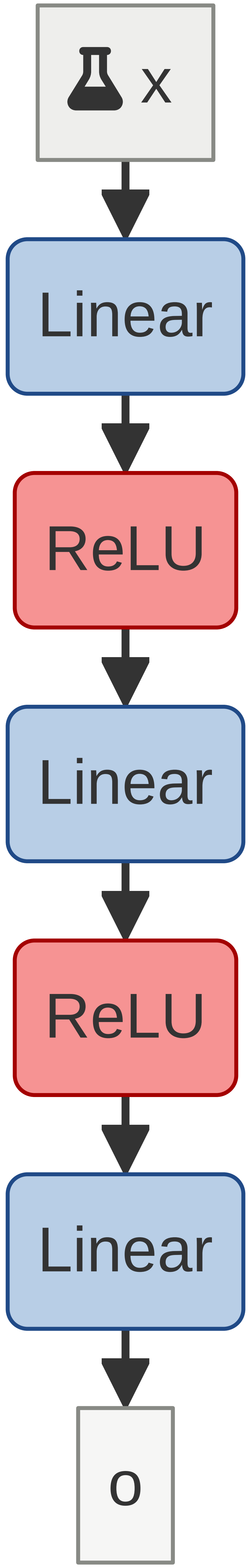
Residuals and Normalizations

Philipp Krähenbühl, UT Austin

Recap: Deep Networks

- Model: $f_\theta : \mathbb{R}^n \to \mathbb{R}^c$
- Parameters: $\theta = (\mathbf{W}_1, \mathbf{b}_1, \ldots, \mathbf{W}_N, \mathbf{b}_N)$
- Computation:

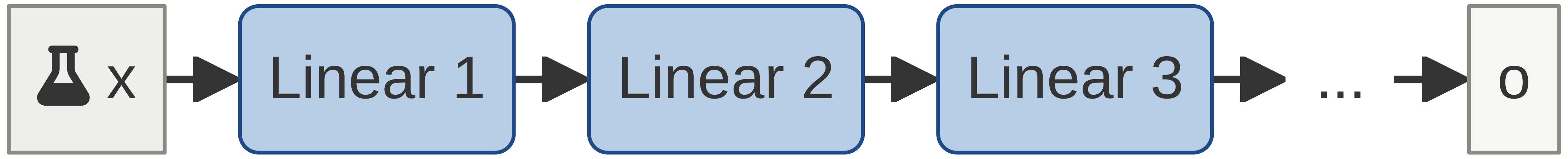
$$\begin{aligned}
\mathbf{z}_1 &= \text{ReLU}(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) \\
\mathbf{z}_2 &= \text{ReLU}(\mathbf{W}_2\mathbf{z}_1 + \mathbf{b}_2) \\
&\vdots \\
\mathbf{z}_{N-1} &= \text{ReLU}(\mathbf{W}_{N-1}\mathbf{z}_{N-2} + \mathbf{b}_{N-1}) \\
f_\theta(\mathbf{x}) &= \mathbf{W}_N\mathbf{z}_{N-1} + \mathbf{b}_N
\end{aligned}$$



Building the deepest network ever built in PyTorch

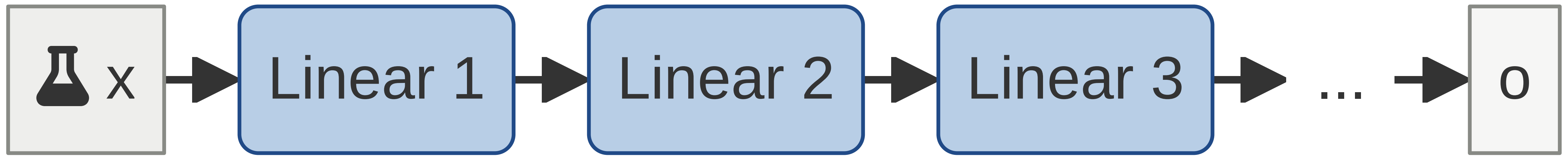
Vanishing and Exploding Gradients

A Simple Example



- n-layer linear network
  - No non-linearities
  - Scalar weight $w_i \in \mathbb{R}, w_i > 0$
  - Bias $b_i \in \mathbb{R}$

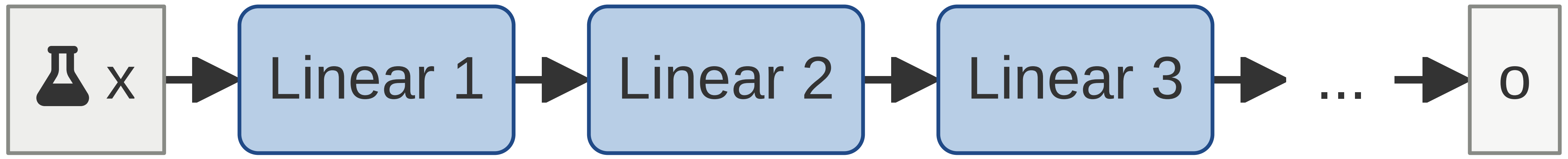
Activations



- Assume $w_i \approx w, b_i \approx b \quad \forall_i$
- Case 1: $w < 1$
- Case 2: $w = 1$
- Case 3: $w > 1$
  -

Layer	**Activation**
$l = 1$	$a_1 \approx wx + b$
$l = 2$	$a_2 \approx w^2x + (w+1)b$
$l = 3$	$a_3 \approx w^3x + (w^2 + w + 1)b$
$l = n$	$a_n \approx w^nx + b\underbrace{\sum_{k=0}^{n-1} w^k}_{\frac{1-w^n}{1-w}}$

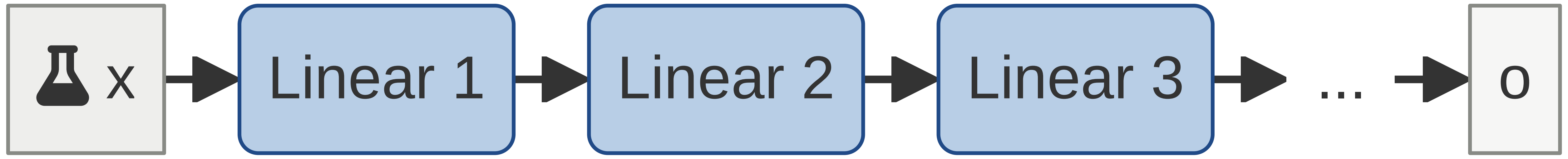
Activations



- Assume $w_i \approx w, b_i \approx b \quad \forall_i$
- Case 1: $w < 1$
  - $w^n x \to 0$ for large n
  - Vanishing inputs: $a_n \approx \dfrac{b}{1-w}$
- Case 2: $w = 1$
  - $a_n = x + nb$
- Case 3: $w > 1$
  - $w^n \to \infty$
  - Exploding activations: $a_n \approx w^n x \to \pm \infty$

Layer	Activation
$l = 1$	$a_1 \approx wx + b$
$l = 2$	$a_2 \approx w^2 x + (w+1)b$
$l = 3$	$a_3 \approx w^3 x + (w^2 + w + 1)b$
$l = n$	$a_n \approx w^n x + b\underbrace{\sum_{k=0}^{n-1} w^k}_{\frac{1-w^n}{1-w}}$

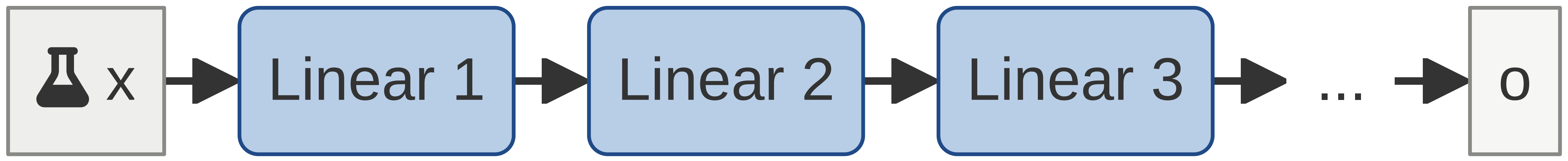
Gradients



- Assume $w_i \approx w, b_i \approx b \quad \forall_i$
- Case 1: $w < 1$
- Case 2: $w = 1$
- Case 3: $w > 1$

Layer	**Gradient** $\nabla_{w_i} y = a_{i-1} \underbrace{\prod_{k=i+1}^{n} w^k}_{\approx w^{n-i}}$
$l = 1$	$\nabla_{w_1} y \approx x w^{n-1}$
$l = 2$	$\nabla_{w_2} y \approx a_1 w^{n-2}$
$l = k$	$\nabla_{w_k} y \approx a_{k-1} w^{n-k}$
$l = n$	$\nabla_{w_n} y \approx a_{n-1}$

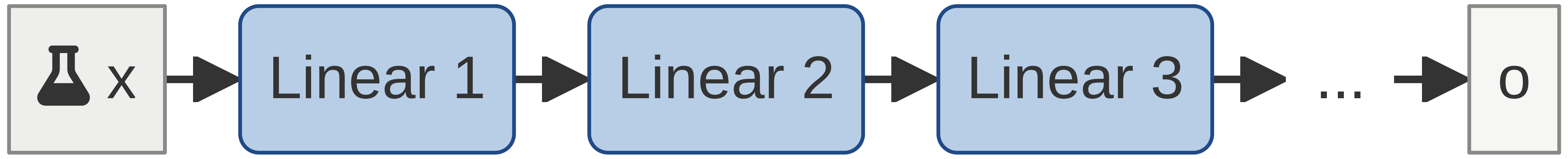
Gradients



Layer	**Gradient** $\nabla_{w_i} y = a_{i-1} \underbrace{\prod_{k=i+1}^{n} w^k}_{\approx w^{n-i}}$
$l = 1$	$\nabla_{w_1} y \approx x w^{n-1}$
$l = 2$	$\nabla_{w_2} y \approx a_1 w^{n-2}$
$l = k$	$\nabla_{w_k} y \approx a_{k-1} w^{n-k}$
$l = n$	$\nabla_{w_n} y \approx a_{n-1}$

- Assume $w_i \approx w, b_i \approx b \quad \forall_i$
- Case 1: $w < 1$
  - Gradient vanishes: $w^{n-i} \to 0$ for large n (and small i)
- Case 2: $w = 1$
  - Gradient stable: $\nabla_{w_i} y \approx a_{i-1}$
- Case 3: $w > 1$
  - Gradient explodes: $w^{n-i-1} \to \infty$ for large n (and small i)

Simple Example - Summary



$w < 1$

- Training stable
- Vanishing activations
- Vanishing gradients
- Network does not train

$w = 1$

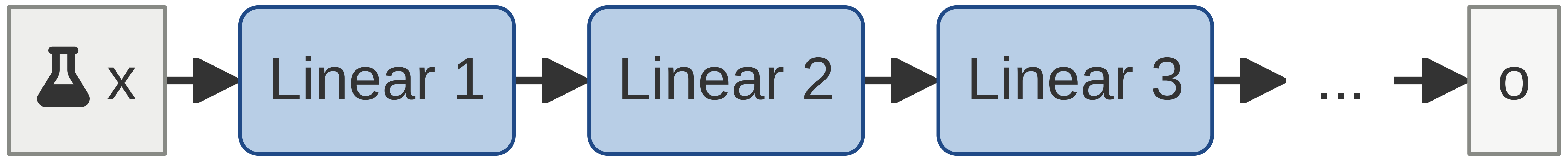
- Training stable
- Network does train
- Nearly impossible to maintain

$w > 1$

- Training explodes
- Exploding activations
- Exploding gradients
- Network does not train

General Linear Networks

Exploding Gradients



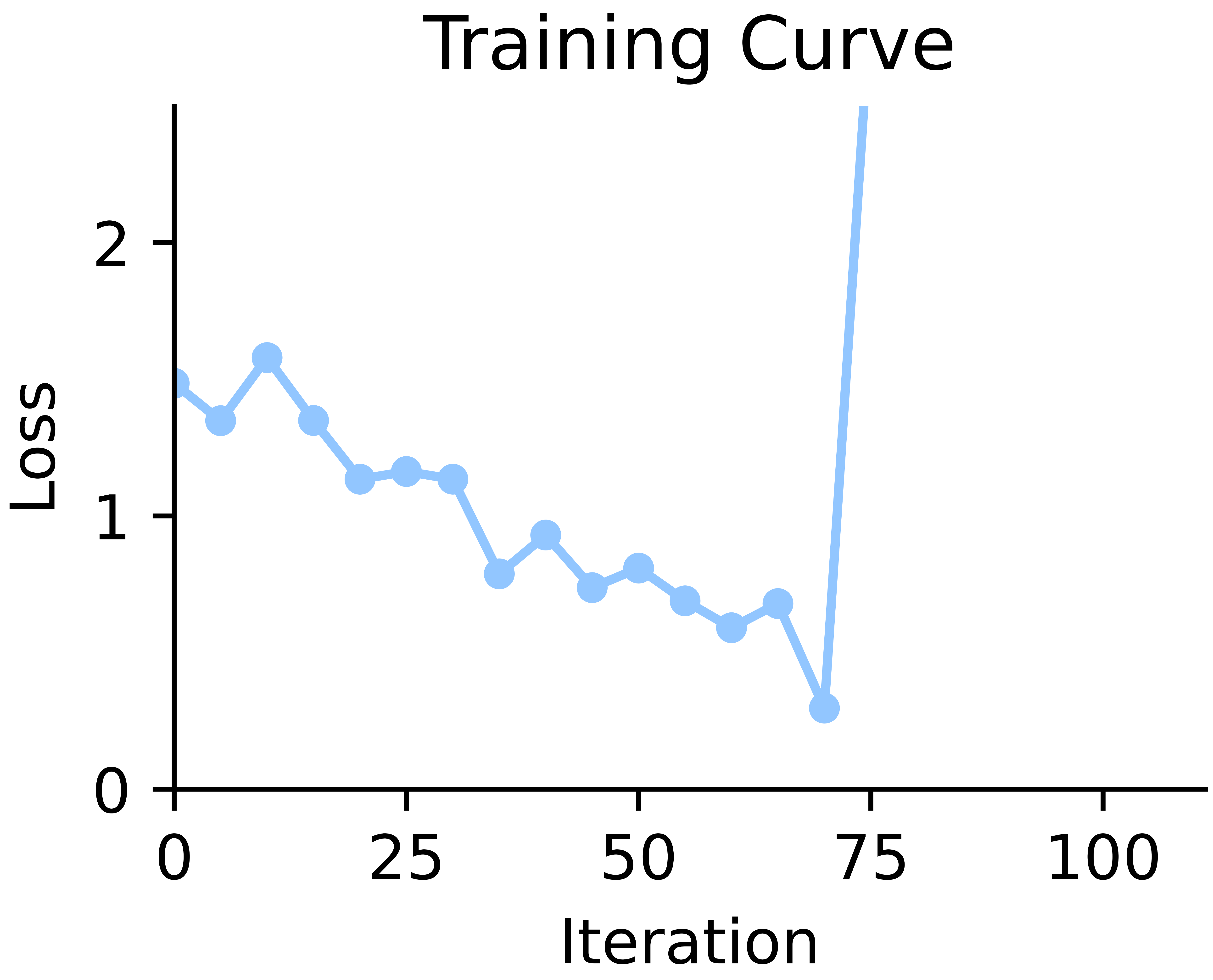
- Weights $W_i \in \mathbb{R}^{n \times n}$
- Exploding activations
$$\|a_k\| = \|\prod_{i=1}^{k} W_k x\| \approx \prod_{i=1}^{k} \|W_k\| \|x\| \to \infty$$
- Exploding gradients
$$\|\nabla_{W_k} o\| \approx \|a_{k-1}\| \prod_{i=k+1}^{n} \|W_i\| \to \infty$$
- Network poorly initialized $\|W_i\| > 1$
- Learning rate too high

