layer_normalization

1. Normalization

1.1 Batch Norm

为什么要进行BN呢?

1. 在深度神经网络训练的过程中，通常以输入网络的每一个mini-batch进行训练，这样每个batch具有不同的分布，使模型训练起来特别困难。
2. 内部协方差Internal Covariate Shift (ICS) 问题：在训练的过程中，激活函数会改变各层数据的分布，随着网络的加深，这种改变（差异）会越来越大，使模型训练起来特别困难，收敛速度很慢，会出现梯度消失的问题。

BN的主要思想： 针对每个神经元，**使数据在进入激活函数之前，沿着通道计算每个batch的均值、方差，'强迫'数据保持均值为0，方差为1的正态分布，** 避免发生梯度消失。具体来说，就是把第1个样本的第1个通道，加上第2个样本第1个通道 ..... 加上第 N 个样本第1个通道，求平均，得到通道 1 的均值（注意是除以 N×H×W 而不是单纯除以 N，最后得到的是一个代表这个 batch 第1个通道平均值的数字，而不是一个 H×W 的矩阵）。求通道 1 的方差也是同理。对所有通道都施加一遍这个操作，就得到了所有通道的均值和方差。

BN的使用位置： 全连接层或卷积操作之后，激活函数之前。

BN算法过程：

1.沿着通道计算每个batch的均值

2.沿着通道计算每个batch的方差

3.做归一化

4.加入缩放和平移变量γ和 β

加入缩放和平移变量的原因是：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。

具体流程：

输入形状：`(N, C, H, W)`

- `N`：batch size（一个批次中有多少张图）
- `C`：通道数
- `H × W`：每张图的空间尺寸（高 × 宽）

Step 1：计算每个通道的均值

对每个通道 `C`，我们把整个 batch 所有样本、所有空间位置上的数值加起来，求平均：

$$\mu_c = \frac{1}{N \times H \times W} \sum_{i=1}^{N} \sum_{h=1}^{H} \sum_{w=1}^{W} x_{i,c,h,w}$$

> **Note**
>
> **为什么是除以 N×H×W?**

> **BatchNorm 在卷积层中所说的“数据”指的就是每个通道下的所有像素点。**因为我们是在这个 batch 中，**第 c 个通道的所有像素点**上做平均。比如一张 3×3 的图就有 9 个点，两个样本就有 2×9 = 18 个点。所以要除以总的像素点数（N×H×W），而不是只除以样本数 N！

Step 2：计算每个通道的方差

同样地，对每个通道 `c`，计算其在整个 batch 和空间上的方差：

$$\sigma_c^2 = \frac{1}{N \times H \times W} \sum_{i=1}^{N} \sum_{h=1}^{H} \sum_{w=1}^{W} (x_{i,c,h,w} - \mu_c)^2$$

Step 3：标准化（归一化）

把每个值减去均值，再除以标准差（加一个小常数防止除零）：

$$\hat{x}_{i,c,h,w} = \frac{x_{i,c,h,w} - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}}$$

Step 4：加入可学习参数 γ 和 β

引入缩放和平移参数，让模型保留表达能力：

$$y_{i,c,h,w} = \gamma_c \cdot \hat{x}_{i,c,h,w} + \beta_c$$

其中：

- γ_c：每个通道的缩放因子（scale）
- β_c：每个通道的平移因子（shift）

BN的作用：

1. 允许较大的学习率；
2. 减弱对初始化的强依赖性
3. 保持隐藏层中数值的均值、方差不变，让数值更稳定，为后面网络提供坚实的基础；
4. 有轻微的正则化作用（相当于给隐藏层加入噪声，类似Dropout）

BN存在的问题：

1. 每次是在一个batch上计算均值、方差，如果batch size太小，则计算的均值、方差不足以代表整个数据分布。
2. **batch size太大：** 会超过内存容量；需要跑更多的epoch，导致总训练时间变长；会直接固定梯度下降的方向，导致很难更新。

1.2 Layer Norm

LayerNorm是大模型也是transformer结构中最常用的归一化操作，简而言之，它的作用是对特征张量按照某一维度或某几个维度进行0均值，1方差的归一化操作，计算公式为：

$$y = \frac{x - \mathrm{E}(x)}{\sqrt{\mathrm{Var}(x) + \epsilon}} \cdot \gamma + \beta$$

