详解深度学习中的Normalization，不只是BN

Juliuszh 4 天前

深度神经网络模型训练之难众所周知，其中一个重要的现象就是 Internal Covariate Shift. Batch Norm 大法自 2015 年由Google 提出之后，就成为深度学习必备之神器。自 BN 之后， Layer Norm / Weight Norm / Cosine Norm 等也横空出世。本文从 Normalization 的背景讲起，用一个公式概括 Normalization 的基本思想与通用框架，将各大主流方法一一对号入座进行深入的对比分析，并从参数和数据的伸缩不变性的角度探讨 Normalization 有效的深层原因。

本文继续沿袭上一个专题（Adam那么棒，为什么还对SGD念念不忘 —— 一个框架看懂优化算法）的写作思路，先提炼一个通用的分析框架，然后再对号入座式地梳理各大主流算法，便于我们分析比较。

目录：

以下是正文，enjoy.

1. 为什么需要 Normalization

1.1 独立同分布与白化

机器学习界的炼丹师们最喜欢的数据有什么特点？窃以为，莫过于“**独立同分布**”了，即 *independent and identically distributed*，简称为 *i.i.d.* 独立同分布并非所有机器学习模型的必然要求（比如 Naive Bayes 模型就建立在特征彼此独立的基础之上，而Logistic Regression 和 神经网络 则在非独立的特征数据上依然可以训练出很好的模型），但独立同分布的数据可以简化常规机器学习模型的训练、提升机器学习模型的预测能力，已经是一个共识。

因此，在把数据喂给机器学习模型之前，“**白化（whitening）**”是一个重要的数据预处理步骤。白化一般包含两个目的：

（1）*去除特征之间的相关性* —> 独立；

（2）*使得所有特征具有相同的均值和方差* —> 同分布。

白化最典型的方法就是PCA，可以参考阅读 PCAWhitening。

1.2 深度学习中的 Internal Covariate Shift

深度神经网络模型的训练为什么会很困难？其中一个重要的原因是，深度神经网络涉及到很多层的叠加，而每一层的参数更新会导致上层的输入数据分布发生变化，通过层层叠加，高层的输入分布变化会非常剧烈，这就使得高层需要不断去重新适应底层的参数更新。为了训好模型，我们需要非常谨慎地去设定学习率、初始化权重、以及尽可能细致的参数更新策略。

Google 将这一现象总结为 Internal Covariate Shift，简称 ICS. 什么是 ICS 呢？@魏秀参 在一个回答中做出了一个很好的解释：

> 大家都知道在统计机器学习中的一个经典假设是“源空间（source domain）和目标空间（target domain）的数据分布（distribution）是一致的”。如果不一致，那么就出现了新的机器学习问题，如 transfer learning / domain adaptation 等。而 covariate shift 就是分布不一致假设之下的一个分支问题，它是指源空间和目标空间的条件概率是一致的，但是其边缘概率不同，即：对所有 $\boldsymbol{x} \in \boldsymbol{\mathcal{X}}$，
>
> $$\boldsymbol{P_s(Y|X=x) = P_t(Y|X=x)}$$
>
> 但是
>
> $$\boldsymbol{P_s(X) \neq P_t(X)}$$
>
> 大家细想便会发现，的确，对于神经网络的各层输出，由于它们经过了层内操作作用，其分布显然与各层对应的输入信号分布不同，而且差异会随着网络深度增大而增大，可是它们所能“指示”的样本标记（label）仍然是不变的，这便符合了covariate shift的定义。由于是对层间信号的分析，也即是“internal”的来由。

1.3 ICS 会导致什么问题?

简而言之，每个神经元的输入数据不再是“独立同分布”。

其一，上层参数需要不断适应新的输入数据分布，降低学习速度。

其二，下层输入的变化可能趋向于变大或者变小，导致上层落入饱和区，使得学习过早停止。

其三，每层的更新都会影响到其它层，因此每层的参数更新策略需要尽可能的谨慎。

---

2. Normalization 的通用框架与基本思想

我们以神经网络中的一个普通神经元为例。神经元接收一组输入向量

$$\mathbf{x} = (x_1, x_2, \cdots, x_d)$$

通过某种运算后，输出一个标量值：

$$y = f(\mathbf{x})$$

由于 ICS 问题的存在， $\mathbf{x}$ 的分布可能相差很大。要解决独立同分布的问题，“理论正确”的方法就是对每一层的数据都进行白化操作。然而标准的白化操作代价高昂，特别是我们还希望白化操作是可微的，保证白化操作可以通过反向传播来更新梯度。