Handling Exploding Gradients

- 💥 Symptoms
- 🔍 Diagnosis
- 🛠️ Remedy

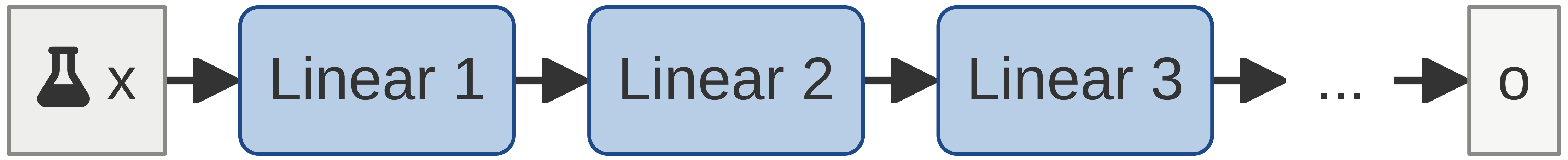
Handling Exploding Gradients

- 💥 Symptoms
  - Weights or loss are ♾️ or NaN
- 🔍 Diagnosis
  - Plot weight norms per layer
  - Plot gradient norms per layer
- 🛠️ Remedy
  - Reduce learning rate
  - Rarely: change initialization



General Linear Networks

Vanishing Gradients



- Weights $W_i \in \mathbb{R}^{n \times n}$
- Vanishing activations

$$\|a_k\| = \| \prod_{i=1}^{k} W_k x \| \leq \prod_{i=1} \|W_k\| \|x\| \to 0$$

- Vanishing gradients

$$\|\nabla_{W_k} o\| \leq \|a_{k-1}\| \prod_{i=k+1}^{n} \|W_i\| \to 0$$

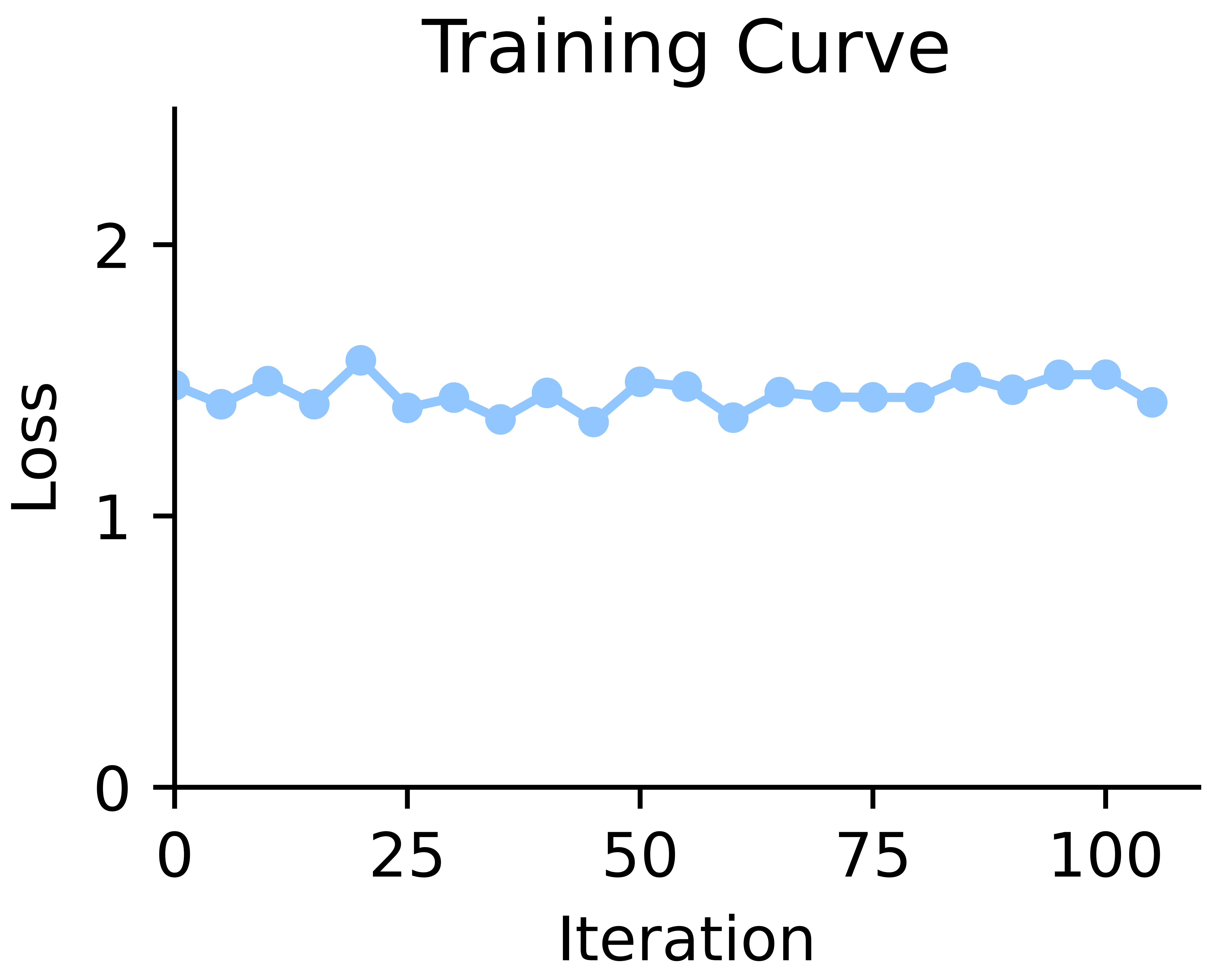
- Occurs in almost all deep networks

Handling Vanishing Gradients

- 💥 Symptoms
- 🔍 Diagnosis
- 🛠️ Remedy

Handling Vanishing Gradients

- 💥 Symptoms
  - Network does not train
- 🔍 Diagnosis
  - Train with 0 learning rate and compare
  - Plot weight norms per layer
  - Plot gradient norms per layer
- 🛠️ Remedy
  - Happens to all but the shallowest networks
  - Tune learning rate
  - Change network architecture



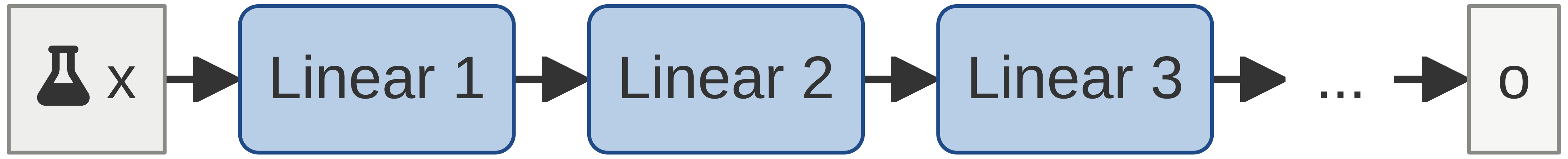
Vanishing and Exploding Gradients - TL;DR

- Vanishing gradients occur in most deep networks
- Exploding gradients lead to NaN; fixed by lower learning rate

Vanishing and Exploding Gradients in PyTorch

Normalizations

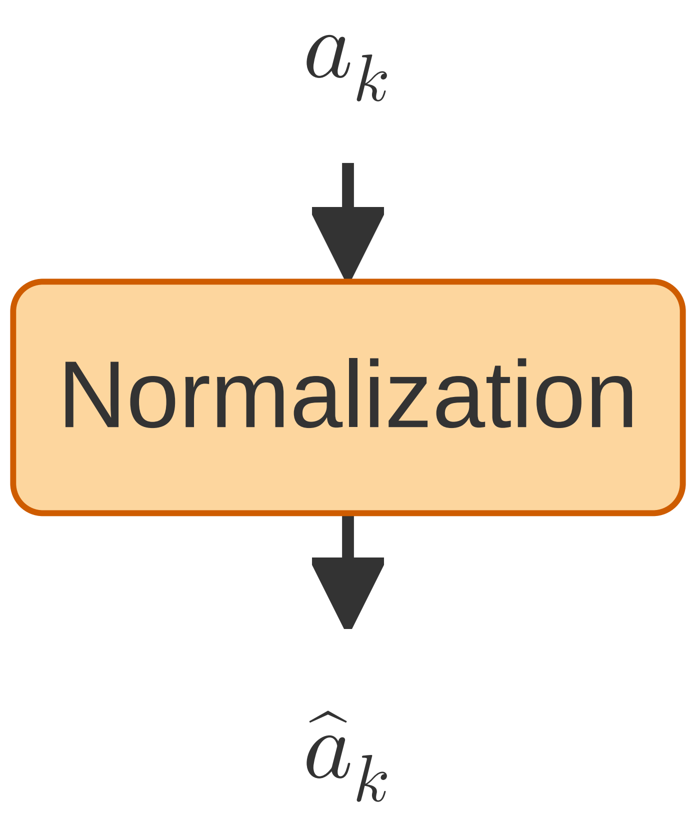
Recap: A Simple Example

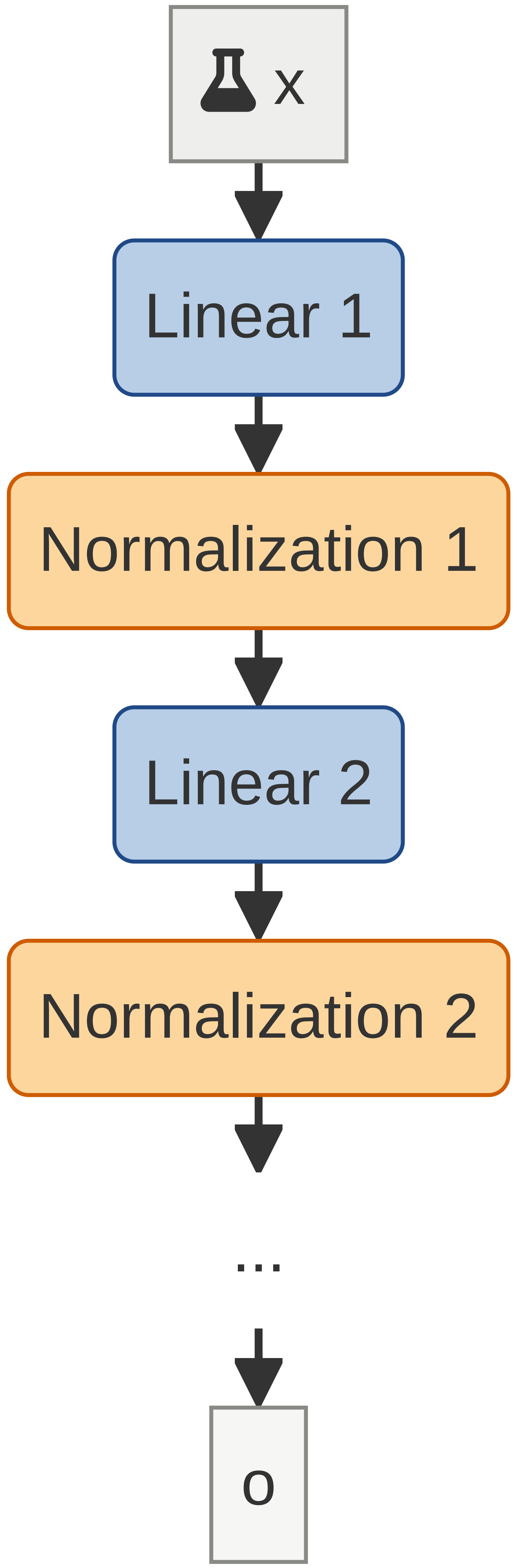


- n-layer linear network
  - No non-linearities
  - Scalar weight $W_i \in \mathbb{R}$
  - Bias $b_i \in \mathbb{R}$
- Major problem: vanishing gradients $\nabla_{w_i} o \to 0$ for large number of layers
- Inconvenience: vanishing activations $$a_n = \underbrace{\prod_{k=1}^{n} W_k}_{\to 0} x + \underbrace{\ldots}_{\text{bias}}$$

Normalization

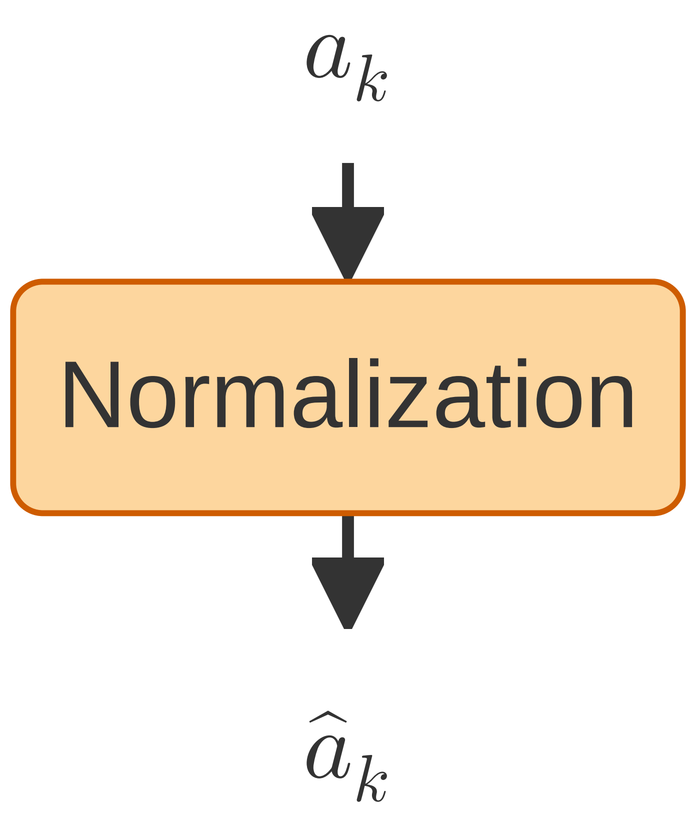
- Rescale and shift activations
  - $\hat{a}_k = \dfrac{a_k - \mu_k}{\sigma_k}$
- Exploding activation $||a_k|| \to \infty$:
  - $\sigma_k \approx ||a_k|| \to \infty$ and $||\hat{a}_k|| \approx 1$
- Vanishing activation $||a_k|| \to 0$:
  - $\sigma_k \approx ||a_k|| \to 0$ and $||\hat{a}_k|| \approx 1$

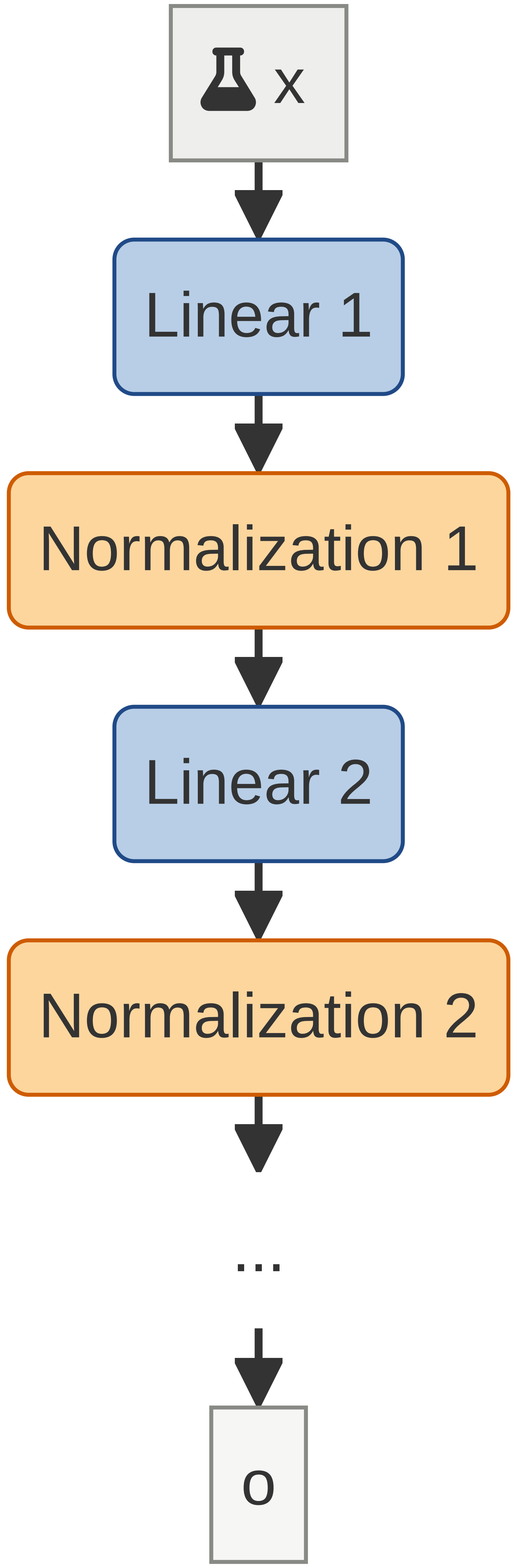




Normalization

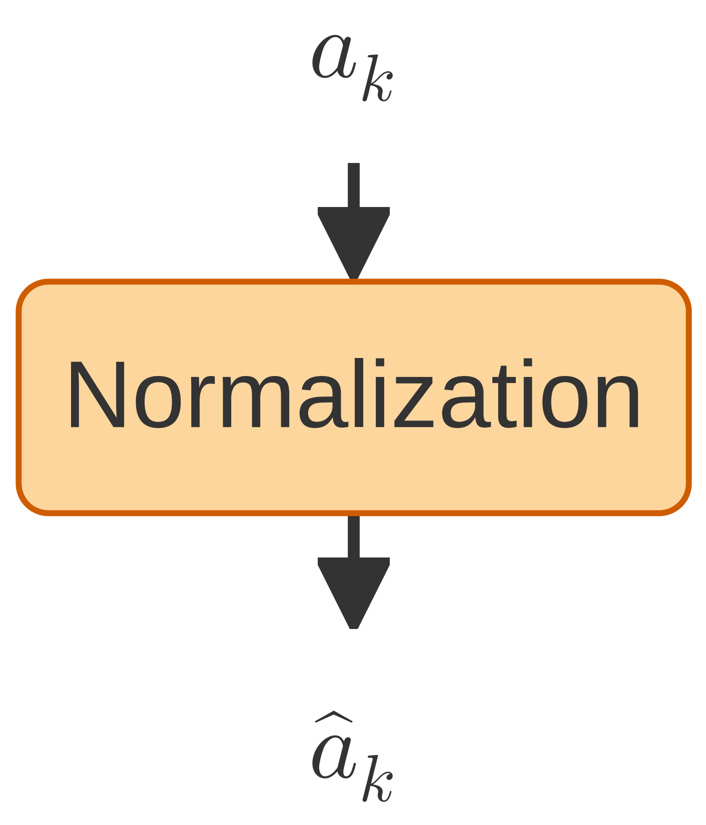
- Rescale and shift activations
- $$\hat{a}_k = \frac{a_k - \mu_k}{\sigma_k}$$
- Where do $\boldsymbol{\mu}_k$ and $\boldsymbol{\sigma}_k$ come from?

