Batch Normalization and Layer Normalization

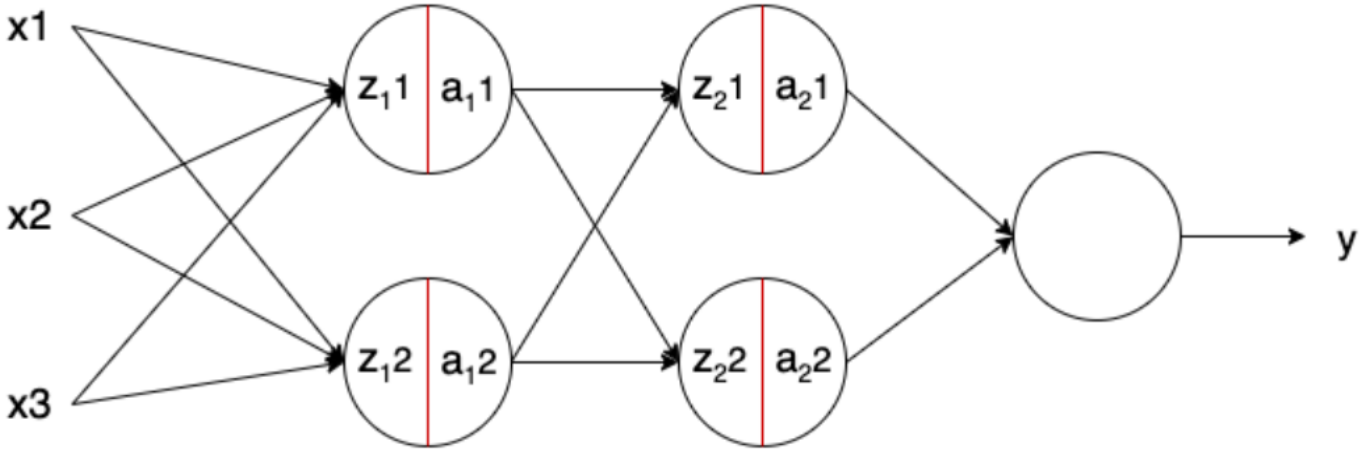
赵洲

浙江大学计算机学院

BN的研究动机

- 解决神经网络的内部协变量偏移问题（Internal Covariate Shift），加速网络收敛速度。

“当上一层网络中参数发生微弱变化时，下一层的输入数据的分布就会发生变化，由于每一层中的线性变换与非线性激活映射，这些微弱变化随着网络层数的加深而被放大，使得我们的模型训练变得困难”

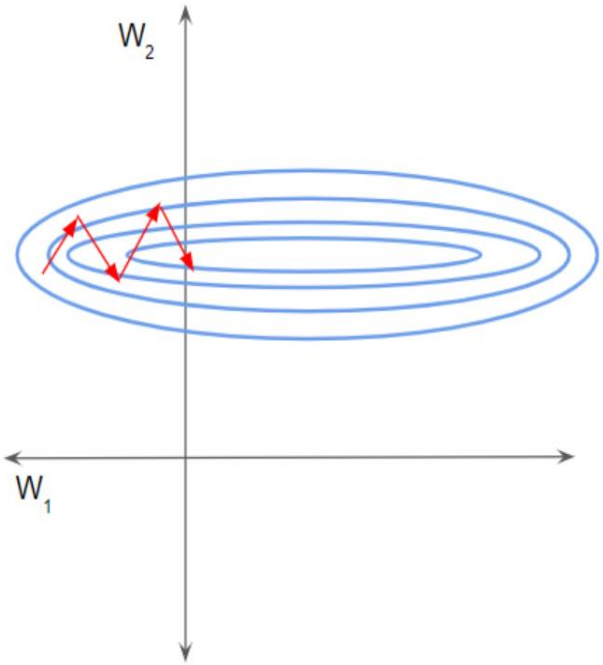


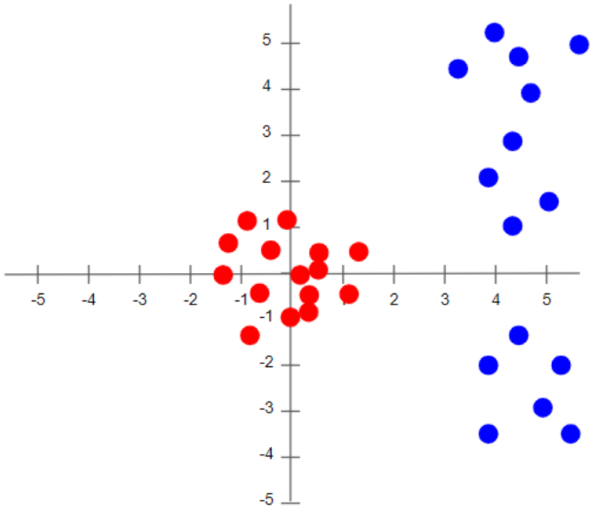
Internal Covariate Shift带来的问题？

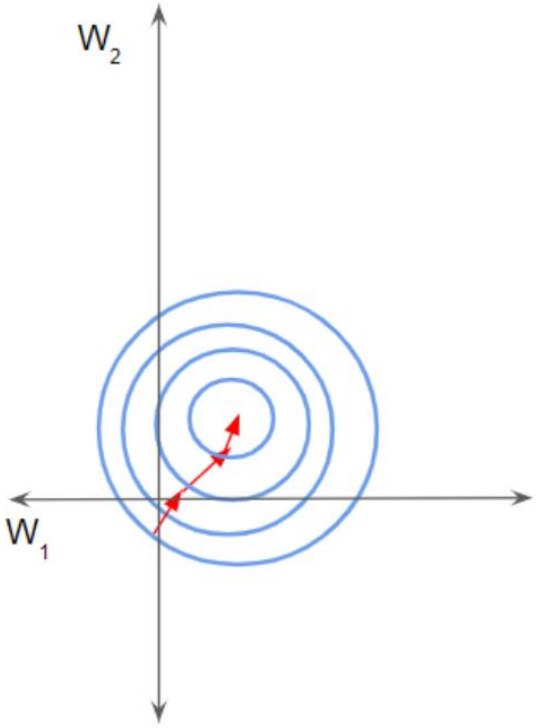
- **网络学习速度降低**
  - 网络某一层的输入数据分布发生变化，这一层网络就需要适应学习这个新的数据分布，导致网络学习速度降低。