这里的 x 可以理解为**张量中具体某一维度的所有元素**，比如对于 shape 为 (2,2,4) 的张量 input，若指定归一化的操作为第三个维度，则会对第三个维度中的四个张量（2,2,1），各进行上述的一次计算。

详细形式：

$$a_i = \sum_{j=1}^{m} w_{ij} x_j, \quad y_i = f(a_i + b_i)$$

$$\bar{a}_i = \frac{a_i - \mu}{\sigma} \cdot g_i, \quad y_i = f(\bar{a}_i + b_i)$$

$$\mu = \frac{1}{n} \sum_{i=1}^{n} a_i, \quad \sigma = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (a_i - \mu)^2}$$

这里结合PyTorch的 `nn.LayerNorm` 算子来看比较明白：

```
nn.LayerNorm(normalized_shape, eps=1e-05, elementwise_affine=True, device=None,
dtype=None)
```

- `normalized_shape`：归一化的维度，int（最后一维）或 list（list 里面的维度）。还是以 `(2,2,4)` 为例，如果输入是 int，则必须是 `4`；如果是 list，则可以是 `[4]`，`[2,4]`，`[2,2,4]`，即最后一维、倒数两维、和所有维度。
- `eps`：加在分母方差上的偏置项，防止分母为0。
- `elementwise_affine`：是否使用可学习的参数 γ 和 β，前者初始为1，后者初始为0。设置该变量为 `True`，则二者均可学习，随着训练过程而变化。

Layer Normalization (LN) 的一个优势是不需要批训练，在单条数据内部就能归一化。LN 不依赖于 batch size 和输入 sequence 的长度，因此可以用于 batch size 为1和 RNN 中。LN 用于 RNN 效果比较明显，但是在 CNN 上，效果不如 BN。

1.3 Instance Norm

IN 针对图像像素做 normalization，最初用于图像的风格化迁移。在图像风格化中，生成结果主要依赖于某个图像实例，feature map 的各个 channel 的均值和方差会影响到最终生成图像的风格。所以对整个 batch 归一化不适合图像风格化中，因而对 H、W 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。

对于 IN，对每个样本的 H、W 维度的数据求均值和标准差，保留 N、C 维度，也就是说，它只在 channel 内部求均值和标准差，其公式如下：

$$y_{tijk} = \frac{x_{tijk} - \mu_{ti}}{\sqrt{\sigma_{ti}^2 + \epsilon}}$$

$$\mu_{ti} = \frac{1}{HW} \sum_{l=1}^{W} \sum_{m=1}^{H} x_{tilm}$$

$$\sigma_{ti}^2 = \frac{1}{HW} \sum_{l=1}^{W} \sum_{m=1}^{H} (x_{tilm} - \mu_{ti})^2$$

> **Note**
>
> **InstanceNorm 是一种针对图像风格任务设计的归一化方法，它在每个样本的每个通道的空间维度（H×W）上进行标准化，保留图像的个性化风格信息，非常适合风格迁移、GAN 等图像生成任务**

当然可以！我们先将你提供的 **pRMSNorm** 内容中的公式用 `$$` 包裹，然后我再帮你**重新解释一遍它的含义**，让你更容易理解。

1.4 RMS Norm

与 LayerNorm 相比，RMS Norm 的主要区别在于去掉了减去均值的部分，计算公式为：

$$\bar{a}_i = \frac{a_i}{\sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2}} \cdot g_i$$

其中 $\sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2}$ 是 均方根RMS 操作（Root Mean Square）。

RMS 中去除了 mean 的统计值的使用，只使用 root mean square (RMS) 进行归一化。

1.5 pRMSNorm

RMSNorm：

- RMSNorm 是一种归一化方法，和 LayerNorm 类似，但只考虑输入张量的平方均值（Root Mean Square），而不减去均值。
- 它的公式是：

$$\mathrm{RMSNorm}(x) = \frac{x}{\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}}$$