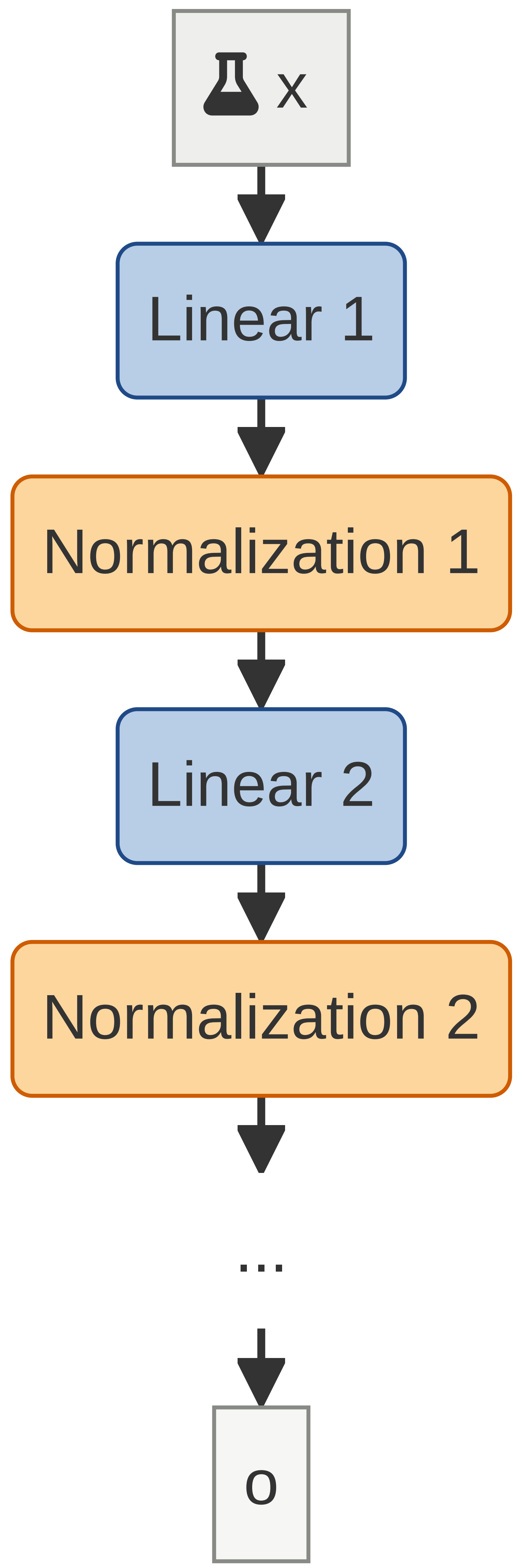




Layer Normalization

- Rescale and shift activations **per feature**
- $\hat{a}_k = \dfrac{a_k - \mu_k}{\sigma_k}$
- Compute μ_k and σ_k across each data element

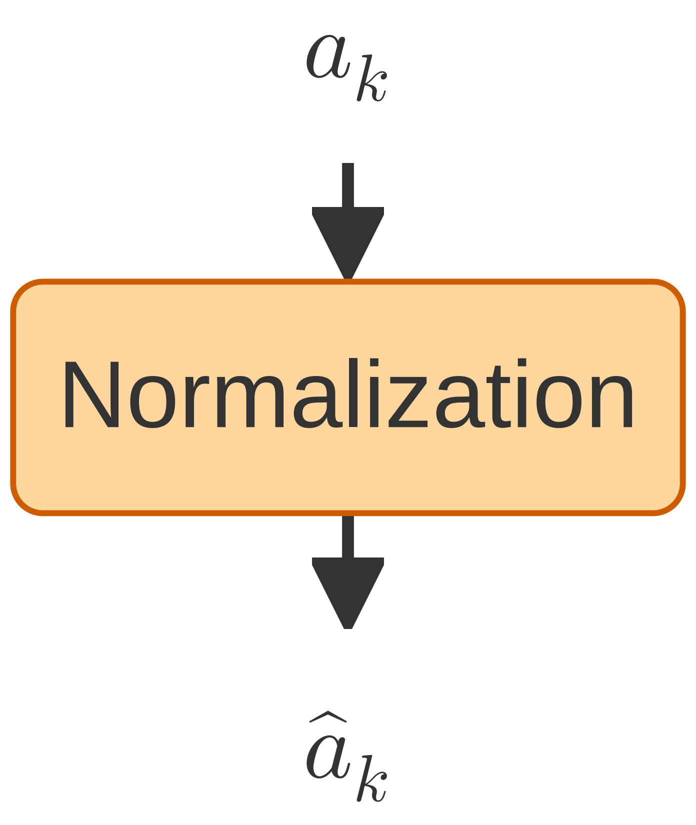


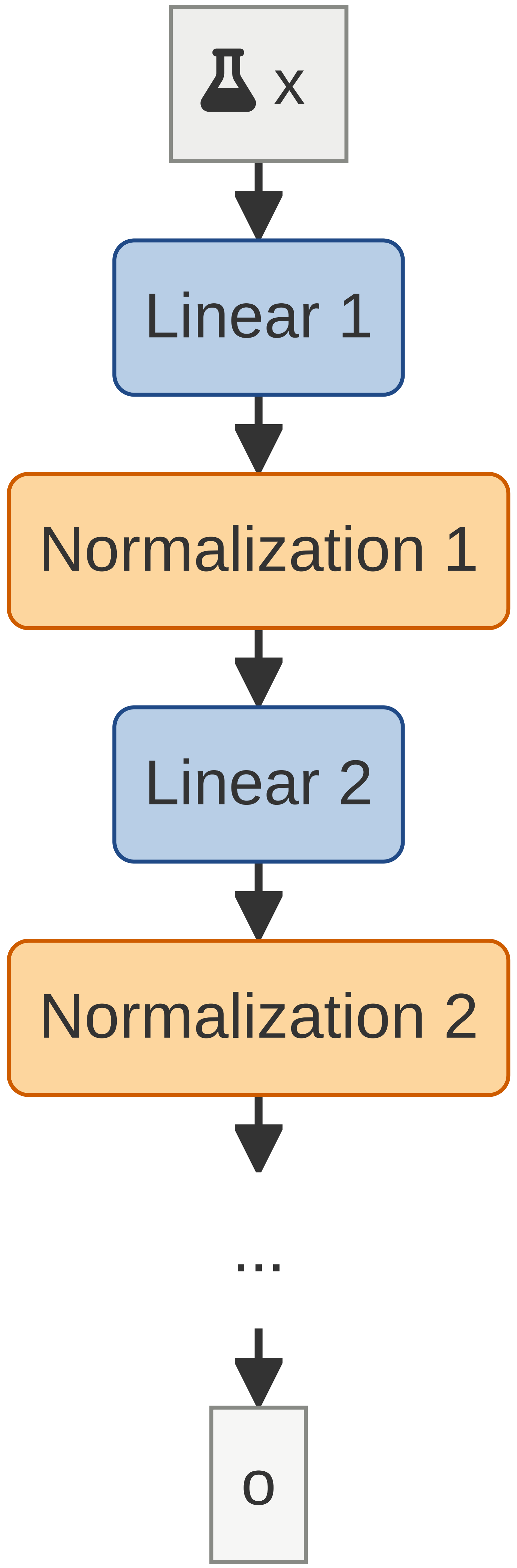


Layer Normalization, Ba, J., Kiros, J. R. and Hinton, G., arXiv preprint arXiv:1607.06450, 2016

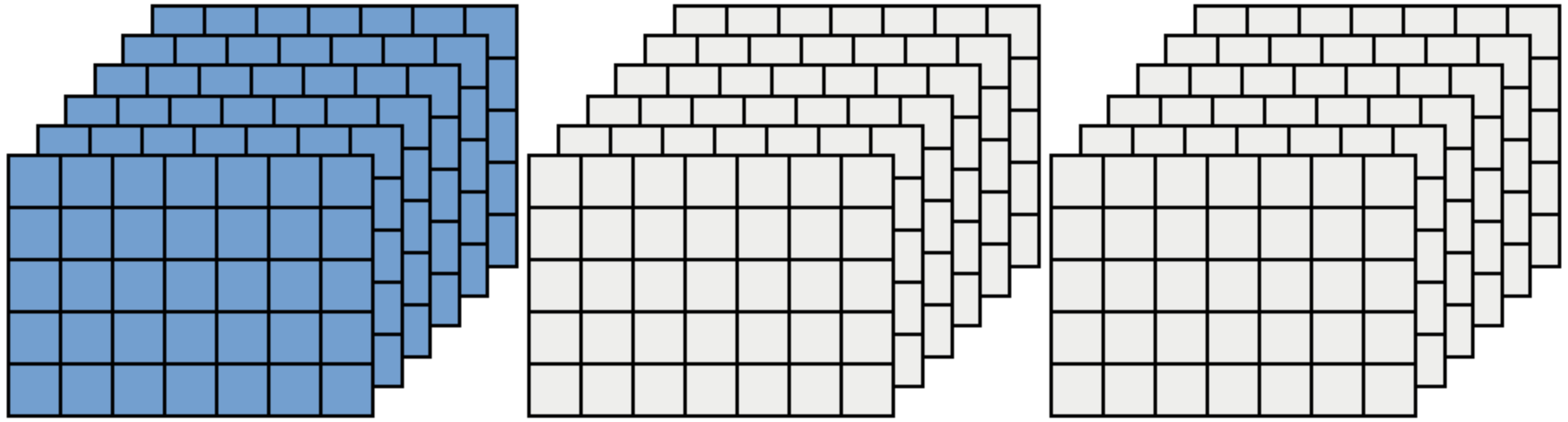
Layer Normalization

- Rescale and shift activations **per feature**
  - mean μ_b
  - stdev σ_b
- $\hat{\mathbf{a}}_{b,c} = \dfrac{\mathbf{a}_{b,c} - \mu_b}{\sigma_b}$
  - $\mu_b = \dfrac{1}{C} \sum_c \mathbf{a}_{b,c}$
  - $\sigma_b^2 = \dfrac{1}{C} \sum_c (\mathbf{a}_{b,c} - \mu_b)^2$



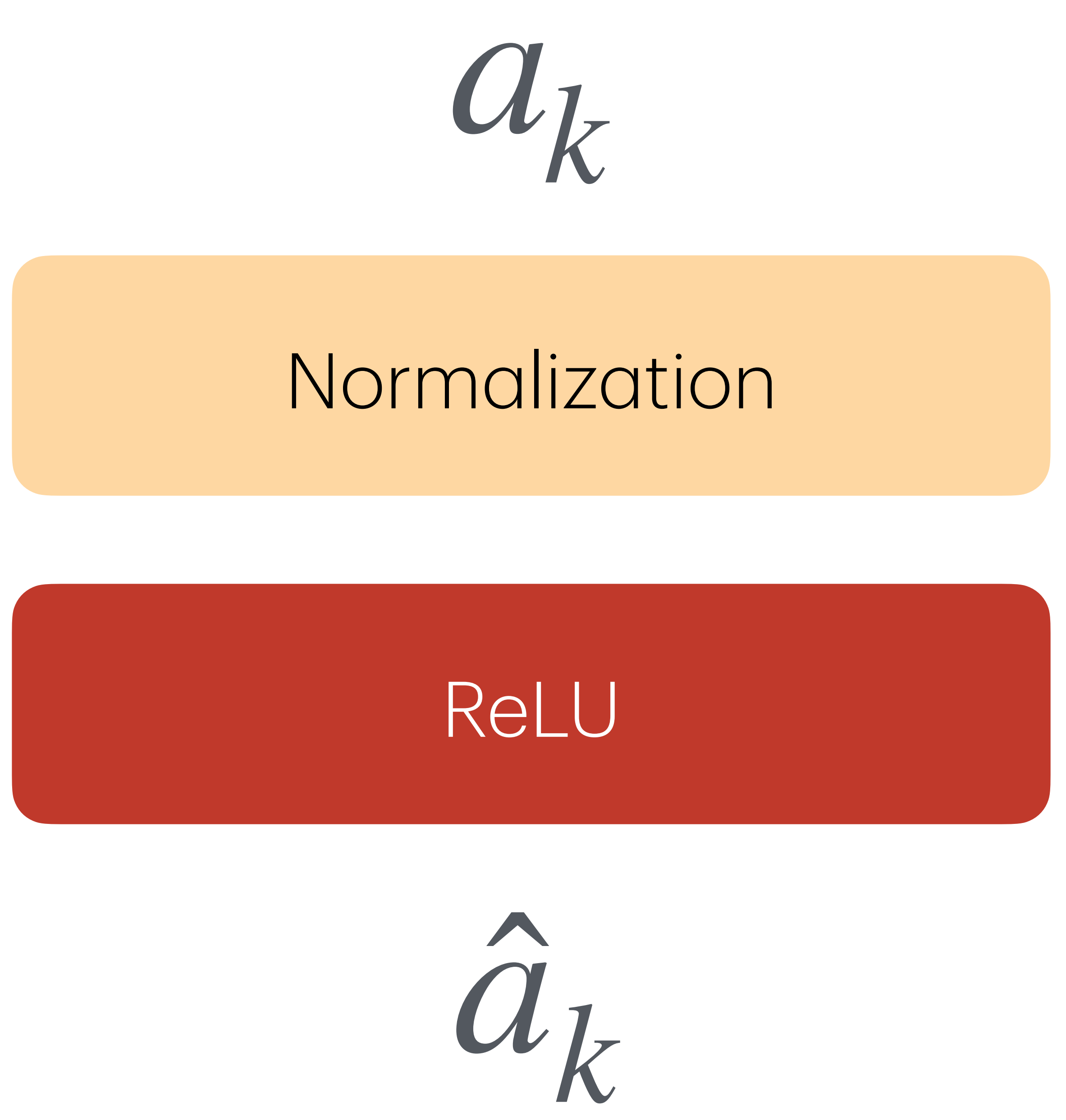


What Does Layer Normalization Do?