- **网络收敛速度减缓**
  - 网络输入尽可能需要落在激活函数敏感区域，可以使得网络学习的梯度变大，加快网络训练的收敛度。

梯度下降

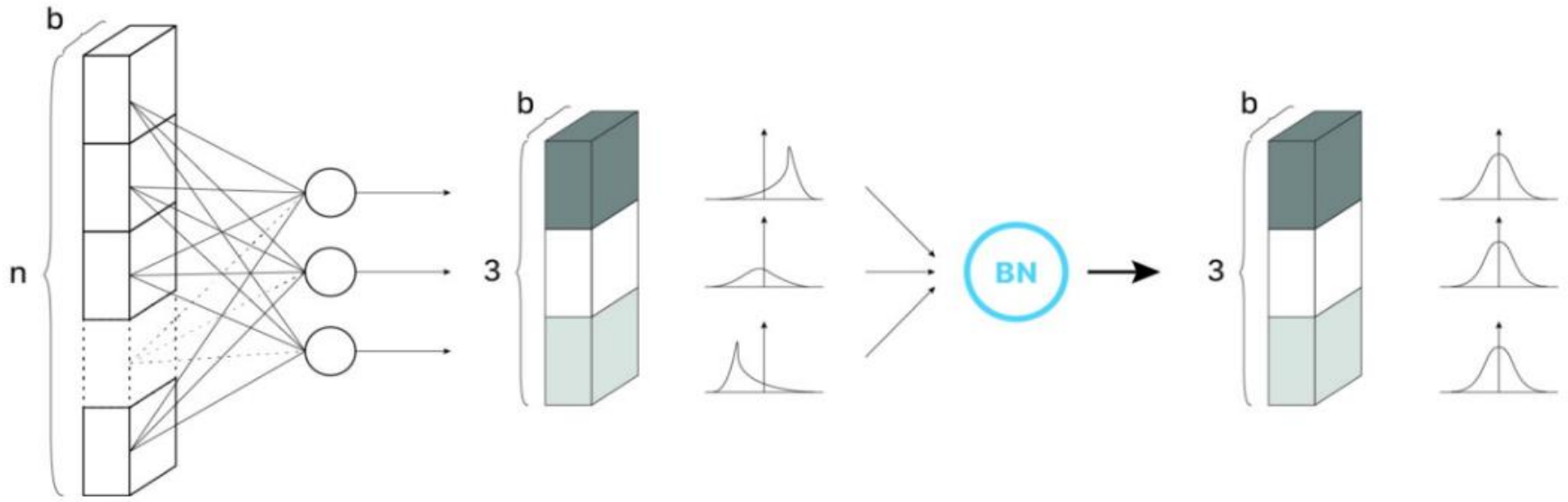






如何缓解Internal Covariate Shift?

- Batch Normalization的基本思想：
  - 通过规范化手段，把每层神经网络输入值的分布（每个特征）强行拉回到均值为0方差为1的标准正态分布。
  - 使得激活函数的输入值落到敏感区域，使得梯度变大，避免梯度消失的问题。



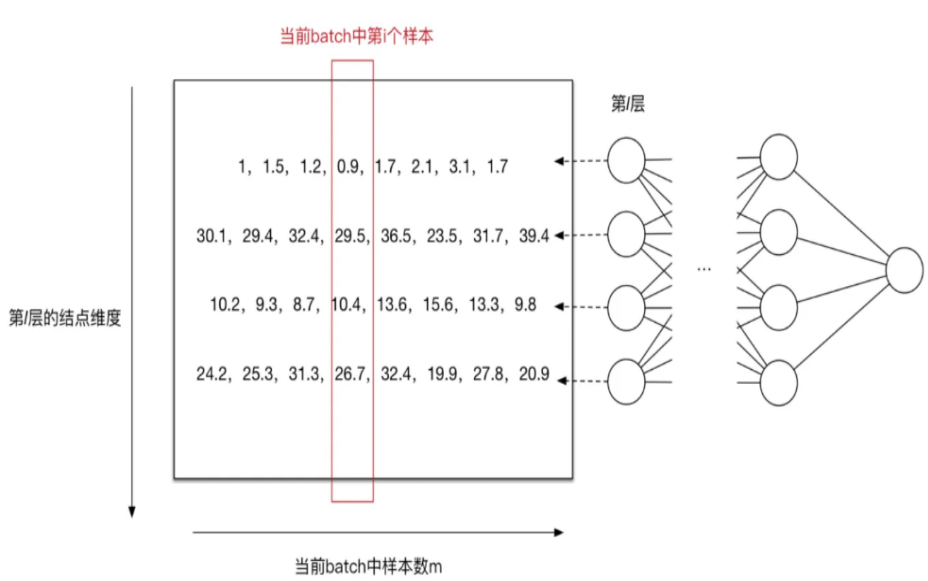
BN的数学原理

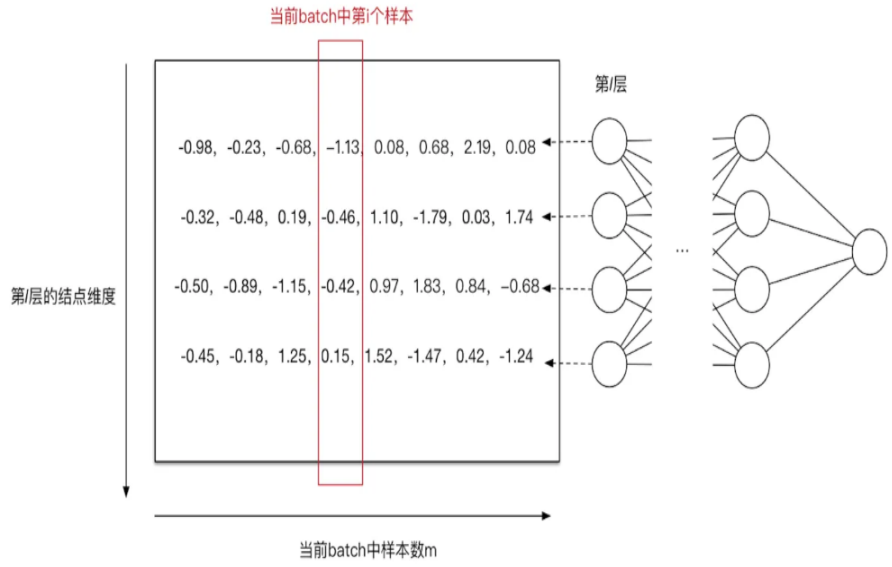
$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} Z_j^{(i)}$$

$$\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(Z_j^{(i)} - \mu_j)^2$$

$$\hat{Z}_j = \frac{Z_j - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

BN例子


当前batch中第i个样本
第l层
1, 1.5, 1.2, 0.9, 1.7, 2.1, 3.1, 1.7
30.1, 29.4, 32.4, 29.5, 36.5, 23.5, 31.7, 39.4
10.2, 9.3, 8.7, 10.4, 13.6, 15.6, 13.3, 9.8
24.2, 25.3, 31.3, 26.7, 32.4, 19.9, 27.8, 20.9
第l层的结点维度
当前batch中样本数m


当前batch中第i个样本
第l层
-0.98, -0.23, -0.68, -1.13, 0.08, 0.68, 2.19, 0.08
-0.32, -0.48, 0.19, -0.46, 1.10, -1.79, 0.03, 1.74
-0.50, -0.89, -1.15, -0.42, 0.97, 1.83, 0.84, -0.68
-0.45, -0.18, 1.25, 0.15, 1.52, -1.47, 0.42, -1.24
第l层的结点维度
当前batch中样本数m

