Normalizations

Recap - A Simple Example

n-layer linear network

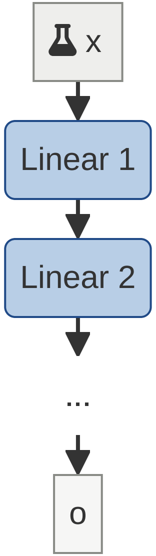
- No non-linearities
- Scalar weight $w_i \in \mathbb{R}, w_i \geq 0$
- Bias $b_i \in \mathbb{R}$

Major problem: vanishing gradients

$$\|\nabla_{W_i} y\| \to 0 \text{ for large } n$$

Inconvenience: vanishing activations

$$a_n = \underbrace{\prod_{k=1}^{n} W_k}_{\to 0} x + \underbrace{\ldots}_{\text{bias}}$$



Normalization

Rescale and shift activations

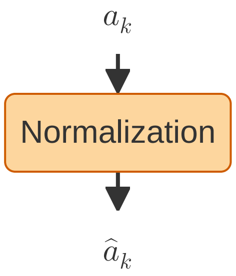
$$\hat{a}_k = \frac{a_k - \mu_k}{\sigma_k}$$

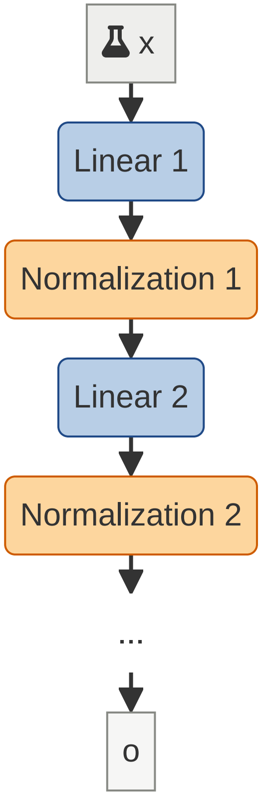
Exploding activation $||a_k|| \to \infty$:

$$\sigma_k \approx ||a_k|| \to \infty \text{ and } ||\hat{a}_k|| \approx 1$$

Vanishing activation $||a_k|| \to 0$:

$$\sigma_k \approx ||a_k|| \to 0 \text{ and } ||\hat{a}_k|| \approx 1$$



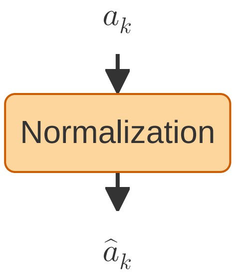


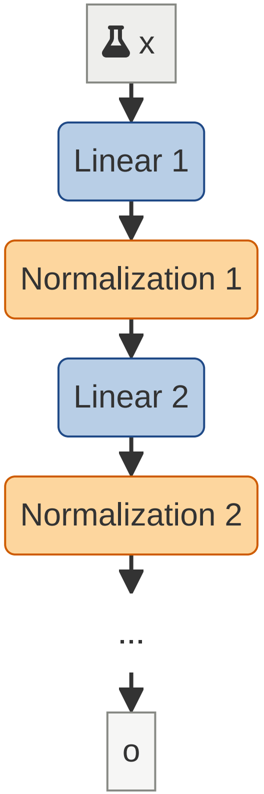
Normalization

Rescale and shift activations

$$\hat{a}_k = \frac{a_k - \mu_k}{\sigma_k}$$

Where do μ_k and σ_k come from?



