Normalization

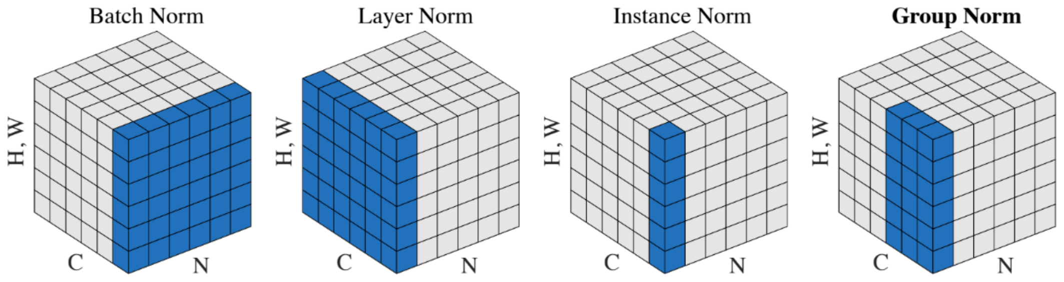
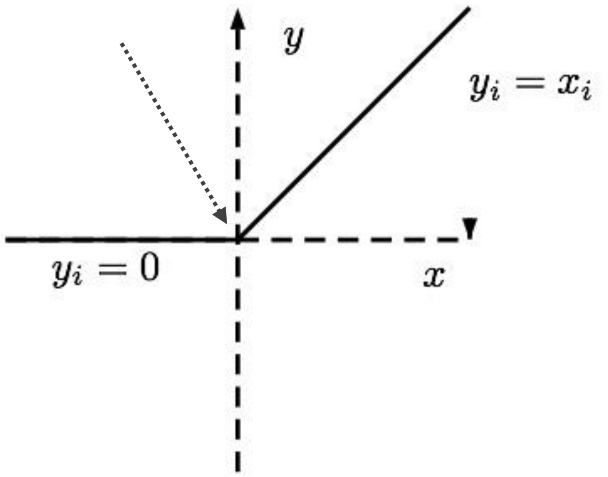


Figure 2. **Normalization methods**. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

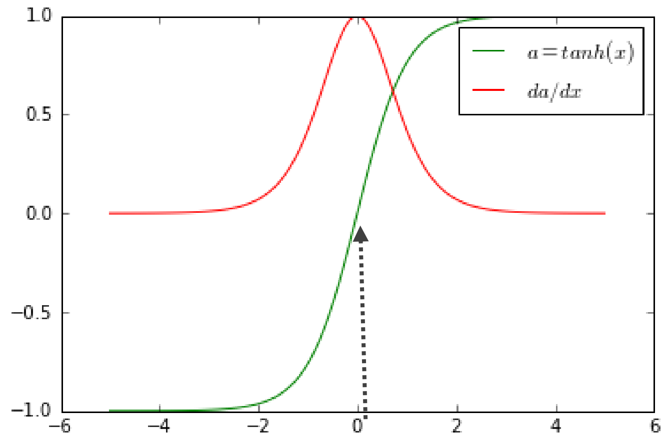
Data pre-processing

- Center data roughly around 0
- Activation functions usually "centered" around 0
  - Propagating to next layer, the mean value remains roughly 0 $\rightarrow$ no shift with depth
  - Also, important for training as often the strongest gradients are around x=0