可学习Batch Normalization

- Batch Normalization通过让每一层的输入数据部分布变得稳定，一定程度缓解ICS问题，但是导致数据表达能力流失。

$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$$

$$\mu = \frac{1}{m} \sum_{i=1}^{m} Z^{[l](i)}$$

$$\sigma^2 = \frac{1}{m} \sum_{i=1}^{m} (Z^{[l](i)} - \mu)^2$$

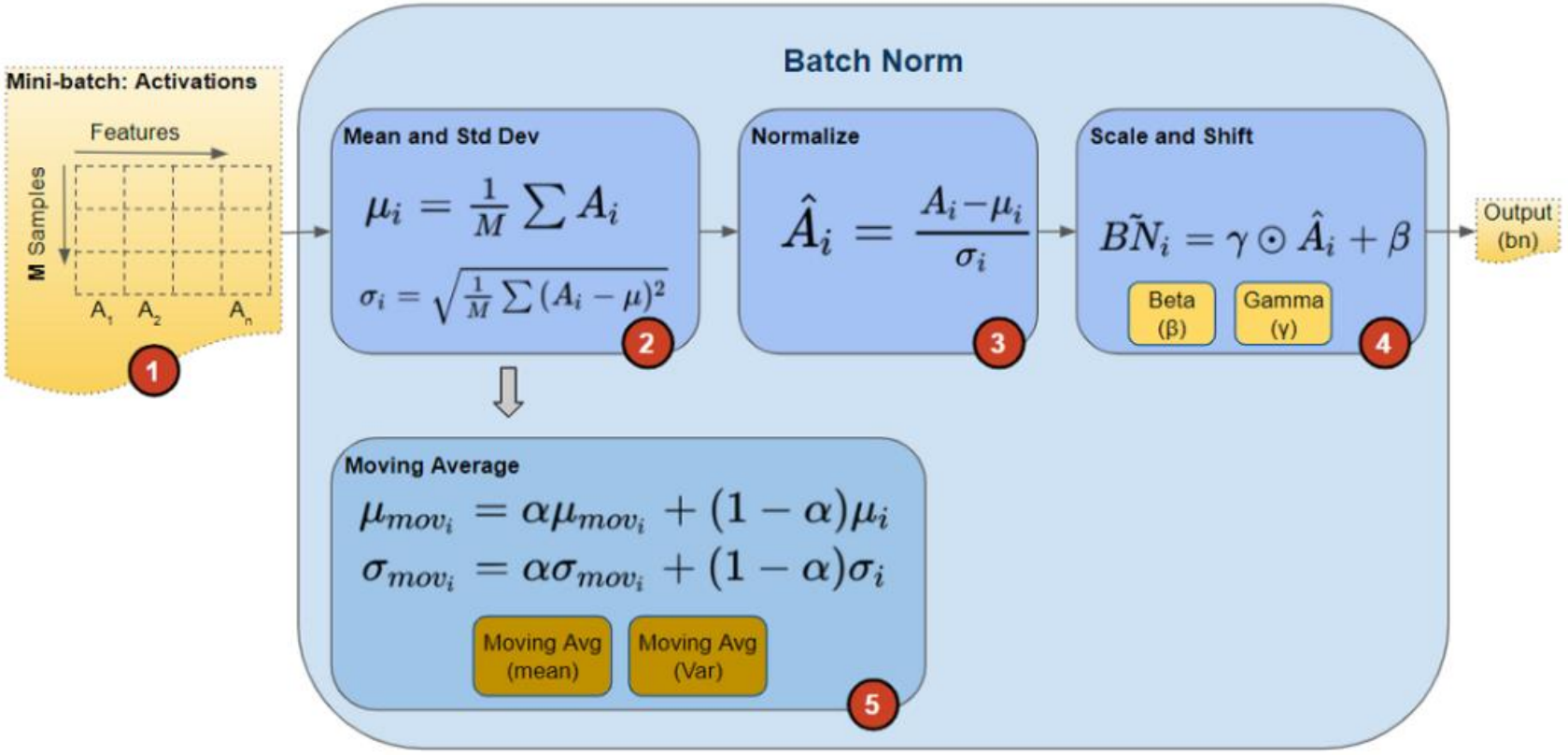
- BN引入可学习参数，拉伸（scale）和偏移（shift），用于恢复数据本省的表达能力（可以实现等价变换保留原始输入特征的分布信息）。

$$\tilde{Z}^{[l]} = \gamma \cdot \frac{Z^{[l]} - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta$$

$$A^{[l]} = g^{[l]}(\tilde{Z}^{[l]})$$

如果批量归一化无益，理论上，学出的模型可以不使用批量归一化

BN示意图


Mini-batch: Activations
Features
M Samples
A1 A2 An
1
Batch Norm
Mean and Std Dev
$\mu_i = \frac{1}{M}\sum A_i$
$\sigma_i = \sqrt{\frac{1}{M}\sum (A_i - \mu)^2}$
2
Normalize
$\hat{A}_i = \frac{A_i - \mu_i}{\sigma_i}$
3
Scale and Shift
$B\tilde{N}_i = \gamma \odot \hat{A}_i + \beta$
Beta (β)
Gamma (γ)
4
Moving Average
$\mu_{mov_i} = \alpha\mu_{mov_i} + (1-\alpha)\mu_i$
$\sigma_{mov_i} = \alpha\sigma_{mov_i} + (1-\alpha)\sigma_i$
Moving Avg (mean)
Moving Avg (Var)
5
Output (bn)

BN的反向传播

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$$

$$\frac{\partial \ell}{\partial \hat{x}_i} = \frac{\partial \ell}{\partial y_i} \cdot \gamma$$

$$\frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \hat{x}_i} \cdot (x_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{2} (\sigma_{\mathcal{B}}^2 + \epsilon)^{-3/2}$$

$$\frac{\partial \ell}{\partial \mu_{\mathcal{B}}} = \left(\sum_{i=1}^{m} \frac{\partial \ell}{\partial \hat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \right) + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{\sum_{i=1}^{m} -2(x_i - \mu_{\mathcal{B}})}{m}$$

$$\frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial \hat{x}_i} \cdot \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{2(x_i - \mu_{\mathcal{B}})}{m} + \frac{\partial \ell}{\partial \mu_{\mathcal{B}}} \cdot \frac{1}{m}$$

$$\frac{\partial \ell}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i} \cdot \hat{x}_i$$

$$\frac{\partial \ell}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i}$$

BN在网络推理阶段

- 测试阶段使用整个样本的无偏估计来对测试数据进行归一化，整个样本的统计量在训练过程中进行保存。

for $k = 1 \ldots K$ **do**

// For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.

Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them:

$$\mathrm{E}[x] \leftarrow \mathrm{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$