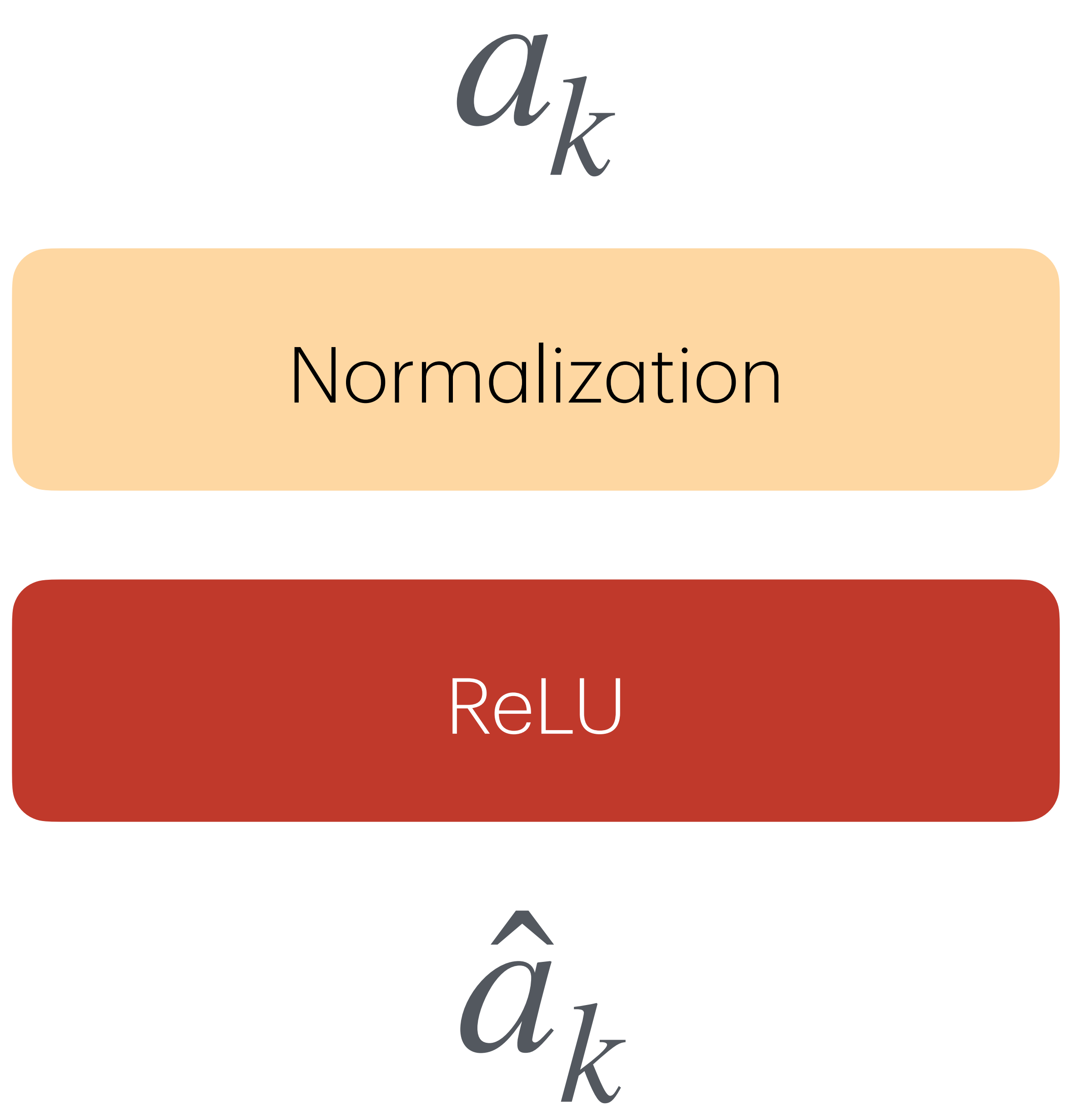
What Does Layer Normalization Do?

- What happens if $a_k = [0,1,2]$?
- $\hat{a}_k =$



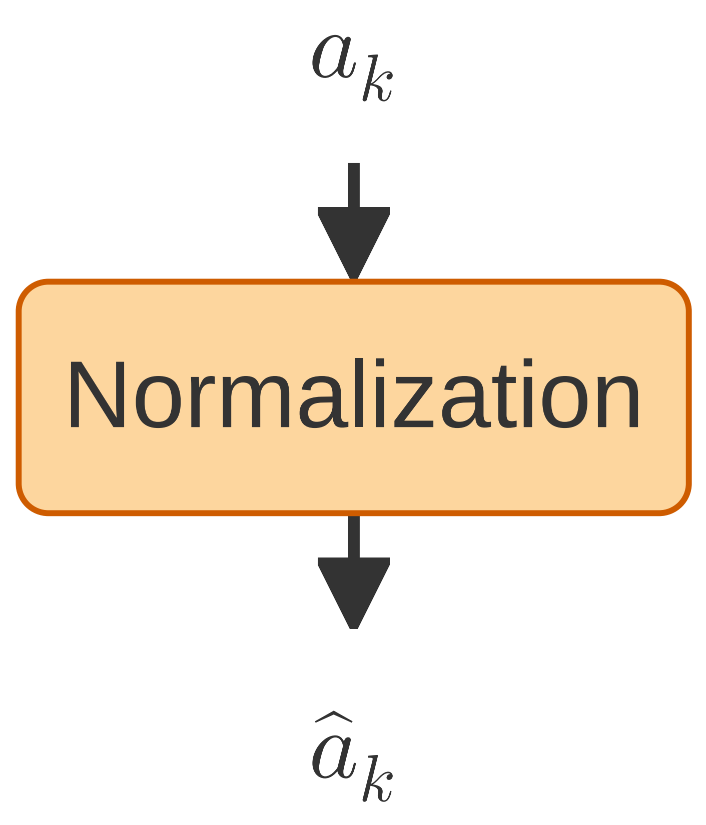
What Does Layer Normalization Do?

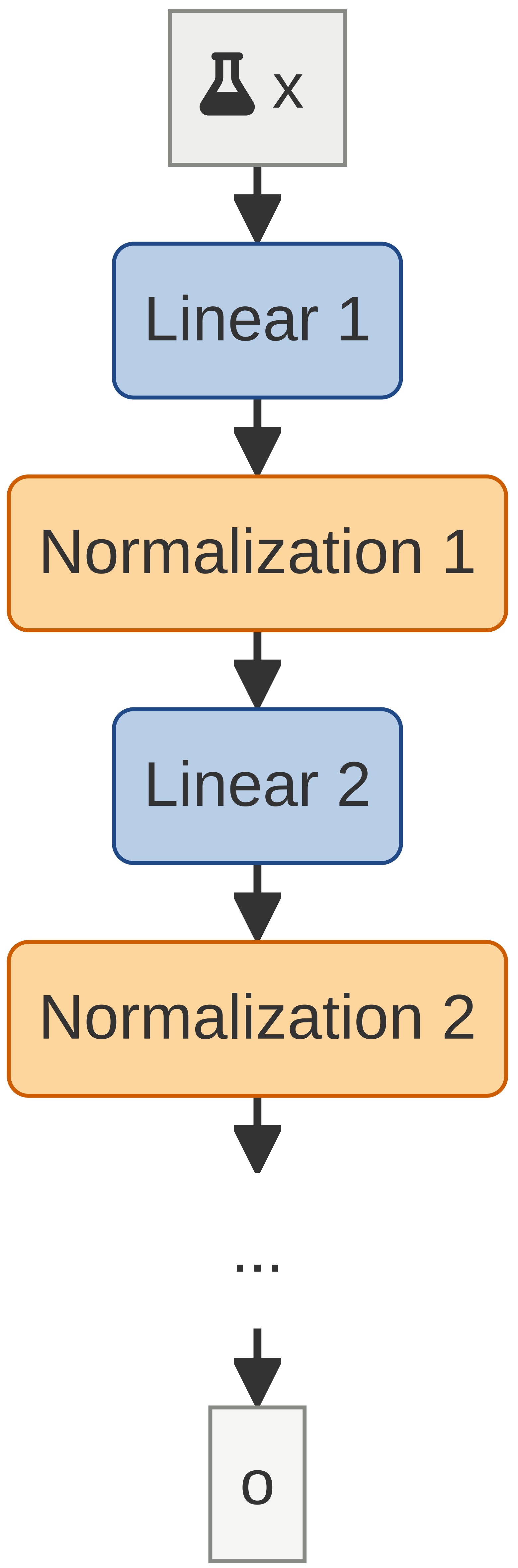
- What happens if $a_k = [0,1,2]$?
- $\hat{a}_k = [0,0,1]$
- About 50% of activations are zero
    - Due to ReLU



Layer Normalization

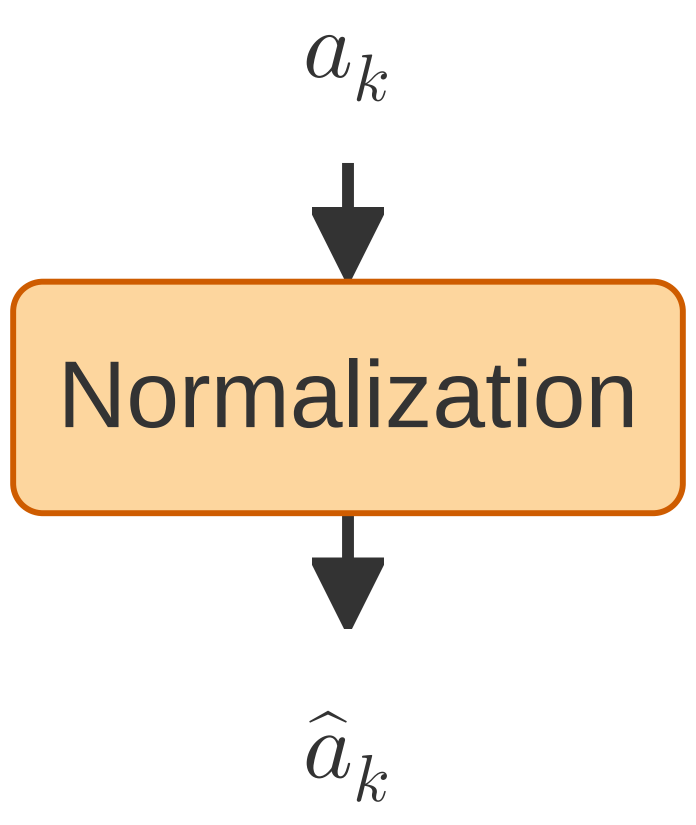
- Rescale and shift activations **per feature**
- $\hat{\mathbf{a}}_{b,c} = \dfrac{\mathbf{a}_{b,c} - \mu_b}{\sigma_b}\gamma_c + \beta_c$
  - $\mu_b = \dfrac{1}{C}\sum_c \mathbf{a}_{b,c}$
  - $\sigma_b^2 = \dfrac{1}{C}\sum_c (\mathbf{a}_{b,c} - \mu_b)^2$

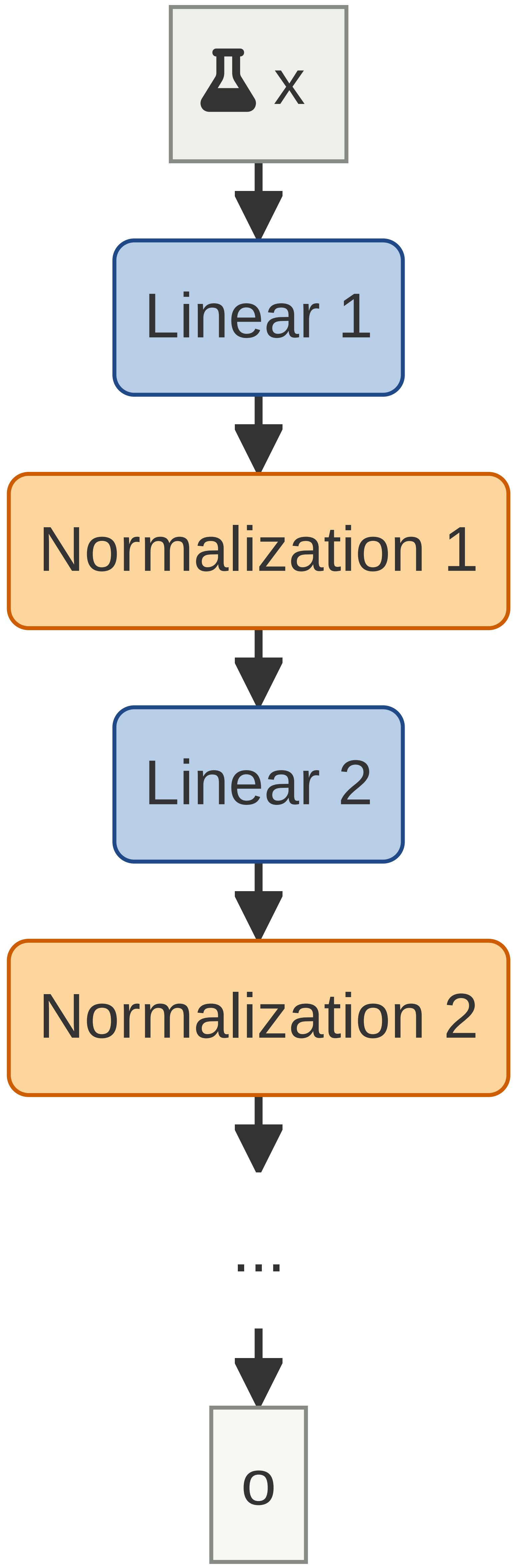




RMSNorm

- Rescale and shift activations **per feature**
- $\hat{\mathbf{a}}_{b,c} = \frac{\mathbf{a}_{b,c}}{\sigma_b}\gamma_c$
  - $\sigma_b^2 = \frac{1}{C}\sum_c (\mathbf{a}_{b,c})^2$

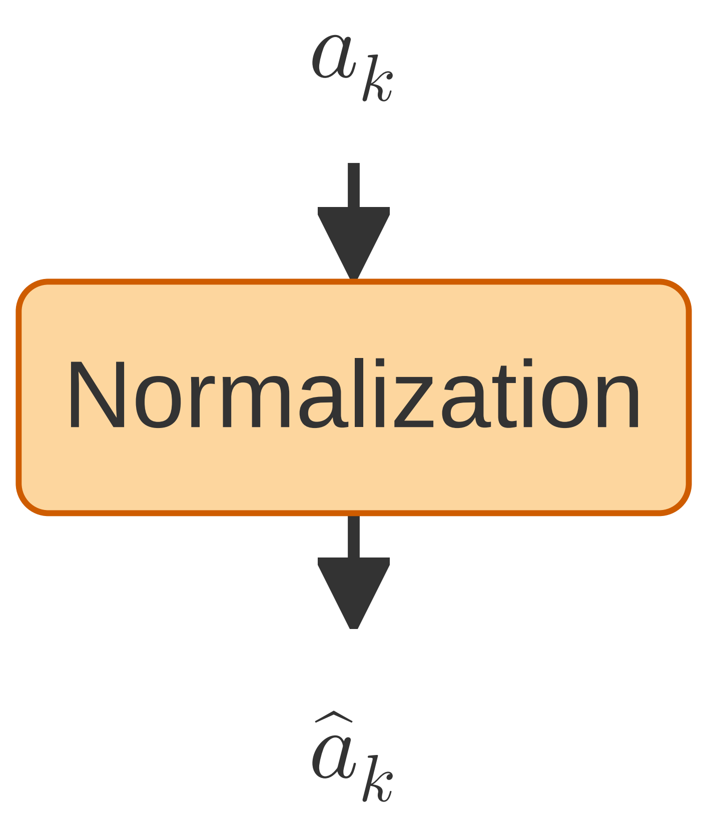


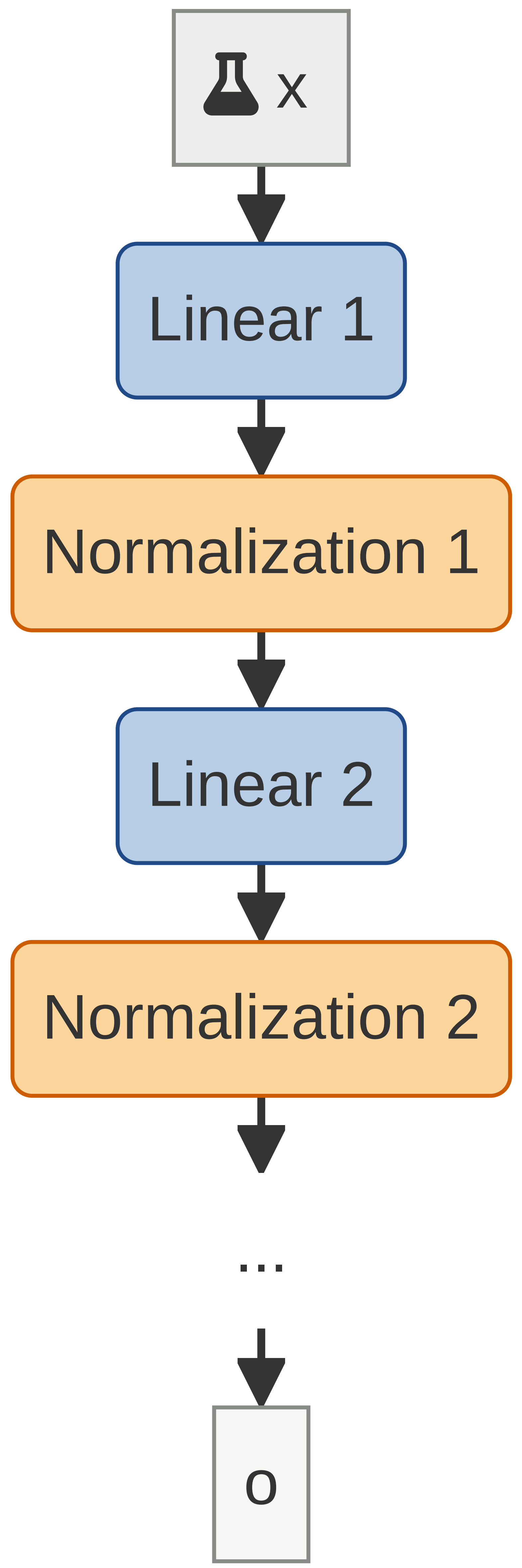


Root Mean Square Layer Normalization, Zhang and Sennrich., arXiv preprint arXiv:1910.07467, 2019