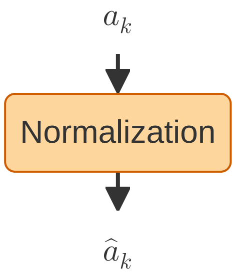


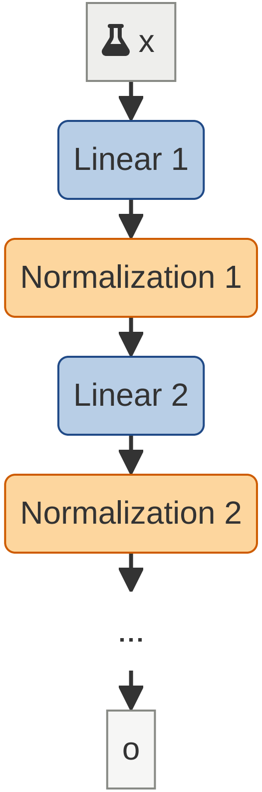
Batch Normalization

Rescale and shift activations ***per channel***

$$\hat{a}_k = \frac{a_k - \mu_k}{\sigma_k}$$

Compute μ_k and σ_k from training batch (on the fly)





S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In ICML, 2015 ↩

Batch Normalization

Rescale and shift activations ***per channel***

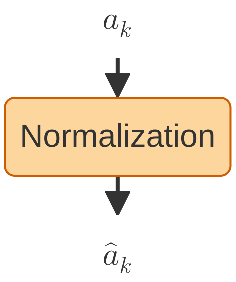
- mean μ_c
- stdev σ_c

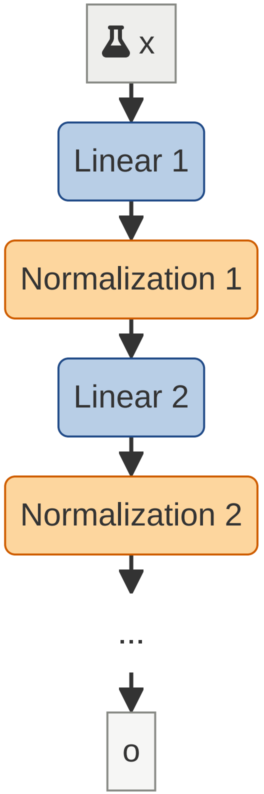
$$\hat{\mathbf{a}}_{b,x,y,c} = \frac{\mathbf{a}_{b,x,y,c} - \mu_c}{\sigma_c}$$

where

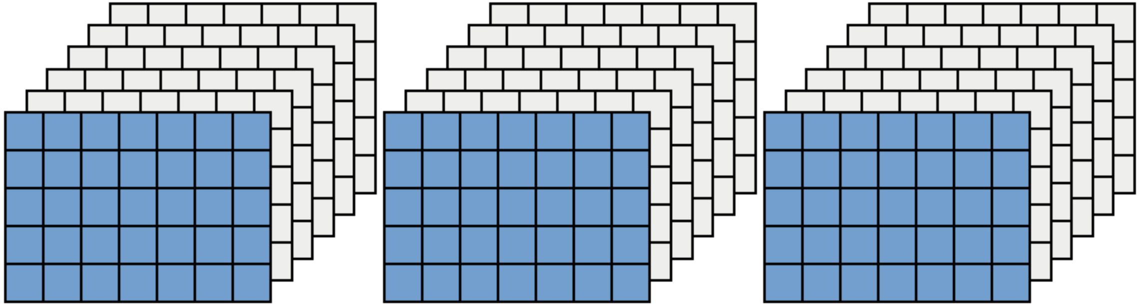
$$\mu_c = \frac{1}{BWH} \sum_{k,x,y} \mathbf{a}_{b,x,y,c}$$

$$\sigma_c^2 = \frac{1}{BWH} \sum_{k,x,y} (\mathbf{a}_{b,x,y,c} - \mu_c)^2$$





What Does Batch Normalization Do?