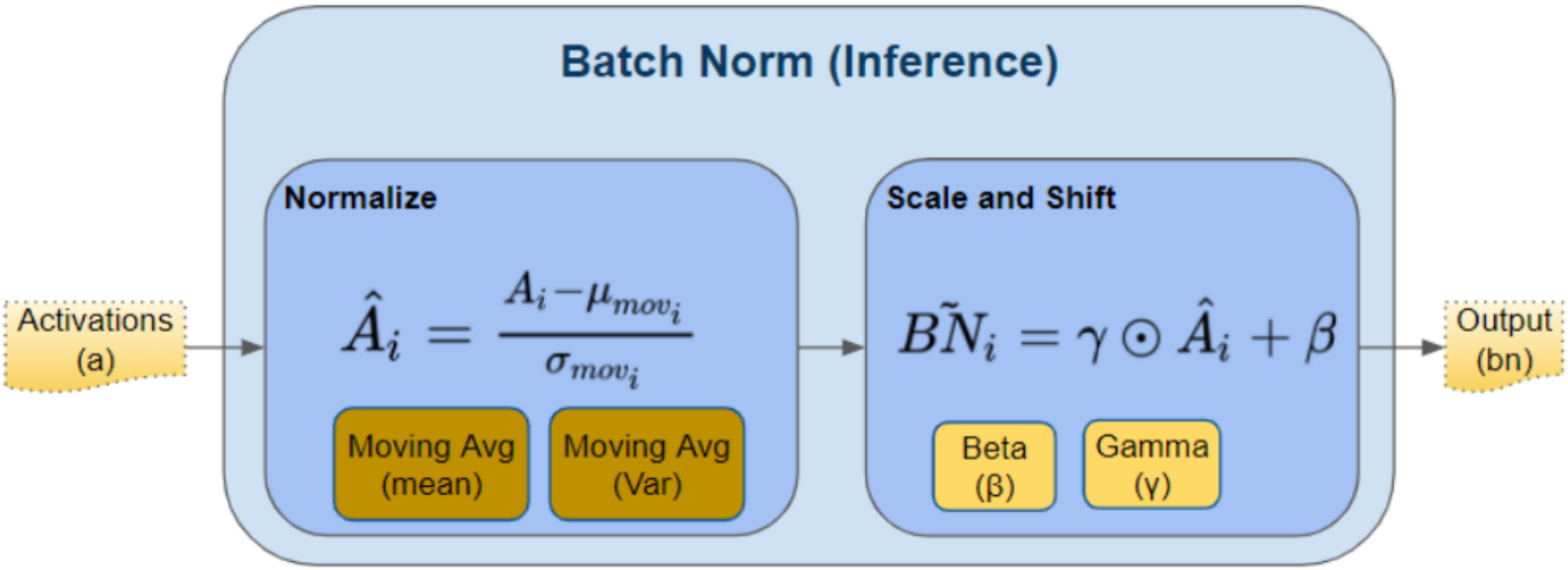
$$\mathrm{Var}[x] \leftarrow \frac{m}{m-1}\mathrm{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

In $N_{\mathrm{BN}}^{\mathrm{inf}}$, replace the transform $y = \mathrm{BN}_{\gamma,\beta}(x)$ with

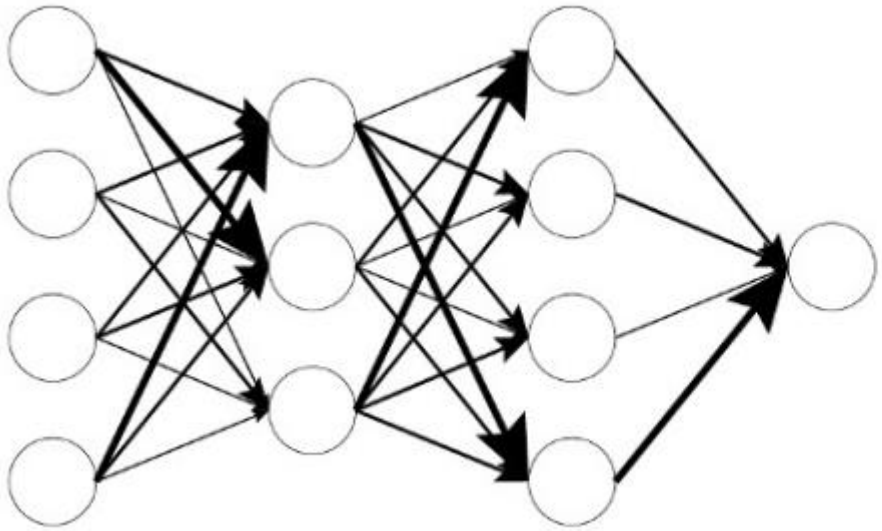
$$y = \frac{\gamma}{\sqrt{\mathrm{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]+\epsilon}}\right)$$

end for

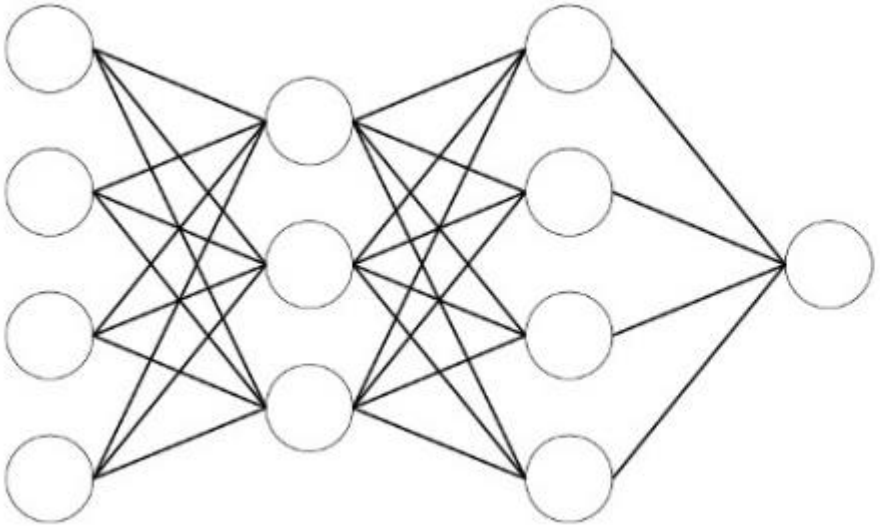
BN推理示意图


Batch Norm (Inference)
Normalize
$\hat{A}_i = \frac{A_i - \mu_{mov_i}}{\sigma_{mov_i}}$
Moving Avg (mean)
Moving Avg (Var)
Scale and Shift
$B\tilde{N}_i = \gamma \odot \hat{A}_i + \beta$
Beta (β)
Gamma (γ)
Activations (a)
Output (bn)

网络对比

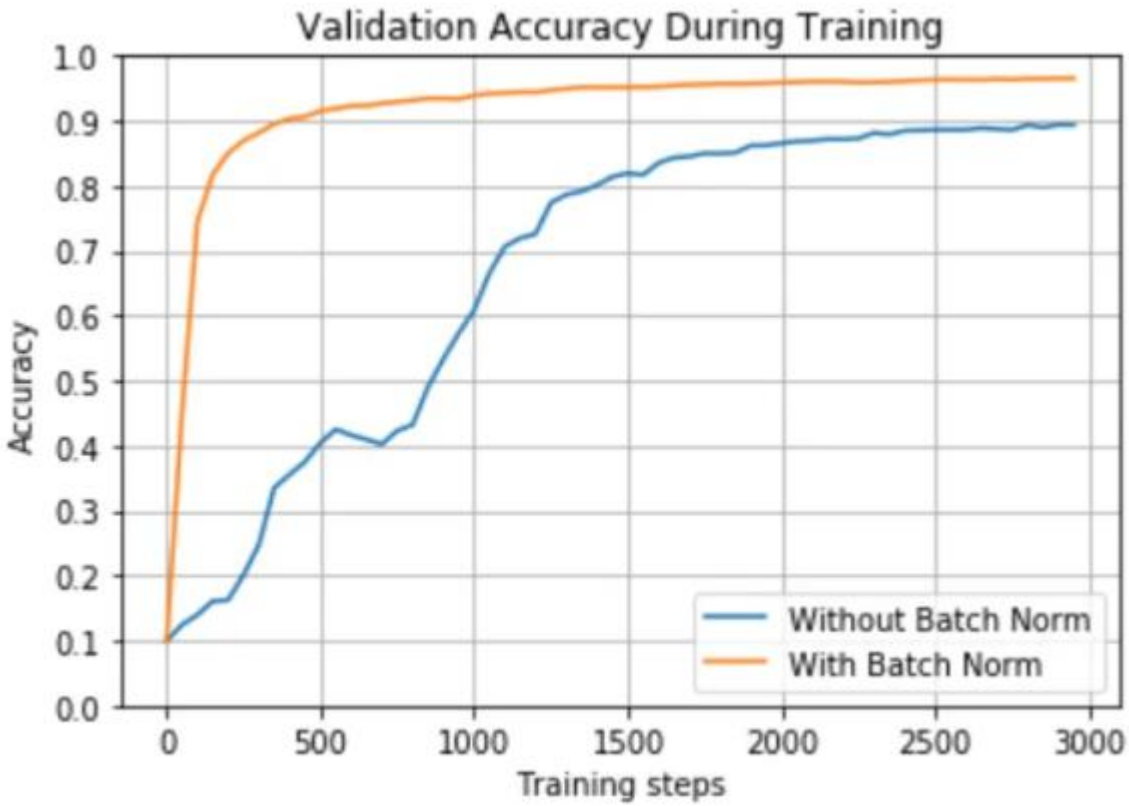


- **Raw** signal
- **High interdependancy** between distributions
- **Slow** and **unstable** training