Zero-mean RMSNorm

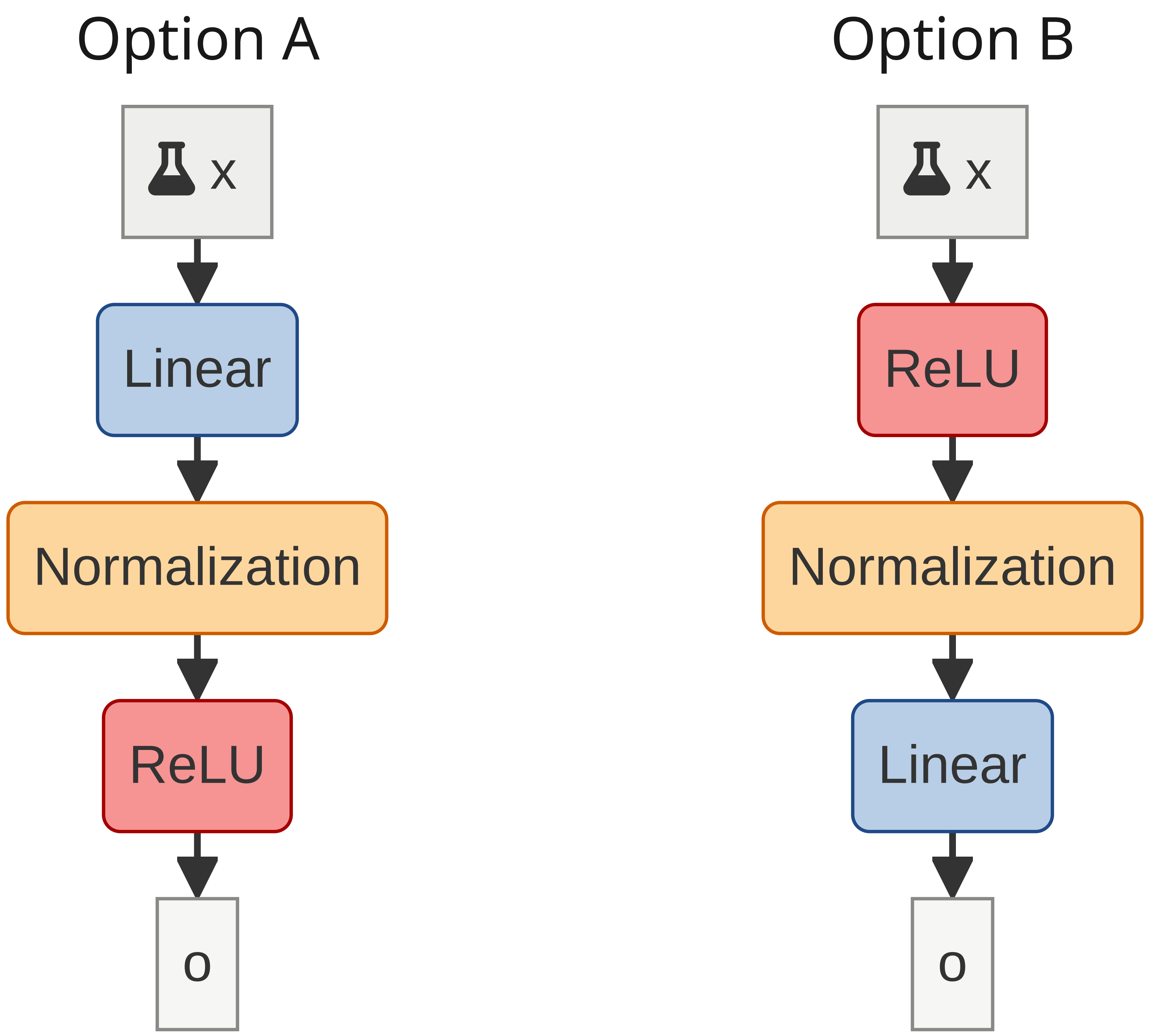
- Rescale and shift activations **per feature**
- $\hat{\mathbf{a}}_{b,c} = \dfrac{\mathbf{a}_{b,c} - \mu_b}{\sigma_b}(1 + \gamma_c)$
  - Regularizer on $\gamma_c \to 0$
  - $\mu_b = \dfrac{1}{C}\sum_c \mathbf{a}_{b,c}$
  - $\sigma_b^2 = \dfrac{1}{C}\sum_c (\mathbf{a}_{b,c} - \mu_b)^2$





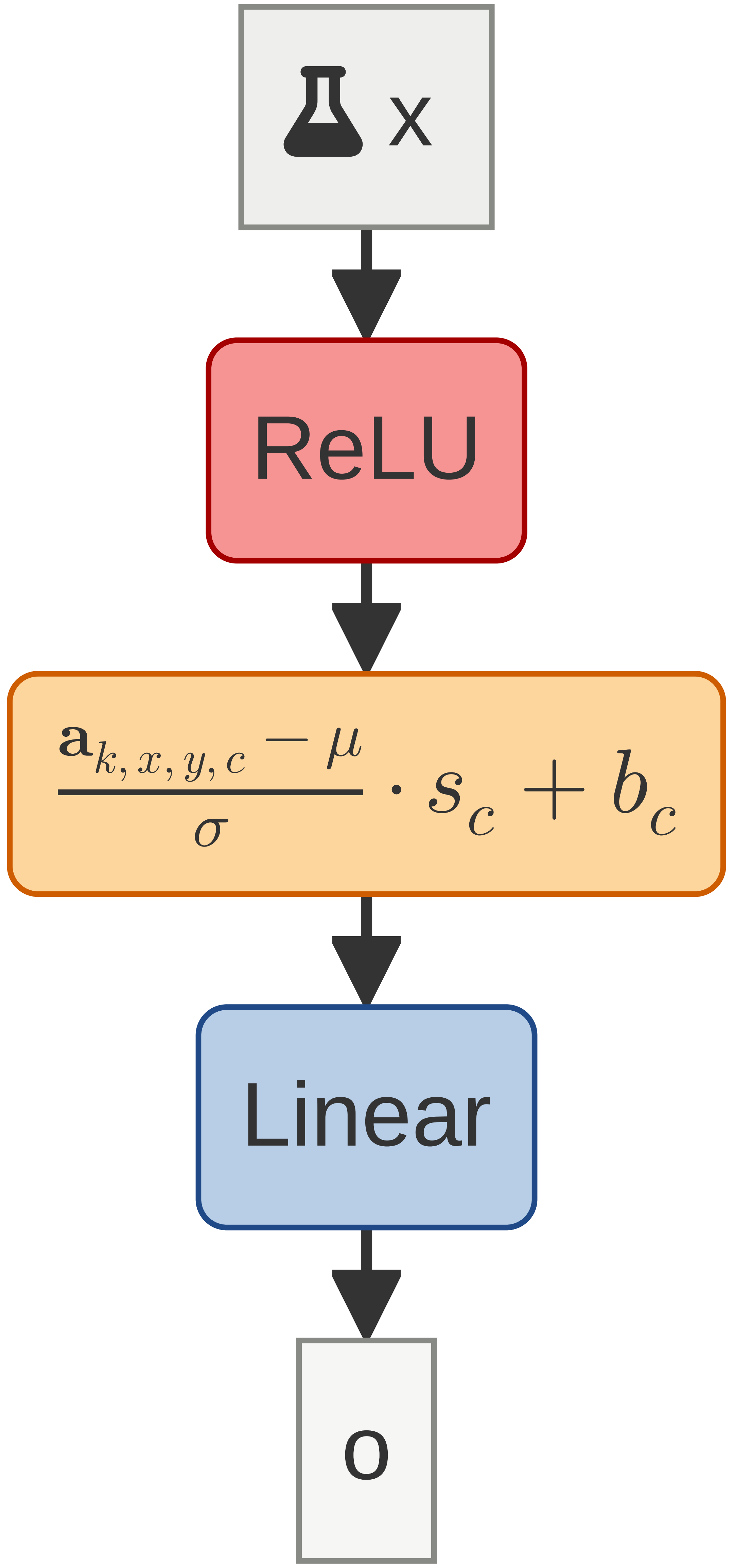
Where to Add Normalization?

- Option A: After Linear, Before Activation
- Option B: After Activation, Before next Linear



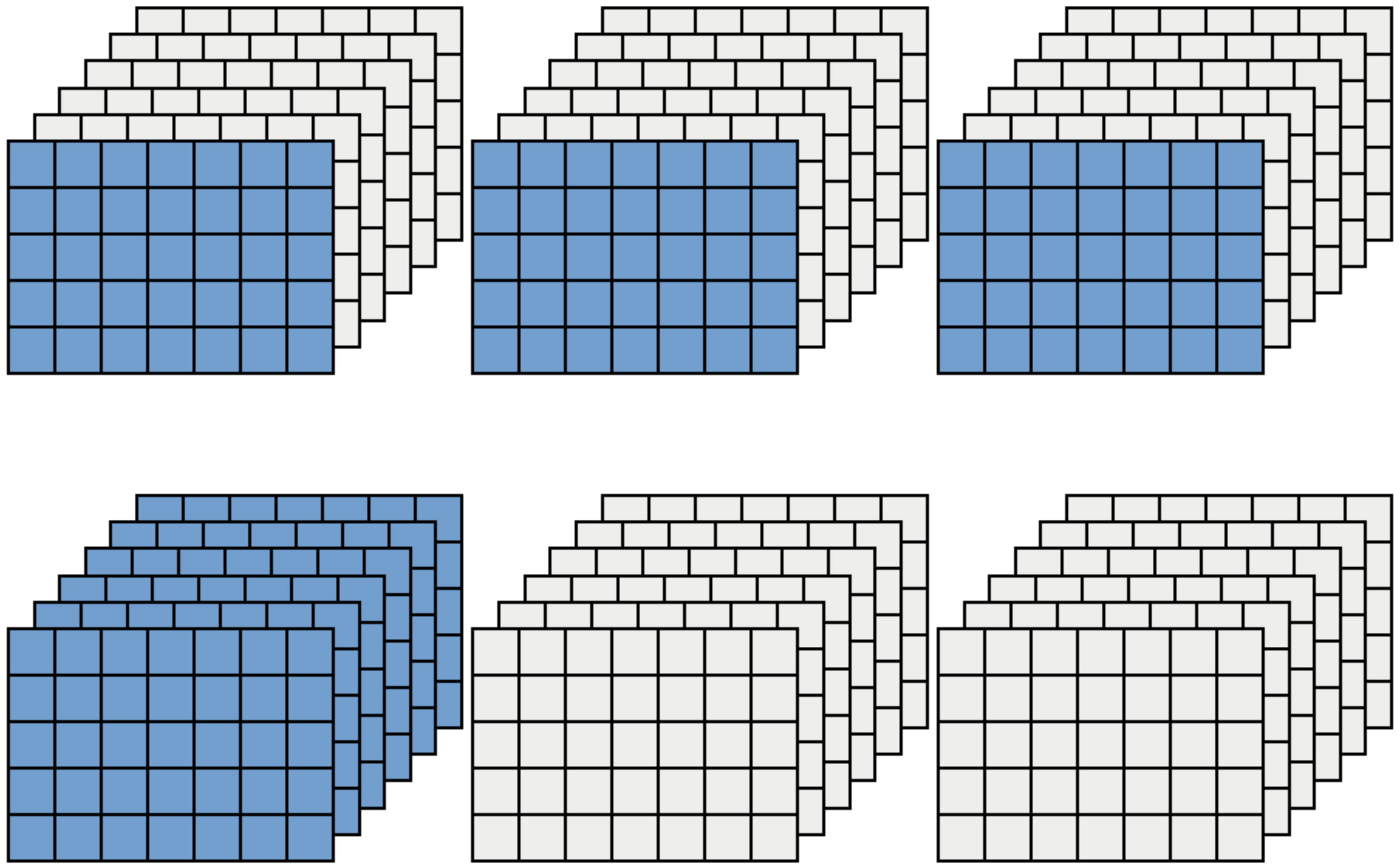
Post-Activation Normalization

- Easier
  - Works better with SwiGLU, etc



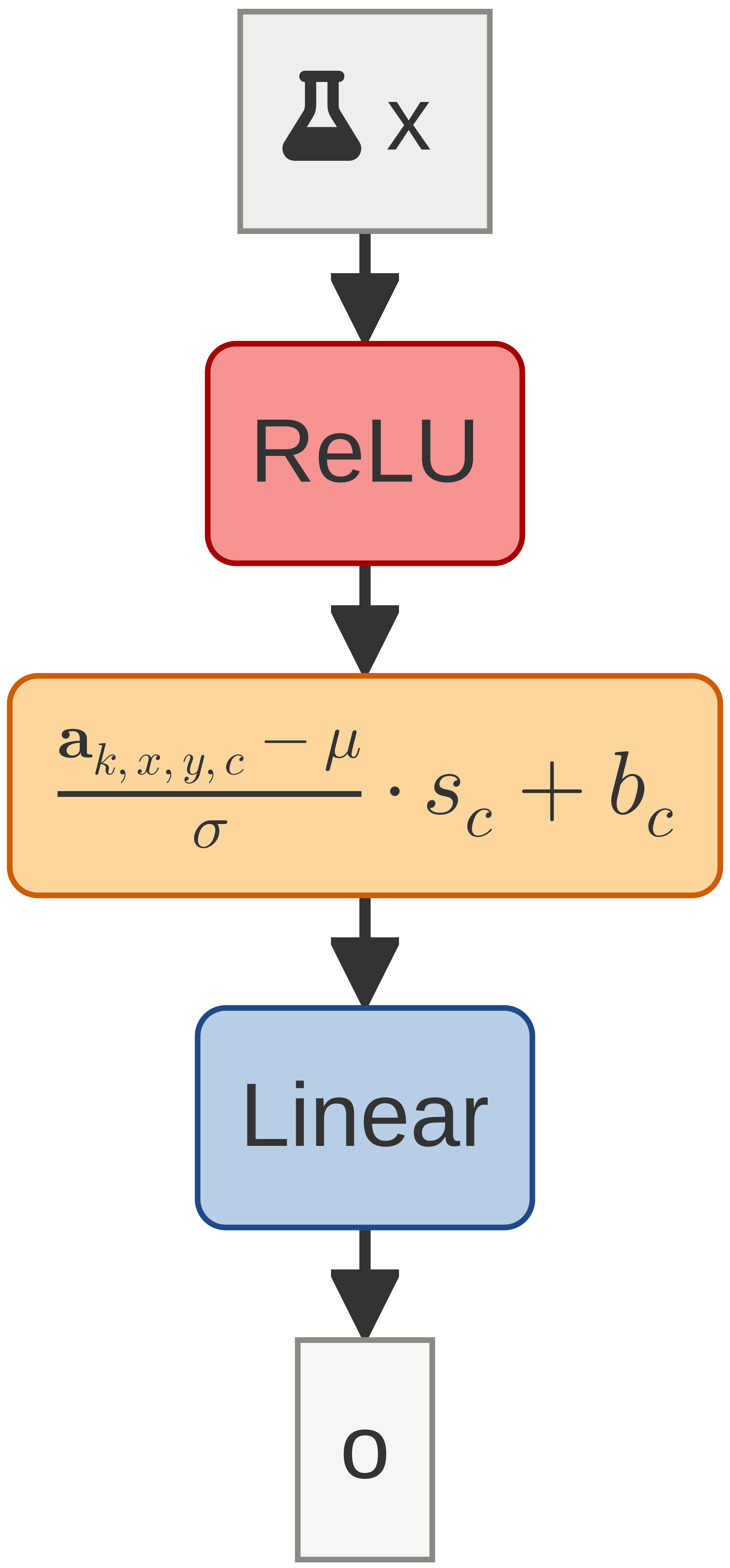
What normalization to use?