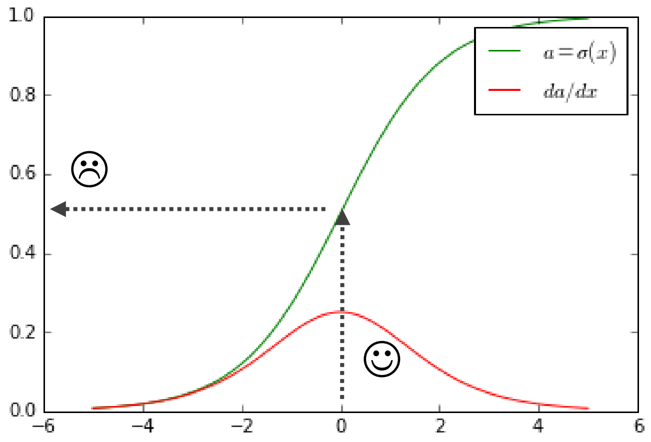
ReLU ☺



tanh(x) ☺



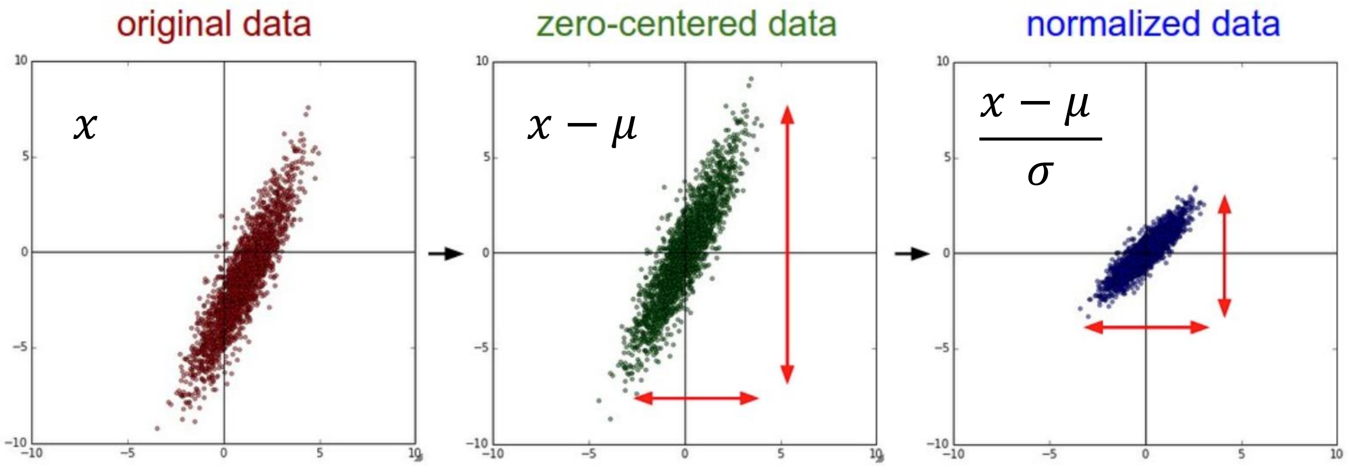
$\sigma(x)$ ☹



Normalizing input to zero-mean, unit variance

- Assume: Input variables follow a Gaussian distribution (roughly)
- Subtract input by the mean
  - Optionally, divide by the standard deviation

$$N(\mu, \sigma^2) \rightarrow N(0, 1)$$



Picture credit: [Stanford Course]

Normalizing intermediate layers

- Batch normalization
- Layer normalization
- Instance normalization
- Group normalization
- Weight normalization

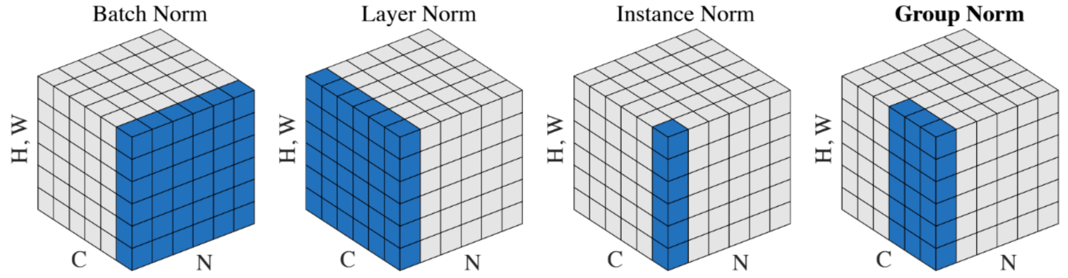


Figure 2. **Normalization methods**. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