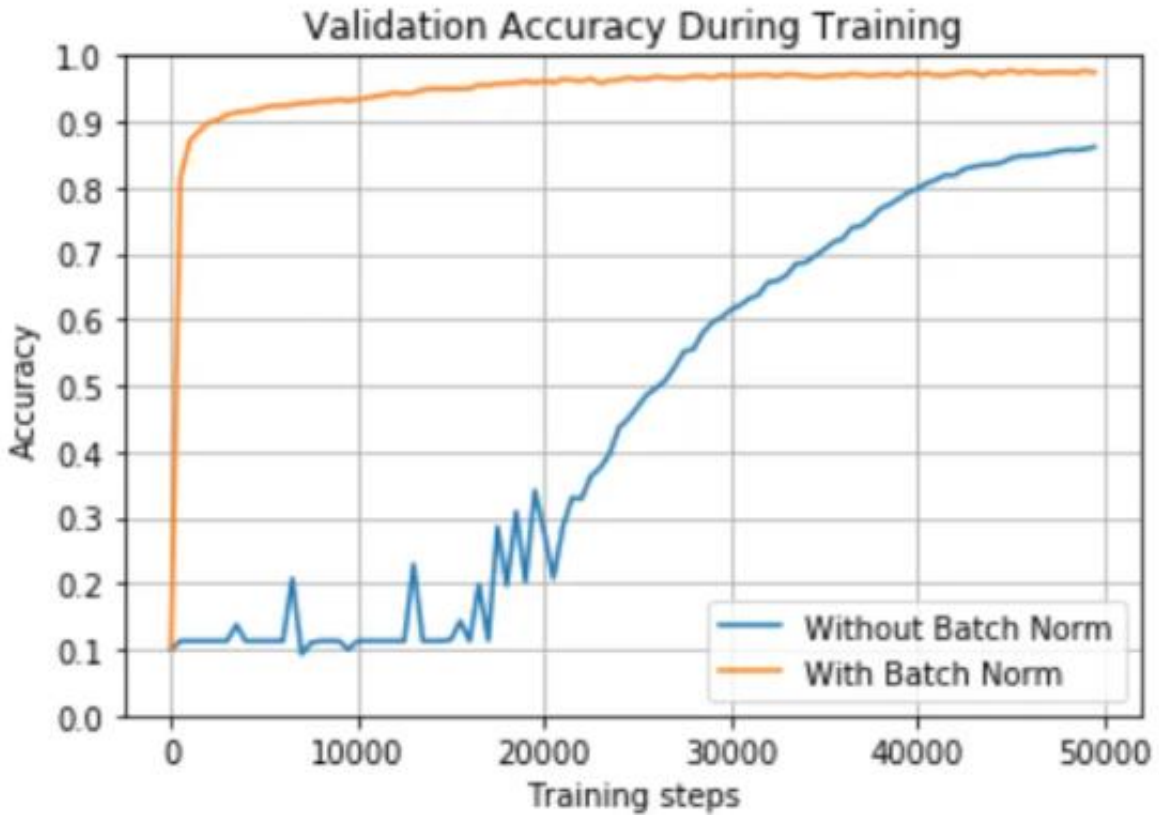


- **Normalized** signal
- **Mitigated interdependancy** between distributions
- **Fast** and **stable** training

ReLU+BN结果


Validation Accuracy During Training
Accuracy
Training steps
Without Batch Norm
With Batch Norm
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
0
500
1000
1500
2000
2500
3000

Sigmoid + BN结果



BN的实现

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

def batch_norm(is_training, X, gamma, beta, moving_mean, moving_var, eps, momentum):
    # 判断当前模式是训练模式还是预测模式
    if not is_training:
        # 如果是在预测模式下，直接使用传入的移动平均所得的均值和方差
        X_hat = (X - moving_mean) / torch.sqrt(moving_var + eps)
    else:
        assert len(X.shape) in (2, 4)
        if len(X.shape) == 2:
            # 使用全连接层的情况，计算特征维上的均值和方差
            mean = X.mean(dim=0)
            var = ((X - mean) ** 2).mean(dim=0)
        else:
            # 使用二维卷积层的情况，计算通道维上（axis=1）的均值和方差。这里我们需要保持
            # X的形状以便后面可以做广播运算
            mean = X.mean(dim=0, keepdim=True).mean(dim=2, keepdim=True).mean(dim=3, keepdim=True)
            var = ((X - mean) ** 2).mean(dim=0, keepdim=True).mean(dim=2, keepdim=True).mean(dim=3, keepdim=True)
        # 训练模式下用当前的均值和方差做标准化
        X_hat = (X - mean) / torch.sqrt(var + eps)
        # 更新移动平均的均值和方差
        moving_mean = momentum * moving_mean + (1.0 - momentum) * mean
        moving_var = momentum * moving_var + (1.0 - momentum) * var
    Y = gamma * X_hat + beta  # 拉伸和偏移
    return Y, moving_mean, moving_var
```

```
class BatchNorm(nn.Module):
    def __init__(self, num_features, num_dims):
        super(BatchNorm, self).__init__()
        if num_dims == 2:
            shape = (1, num_features)
        else:
            shape = (1, num_features, 1, 1)
        # 参与求梯度和迭代的拉伸和偏移参数，分别初始化成0和1
        self.gamma = nn.Parameter(torch.ones(shape))
        self.beta = nn.Parameter(torch.zeros(shape))
        # 不参与求梯度和迭代的变量，全在内存上初始化成0
        self.moving_mean = torch.zeros(shape)
        self.moving_var = torch.zeros(shape)

