# 1、综述

## 1.1 论文链接

1、Batch Normalization

https://arxiv.org/pdf/1502.03167.pdf

2、Layer Normalizaiton

https://arxiv.org/pdf/1607.06450v1.pdf

3、Instance Normalization

https://arxiv.org/pdf/1607.08022.pdf

https://github.com/DmitryUlyanov/texture_nets

4、Group Normalization

https://arxiv.org/pdf/1803.08494.pdf

5、Switchable Normalization

https://arxiv.org/pdf/1806.10779.pdf

https://github.com/switchablenorms/Switchable-Normalization

### 1.2 介绍

归一化层，目前主要有这几个方法，batch normalization(2015),layer normalization(2016),instance normaliztion(2017),group normalization(2018),switchable normalization(2018)。

将输入的图像shape记为[N,C,H,W]，这几个方法主要的区别就是在，n：样本数量 c：图像通道数 w：图像宽度 h：图像高度

- batchnorm是在batch上，对NHW做归一化，对小batchsize效果不好；
- layernorm是在通道方向上，对CHW归一化，主要对rnn作用明显；
- instancenorm在图像像素上，对HW做归一化，用于风格化迁移；
- groupnorm将channel分组，然后再做归一化
- switchablenorm是将BN LN IN结合，赋予权重，让网络自己去学习归一化层应该使用什么方法。

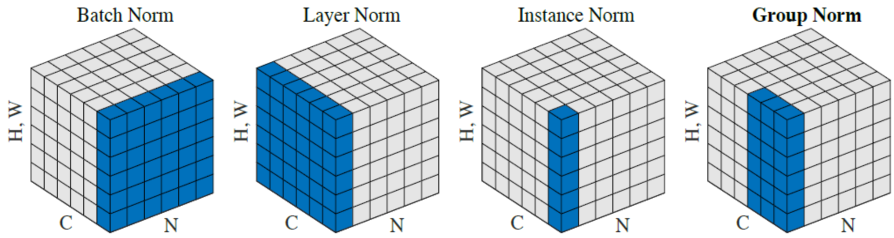


Figure 2. **Normalization methods**. Each subplot shows a feature map tensor, with $N$ as the batch axis, $C$ as the channel axis, and $(H, W)$ as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.



# 2.Batch Normalization

首先，在进行训练之前，一般要对数据做归一化，使其分布一致，但是在深度神经网络训练过程中，通常以送入网络的每一个batch训练，这样每个batch具有不同的分布；此外，为了解决internal covarivate shift问题，这个问题定义是随着batch normalizaiton这篇论文提出的，在训练过程中，数据分布会发生变化，对下一层网络的学习带来困难。

所以batch normalization就是强行将数据拉回到均值为0，方差为1的正太分布上，这样不仅数据分布一致，而且避免发生梯度消失。

此外，internal corvariate shift和covariate shift是两回事，前者是网络内部，后者是针对输入数据，比如我们在训练数据前做归一化等预处理操作。

**Input:** Values of $x$ over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;
Parameters to be learned: $\gamma$, $\beta$

**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.



算法过程：

- 沿着通道计算每个batch的均值u
- 沿着通道计算每个batch的方差σ^2
- 对x做归一化，x’ =(x-u)/开根号(σ^2+ε)
- 加入缩放和平移变量γ和β ,归一化后的值，y=γx’ +β

加入缩放平移变量的原因是：保证每一次数据经过归一化后还保留原有学习来的特征，同时又能完成归一化操作，加速训练。 这两个参数是用来学习的参数。

```
import numpy as np

def Batchnorm(x, gamma, beta, bn_param):

    # x_shape:[B, C, H, W]
    running_mean = bn_param['running_mean']
    running_var = bn_param['running_var']
    results = 0.
    eps = 1e-5

    x_mean = np.mean(x, axis=(0, 2, 3), keepdims=True)
    x_var = np.var(x, axis=(0, 2, 3), keepdims=True0)
    x_normalized = (x - x_mean) / np.sqrt(x_var + eps)
    results = gamma * x_normalized + beta

    # 因为在测试时是单个图片测试，这里保留训练时的均值和方差，用在后面测试时用
    running_mean = momentum * running_mean + (1 - momentum) * x_mean
    running_var = momentum * running_var + (1 - momentum) * x_var

    bn_param['running_mean'] = running_mean
    bn_param['running_var'] = running_var

    return results, bn_param
```

# 3. Layer Normalization

batch normalization存在以下缺点：

- 对batchsize的大小比较敏感，由于每次计算均值和方差是在一个batch上，所以如果batchsize太小，则计算的均值、方差不足以代表整个数据分布；
- BN实际使用时需要计算并且保存某一层神经网络batch的均值和方差等统计信息，对于对一个固定深度的前向神经网络（DNN，CNN）使用BN，很方便；但对于RNN来说，sequence的长度是不一致的，换句话说RNN的深度不是固定的，不同的time-step需要保存不同的statics特征，可能存在一个特殊sequence比其他sequence长很多，这样training时，计算很麻烦。

与BN不同，LN是针对深度网络的某一层的所有神经元的输入按以下公式进行normalize操作。

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \qquad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^l - \mu^l\right)^2}$$



BN与LN的区别在于：

LN中同层神经元输入拥有相同的均值和方差，不同的输入样本有不同的均值和方差；

BN中则针对不同神经元输入计算均值和方差，同一个batch中的输入拥有相同的均值和方差。

所以，LN不依赖于batch的大小和输入sequence的深度，因此可以用于batchsize为1和RNN中对边长的输入sequence的normalize操作。