Link

Batch normalization

- Input distributions change for per layer, especially during training
- Normalize the layer inputs with batch normalization
  - Normalize $a_l \sim N(0, 1)$
  - Followed by affine transformation

$$a_l \leftarrow \gamma a_l + \beta$$

- The parameters γ and β are trainable

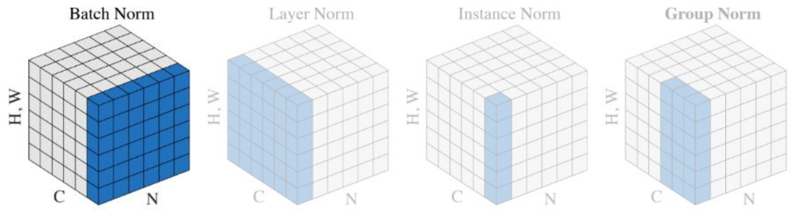
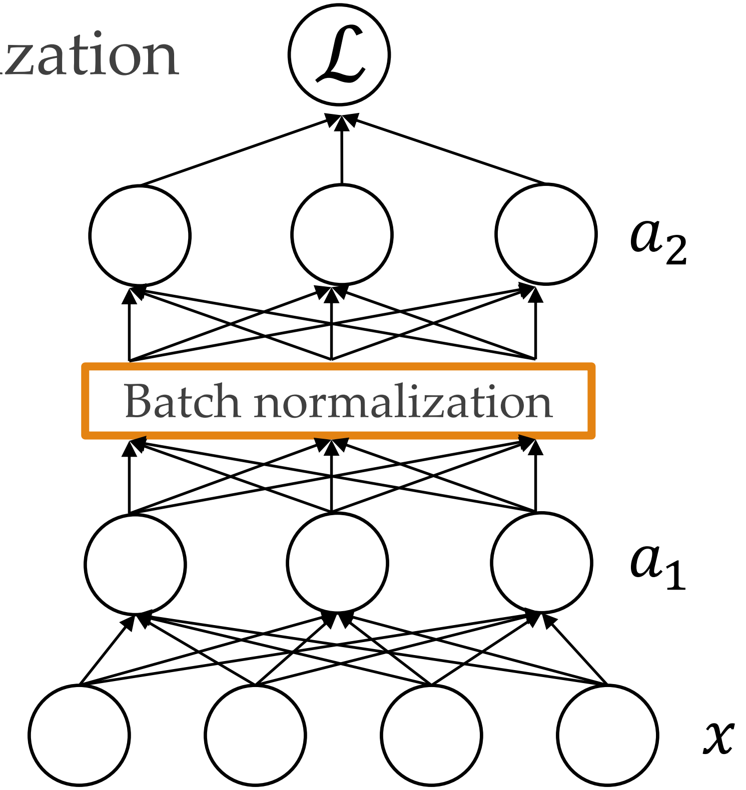


Figure 2. **Normalization methods**. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.



Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, Ioffe, Szegedy, 2015

Batch normalization – The algorithm

- i runs over mini-batch samples, j over the feature dimensions

$\mu_j \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_{ij}$ [compute mini-batch mean]

$\sigma_j^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}\left(x_{ij}-\mu_j\right)^2$ [compute mini-batch variance]

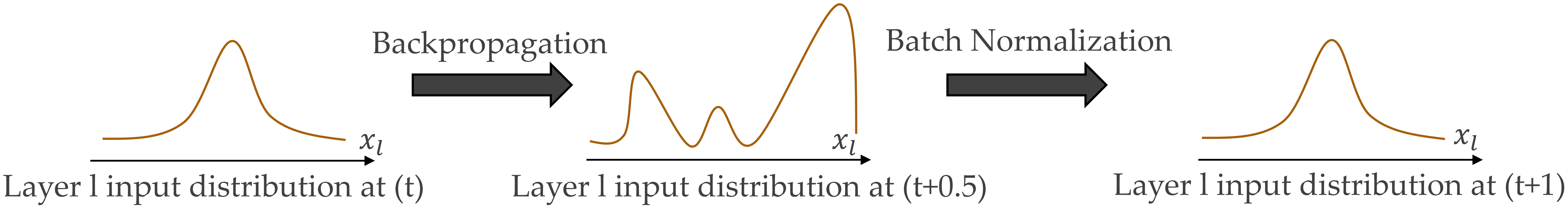
$\hat{x}_{ij} \leftarrow \frac{x_{ij}-\mu_j}{\sqrt{\sigma_j^2+\varepsilon}}$ [normalize input]

$\hat{x}_{ij} \leftarrow \gamma\hat{x}_{ij} + \beta$ [scale and shift input]

Trainable parameters

Batch normalization – Interpretation I

- Covariate shift
  - Per gradient update a module must adapt the weights to fit better the data
  - But also adapt to the change of its input distribution
  - Remember, each module inputs depend on other parameterized modules
- The distribution fed to the layers of a network should be somewhat:
  - Zero-centered
  - Constant through time and data



Batch normalization – Interpretation II

- Batch norm simplifies the learning dynamics
  - Neural network outputs determined by higher order layer interactions
  - They complicate the gradient update
  - Mean of BatchNorm output is β, std is γ
  - They are independent of the activation values themselves
  - Higher order interactions suppressed, training becomes easier
- This angle better explains practical observations:
  - Why batch norm works better after the nonlinearity?
  - Why have γ and β if the problem is the covariate shift?