    def forward(self, X):
        # 如果X不在内存上，将moving_mean和moving_var复制到X所在显存上
        if self.moving_mean.device != X.device:
            self.moving_mean = self.moving_mean.to(X.device)
            self.moving_var = self.moving_var.to(X.device)
        # 保存更新过的moving_mean和moving_var，Module实例的traning属性默认为true，调用.eval()后设成false
        Y, self.moving_mean, self.moving_var = batch_norm(self.training,
            X, self.gamma, self.beta, self.moving_mean,
            self.moving_var, eps=1e-5, momentum=0.9)
        return Y
```

BN的调用

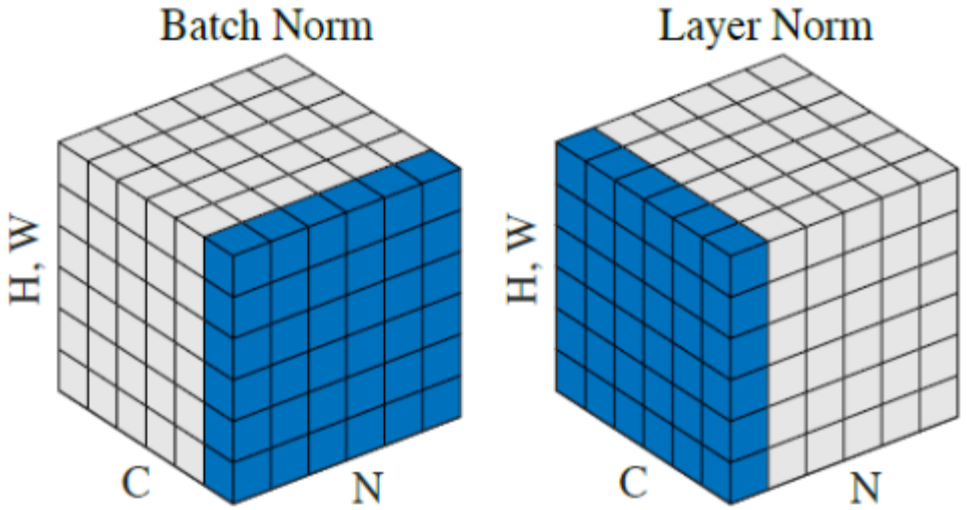
```
net = nn.Sequential(
            nn.Conv2d(1, 6, 5), # in_channels, out_channels, kernel_size
            nn.BatchNorm2d(6),
            nn.Sigmoid(),
            nn.MaxPool2d(2, 2), # kernel_size, stride
            nn.Conv2d(6, 16, 5),
            nn.BatchNorm2d(16),
            nn.Sigmoid(),
            nn.MaxPool2d(2, 2),
            d2l.FlattenLayer(),
            nn.Linear(16*4*4, 120),
            nn.BatchNorm1d(120),
            nn.Sigmoid(),
            nn.Linear(120, 84),
            nn.BatchNorm1d(84),
            nn.Sigmoid(),
            nn.Linear(84, 10)
        )
```

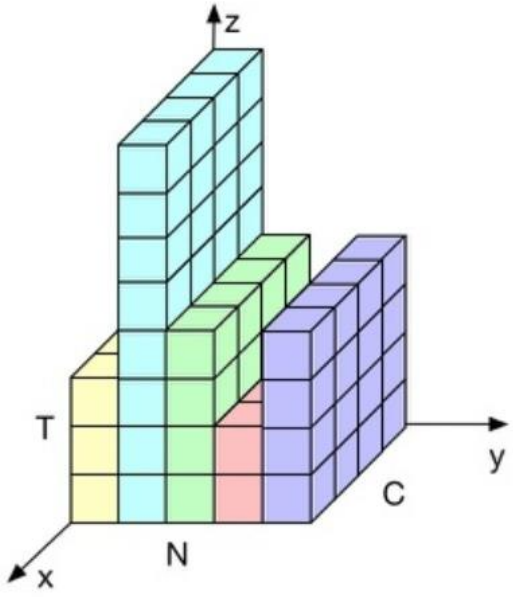
```
batch_size = 256
train_iter, test_iter = d2l.load_data_fashion_mnist(batch_size=batch_size)

lr, num_epochs = 0.001, 5
optimizer = torch.optim.Adam(net.parameters(), lr=lr)
d2l.train_ch5(net, train_iter, test_iter, batch_size, optimizer, device, num_epochs)
```

```
training on  cuda
epoch 1, loss 0.0054, train acc 0.767, test acc 0.795, time 2.0 sec
epoch 2, loss 0.0024, train acc 0.851, test acc 0.748, time 2.0 sec
epoch 3, loss 0.0017, train acc 0.872, test acc 0.814, time 2.2 sec
epoch 4, loss 0.0014, train acc 0.883, test acc 0.818, time 2.1 sec
epoch 5, loss 0.0013, train acc 0.889, test acc 0.734, time 1.8 sec
```

LN的研究动机

- 虽然BN使用mini-batch的均值和标准差对网络隐藏层的输入进行标准化，提升网络训练速度，但是：
  - BN效果受制于batch的大小，小batch无法取得预期效果
  - 针对序列网络，不同的mini-batch具有不同的输入序列长度（深度），计算统计信息比较困难。





LN + MLP

- LN: 根据样本的特征做归一化。

- H是隐层节点数量，l是MLP的层数。

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

$$\hat{\mathbf{a}}^l = \frac{\mathbf{a}^l - \mu^l}{\sqrt{(\sigma^l)^2 + \epsilon}}$$

$$\mathbf{h}^l = f(\mathbf{g}^l \odot \hat{\mathbf{a}}^l + \mathbf{b}^l)$$

LN + RNN

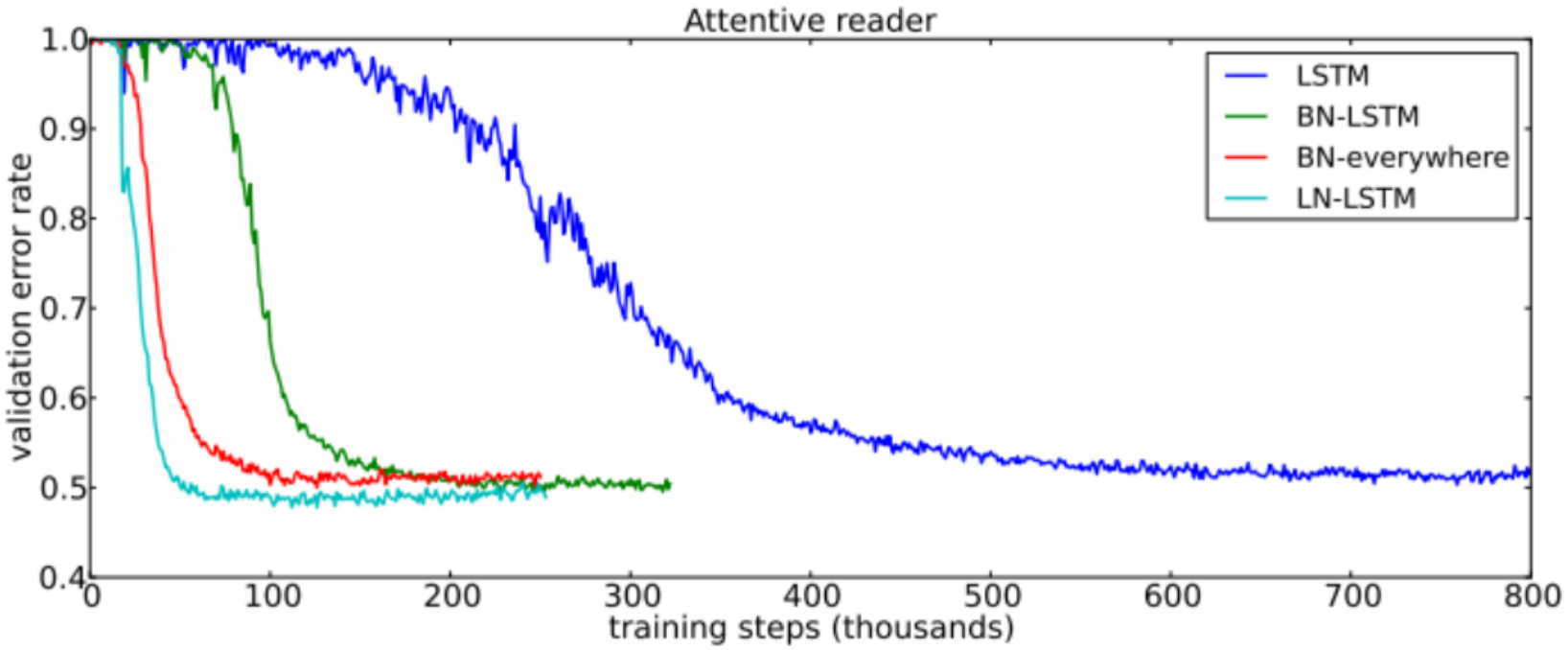
- 给定RNN的表示：

$$\mathbf{a}^t = W_{hh} h^{t-1} + W_{xh} \mathbf{x}^t$$

$$\mathbf{h}^t = f\left(\frac{\mathbf{g}}{\sqrt{(\sigma^t)^2 + \epsilon}} \odot (\mathbf{a}^t - \mu^t) + \mathbf{b}\right) \qquad \mu^t = \frac{1}{H}\sum_{i=1}^{H} a_i^t$$

$$\sigma^t = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^t - \mu^t)^2}$$

LN实验结果


Attentive reader
validation error rate
training steps (thousands)
LSTM
BN-LSTM
BN-everywhere
LN-LSTM
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0
100
200
300
400
500
600
700
800

LN的实现