因此，以 BN 为代表的 Normalization 方法退而求其次，进行了简化的白化操作。基本思想是：在将 $\mathbf{x}$ 送给神经元之前，先对其做**平移和伸缩变换**， 将 $\mathbf{x}$ 的分布规范化成在固定区间范围的标准分布。

通用变换框架就如下所示：

$$h = f\left(\mathbf{g} \cdot \frac{\mathbf{x} - \mu}{\sigma} + \mathbf{b}\right)$$

我们来看看这个公式中的各个参数。

（1） μ 是**平移参数**（shift parameter）， σ 是**缩放参数**（scale parameter）。通过这两个参数进行 shift 和 scale 变换：

$$\hat{\mathbf{x}} = \frac{\mathbf{x} - \mu}{\sigma}$$

得到的数据符合均值为 0、方差为 1 的标准分布。

（2） $\mathbf{b}$ 是**再平移参数**（re-shift parameter）， $\mathbf{g}$ 是**再缩放参数**（re-scale parameter）。将上一步得到的 $\hat{\mathbf{x}}$ 进一步变换为：

$$\mathbf{y} = \mathbf{g} \cdot \hat{\mathbf{x}} + \mathbf{b}$$

最终得到的数据符合均值为 $\mathbf{b}$ 、方差为 $\mathbf{g}^2$ 的分布。

奇不奇怪？奇不奇怪？

说好的处理 ICS，第一步都已经得到了标准分布，第二步怎么又给变走了？

答案是——**为了保证模型的表达能力不因为规范化而下降。**

我们可以看到，第一步的变换将输入数据限制到了一个全局统一的确定范围（均值为 0、方差为 1）。下层神经元可能很努力地在学习，但不论其如何变化，其输出的结果在交给上层神经元进行处理之前，将被粗暴地重新调整到这一固定范围。

沮不沮丧？沮不沮丧？

难道我们底层神经元人民就在做无用功吗？

所以，为了尊重底层神经网络的学习结果，我们将规范化后的数据进行再平移和再缩放，使得每个神经元对应的输入范围是针对该神经元量身定制的一个确定范围（均值为 $\mathbf{b}$ 、方差为 $\mathbf{g}^2$ ）。rescale 和 reshift 的参数都是可学习的，这就使得 Normalization 层可以学习如何去尊重底层的学习结果。

除了充分利用底层学习的能力，另一方面的重要意义在于保证获得非线性的表达能力。Sigmoid 等激活函数在神经网络中有着重要作用，通过区分饱和区和非饱和区，使得神经网络的数据变换具有了非线性计算能力。而第一步的规范化会将几乎所有数据映射到激活函数的非饱和区（线性区），仅利用到了线性变化能力，从而降低了神经网络的表达能力。而进行再变换，则可以将数据从线性区变换到非线性区，恢复模型的表达能力。

那么问题又来了——

经过这么的变回来再变过去，会不会跟没变一样?

不会。因为，再变换引入的两个新参数 g 和 b，可以表示旧参数作为输入的同一族函数，但是新参数有不同的学习动态。在旧参数中， $\mathbf{x}$ 的均值取决于下层神经网络的复杂关联；但在新参数中，$\mathbf{y} = \mathbf{g} \cdot \hat{\mathbf{x}} + \mathbf{b}$ 仅由 $\mathbf{b}$ 来确定，去除了与下层计算的密切耦合。新参数很容易通过梯度下降来学习，简化了神经网络的训练。

那么还有一个问题——

这样的 Normalization 离标准的白化还有多远?