- Try LayerNorm
  - Should always work for this class
- Try RMSNorm
- Try zero-mean RMSNorm
  - Only matters for LARGE models



Why does normalization work?

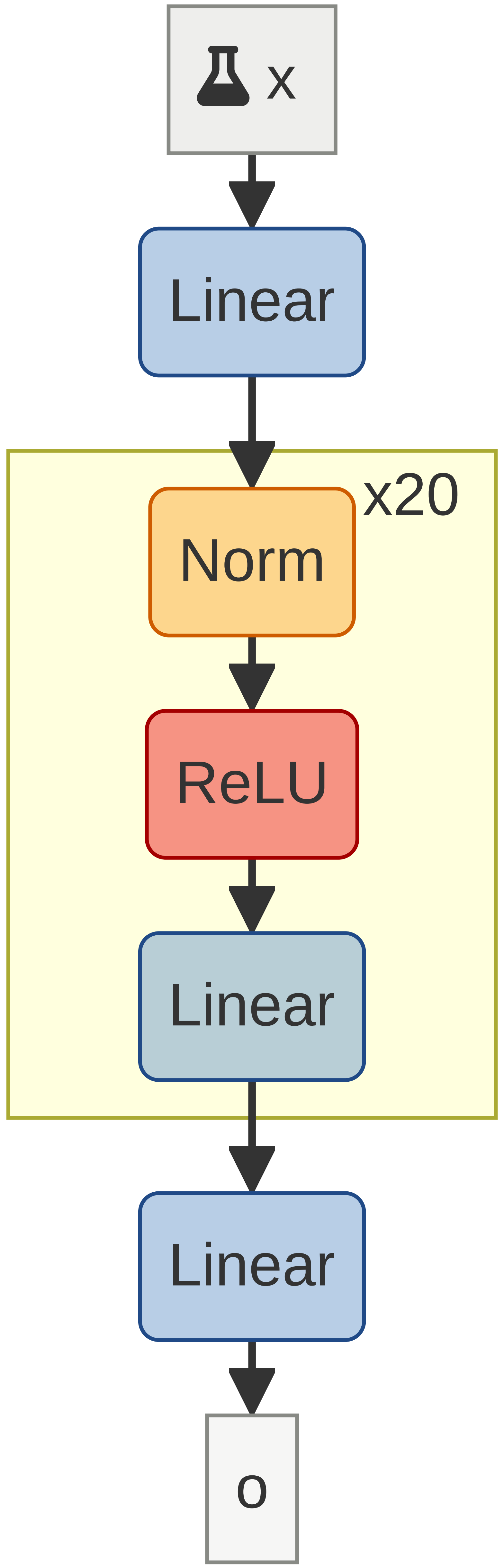
- Normalization fixes vanishing activations
    - Handle badly scaled weights
    - Activations cannot vanish (assuming $b_c = 0$)
    - "Eigenvalues" are close to 1
- The same holds true for gradients [1]



[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 layers



Normalization - TL;DR

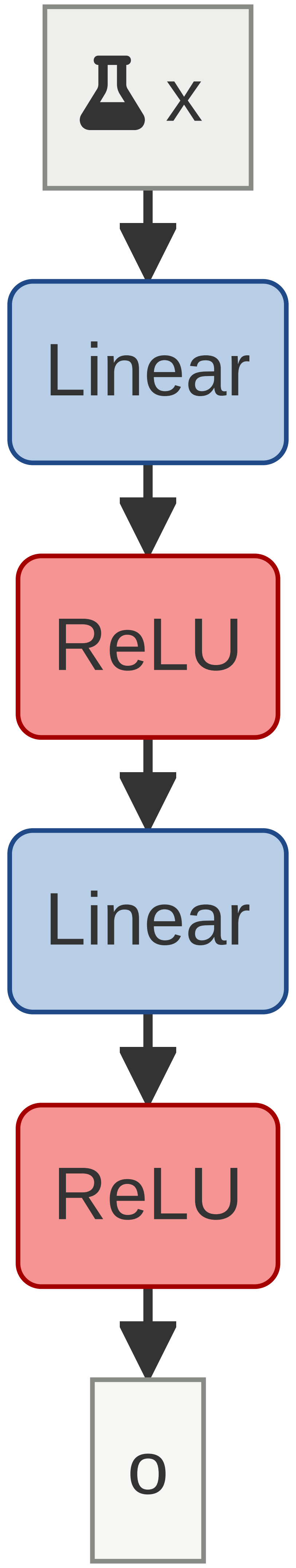
- Normalizations handle vanishing gradients

Normalizations in PyTorch

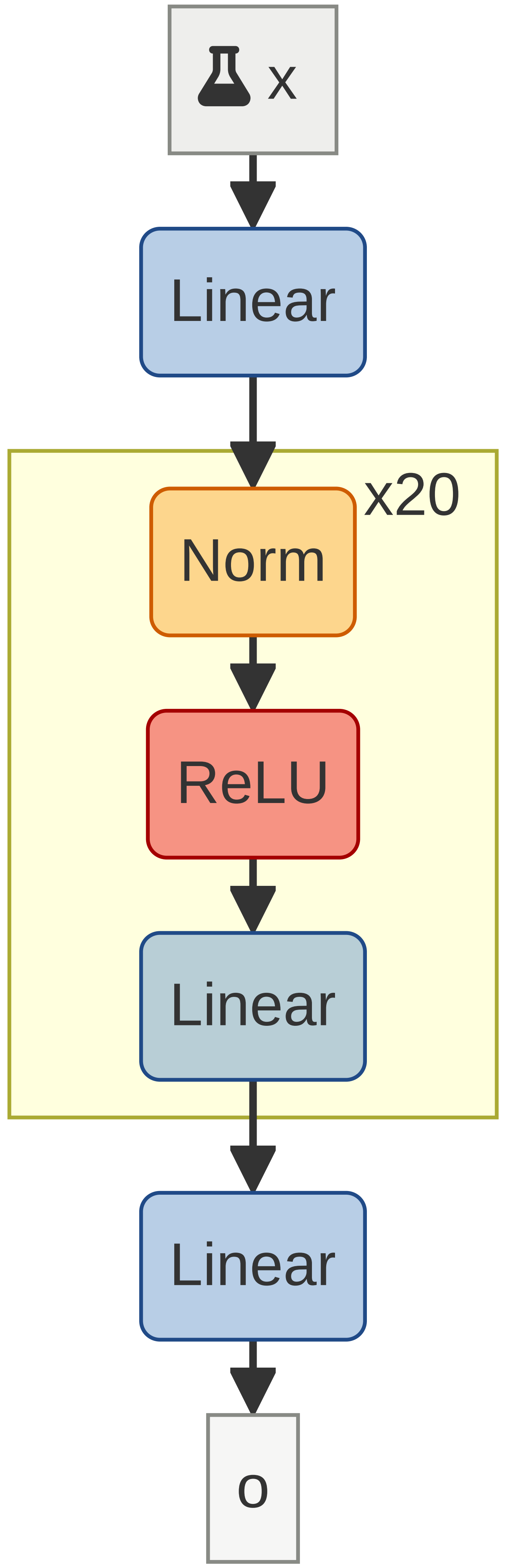
Residual Connections

Recap: Deep Networks

- Without normalization
  - Max depth: 10-12 layers



- With normalization
  - Max depth: 20-30 layers

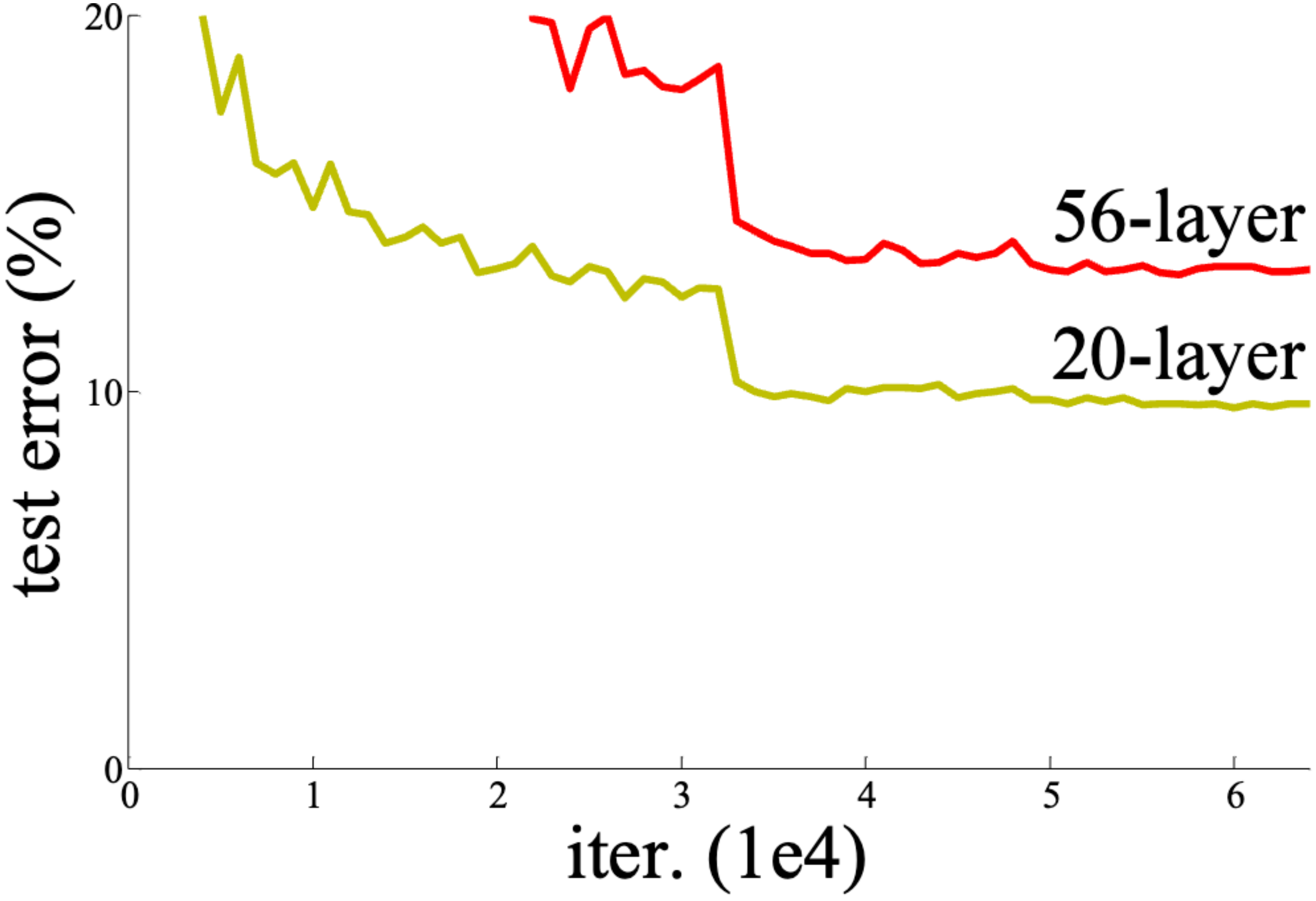


What Happens to Deeper Networks?

- See PyTorch

What Happens to Deeper Networks?

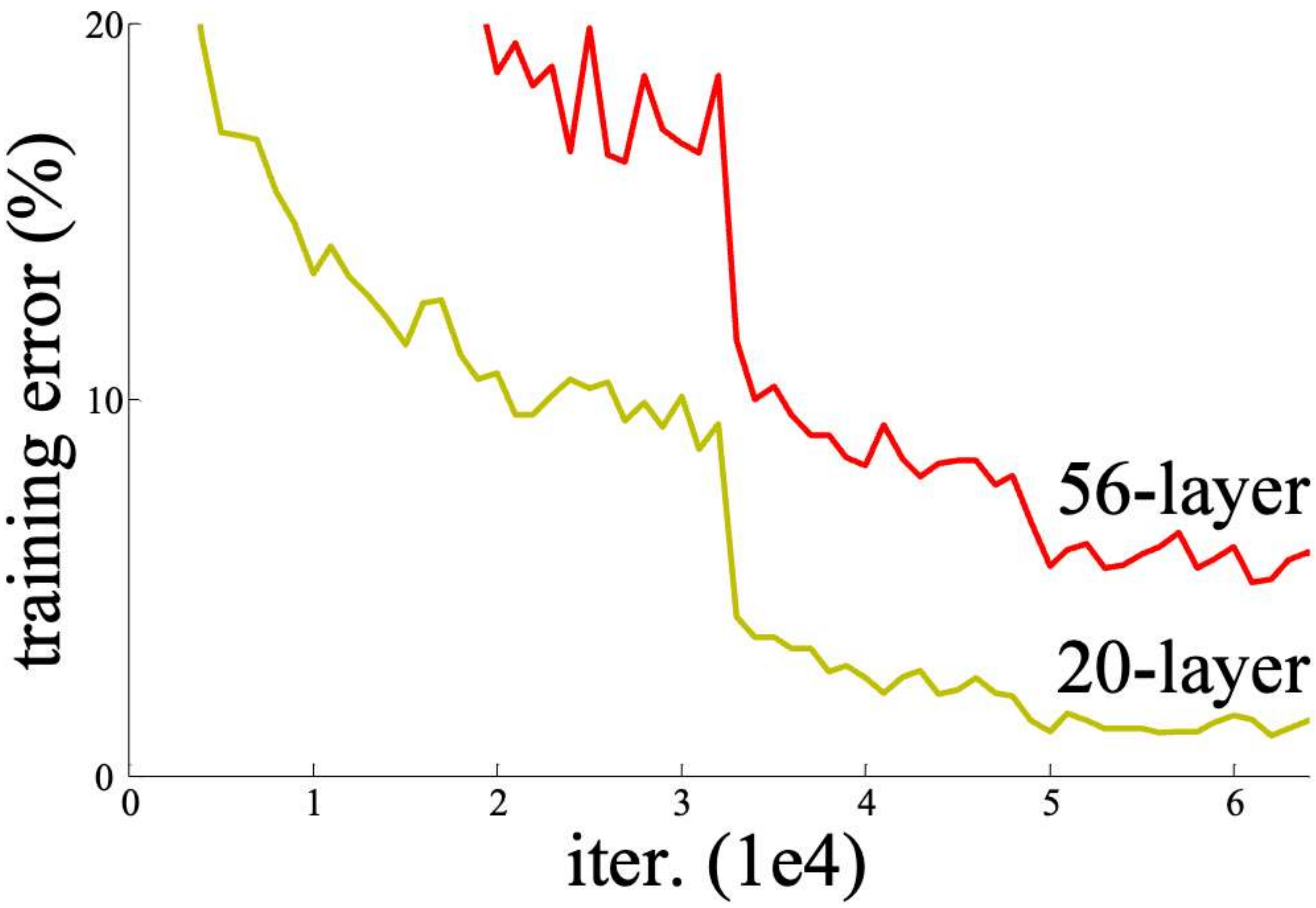
- Thy don't **perform** well!