```python
import torch
import torch.nn as nn
class LayerNorm(nn.Module):
    def __init__(self, hidden_size, eps=1e-12):
        super(LayerNorm, self).__init__()
        self.weight = nn.Parameter(torch.ones(hidden_size))
        self.bias = nn.Parameter(torch.zeros(hidden_size))
        self.variance_epsilon = eps

    def forward(self, x):
        # x 的最后一维代表相应层的神经元数量。
        u = x.mean(-1, keepdim=True)
        s = (x - u).pow(2).mean(-1, keepdim=True)
        x = (x - u) / torch.sqrt(s + self.variance_epsilon)
        return self.weight * x + self.bias
```

LN的调用

```
>>> # NLP Example
>>> batch, sentence_length, embedding_dim = 20, 5, 10
>>> embedding = torch.randn(batch, sentence_length, embedding_dim)
>>> layer_norm = nn.LayerNorm(embedding_dim)
>>> # Activate module
>>> layer_norm(embedding)
>>>
>>> # Image Example
>>> N, C, H, W = 20, 5, 10, 10
>>> input = torch.randn(N, C, H, W)
>>> # Normalize over the last three dimensions (i.e. the channel and spatial dimensions)
>>> # as shown in the image below
>>> layer_norm = nn.LayerNorm([C, H, W])
>>> output = layer_norm(input)
```

LN在Transformer的位置

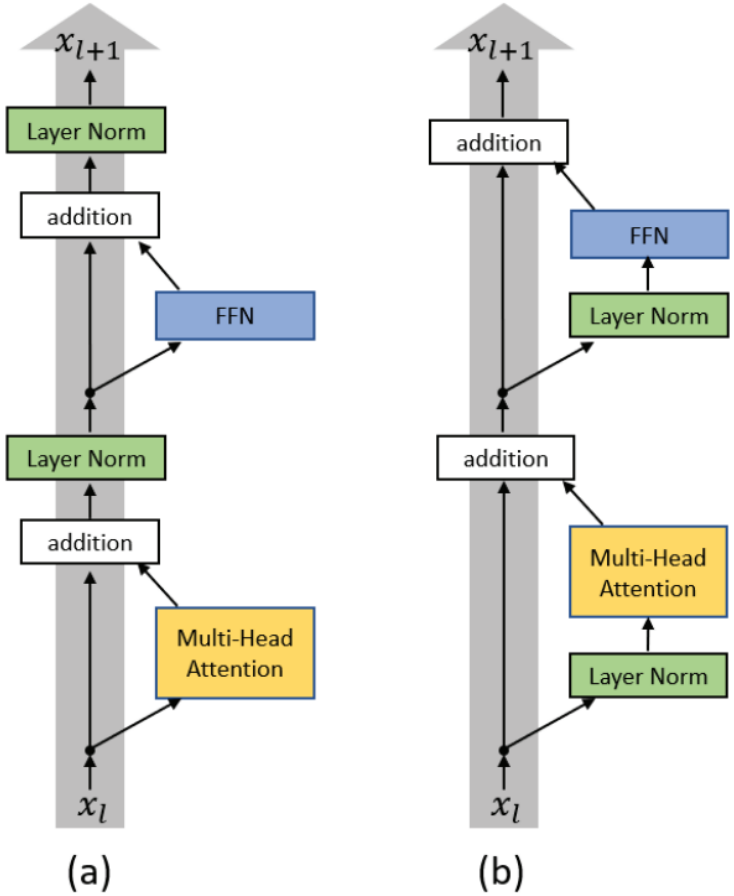


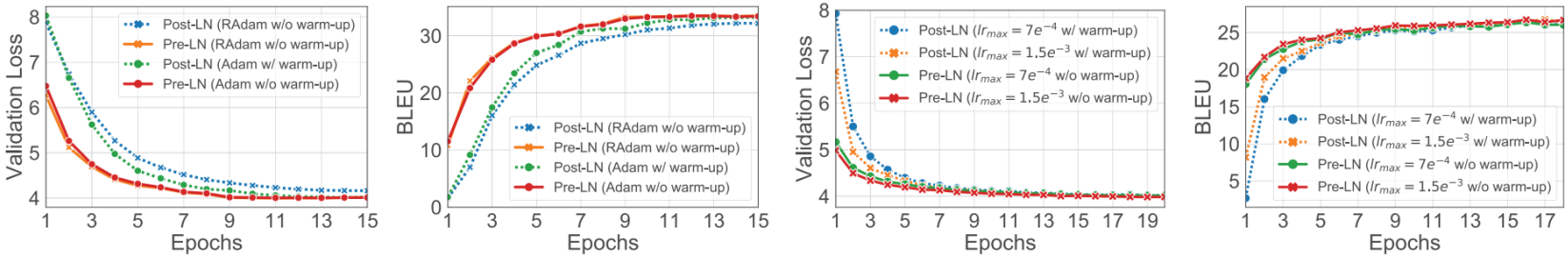
Figure 1. (a) Post-LN Transformer layer; (b) Pre-LN Transformer layer.

LN在Transformer的位置

Table 1. Post-LN Transformer v.s. Pre-LN Transformer

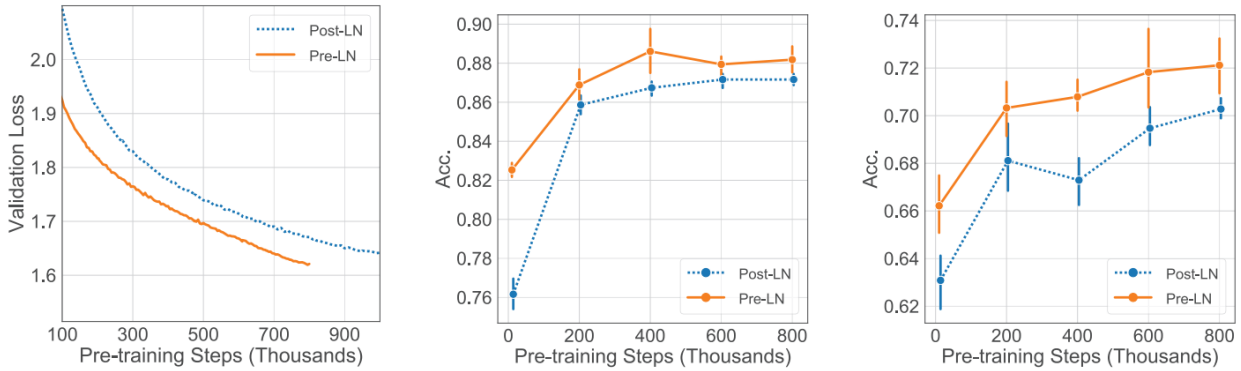
Post-LN Transformer	Pre-LN Transformer
$x_{l,i}^{post,1} = \text{MultiHeadAtt}(x_{l,i}^{post}, [x_{l,1}^{post}, \cdots, x_{l,n}^{post}])$	$x_{l,i}^{pre,1} = \text{LayerNorm}(x_{l,i}^{pre})$
$x_{l,i}^{post,2} = x_{l,i}^{post} + x_{l,i}^{post,1}$	$x_{l,i}^{pre,2} = \text{MultiHeadAtt}(x_{l,i}^{pre,1}, [x_{l,1}^{pre,1}, \cdots, x_{l,n}^{pre,1}])$
