归一化

Pytorch中的归一化方式主要分为以下几种：

- BatchNorm（2015年）
- LayerNorm（2016年）
- InstanceNorm（2017年）
- GroupNorm（2018年）

BatchNorm2D[1]

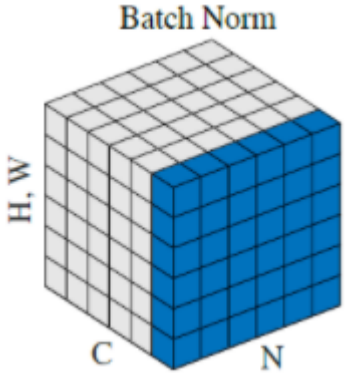
公式：

$$y=\frac{x-\mathbf E[x]}{\sqrt{\mathbf {Var}[x]+\epsilon}}*\gamma+\beta$$

其中前一项是归一化过程。分母中的 ϵ 是一个非常小的数，作用是防止数值计算不稳定。 γ 和 β 是仿射参数，将归一化后的数据再次放缩得到新的数据， γ 可以理解为标准差， β 可以理解为均值，它们两个一般是可学习的。可以发现， γ 和 β 是BatchNorm2D层仅有的可学习参数。

说明：

它是**沿着输入的第二维**（即channel维度）算均值和方差的。比如输入大小为(N,C,H,W)，则均值 $\mathbf E[x]$ 为 input.mean((0,2,3)) 。



代码：

```
torch.nn.BatchNorm2d(num_features, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True, device=None, dtype=None)
```

主要参数的含义：

- num_features：通道数C。
- eps：归一化时加到分母上，防止分母过小导致数值计算不稳定。**不用管这一项，一般用默认的就好。**
- momentum：在track_running_stats为True时，momentum是训练过程中对均值和方差进行动量更新的动量参数；在track_running_stats为False时，momentum不起作用。
- affine：当为True时， \gamma 和 \beta 参数是可学习的；反之，是不可学习的。
- track_running_stats：当为True时，在训练时会始终记录并更新（通过动量方法更新）全局的均值和方差，然后在**测试**时可以用这个均值和方差来归一化（**为什么要这样做？你可以理解为这个均值和方差是所有训练样本的均值和方差，是全局的，对整个样本集的统计信息的描述更加准确一些**）；当为False时，不记录更新全局的均值和方差，这样的话，**测试**时用那个batch的测试数据本身的样本和方差来归一化。

输入输出的维度：

输入维度：(N, C, H, W) (N为batchsize，下文不再赘述)

输出维度：(N, C, H, W)

BatchNorm1D

它也是**沿着输入的第二维**（即channel维）算均值和方差的。它与BatchNorm2D的不同之处在于，它的输入和输出维度可以是更低：

输入维度：(N, C) 或 (N,C,L)

输出维度：(N, C) 或 (N,C,L)

因此，它又可以被称为是**时间BN**

BatchNorm3D

它也是**沿着输入的第二维**（即channel维）算均值和方差的。它与BatchNorm2D的不同之处在于，它的输入和输出维度可以是更高：

输入维度： (N,C,D,H,W)

输出维度： (N,C,D,H,W)

因此，它又可以被称为是**体积BN**或者**时空BN**

LayerNorm[2]

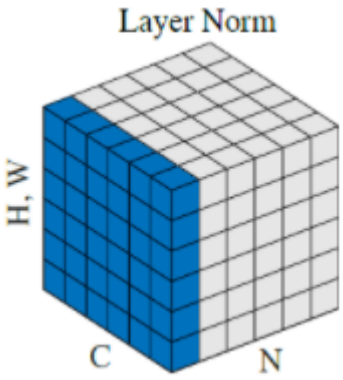
公式：

$$y=\frac{x-\mathbf E[x]}{\sqrt{\mathbf {Var}[x]+\epsilon}}*\gamma+\beta$$

其中前一项是归一化过程。分母中的 \epsilon 是一个非常小的数，作用是防止数值计算不稳定。 \gamma 和 \beta 是仿射参数，将归一化后的数据再次放缩得到新的数据， \gamma 可以理解为标准差， \beta 可以理解为均值，它们两个一般是可学习的。可以发现， \gamma 和 \beta 是LayerNorm层仅有的可学习参数。

说明：

它是**对输入的后几维（具体是几维取决于初始化参数normalized_shape）合并在一起**算均值和方差的。比如输入大小为 (N,C,H,W) （像图像一样），若normalized_shape为三维，则均值 \mathbf E[x] 为 input.mean((-3,-2, -1)) 。比如输入大小为 (N,L,D) （像文本一样），若normalized_shape为一维，则均值 \mathbf E[x] 为 input.mean((-1)) ，即Transformer中使用的LayerNorm。