Kaiming He et al., "Deep Residual Learning for Image Recognition", CVPR 2016

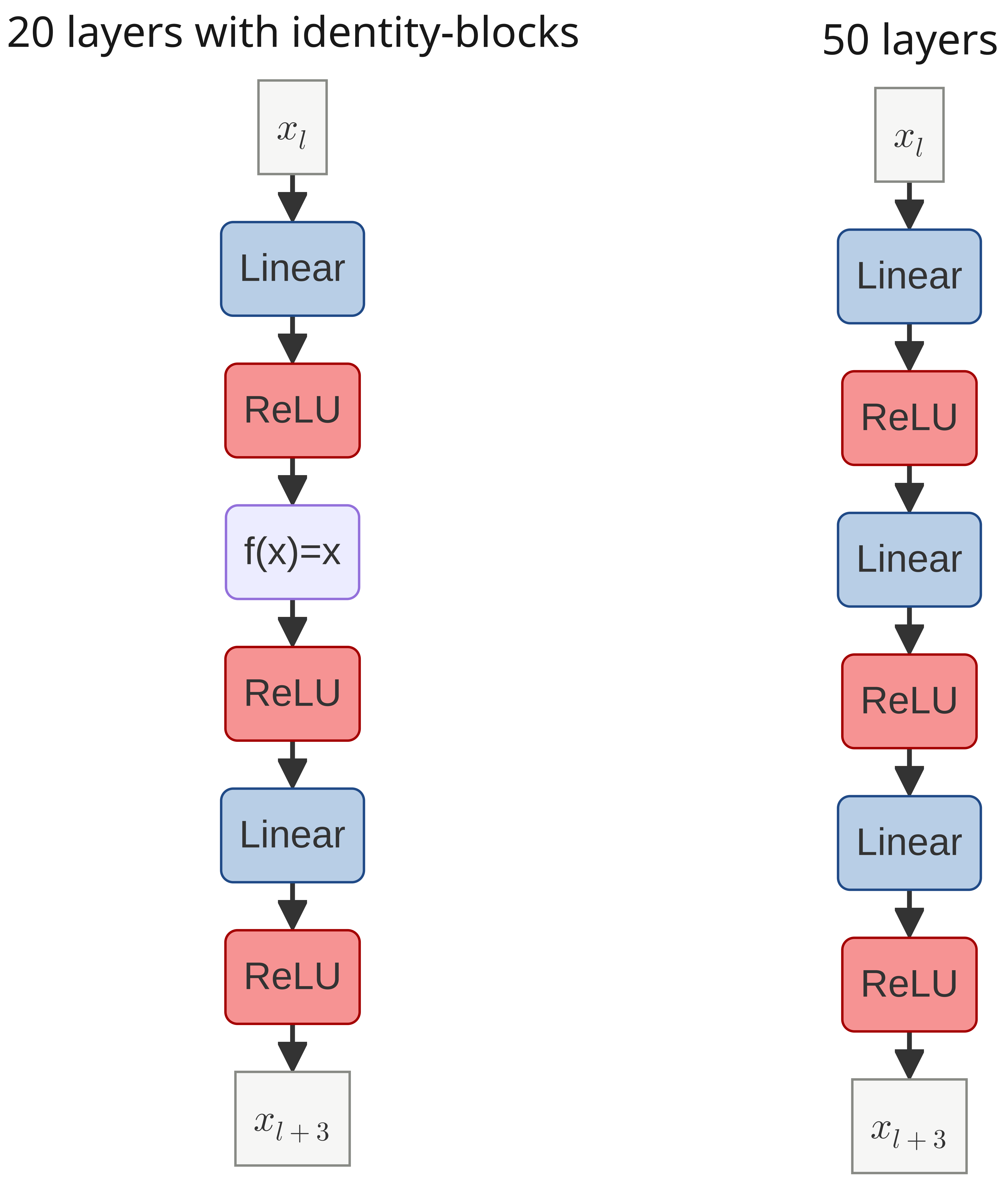
What Happens to Deeper Networks?

- Thy don’t **train** well!



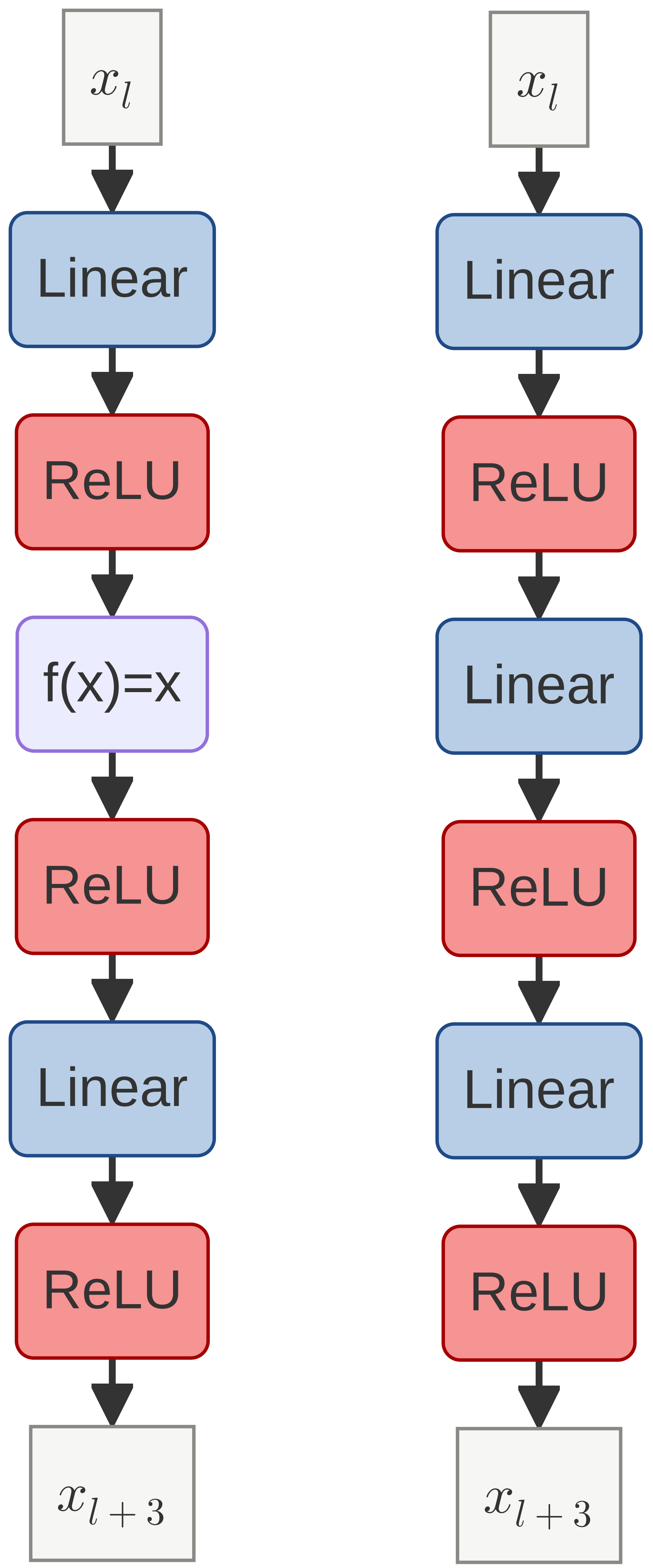
Kaiming He et al., "Deep Residual Learning for Image Recognition", CVPR 2016

20 vs 50 Layers Networks



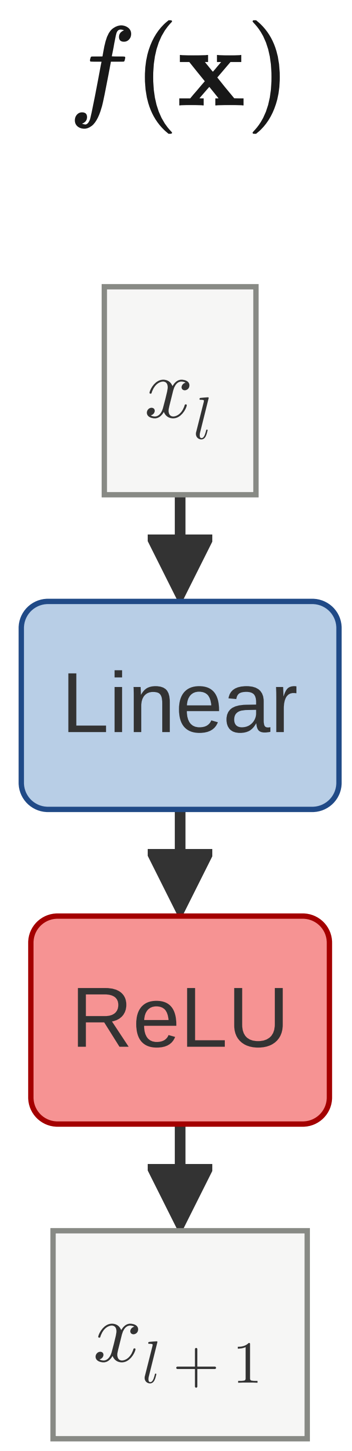
Why don't they train well?

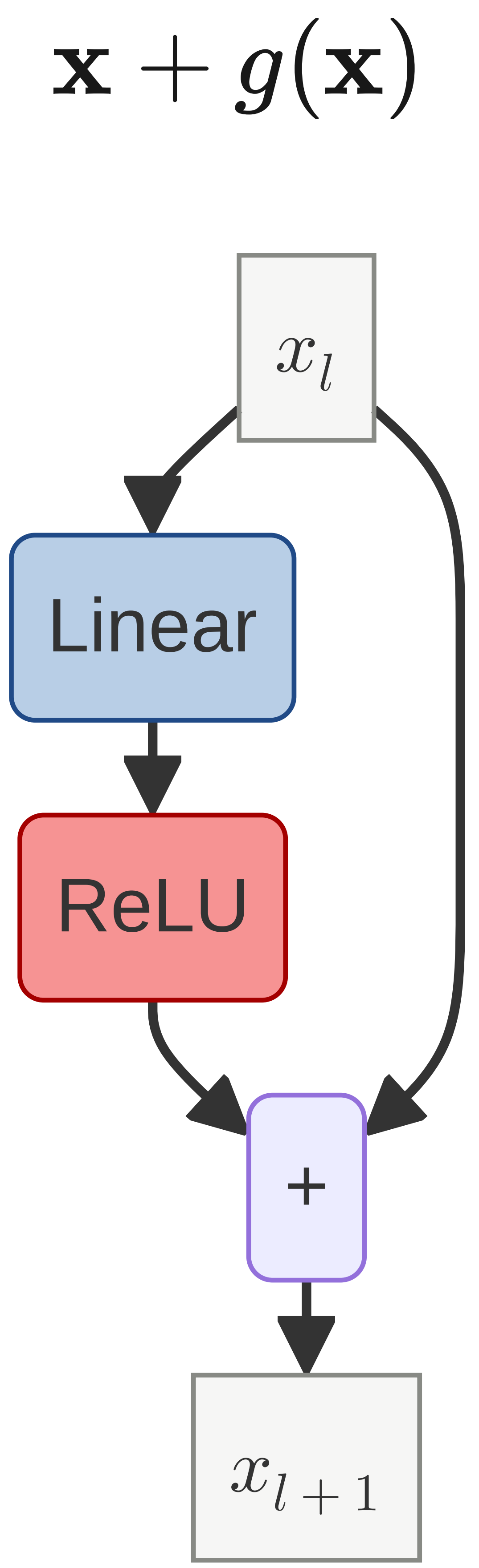
- Initial updates are hard
  - Initial weights: Random Gaussian
  - After a few layers:
    - Inputs look Gaussian (random noise)
    - Gradients look Gaussian (random noise



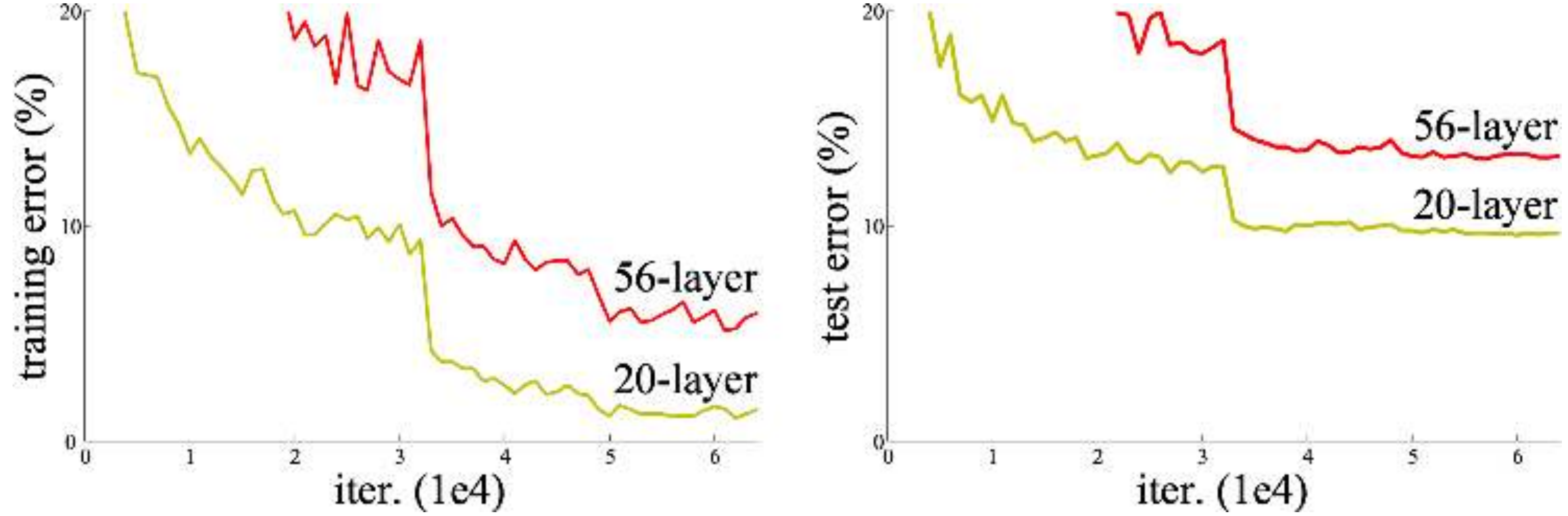
Solution: Residual Connections

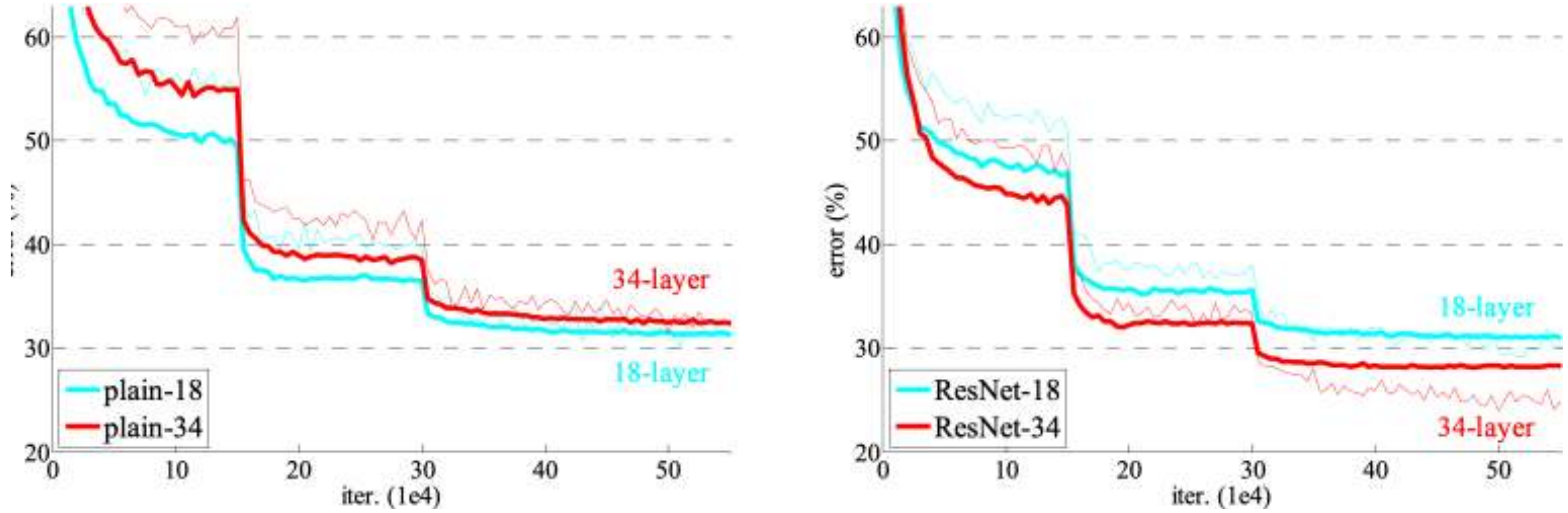
- Parametrize layers as
  - $f(x) = x + g(x)$
  - and learn $g(x)$