Batch normalization - Benefits

- Higher learning rates → faster training
- Neurons of all layers activated in near optimal "regime"
- Model regularization
  - Add some noise to per mini-batch mean and variance
  - The added noise reduces overfitting

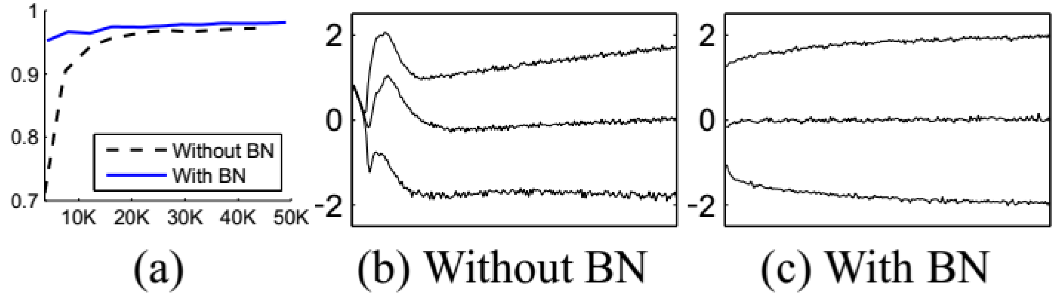


Figure 1: (a) *The test accuracy of the MNIST network trained with and without Batch Normalization, vs. the number of training steps. Batch Normalization helps the network train faster and achieve higher accuracy.* (b, c) *The evolution of input distributions to a typical sigmoid, over the course of training, shown as* $\{15, 50, 85\}$*th percentiles. Batch Normalization makes the distribution more stable and reduces the internal covariate shift.*

From training to test time

- How do we ship the Batch Norm layer after training?
  - We might not have batches at test time
- Usually: keep a moving average of the mean and variance during training
  - Plug them in at test time
  - To the limit, the moving average of mini-batch statistics approaches the batch statistics

- $\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$
- $\sigma_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$
- $\widehat{x_i} \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \varepsilon}}$
- $\widehat{y_i} \leftarrow \gamma \widehat{x_i} + \beta$

Disadvantages of batch normalizations

- Requires large mini-batches
  - Cannot work with mini-batch of size 1 ($\sigma = 0$)
  - And for small mini-batches we don't get very accurate gradients anyways
- Awkward to use with recurrent neural networks
  - Must interleave it between recurrent layers
  - Also, store statistics per time step
- Alternatives have been explored
  - For a good summary check this blogpost

Layer normalization

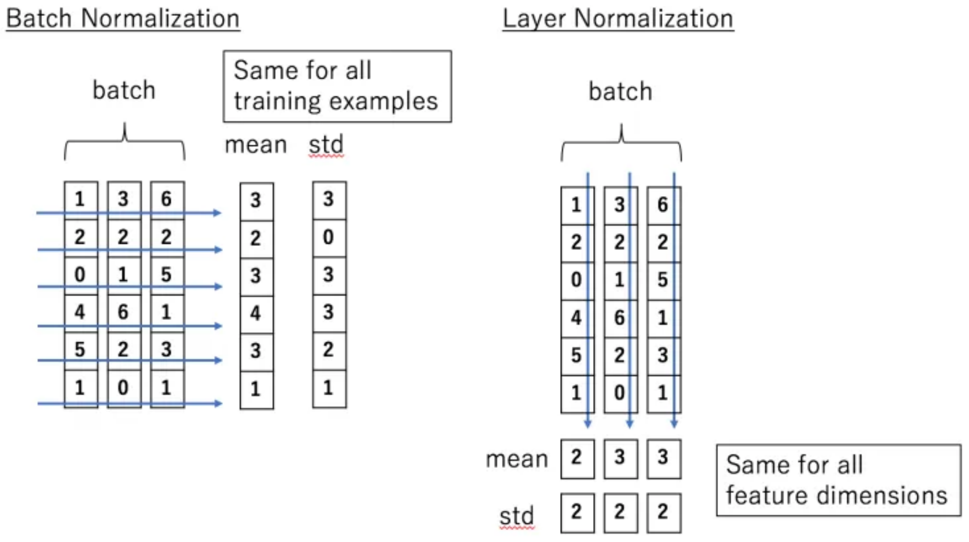
- i runs over mini-batch samples, j over the feature dimensions