也就是说，它只保留了数据的“整体幅度”，不关心其均值。

为了进一步减少计算量，有人提出了 **pRMSNorm（partial RMSNorm）**：

> 在实际训练过程中发现，并不需要对所有数据都进行 RMS 的计算，只需要用其中一部分数据就可以达到差不多的效果。

所以，**pRMSNorm 的思想是：**

- 只取输入向量中前 `p%` 的元素
- 用这些元素来估算整个向量的 RMS 值
- 然后用这个估算值去做归一化

公式如下：

$$\overline{\mathrm{RMS}}(a) = \sqrt{\frac{1}{k}\sum_{i=1}^{k} a_i^2}$$

- a 是一个长度为 `n` 的向量（比如神经元输出、token 向量等）
- $k = n \times p\%$：从这个向量中取前 `k` 个元素来做 RMS 计算
- 实验发现，即使只取 **6.25% 的数据**（也就是 p = 6.25），也能让模型正常训练并收敛

当然可以！以下是你的 **Group Normalization（组归一化，GN）** 内容整理：

1.6 Group Norm

GroupNorm 是一种不依赖 batch 的归一化方法，它将每个样本的通道分成若干组，每组内部在通道和空间维度上进行均值和方差统计，特别适合 batch size 较小的图像任务，是 LayerNorm 和 InstanceNorm 的折中方案。

GN 是为了解决 BN 对较小的 mini-batch size 效果差的问题。GN 适用于占用显存比较大的任务，例如图像分割。对这类任务，可能 batch size 只能是个位数，再大显存就不够用了。而当 batch size 是个位数时，BN 的表现很差，因为没办法通过几个样本的数据量，来近似总体的均值和标准差。GN 也是独立于 batch 的，它是 LN 和 IN 的折中。

GroupNorm 的核心思想：GN 计算均值和标准差时，把每一个样本 feature map 的 channel 分成 G 组，每组将有 C/G 个 channel，然后将这些 channel 中的元素求均值和标准差。各组 channel 用其对应的归一化参数独立地归一化。

$$\mu_{ng}(x) = \frac{1}{(C/G)HW} \sum_{c=gC/G}^{(g+1)C/G} \sum_{h=1}^{H} \sum_{w=1}^{W} x_{nchw}$$

$$\sigma_{ng}(x) = \sqrt{\frac{1}{(C/G)HW} \sum_{c=gC/G}^{(g+1)C/G} \sum_{h=1}^{H} \sum_{w=1}^{W} (x_{nchw} - \mu_{ng}(x))^2 + \epsilon}$$

具体做法：

1. 对每个样本单独处理（不依赖 batch）
2. 将该样本的 feature map 的所有通道分成 `G` 组
3. 每组包含 `C / G` 个通道
4. 在每一组内，沿着空间维度（H × W）计算均值和标准差
5. 然后对该组内的数据进行标准化

1.7 Deep Norm

Post-LN（Post Layer Normalization）是指在 Transformer 模型中将 **Layer Normalization** 应用在 **残差连接之后** 的一种做法。这是原始 Transformer 模型中采用的标准归一化方式 。

其计算流程如下：

1. 输入经过子层（如 Multi-Head Attention 或 FFN）处理得到输出。
2. 将输出与输入进行残差连接（Residual Connection）。
3. 最后对残差连接后的结果进行 LayerNorm 归一化操作。

公式表示为：

$$Post - LN(x) = LayerNorm(x + Sublayer(x))$$

Deep Norm 是对 Post-LN 的改进，具体包括以下两点：

1. **在进行 Layer Norm 之前会以 α 参数扩大残差连接**
2. **在 Xavier 参数初始化过程中以 β 减小部分参数的初始化范围**

```
def deepnorm(x):
    return LayerNorm(x * α + f(x))

def deepnorm_init(w):
    if w is ['ffn', 'v_proj', 'out_proj']:
        nn.init.xavier_normal_(w, gain=β)
    elif w is ['q_proj', 'k_proj']:
        nn.init.xavier_normal_(w, gain=1)
```

一些模型的具体参数使用方法如下：

Architectures	Encoder		Decoder	
	α	β	α	β
Encoder-only (e.g., BERT)	$(2N)^{\frac{1}{4}}$	$(8N)^{-\frac{1}{4}}$	-	-
Decoder-only (e.g., GPT)	-	-	$(2M)^{\frac{1}{4}}$	$(8M)^{-\frac{1}{4}}$
Encoder-decoder (e.g., NMT, T5)	$0.81(N^4M)^{\frac{1}{16}}$	$0.87(N^4M)^{-\frac{1}{16}}$	$(3M)^{\frac{1}{4}}$	$(12M)^{-\frac{1}{4}}$