代码：

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

主要参数的含义：

- normalized_shape：LayerNorm的输入的大小（除去第一维batchsize维度）。比如想让LayerNorm的输入大小为 (N, C, H, W)，那么normalized_shape可以是一个 [C, H, W] 的list。
- eps：归一化时加到分母上，防止分母过小导致数值计算不稳定。**不用管这一项，一般用默认的就好。**
- elementwise_affine：当为True时， \gamma 和 \beta 参数是可学习的；反之，是不可学习的。

输入输出的维度：

输入维度： (N,*)

输出维度： (N,*)

例子：

```
x = torch.randint(4,(2,5,3)).float()
n1 = torch.nn.LayerNorm(3)
n2 = torch.nn.LayerNorm([5,3])
print(x)
print(n1(x))
print(n2(x))

输出为：
tensor([[[2., 3., 0.],
         [1., 0., 0.],
         [0., 2., 3.],
         [2., 0., 0.],
         [1., 2., 3.]],

        [[0., 0., 3.],
         [2., 3., 0.],
         [0., 2., 3.],
```

```
         [3., 1., 2.],
         [0., 1., 1.]]])
tensor([[[ 0.2673,  1.0690, -1.3363],
         [ 1.4142, -0.7071, -0.7071],
         [-1.3363,  0.2673,  1.0690],
         [ 1.4142, -0.7071, -0.7071],
         [-1.2247,  0.0000,  1.2247]],

        [[-0.7071, -0.7071,  1.4142],
         [ 0.2673,  1.0690, -1.3363],
         [-1.3363,  0.2673,  1.0690],
         [ 1.2247, -1.2247,  0.0000],
         [-1.4142,  0.7071,  0.7071]]], grad_fn=<NativeLayerNormBackward>)
tensor([[[ 0.6208,  1.4673, -1.0722],
         [-0.2257, -1.0722, -1.0722],
         [-1.0722,  0.6208,  1.4673],
         [ 0.6208, -1.0722, -1.0722],
         [-0.2257,  0.6208,  1.4673]],

        [[-1.1667, -1.1667,  1.3333],
         [ 0.5000,  1.3333, -1.1667],
         [-1.1667,  0.5000,  1.3333],
         [ 1.3333, -0.3333,  0.5000],
         [-1.1667, -0.3333, -0.3333]]], grad_fn=<NativeLayerNormBackward>)
```

拓展：

我们可以看到，在输入为文本或其他时域信息时，假如shape为 (N,L,D)， L 为输入长度， D 是特征维度，我们使用LayerNorm的目的是沿着最后一维（即特征维）归一化。那假如shape为 (N,D,L) 呢，我们如果想沿着特征维做LayerNorm该如何做？一种最简单的方式是使用permute函数在LayerNorm前后交换后二维。但我们也可以直接自己写一个LayerNorm，使其支持对shape为 (N,D,L) 的输入沿着第二维（特征维，也称通道维）来归一化。代码如下：

```python
class LayerNorm(nn.Module):
    """
    LayerNorm that supports inputs of shape (N, D, L)
    """
    def __init__(
        self,
```

```
        num_channels,   # D
        eps = 1e-5,
        affine = True,
        device = None,
        dtype = None,
    ):
        super().__init__()
        factory_kwargs = {'device': device, 'dtype':
dtype}
        self.num_channels = num_channels
        self.eps = eps
        self.affine = affine

        if self.affine:
            self.weight = nn.Parameter(
                torch.ones([1, num_channels, 1],
**factory_kwargs))
            self.bias = nn.Parameter(
                torch.zeros([1, num_channels, 1],
**factory_kwargs))
        else:
            self.register_parameter('weight', None)
            self.register_parameter('bias', None)

    def forward(self, x):
        # only support inputs of shape (N, D, L)
        assert x.dim() == 3
        assert x.shape[1] == self.num_channels

        # normalization along D channels
        mu = torch.mean(x, dim=1, keepdim=True)
        res_x = x - mu
        sigma = torch.mean(res_x**2, dim=1, keepdim=True)
        out = res_x / torch.sqrt(sigma + self.eps)

        # apply weight and bias
        if self.affine:
            out *= self.weight
            out += self.bias

        return out
```

InstanceNorm2D[3]