The Good:

- Regularizes the network
- Handles badly scaled weights

The Bad:

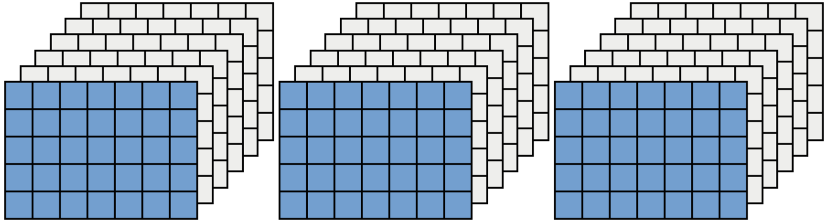
- Mixes gradient info between samples

In General:

- Large batch sizes work better
- More stable mean and stdev estimates

Batch Norm at Test Time

Issue: There is no batch at test time



Solution: Use training mean and stdev

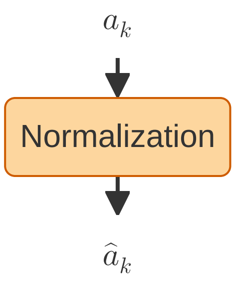
- Keep track of mean and stdev during training
- Implemented via running averages

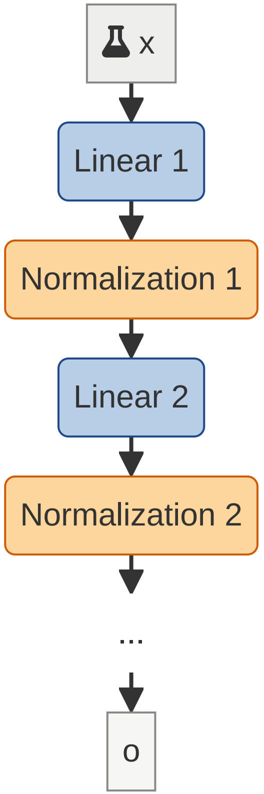
Layer Normalization

Rescale and shift activations **per feature**

$$\hat{a}_k = \frac{a_k - \mu_k}{\sigma_k}$$

Compute μ_k and σ_k across each data element





Layer Normalization, Ba, J., Kiros, J. R. and Hinton, G., arXiv preprint arXiv:1607.06450, 2016 ↩️

Layer Normalization

Rescale and shift activations **per feature**

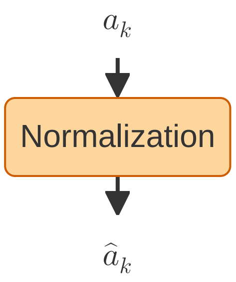
- mean μ_b
- stdev σ_b

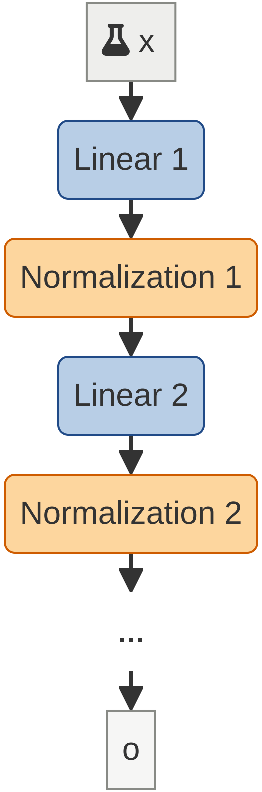
$$\hat{\mathbf{a}}_{b,x,y,c} = \frac{\mathbf{a}_{b,x,y,c} - \mu_b}{\sigma_b}$$

where

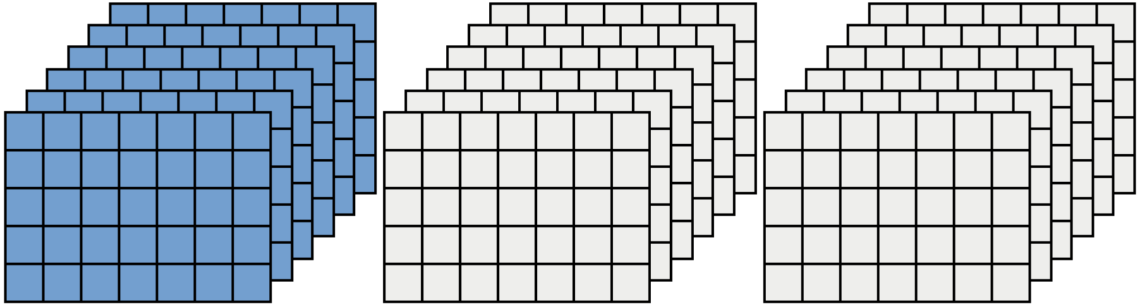
$$\mu_b = \frac{1}{WHC} \sum_{x,y,c} \mathbf{a}_{b,x,y,c}$$

$$\sigma_b^2 = \frac{1}{WHC} \sum_{x,y,c} (\mathbf{a}_{b,x,y,c} - \mu_b)^2$$





What Does Layer Normalization Do?