标准白化操作的目的是“独立同分布”。独立就不说了，暂不考虑。变换为均值为 $\mathbf{b}$ 、方差为 $\mathbf{g}^2$ 的分布，也并不是严格的同分布，只是映射到了一个确定的区间范围而已。（所以，这个坑还有得研究呢！）

---

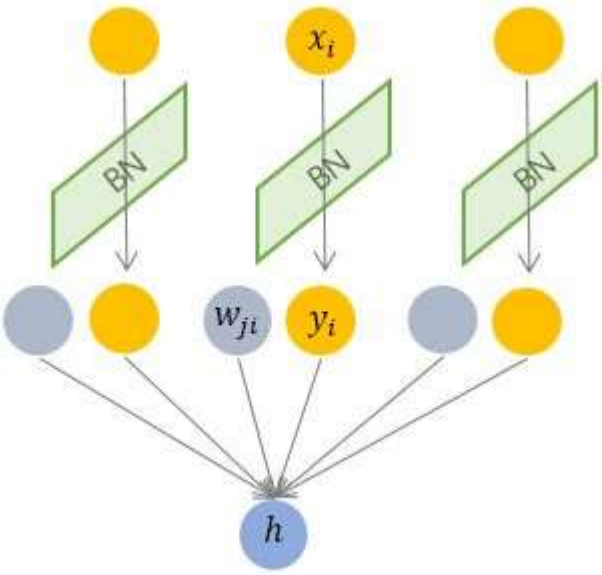
3. 主流 Normalization 方法梳理

在上一节中，我们提炼了 Normalization 的通用公式：

$$h = f\left(\mathbf{g} \cdot \frac{\mathbf{x} - \mu}{\sigma} + \mathbf{b}\right)$$

对照于这一公式，我们来梳理主流的四种规范化方法。

3.1 Batch Normalization —— 纵向规范化



Batch Normalization 于2015年由 Google 提出，开 Normalization 之先河。其规范化针对单个神经元进行，利用网络训练时一个 mini-batch 的数据来计算该神经元 x_i 的均值和方差,因而称为 Batch Normalization。

$$\mu_i = \frac{1}{M}\sum x_i, \quad \sigma_i = \sqrt{\frac{1}{M}\sum (x_i - \mu_i)^2 + \epsilon}$$

其中 M 是 mini-batch 的大小。

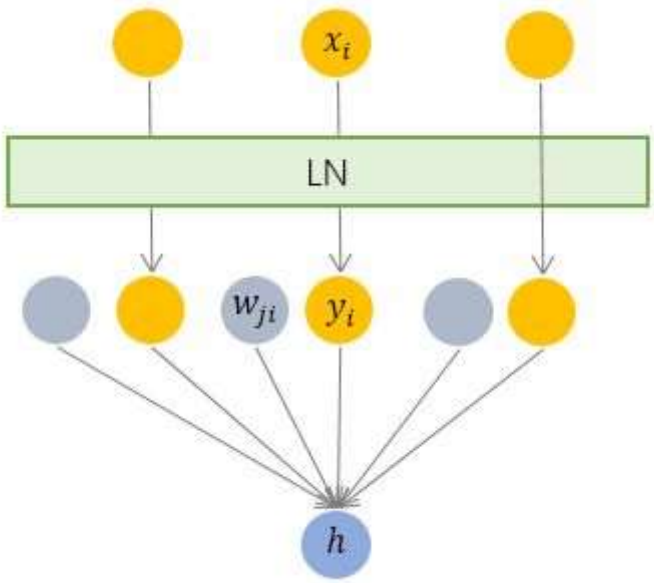
按上图所示，相对于一层神经元的水平排列，BN 可以看做一种纵向的规范化。由于 BN 是针对单个维度定义的，因此标准公式中的计算均为 element-wise 的。

BN 独立地规范化每一个输入维度 x_i ，但规范化的参数是一个 mini-batch 的一阶统计量和二阶统计量。这就要求 每一个 mini-batch 的统计量是整体统计量的近似估计，或者说每一个 mini-batch 彼此之间，以及和整体数据，都应该是近似同分布的。分布差距较小的 mini-batch 可以看做是为规范化操作和模型训练引入了噪声，可以增加模型的鲁棒性；但如果每个 mini-batch的原始分布差别很大，那么不同 mini-batch 的数据将会进行不一样的数据变换，这就增加了模型训练的难度。