论文中，作者认为 Post-LN 的不稳定性部分来自于梯度消失以及太大的模型更新，同时，有以下几个理论分析：

- 定义了“预期模型更新”的概念，表示模型更新的规模量级
- 证明了 W_Q 和 W_K 不会改变注意力输出大小数量级的界限，因而 β 并没有缩小这部分参数
- 模型倾向于累积每个子层的更新，从而导致模型更新量呈爆炸式增长，从而使早期优化变得不稳定
- 使用 Deep Norm 的“预期模型更新”，在参数 α, β 取值适当的时候，以常数为界

同时，作者通过实验证实了 Deep Norm 在训练深层 Transformer 模型的时候具备近乎恒定的更新规模，成功训练了 1000 层 Transformer 的模型。认为 Deep Norm 在具备 Post-LN 的良好性能的同时又有 Pre-LN 的稳定训练

代码实现：[microsoft/torchscale: Foundation Architecture for (M)LLMs]

2. BN & LN & IN & GN

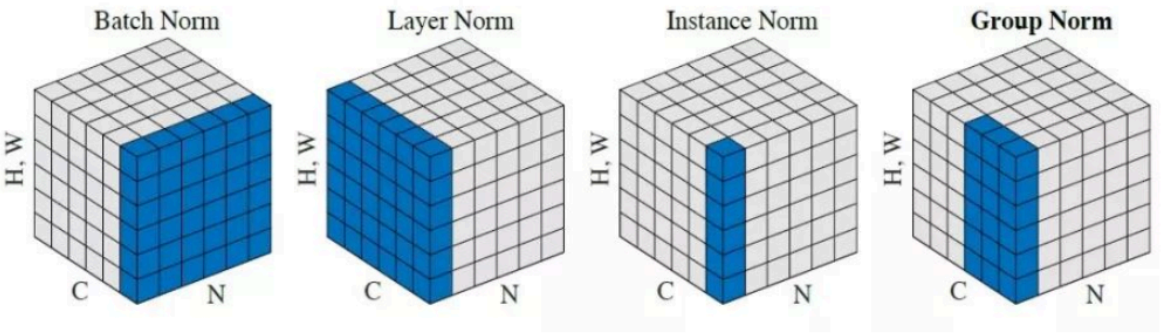
常用的Normalization方法主要有：

- Batch Normalization（BN，2015年）
- Layer Normalization（LN，2016年）
- Instance Normalization（IN，2017年）
- Group Normalization（GN，2018年）

它们都是从激活函数的输入来考虑、做文章的，以不同的方式**对激活函数的输入进行 Norm** 的。

将输入的 **feature map shape** 记为 `[N, C, H, W]`，其中N表示batch size，即N个样本；C表示通道数；H、W分别表示特征图的高度、宽度。这几个方法主要的区别就是在：

1. BN是在batch上，对N、H、W做归一化，而保留通道 C 的维度。**BN对较小的batch size效果不好。BN适用于固定深度的前向神经网络**，如CNN，不适用于RNN；
2. LN在通道方向上，对C、H、W归一化，主要对RNN效果明显；
3. IN在图像像素上，对H、W做归一化，用在风格化迁移；
4. GN将channel分组，然后再做归一化。



比喻成一摞书，这摞书总共有 N 本，每本有 C 页，每页有 H 行，每行 有W 个字符。

1. **BN 是把一个 batch 中所有样本的相同通道、相同位置上的数值拿出来，一起算平均值和标准差。**
   例如：对于第1张图的第3通道第(5,6)个像素，和第2张图的第3通道第(5,6)个像素……都拿出来一起算。这就像是在计算一本“平均书”，每一“页+位置”上只有一个平均值。所以 BN 是对 batch 中相同位置的数据做归一化。
2. **LN 是针对每一个样本单独处理。它会把这个样本中所有通道、所有位置上的数值全部加在一起，然后除以总数量（C×H×W）来算平均值。**就像是把一本书里的所有字都加起来，算出整本书的“平均字”，然后再用这个平均值去标准化每一个字。所以 LN 不依赖 batch，适合用于长度不固定的模型，比如 RNN 和 Transformer。
3. **IN 是对每个样本、每个通道单独处理。它会在这个通道的所有空间位置（也就是 H×W 个数值）上求平均值和标准差。**
   就像是一本书的每一页单独来看，把这一页上的所有内容加起来，再除以这一页有多少个字（H×W），得到这一页的“平均字”。然后用这个平均值和标准差来标准化这一页的内容。IN 常用于图像生成任务，如风格迁移。
4. **GN 是把一个样本的所有通道分成若干组（比如把 64 个通道分成 8 组，每组 8 个通道）。然后在每一组内部，把这组里所有通道的所有空间位置上的数值拿来做归一化。**
   就像是把一本书分成几本小册子，每一本小册子里的页面内容放在一起，算出它们的平均值和标准差。GN 结合了 BN 和 LN 的优点，不依赖 batch 大小，在 batch 小的时候表现很好，常用于目标检测、分割等任务。