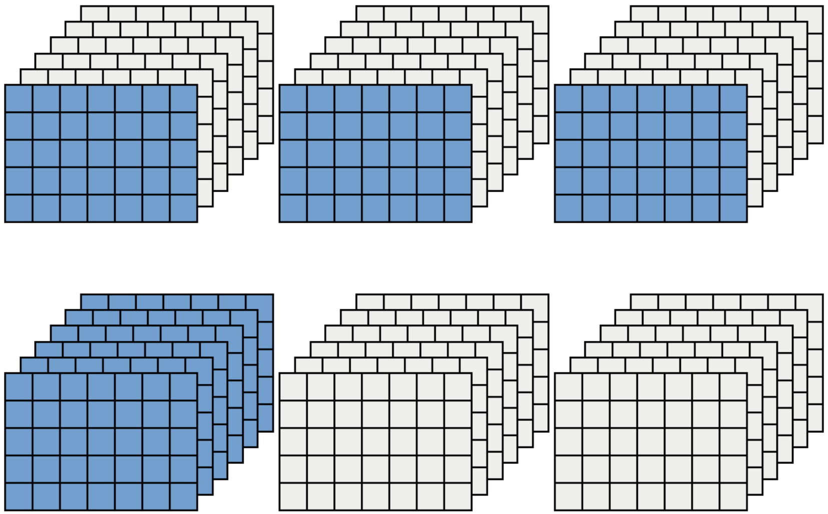
Comparison to Batch Norm

No Summary Statistics

- Training and testing are the same

In General:

- Works well for sequence models
- Does not normalize activations individually



Group Normalization

Rescale and shift activations ***per group***

- mean $\mu_{k,g}$
- stdev $\sigma_{k,g}$

$$\mathbf{a}_{b,x,y,c} = \frac{\mathbf{a}_{b,x,y,c} - \mu_{k,g}}{\sigma_{k,g}}$$

where

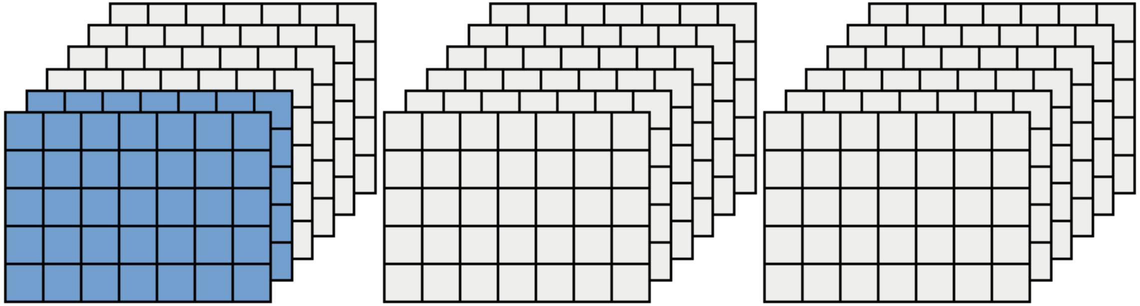
$$g = \lfloor c/G \rfloor$$

$$\mu_{k,g} = \frac{1}{WHG} \sum_{c=gG}^{(g+1)G-1} \sum_{x,y} \mathbf{a}_{b,x,y,c}$$

$$\sigma^2_{k,g} = \frac{1}{WHG} \sum_{c=gG}^{(g+1)G-1} \sum_{x,y} (\mathbf{a}_{b,x,y,c} - \mu_{k,g})^2$$

Group normalization. Yuxin Wu, and Kaiming He. ECCV. 2018. ↩️

What Does Group Normalization Do?