# 4、Instance Normalization

BN注重对每个batch进行归一化，保证数据分布一致，因为判别模型中结果取决于数据整体分布。

但是图像风格化中，生成结果主要依赖于某个图像实例，所以对整个batch归一化不适合图像风格化中，因而对HW做归一化。可以加速模型收敛，并且保持每个图像实例之间的独立。

$$y_{tijk} = \frac{x_{tijk} - \mu_{ti}}{\sqrt{\sigma_{ti}^2 + \epsilon}}, \quad \mu_{ti} = \frac{1}{HW}\sum_{l=1}^{W}\sum_{m=1}^{H} x_{tilm}, \quad \sigma_{ti}^2 = \frac{1}{HW}\sum_{l=1}^{W}\sum_{m=1}^{H}(x_{tilm} - mu_{ti})^2.$$

```
def Instancenorm(x, gamma, beta):

    # x_shape:[B, C, H, W]
    results = 0.
    eps = 1e-5

    x_mean = np.mean(x, axis=(2, 3), keepdims=True)
    x_var = np.var(x, axis=(2, 3), keepdims=True0)
    x_normalized = (x - x_mean) / np.sqrt(x_var + eps)
    results = gamma * x_normalized + beta
    return results
```

## 5、Group Normalization

主要是针对Batch Normalization对小batchsize效果差，GN将channel方向分group，然后每个group内做归一化，算(C//G)*H*W的均值，这样与batchsize无关，不受其约束。

**Group Norm.** Formally, a Group Norm layer computes $\mu$ and $\sigma$ in a set $\mathcal{S}_i$ defined as:

$$\mathcal{S}_i = \{k \mid k_N = i_N, \lfloor \frac{k_C}{C/G} \rfloor = \lfloor \frac{i_C}{C/G} \rfloor \}. \tag{7}$$

Here $G$ is the number of groups, which is a pre-defined hyper-parameter ($G = 32$ by default). $C/G$ is the number of channels per group. $\lfloor \cdot \rfloor$ is the floor operation, and "$\lfloor \frac{k_C}{C/G} \rfloor = \lfloor \frac{i_C}{C/G} \rfloor$" means that the indexes $i$ and $k$ are in the same group of channels, assuming each group of channels are stored in a sequential order along the $C$ axis. GN computes $\mu$ and $\sigma$ along the $(H, W)$ axes and along a group of $\frac{C}{G}$ channels. The computation of GN is illustrated in Figure 2 (rightmost), which is a simple case of 2 groups ($G = 2$) each having 3 channels. 

```
def GroupNorm(x, gamma, beta, G=16):

    # x_shape:[B, C, H, W]
    results = 0.
    eps = 1e-5
    x = np.reshape(x, (x.shape[0], G, x.shape[1]/16, x.shape[2], x.shape[3]))

    x_mean = np.mean(x, axis=(2, 3, 4), keepdims=True)
    x_var = np.var(x, axis=(2, 3, 4), keepdims=True0)
    x_normalized = (x - x_mean) / np.sqrt(x_var + eps)
    results = gamma * x_normalized + beta
    return results
```

## 6、Switchable Normalization

本篇论文作者认为，

第一，归一化虽然提高模型泛化能力，然而归一化层的操作是人工设计的。在实际应用中，解决不同的问题原则上需要设计不同的归一化操作，并没有一个通用的归一化方法能够解决所有应用问题；
第二，一个深度神经网络往往包含几十个归一化层，通常这些归一化层都使用同样的归一化操作，因为手工为每一个归一化层设计操作需要进行大量的实验。
因此作者提出自适配归一化方法——Switchable Normalization（SN）来解决上述问题。与强化学习不同，SN使用可微分学习，为一个深度网络中的每一个归一化层确定合适的归一化操作。

公式：

SN has an intuitive expression

$$\hat{h}_{ncij} = \gamma \frac{h_{ncij} - \Sigma_{k\in\Omega} w_k \mu_k}{\sqrt{\Sigma_{k\in\Omega} w'_k \sigma_k^2 + \epsilon}} + \beta, \tag{3}$$



$$w_k = \frac{e^{\lambda_k}}{\Sigma_{z\in\{\text{in},\text{ln},\text{bn}\}} e^{\lambda_z}}, \quad k \in \{\text{in}, \text{ln}, \text{bn}\},$$



$$\begin{aligned}
\mu_{\text{in}} &= \frac{1}{HW}\sum_{i,j}^{H,W} h_{ncij}, \ \sigma_{\text{in}}^2 = \frac{1}{HW}\sum_{i,j}^{H,W}(h_{ncij} - \mu_{\text{in}})^2, \\
\mu_{\text{ln}} &= \frac{1}{C}\sum_{c=1}^{C}\mu_{\text{in}}, \ \ \sigma_{\text{ln}}^2 = \frac{1}{C}\sum_{c=1}^{C}(\sigma_{\text{in}}^2 + \mu_{\text{in}}^2) - \mu_{\text{ln}}^2, \\
\mu_{\text{bn}} &= \frac{1}{N}\sum_{n=1}^{N}\mu_{\text{in}}, \ \ \sigma_{\text{bn}}^2 = \frac{1}{N}\sum_{n=1}^{N}(\sigma_{\text{in}}^2 + \mu_{\text{in}}^2) - \mu_{\text{bn}}^2, \quad (5)
\end{aligned}$$