> **Note**
>
> **BN 是“按页找平均”，LN 是“整本书找平均”，IN 是“每页单独平均”，GN 是“把书分组找平均”。**

当然可以，以下是将你提供的内容整理成一个**表格形式**，方便你对比查看和做笔记：

方法	适用场景	原因说明
BN	CNN	- CNN 输入结构固定，batch 内数据分布相似 - 可以利用 batch 的统计信息提升训练稳定性
		- 不适合 batch 很小的情况 - 不适合 RNN（输入长度不固定）

方法	适用场景	原因说明
LN	RNN、Transformer	- RNN 输入序列长度可能不同，无法统一按 batch 统计 - LN 只依赖每个样本自身的信息，不受 batch 影响
		- 更适合处理序列模型
IN	风格迁移	- 风格迁移希望保留图像结构，只调整像素分布 - IN 对每张图的每个通道分别处理，能保留风格特征
		- 更适合图像生成类任务，如图像风格转换
GN	分割、检测	- 不依赖 batch 大小，即使 batch=1 也能正常工作
		- 在通道上分组，兼顾了 BN 和 LN 的优点 - 在目标检测、语义分割中表现优于 BN，尤其 batch 小时

3.Post-LN 和 Pre-LN

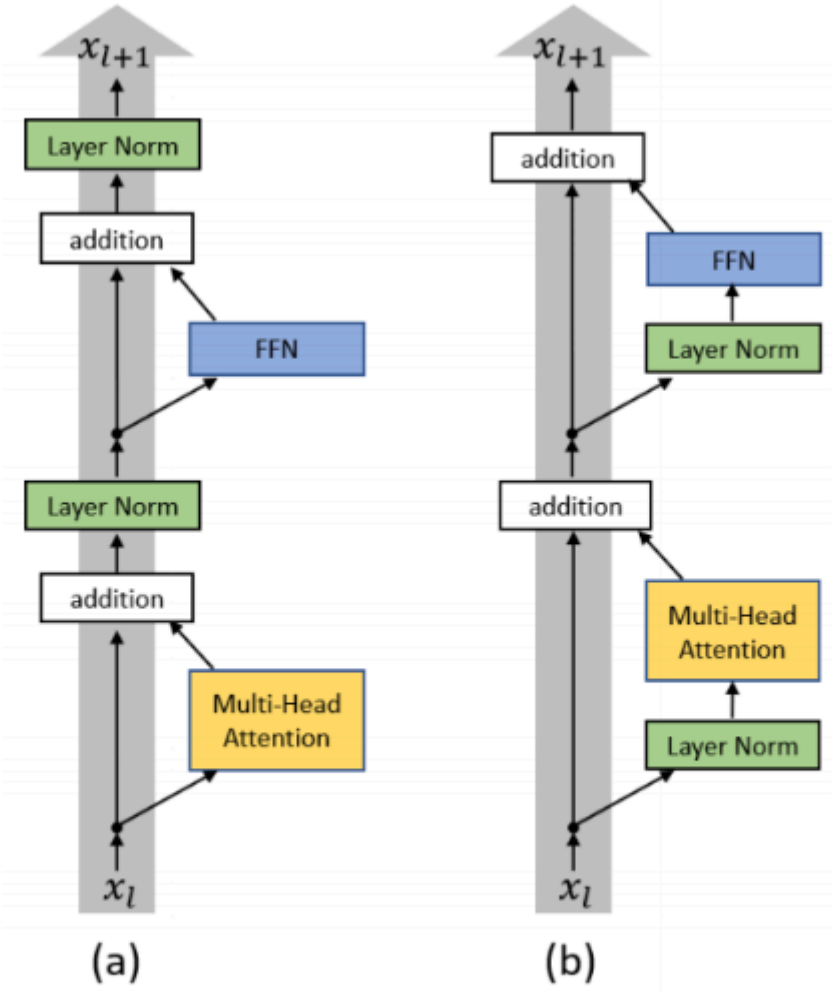
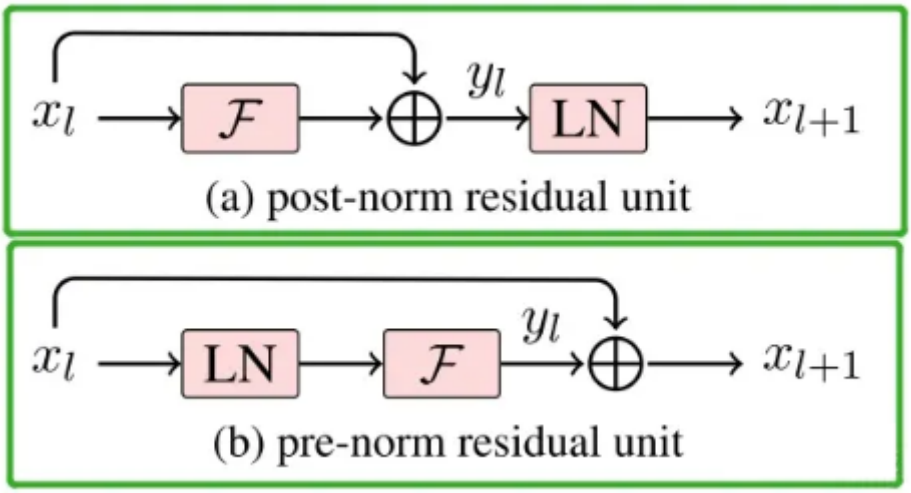