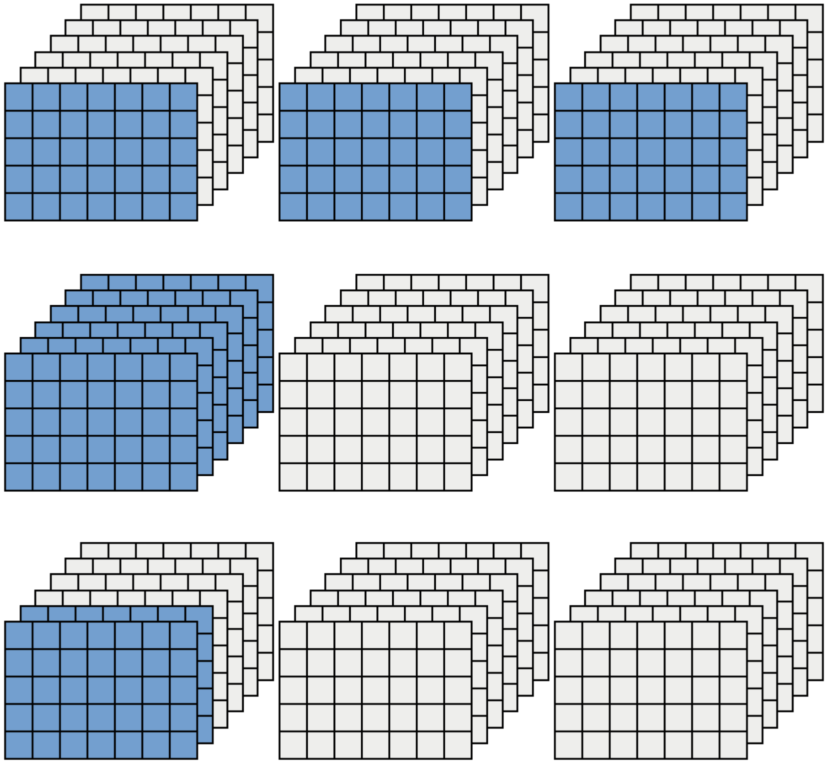
Group Normalization Comparison

Key Characteristics

- Splits channels into groups
- Layer norm is a special case of group norm (G=1)

In Practice:

- Common in UNet/diffusion architectures



Local response normalization

"Generalization" of group norm

- Hyperparameters α and β
- $\mathbf{a} \in \mathbb{R}^{B \times W \times H \times C}$

$$\hat{\mathbf{a}}_{b,x,y,c} = \mathbf{a}_{b,x,y,c}\left(\gamma + \frac{\alpha}{n} \sum_{c'=c-n/2}^{c+n/2} \mathbf{a}^2_{b,x,y,c}\right)^{-\beta}$$

