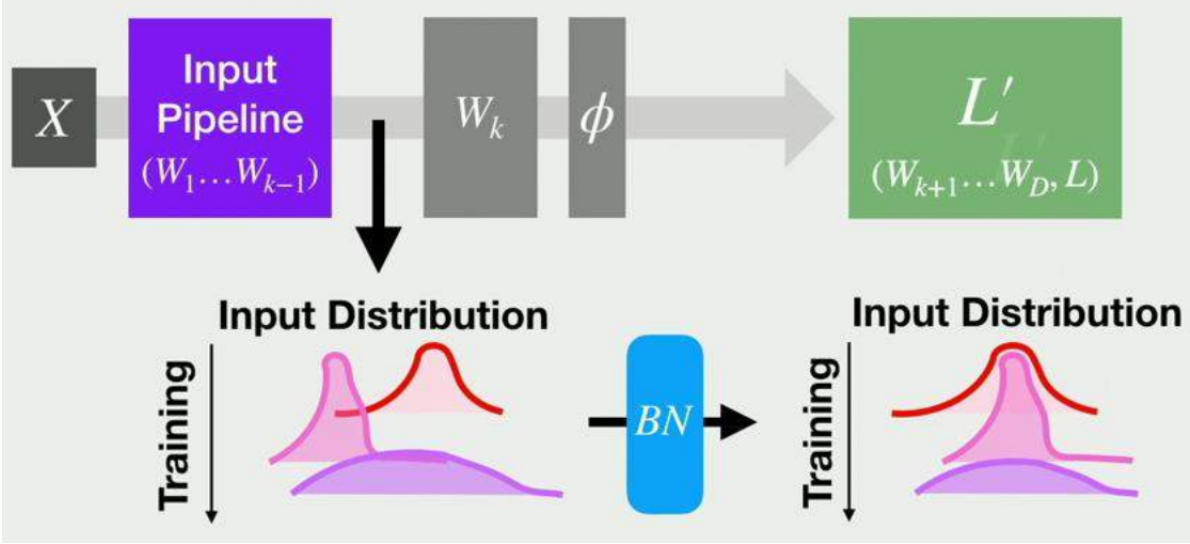
> —— Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

但是，在**caffenet-benchmark-batchnorm**中，作者基于caffenet在ImageNet2012上做了如下对比实验，

Name	Accuracy	LogLoss	Comments
Before	0.474	2.35	As in paper
Before + scale&bias layer	0.478	2.33	As in paper
After	**0.499**	**2.21**	
After + scale&bias layer	0.493	2.24	

实验表明，放在前后的差异似乎不大，**甚至放在ReLU后还好一些。**

放在ReLU后相当于直接对每层的输入进行归一化，如下图所示，这与浅层模型的Standardization是一致的。



caffenet-benchmark-batchnorm中，还有BN层与不同激活函数、不同初始化方法、dropout等排列组合的对比实验，可以看看。

所以，BN究竟应该放在激活的前面还是后面？以及，BN与其他变量，如激活函数、初始化方法、dropout等，如何组合才是最优？**可能只有直觉和经验性的指导意见，具体问题的具体答案可能还是得实验说了算（微笑）。**

BN层为什么有效?