Residual Networks

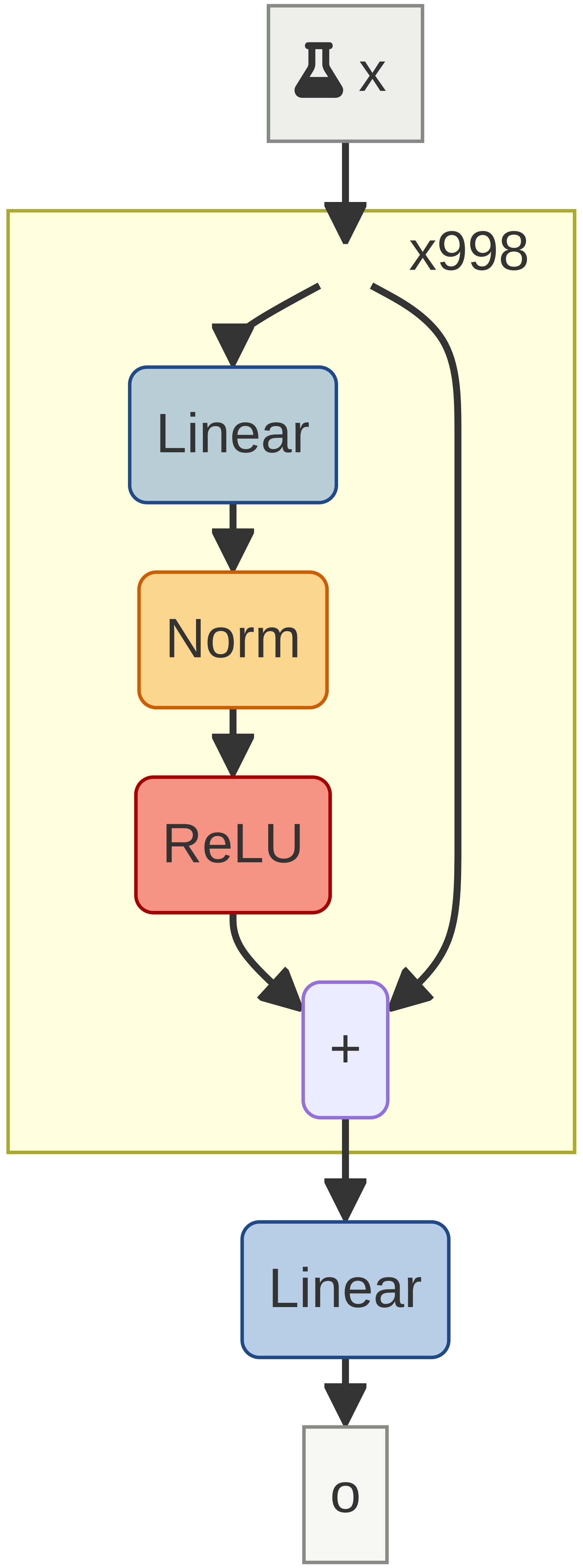




Kaiming He et al., "Deep Residual Learning for Image Recognition", CVPR 2016

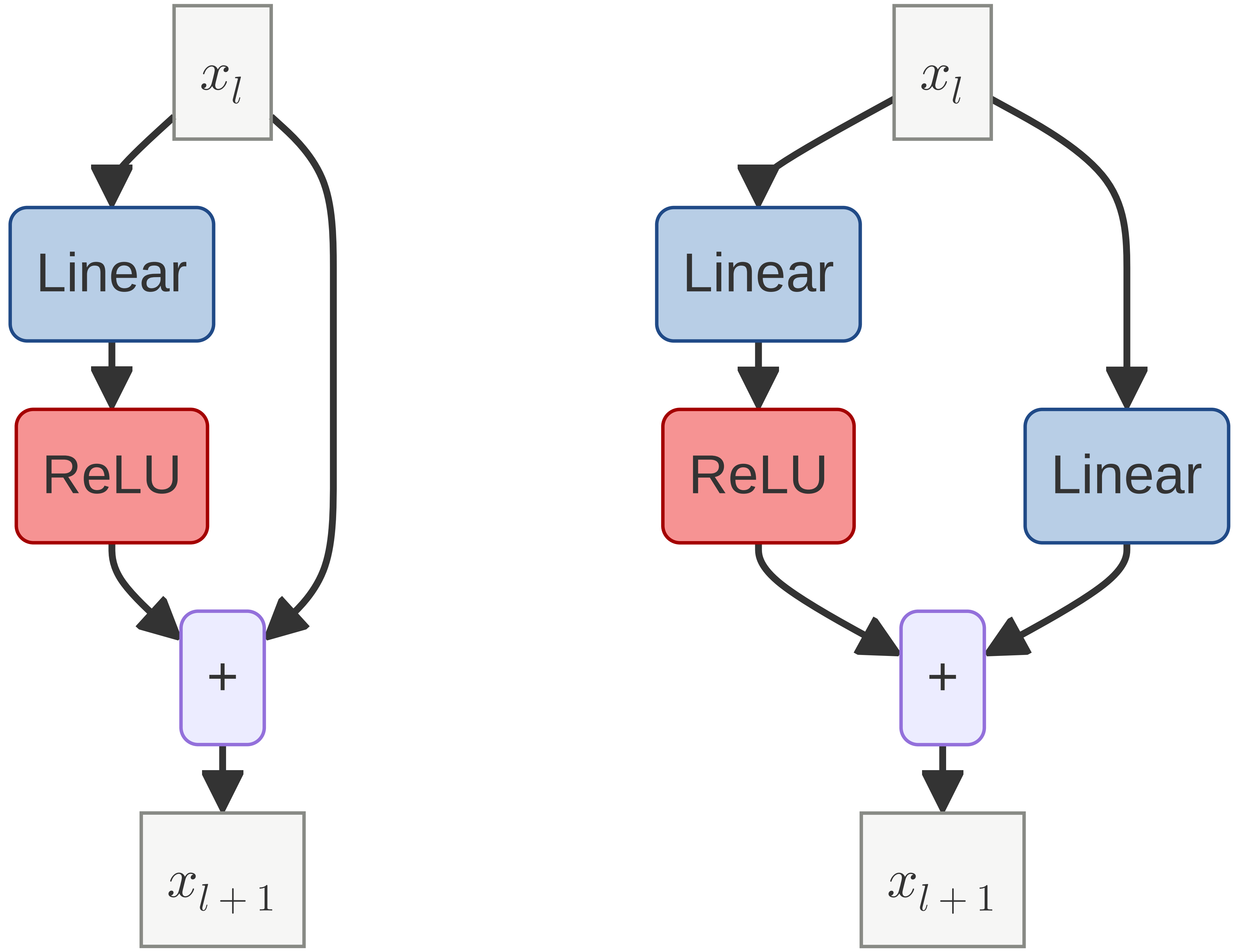
How Well Do Residual Connections Work?

- Can train networks of **up to 1000 layers**



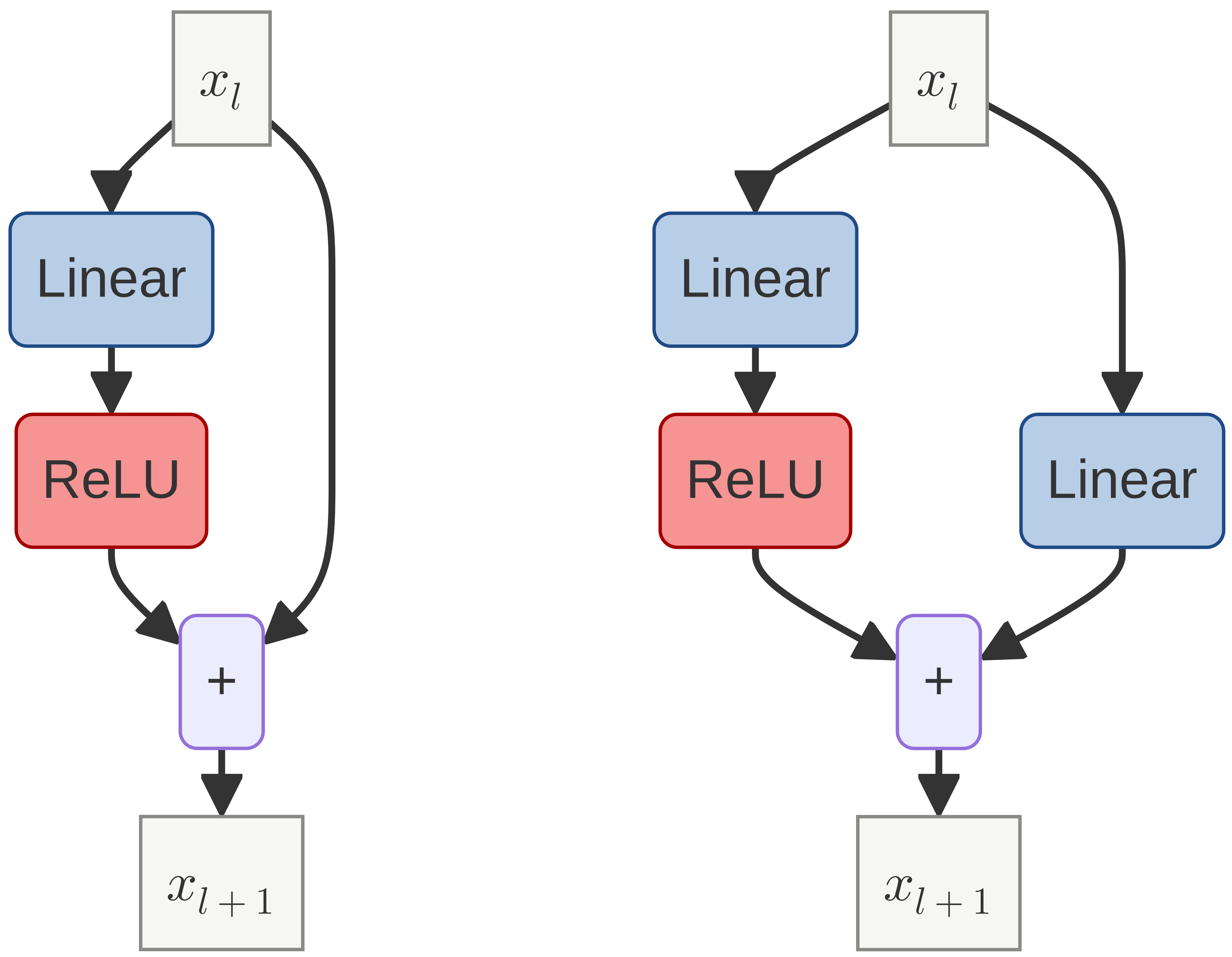
What if Input and Output Are Not the Same Size?

- Add a linear layer to reshape
  - Minimize number of dimension changes
  - For most layers, choose `input_shape = output_shape`
- Less relevant for transformers



Why Do Residual Connection Work? - Practical Answer

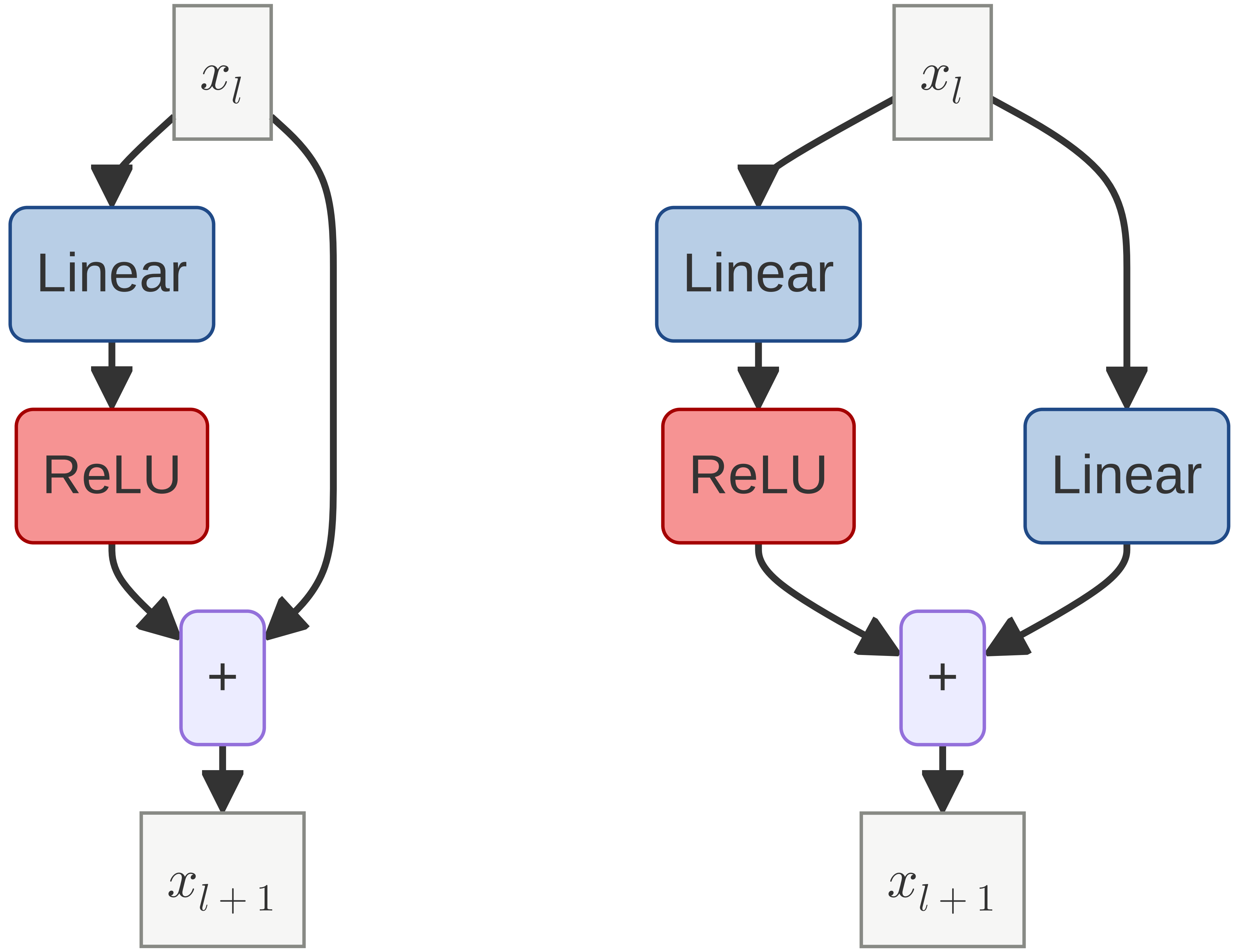
- Gradient Travels Further
  - Another way to prevent vanishing gradients
- Reuse of features / patterns
  - Only update patterns
- Can even drop some layers [1]
  - Dropping some layers still does well



[1] Gao Huang et al., "Deep Networks with Stochastic Depth", ECCV 2016

Why Do Residual Connection Work? - Theoretical Answer

- Optimization [1,2]
  - Invertibility
  - Model capacity: very wide
  - Simplified "loss landscape" for SGD



[1] Simon S. Du, et al., "Gradient Descent Finds Global Minima of Deep Neural Networks", ICML 2019
[2] Moritz Hardt and Tengyu Ma, “Identity matters in deep learning”, ICLR 2017

Residual TL;DR

- Go deeper with residual connections
- Residuals + Normalization fixes vanishing activations and gradients

Residual Connections in PyTorch

Building the deepest network ever built in PyTorch

Residuals and Normalizations - Summary

Let's train really deep networks

- Vanishing gradients and activations are normal
  - Better than explosions
- Residual connections and normalization deal with vanishing gradients