因此，BN 比较适用的场景是：每个 mini-batch 比较大，数据分布比较接近。在进行训练之前，要做好充分的 shuffle. 否则效果会差很多。

另外，由于 BN 需要在运行过程中统计每个 mini-batch 的一阶统计量和二阶统计量，因此不适用于 动态的网络结构 和 RNN 网络。不过，也有研究者专门提出了适用于 RNN 的 BN 使用方法，这里先不展开了。

3.2 Layer Normalization —— 横向规范化



层规范化就是针对 BN 的上述不足而提出的。与 BN 不同，LN 是一种横向的规范化，如图所示。它综合考虑一层所有维度的输入，计算该层的平均输入值和输入方差，然后用同一个规范化操作来转换各个维度的输入。

$$\mu = \sum_i x_i, \quad \sigma = \sqrt{\sum_i (x_i - \mu)^2 + \epsilon}$$

其中 i 枚举了该层所有的输入神经元。对应到标准公式中，四大参数 μ，σ，$\mathbf{g}$，$\mathbf{b}$ 均为标量（BN 中是向量），所有输入共享一个规范化变换。

LN 针对单个训练样本进行，不依赖于其他数据，因此可以避免 BN 中受 mini-batch 数据分布影响的问题，可以用于 小mini-batch场景、动态网络场景和 RNN，特别是自然语言处理领域。此外，LN 不需要保存 mini-batch 的均值和方差，节省了额外的存储空间。

但是，BN 的转换是针对单个神经元可训练的——不同神经元的输入经过再平移和再缩放后分布在不同的区间，而 LN 对于一整层的神经元训练得到同一个转换——所有的输入都在同一个区间范围内。如果不同输入特征不属于相似的类别（比如颜色和大小），那么 LN 的处理可能会降低模型的表达能力。

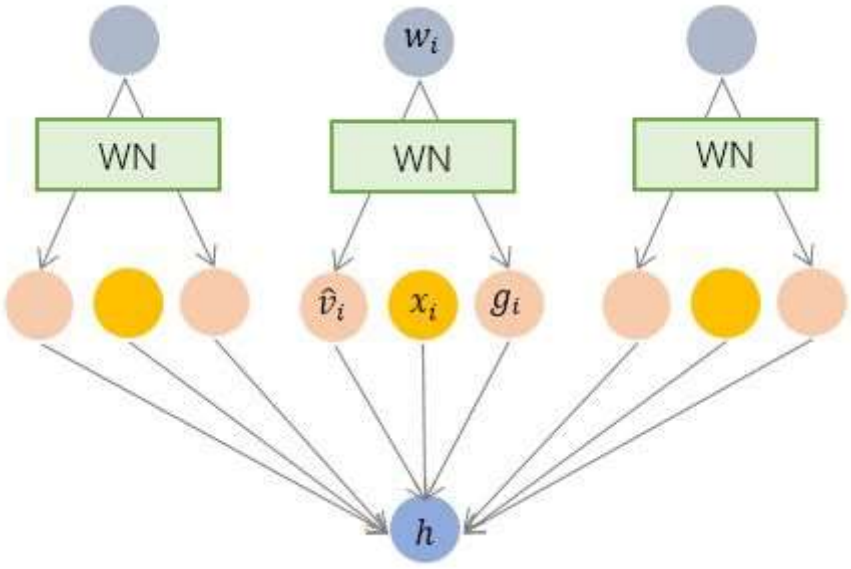
3.3 Weight Normalization —— 参数规范化

前面我们讲的模型框架

$$h = f\left(\mathbf{g} \cdot \frac{\mathbf{x} - \mu}{\sigma} + \mathbf{b}\right)$$

中，经过规范化之后的 $\mathbf{y}$ 作为输入送到下一个神经元，应用以 $\mathbf{w}$ 为参数的 $f_{\mathbf{w}}(\cdot)$ 函数定义的变换。最普遍的变换是线性变换，即 $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$.