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012

Differences Between LRN and GN

Group Normalization

$$\mathbf{a}_{b,x,y,c} = \frac{\mathbf{a}_{b,x,y,c} - \mu_{k,g}}{\sigma_{k,g}}$$

- Normalize over all spatial locations
- Subtract mean

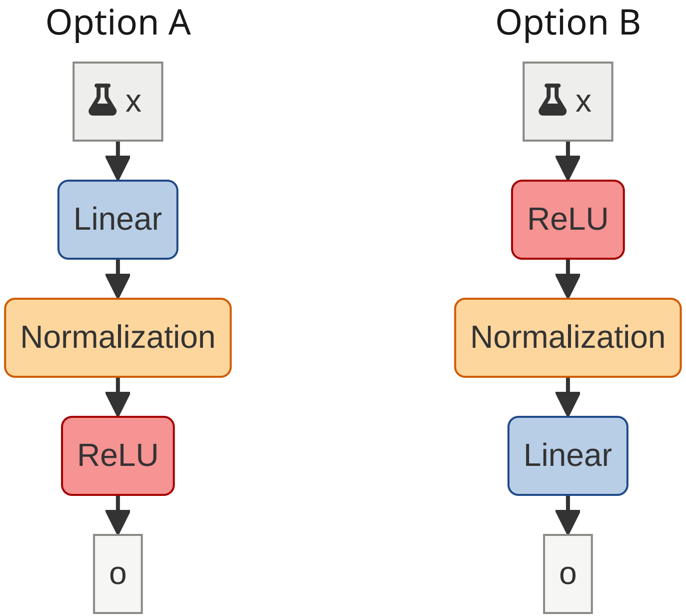
Local response normalization

$$\hat{\mathbf{a}}_{b,x,y,c} = \mathbf{a}_{b,x,y,c}\left(\gamma + \frac{\alpha}{n}\sum_{c'=c-n/2}^{c+n/2} \mathbf{a}^2_{b,x,y,c}\right)^{-\beta}$$

- More flexible parametrization
- Sliding window

Where to Add Normalization?

- Option A: after linear
- Option B: after activation



Option A: Post-Linear Normalization

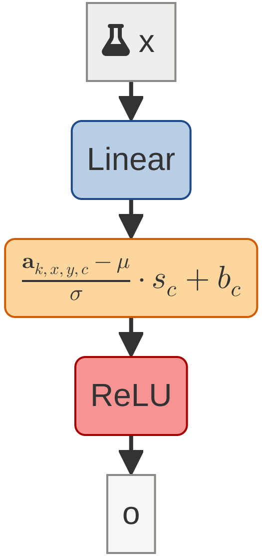
Output activation mean $\hat{\mu}_k = 0$

- No need for bias b_c in linear layer

Issue: about half activations negative

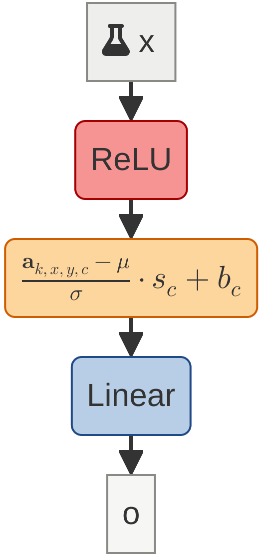
- Zero-ed out by ReLU

Solution: learn a scale s_c and bias b_c after norm



Option B: Post-Activation Normalization

Scale s_c and bias b_c optional



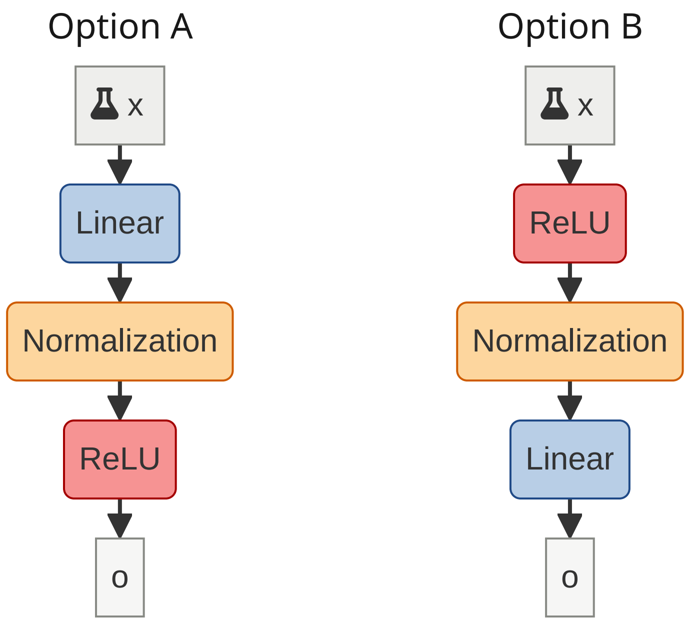
Where to Add Normalization?