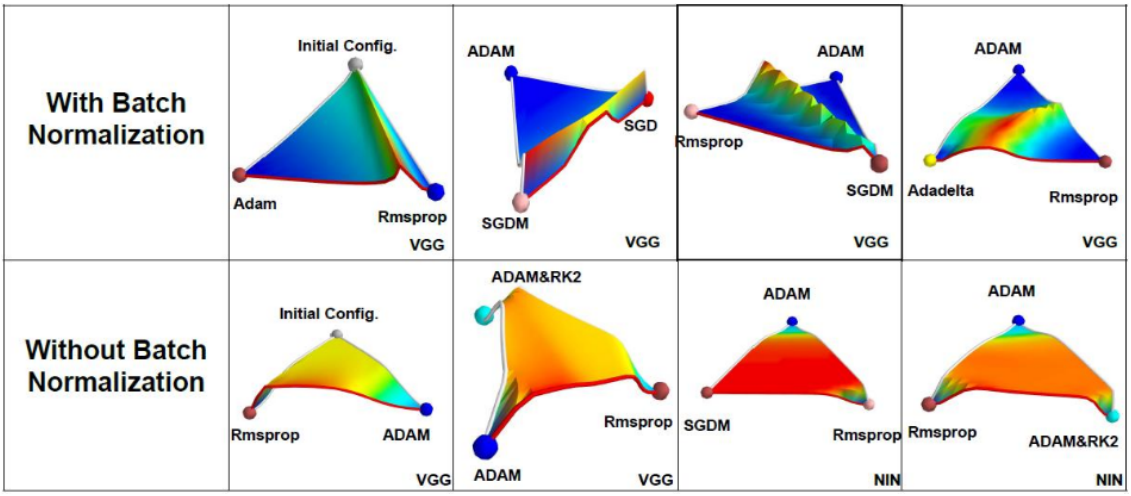
BN层的有效性已有目共睹，但为什么有效可能还需要进一步研究，这里有一些解释：

- **BN层让损失函数更平滑。**论文How Does Batch Normalization Help Optimization中，通过分析训练过程中每步梯度方向上步长变化引起的损失变化范围、梯度幅值的变化范围、光滑度的变化，认为添**加BN层后，损失函数的landscape(loss surface)变得更平滑，相比高低不平上下起伏的loss surface，平滑loss surface的梯度预测性更好，可以选取较大的步长。**如下图所示，

(a) loss landscape (b) gradient predictiveness (c) "effective" β-smoothness

Figure 4: Analysis of the optimization landscape of VGG networks. At a particular training step, we measure the variation (shaded region) in loss (a) and ℓ_2 changes in the gradient (b) as we move in the gradient direction. The "effective" β-smoothness (c) refers to the maximum difference (in ℓ_2-norm) in gradient over distance moved in that direction. There is a clear improvement in all of these measures in networks with BatchNorm, indicating a more well-behaved loss landscape. (Here, we cap the maximum distance to be $\eta = 0.4\times$ the gradient since for larger steps the standard network just performs worse (see Figure 1). BatchNorm however continues to provide smoothing for even larger distances.) Note that these results are supported by our theoretical findings (Section 4).

- **BN更有利于梯度下降。**论文[An empirical analysis of the optimization of deep network loss surfaces]中，绘制了VGG和NIN网络在有无BN层的情况下，loss surface的差异，包含初始点位置以及不同优化算法最终收敛到的local minima位置，如下图所示。**没有BN层的，其loss surface存在较大的高原，有BN层的则没有高原，而是山峰，因此更容易下降。**



- 这里再提供一个直觉上的理解，没有BN层的情况下，网络没办法直接控制每层输入的分布，其分布前面层的权重共同决定，或者说分布的均值和方差"隐藏"在前面层的每个权重中，网络若想调整其分布，需要通过复杂的反向传播过程调整前面的每个权重实现，**BN层的存在相当于将分布的均值和方差从权重中剥离了出来 (重参数化，reparametrization)，只需调整 γ 和 β 两个参数就可以直接调整分布，让分布和权重的配合变得更加容易。**

这里多说一句，论文[How Does Batch Normalization Help Optimization]中对比了标准VGG以及加了BN层的VGG每层分布随训练过程的变化，发现两者并无明显差异，认为BatchNorm并没有改善 **Internal Covariate Shift。但这里有个问题是**，两者的训练都可以收敛，对于不能收敛或者训练过程十分震荡的模型呢，其分布变化是怎样的？我也不知道，没做过实验（微笑）。

参考

- arxiv-Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift
- arxiv-How Does Batch Normalization Help Optimization?
- arxiv-An empirical analysis of the optimization of deep network loss surfaces
- talk-How does Batch Normalization Help Optimization?
- How does Batch Normalization Help Optimization?
- Understanding the backward pass through Batch Normalization Layer

- Batch Normalization — What the hey?
- Why Does Batch Normalization Work?