BN 和 LN 均将规范化应用于输入的特征数据 $\mathbf{x}$ ，而 WN 则另辟蹊径，将规范化应用于线性变换函数的权重 $\mathbf{w}$ ，这就是 WN 名称的来源。



具体而言，WN 提出的方案是，**将权重向量 $\mathbf{w}$ 分解为向量方向 $\hat{\mathbf{v}}$ 和向量模 g 两部分**：

$$\mathbf{w} = g \cdot \hat{\mathbf{v}} = g \cdot \frac{\mathbf{v}}{||\mathbf{v}||}$$

其中 $\mathbf{v}$ 是与 $\mathbf{w}$ 同维度的向量，$||\mathbf{v}||$ 是欧氏范数，因此 $\hat{\mathbf{v}}$ 是单位向量，决定了 $\mathbf{w}$ 的方向；g 是标量，决定了 $\mathbf{w}$ 的长度。由于 $||\mathbf{w}|| \equiv |g|$ ，因此这一权重分解的方式将权重向量的欧氏范数进行了固定，从而实现了正则化的效果。

乍一看，这一方法似乎脱离了我们前文所讲的通用框架?

并没有。其实从最终实现的效果来看，异曲同工。我们来推导一下看。

$$f_{\mathbf{w}}(WN(\mathbf{x})) = \mathbf{w} \cdot WN(\mathbf{x}) = g \cdot \frac{\mathbf{v}}{||\mathbf{v}||} \cdot \mathbf{x}$$
$$= \mathbf{v} \cdot g \cdot \frac{\mathbf{x}}{||\mathbf{v}||} = f_{\mathbf{v}}(g \cdot \frac{\mathbf{x}}{||\mathbf{v}||})$$

对照一下前述框架：

$$h = f\left(\mathbf{g} \cdot \frac{\mathbf{x} - \mu}{\sigma} + \mathbf{b}\right)$$

我们只需令：

$$\sigma = ||\mathbf{v}||, \quad \mu = 0, \quad \mathbf{b} = 0$$

就完美地对号入座了！

回忆一下，BN 和 LN 是用输入的特征数据的方差对输入数据进行 scale，而 WN 则是用 神经元的权重的欧氏范式对输入数据进行 scale。**虽然在原始方法中分别进行的是特征数据规范化和参数的规范化，但本质上都实现了对数据的规范化，只是用于 scale 的参数来源不同。**

另外，我们看到这里的规范化只是对数据进行了 scale，而没有进行 shift，因为我们简单地令 $\mu = 0$. 但事实上，这里留下了与 BN 或者 LN 相结合的余地——那就是利用 BN 或者 LN 的方法来计算输入数据的均值 μ 。

WN 的规范化不直接使用输入数据的统计量，因此避免了 BN 过于依赖 mini-batch 的不足，以及 LN 每层唯一转换器的限制，同时也可以用于动态网络结构。

3.4 Cosine Normalization —— 余弦规范化

Normalization 还能怎么做?

我们再来看看神经元的经典变换 $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$.

对输入数据 $\mathbf{x}$ 的变换已经做过了，横着来是 LN，纵着来是 BN。

对模型参数 $\mathbf{w}$ 的变换也已经做过了，就是 WN。

好像没啥可做的了。

然而天才的研究员们盯上了中间的那个点，对，就是 $\cdot$

他们说，我们要对数据进行规范化的原因，是数据经过神经网络的计算之后可能会变得很大，导致数据分布的方差爆炸，而这一问题的根源就是我们的计算方式——点积，权重向量 $\mathbf{w}$ 和 特征数据向量 $\mathbf{x}$ 的点积。向量点积是无界（unbounded）的啊！

那怎么办呢？我们知道向量点积是衡量两个向量相似度的方法之一。哪还有没有其他的相似度衡量方法呢？有啊，很多啊！夹角余弦就是其中之一啊！而且关键的是，夹角余弦是有确定界的啊，[-1, 1] 的取值范围，多么的美好！仿佛看到了新的世界！