$x_{l,i}^{post,3} = \text{LayerNorm}(x_{l,i}^{post,2})$	$x_{l,i}^{pre,3} = x_{l,i}^{pre} + x_{l,i}^{pre,2}$
$x_{l,i}^{post,4} = \text{ReLU}(x_{l,i}^{post,3} W^{1,l} + b^{1,l}) W^{2,l} + b^{2,l}$	$x_{l,i}^{pre,4} = \text{LayerNorm}(x_{l,i}^{pre,3})$
$x_{l,i}^{post,5} = x_{l,i}^{post,3} + x_{l,i}^{post,4}$	$x_{l,i}^{pre,5} = \text{ReLU}(x_{l,i}^{pre,4} W^{1,l} + b^{1,l}) W^{2,l} + b^{2,l}$
$x_{l+1,i}^{post} = \text{LayerNorm}(x_{l,i}^{post,5})$	$x_{l+1,i}^{pre} = x_{l,i}^{pre,5} + x_{l,i}^{pre,3}$
	Final LayerNorm: $x_{Final,i}^{pre} \leftarrow \text{LayerNorm}(x_{L+1,i}^{pre})$

LN在Transformer的位置



(a) Validation Loss (IWSLT) (b) BLEU (IWSLT) (c) Validation Loss (WMT) (d) BLEU (WMT)

Figure 4. Performances of the models on the IWSLT14 De-En task and WMT14 En-De task

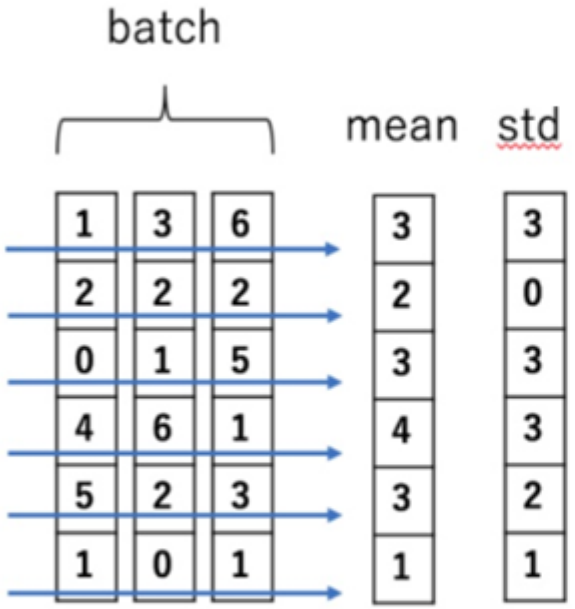


(a) Validation Loss on BERT (b) Accuracy on MRPC (c) Accuracy on RTE

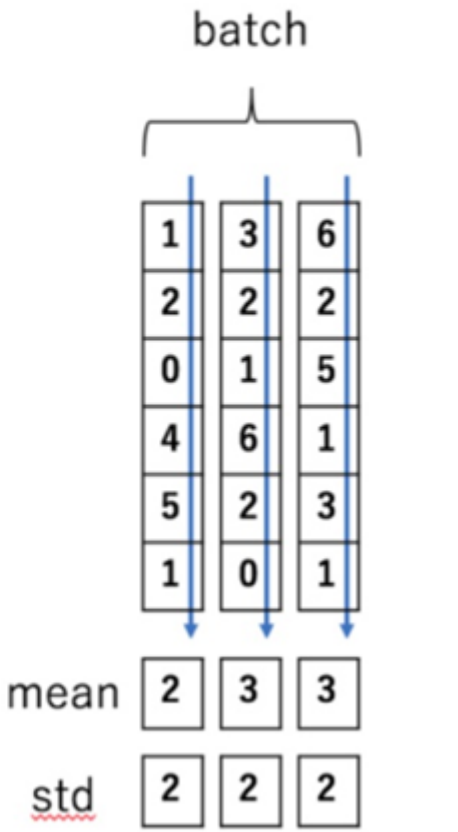
Figure 5. Performances of the models on unsupervised pre-training (BERT) and downstream tasks

BN和LN的区别总结

Batch Normalization



Layer Normalization

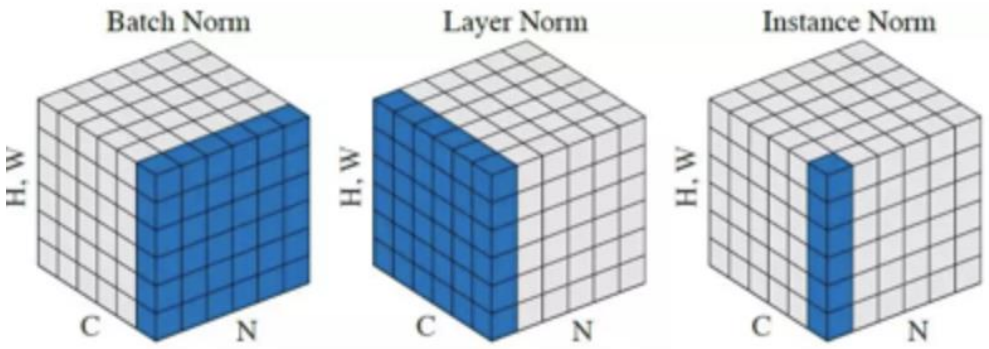


预告

- **Instance Normalization**

Stylization

Figure 4: Stylization examples using proposed method. First row: style images; second row: original image and its stylized versions.



BN v.s. IN

$$\mu_i = \frac{1}{HWT}\sum_{t=1}^{T}\sum_{l=1}^{W}\sum_{m=1}^{H} x_{tilm} \qquad \sigma_i^2 = \frac{1}{HWT}\sum_{t=1}^{T}\sum_{l=1}^{W}\sum_{m=1}^{H}(x_{tilm} - \mu_i)^2 \quad (1)$$

$$y_{tijk} = \frac{x_{tijk} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$$

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

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