Figure 1: (a) Post-LN Transformer layer; (b) Pre-LN Transformer layer.

> **Note**
>
> 目前比较明确的结论是：**同一设置之下，Pre Norm结构往往更容易训练，但最终效果通常不如Post Norm。Pre-LN 更稳，Post-LN 更强**，前者保训练稳定，后者提模型上限。

左边是原版Transformer的Post-LN，即将LN放在addition之后；右边是改进之后的Pre-LN，即把LN放在FFN和MHA之前。



3.1Post-Norm:

一般认为，Post-Norm在残差之后做归一化，对参数正则化的效果更强，进而模型的收敛性也会更好；

优点：

1. LayerNorm 在残差连接之后，对整个输出进行标准化，所以对参数的正则化更强。
2. 输出特征分布更稳定，每一层输出都受到统一约束，有利于后续层的学习
3. 最终模型性能更好，实验表明收敛性和泛化能力通常优于 Pre-Norm

缺点：

1. 训练不够稳定，容易出现梯度消失或爆炸，尤其在深层模型中
2. 不利于深层网络训练。信息传递容易被干扰，训练难度大

3.2Pre-Norm:

Pre-Norm有一部分参数直接加在了后面，没有对这部分参数进行正则化，可以在反向时防止梯度爆炸或者梯度消失，大模型的训练难度大，因而使用Pre-Norm较多。

优点:

1. 更容易训练,残差路径清晰，信息流动顺畅
2. 适合深层模型,可有效缓解梯度消失/爆炸问题，支持百层以上训练
3. 稳定性高,输入数据规范，训练过程更加可控

缺点：

1. 参数正则化较弱，输出结果没有再次标准化，特征分布可能不稳定
2. 最终效果略逊于 Post-Norm，虽然训练稳定，但在精度和泛化能力上通常稍差一些

为什么 Pre-Norm 更容易训练但效果不如 Post-Norm?

> 因为 **Pre-Norm 的恒等路径更清晰**，信息可以直接从底层传到高层，不容易出错，所以更容易训练；
> 但与此同时，**它缺乏对最终输出的强正则化处理**，导致模型表达能力和泛化能力略逊于 Post-Norm，因此最终效果稍差。

博客：为什么Pre Norm的效果不如Post Norm？