于是，Cosine Normalization 就出世了。他们不处理权重向量 $\mathbf{w}$ ，也不处理特征数据向量 $\mathbf{x}$ ，就改了一下线性变换的函数：

$$f_{\mathbf{w}}(\mathbf{x}) = cos\theta = \frac{\mathbf{w} \cdot \mathbf{x}}{||\mathbf{w}|| \cdot ||\mathbf{x}||}$$

其中 θ 是 $\mathbf{w}$ 和 $\mathbf{x}$ 的夹角。然后就没有然后了，所有的数据就都是 [-1, 1] 区间范围之内的了！

不过，回过头来看，CN 与 WN 还是很相似的。我们看到上式中，分子还是 $\mathbf{w}$ 和 $\mathbf{x}$ 的内积，而分母则可以看做用 $\mathbf{w}$ 和 $\mathbf{x}$ 二者的模之积进行规范化。对比一下 WN 的公式：

$$f_{\mathbf{w}}(WN(\mathbf{x})) = f_{\mathbf{v}}(g \cdot \frac{\mathbf{x}}{||\mathbf{v}||})$$

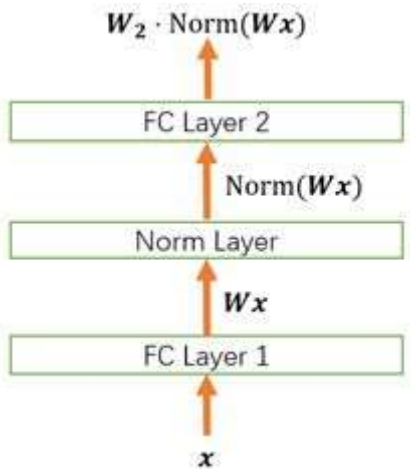
一定程度上可以理解为，WN 用 权重的模 $||\mathbf{v}||$ 对输入向量进行 scale，而 CN 在此基础上用输入向量的模 $||\mathbf{x}||$ 对输入向量进行了进一步的 scale.

CN 通过用余弦计算代替内积计算实现了规范化，但成也萧何败萧何。原始的内积计算，其几何意义是 输入向量在权重向量上的投影，既包含 二者的夹角信息，也包含 两个向量的scale信息。去掉scale信息，可能导致表达能力的下降，因此也引起了一些争议和讨论。具体效果如何，可能需要在特定的场景下深入实验。

现在，BN, LN, WN 和 CN 之间的来龙去脉是不是清楚多了?

4. Normalization 为什么会有效?

我们以下面这个简化的神经网络为例来分析。



4.1 Normalization 的权重伸缩不变性

权重伸缩不变性（weight scale invariance）指的是，当权重 $\mathbf{W}$ 按照常量 λ 进行伸缩时，得到的规范化后的值保持不变，即：

$$Norm(\mathbf{W}'\mathbf{x}) = Norm(\mathbf{W}\mathbf{x})$$

其中 $\mathbf{W}' = \lambda\mathbf{W}$ 。

上述规范化方法均有这一性质，这是因为，当权重 $\mathbf{W}$ 伸缩时，对应的均值和标准差均等比例伸缩，分子分母相抵。

$$\begin{aligned} Norm(\mathbf{W}'\mathbf{x}) &= Norm\left(\mathbf{g}\cdot\frac{\mathbf{W}'\mathbf{x}-\mu'}{\sigma'}+\mathbf{b}\right) \\ &= Norm\left(\mathbf{g}\cdot\frac{\lambda\mathbf{W}\mathbf{x}-\lambda\mu}{\lambda\sigma}+\mathbf{b}\right) \\ &= Norm\left(\mathbf{g}\cdot\frac{\mathbf{W}\mathbf{x}-\mu}{\sigma}+\mathbf{b}\right) = Norm(\mathbf{W}\mathbf{x}) \end{aligned}$$

权重伸缩不变性可以有效地提高反向传播的效率。

由于

$$\frac{\partial Norm(\mathbf{W}'\mathbf{x})}{\partial \mathbf{x}} = \frac{\partial Norm(\mathbf{W}\mathbf{x})}{\partial \mathbf{x}}$$