$$\mu_i \leftarrow \frac{1}{m}\sum_{j=1}^{m} x_{ij} \qquad \text{[mean over features]}$$

$$\sigma_i^2 \leftarrow \frac{1}{m}\sum_{j=1}^{m}(x_i - \mu_\mathcal{B})^2 \qquad \text{[variance over features]}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_i}{\sqrt{\sigma_i^2 + \varepsilon}} \qquad \text{[normalize input]}$$

$$\hat{y}_i \leftarrow \gamma \hat{x}_i + \beta \qquad \text{[scale and shift input]}$$



- Originally proposed for RNNs
  - Not as good in image classification

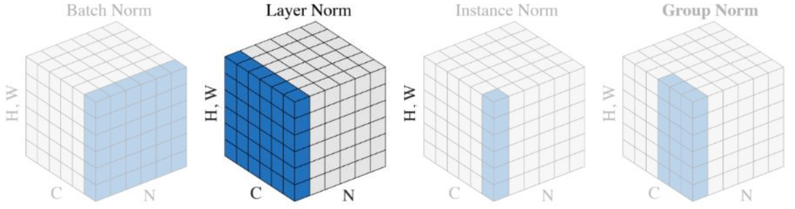


Figure 2. **Normalization methods**. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

Layer Normalization, Ba, Kiros, Hinton, 2016

Instance normalization

- Similar to layer normalization but per channel per training example
- Basic idea: network should be agnostic to the contrast of the original image
- Originally proposed for style transfer
  - Not as good in image classification

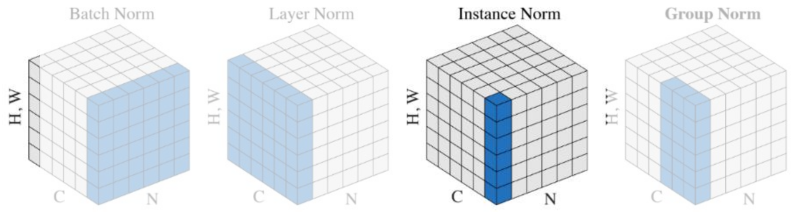


Figure 2. **Normalization methods**. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

Instance Normalization: The Missing Ingredient for Fast Stylization, Ulyanov, Vedaldi, Lempitsky, 2017

Group normalization

- Same as instance norm but over groups of channels
  - Between layer normalization and instance normalization
- Better than batch normalization for small batches (e.g., <32)
  - Competitive for larger batches
- Useful for object detection/segmentation networks
  - They rely on high resolution images and cannot have big mini-batches

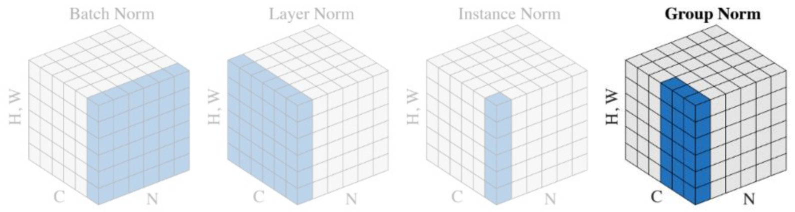


Figure 2. **Normalization methods**. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

Group Normalization, We, He, 2018

Weight normalization

- Instead of normalizing activations, normalize weights
- Re-parameterize weights

$$\boldsymbol{w} = g \frac{\boldsymbol{v}}{||\boldsymbol{v}||}$$

- Separate the norm from the direction
- Similar to dividing by standard deviation in batch normalization
- Can be combined with mean-only batch normalization
  - Subtract the mean (but not divide by the standard deviation)
  - Then, apply weight normalization

Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks, Salimans, Kingma, 2016