Both work

Option A: **more popular**

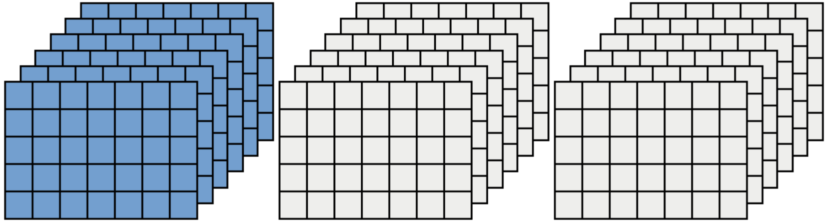
Option B: **easier**

- Scale and bias optional
- Layer unchanged

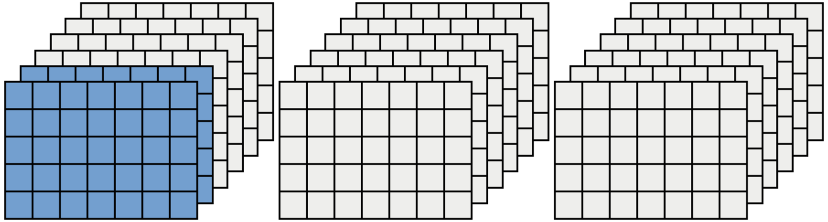


What Normalization to Use?

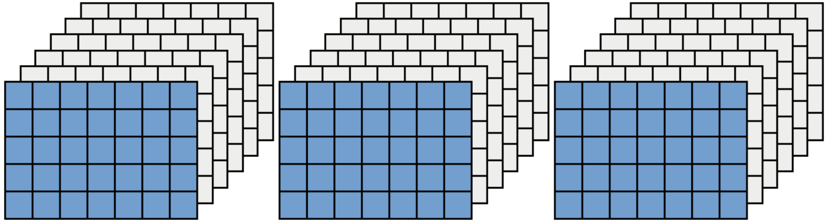
1. Try LayerNorm



2. Try GroupNorm



3. Try BatchNorm
  - Most suitable for image-like data
  - Do **NOT** use on vanilla linear layers

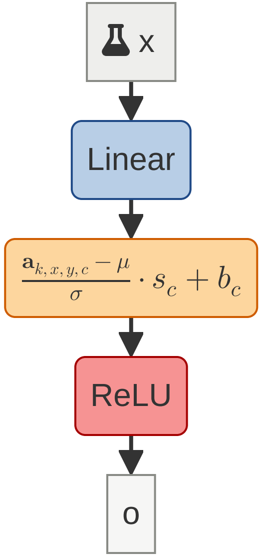


Why Does Normalization Work?

Normalization fixes vanishing activations

- Handle badly scaled weights
- Activations cannot vanish (assumming $b_c = 0$)
- "Eigenvalues" are close to 1

The same holds true for gradients [1].

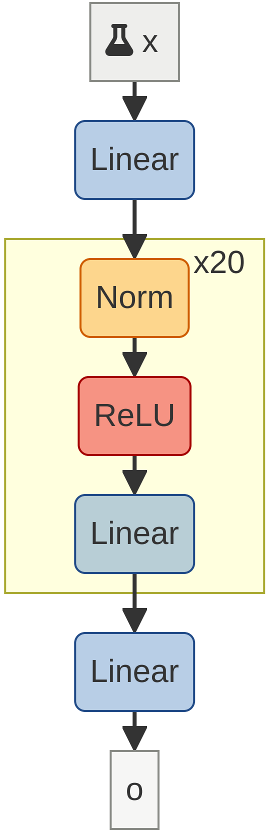


1. Li, H., Xu, Z., Taylor, G., Studer, C. and Goldstein, T., Visualizing the Loss Landscape of Neural Nets. In NeurIPS 2018. ↩

How Deep Can These Networks Go?

With normalization

- Max depth 20-30



Normalization - TL;DR

Normalizations handle vanishing gradients