因此，权重的伸缩变化不会影响反向梯度的 Jacobian 矩阵，因此也就对反向传播没有影响，避免了反向传播时因为权重过大或过小导致的梯度消失或梯度爆炸问题，从而加速了神经网络的训练。

权重伸缩不变性还具有参数正则化的效果，可以使用更高的学习率。

由于

$$\frac{\partial Norm(\mathbf{W}'\mathbf{x})}{\partial \mathbf{W}'} = \frac{1}{\lambda} \cdot \frac{\partial Norm(\mathbf{W}\mathbf{x})}{\partial \mathbf{W}}$$

因此，下层的权重值越大，其梯度就越小。这样，参数的变化就越稳定，相当于实现了参数正则化的效果，避免参数的大幅震荡，提高网络的泛化性能。

4.2 Normalization 的数据伸缩不变性

数据伸缩不变性（data scale invariance）指的是，当数据 $\mathbf{x}$ 按照常量 λ 进行伸缩时，得到的规范化后的值保持不变，即：

$$Norm(\mathbf{W}\mathbf{x}') = Norm(\mathbf{W}\mathbf{x})$$

其中 $\mathbf{x}' = \lambda\mathbf{x}$ 。

数据伸缩不变性仅对 BN、LN 和 CN 成立。因为这三者对输入数据进行规范化，因此当数据进行常量伸缩时，其均值和方差都会相应变化，分子分母互相抵消。而 WN 不具有这一性质。

数据伸缩不变性可以有效地减少梯度弥散，简化对学习率的选择。

对于某一层神经元 $h_l = f_{\mathbf{W}_l}(\mathbf{x}_l)$ 而言，展开可得

$$h_l = f_{\mathbf{W}_l}(\mathbf{x}_l) = f_{\mathbf{W}_l}(f_{\mathbf{W}_{l-1}}(\mathbf{x}_{l-1})) = \cdots = \mathbf{x}_0 \prod_{k=0}^{l} \mathbf{W}_k$$

每一层神经元的输出依赖于底下各层的计算结果。如果没有正则化，当下层输入发生伸缩变化时，经过层层传递，可能会导致数据发生剧烈的膨胀或者弥散，从而也导致了反向计算时的梯度爆炸或梯度弥散。

加入 Normalization 之后，不论底层的数据如何变化，对于某一层神经元 $h_l = f_{\mathbf{W}_l}(\mathbf{x}_l)$ 而言，其输入 $\mathbf{x}_l$ 永远保持标准的分布，这就使得高层的训练更加简单。从梯度的计算公式来看：

$$\frac{\partial Norm(\mathbf{W}\mathbf{x}')}{\partial \mathbf{W}} = \frac{\partial Norm(\mathbf{W}\mathbf{x})}{\partial \mathbf{W}}$$

数据的伸缩变化也不会影响到对该层的权重参数更新，使得训练过程更加鲁棒，简化了对学习率的选择。

参考文献

[1] Sergey Ioffe and Christian Szegedy. Accelerating Deep Network Training by Reducing Internal Covariate Shift.

[2] Jimmy L. Ba, Jamie R. Kiros, Geoffrey E. Hinton. [1607.06450] Layer Normalization.

[3] Tim Salimans, Diederik P. Kingma. A Simple Reparameterization to Accelerate Training of Deep Neural Networks.

[4] Chunjie Luo, Jianfeng Zhan, Lei Wang, Qiang Yang. Using Cosine Similarity Instead of Dot Product in Neural Networks.

[5] Ian Goodfellow, Yoshua Bengio, Aaron Courville. Deep Learning.

本文在写作过程中，参考了以下各位的回答，特此致谢。

@魏秀参 的回答： 深度学习中 Batch Normalization为什么效果好?

@孔涛 的回答： 深度学习中 Batch Normalization为什么效果好?

@王峰 的回答： 深度学习中 Batch Normalization为什么效果好?

@lqfarmer 的回答： Weight Normalization 相比batch Normalization 有什么优点呢?

@Naiyan Wang 的回答： Batch normalization和Instance normalization的对比?

@YJango 的文章： YJango的Batch Normalization--介绍