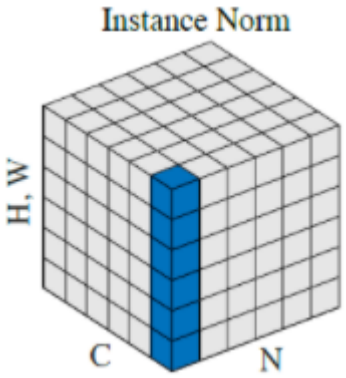
公式：

$$y=\frac{x-\mathbf E[x]}{\sqrt{\mathbf {Var}[x]+\epsilon}}*\gamma+\beta$$

其中前一项是归一化过程。分母中的 ϵ 是一个非常小的数，作用是防止数值计算不稳定。 γ 和 β 是仿射参数，将归一化后的数据再次放缩得到新的数据， γ 可以理解为标准差， β 可以理解为均值，它们两个一般是不可学习的。

说明：

它是**将输入的后两维（即除了batchsize维和channel维）合并在一起**算均值和方差的。比如输入大小为 (N,C,H,W) ，则均值 $\mathbf E[x]$ 为 input.mean((-2,-1)) 。



代码：

```
torch.nn.InstanceNorm2d(num_features, eps=1e-05, momentum=0.1, affine=False, track_running_stats=False, device=None, dtype=None)
```

主要参数的含义：

- num_features：通道数C。
- eps：归一化时加到分母上，防止分母过小导致数值计算不稳定。**不用管这一项，一般用默认的就好。**
- momentum：在track_running_stats为True时，momentum是训练过程中对均值和方差进行动量更新的动量参数；在track_running_stats为False时，momentum不起作用。

- affine：当为True时， \gamma 和 \beta 参数是可学习的；反之，是不可学习的。
- track_running_stats：当为True时，在训练时会始终记录并更新（通过动量方法更新）全局的均值和方差，然后在**测试**时可以用这个均值和方差来归一化（**为什么要这样做？你可以理解为这个均值和方差是所有训练样本的均值和方差，是全局的，对整个样本集的统计信息的描述更加准确一些**）；当为False时，不记录更新全局的均值和方差，这样的话，**测试**时用那个batch的测试数据本身的样本和方差来归一化。

输入输出的维度：

输入维度：(N, C, H, W)

输出维度：(N, C, H, W)

InstanceNorm1D

它是**对输入的最后一维（即除了batchsize维和channel维）**算均值和方差的。它与BatchNorm2D的不同之处在于，它的输入和输出维度可以是更低：

输入维度：(N,C,L)

输出维度：(N,C,L)

InstanceNorm3D

它是**将输入的后三维（即除了batchsize维和channel维）合并在一起**算均值和方差的。它与BatchNorm2D的不同之处在于，它的输入和输出维度可以是更高：

输入维度： (N,C,D,H,W)

输出维度： (N,C,D,H,W)

GroupNorm[4]

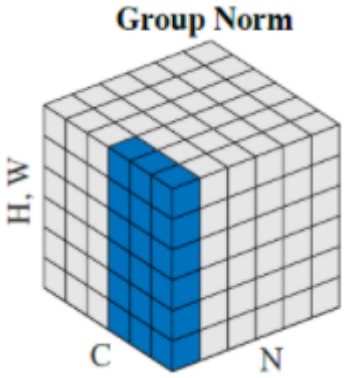
公式：

y=\frac{x-\mathbf E[x]}{\sqrt{\mathbf {Var}[x]+\epsilon}}*\gamma+\beta

其中前一项是归一化过程。分母中的 \epsilon 是一个非常小的数，作用是防止数值计算不稳定。 \gamma 和 \beta 是仿射参数，将归一化后的数据再次放缩得到新的数据， \gamma 可以理解为标准差， \beta 可以理解为均值，它们两个一般是可学习的。

说明：

它是**将输入的通道分组后，对输入的每组通道以及后面的维度合并在一起**算均值和方差的。**如果分为C组，即每组一个channel，则等价于InstanceNorm；如果只分为1组，即所有channel为一组，则等价于LayerNorm。**



代码：

```
torch.nn.GroupNorm(num_groups, num_channels, eps=1e-05, affine=True, device=None, dtype=None)
```

主要参数的含义：

- num_groups：通道分为几组。
- num_channels：通道数C。
- eps：归一化时加到分母上，防止分母过小导致数值计算不稳定。**不用管这一项，一般用默认的就好。**
- affine：当为True时， \gamma 和 \beta 参数是可学习的；反之，是不可学习的。

输入输出的维度：

输入维度：(N, C, *)

输出维度：(N, C, *)

参考

1. ^https://arxiv.org/pdf/1502.03167.pdf
2. ^https://arxiv.org/pdf/1607.06450.pdf
3. ^https://arxiv.org/pdf/1607.08022.pdf
4. ^https://arxiv.org/pdf/1803.08494.pdf