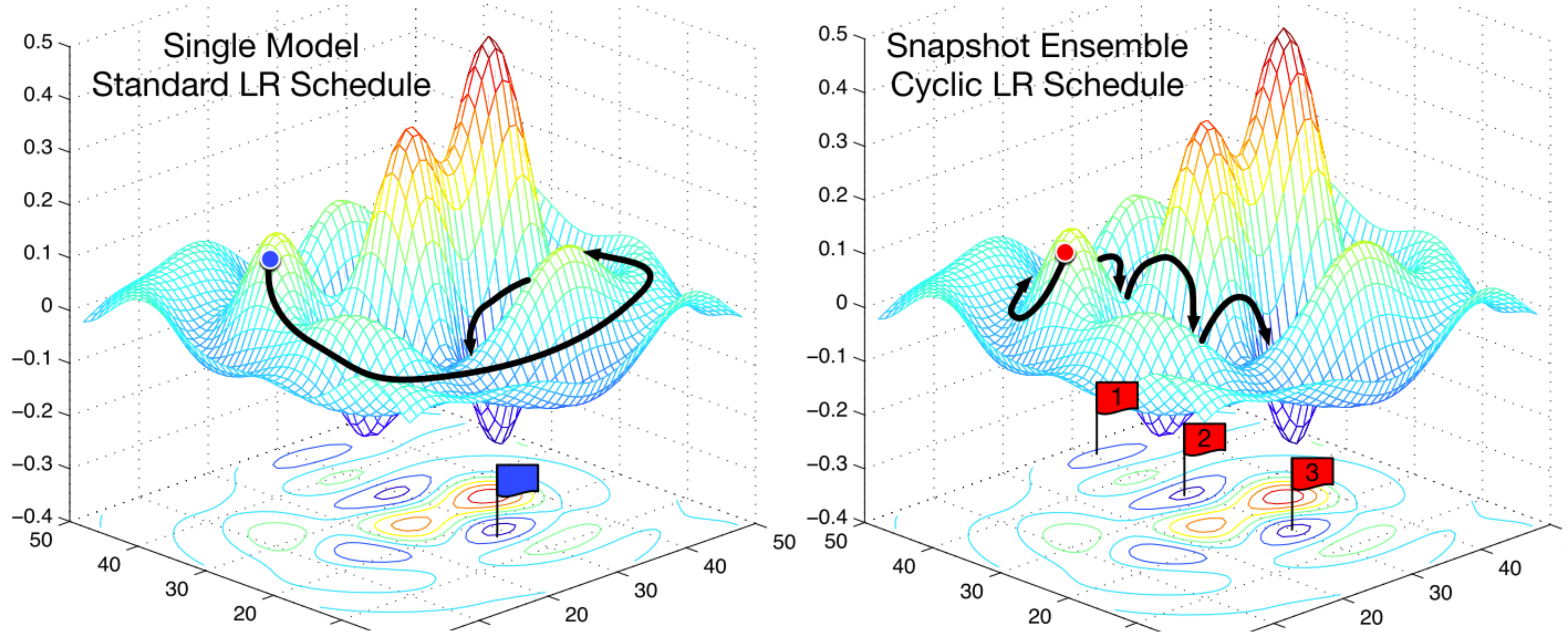
16. Training your neural network

EECE454 Introduction to
Machine Learning Systems

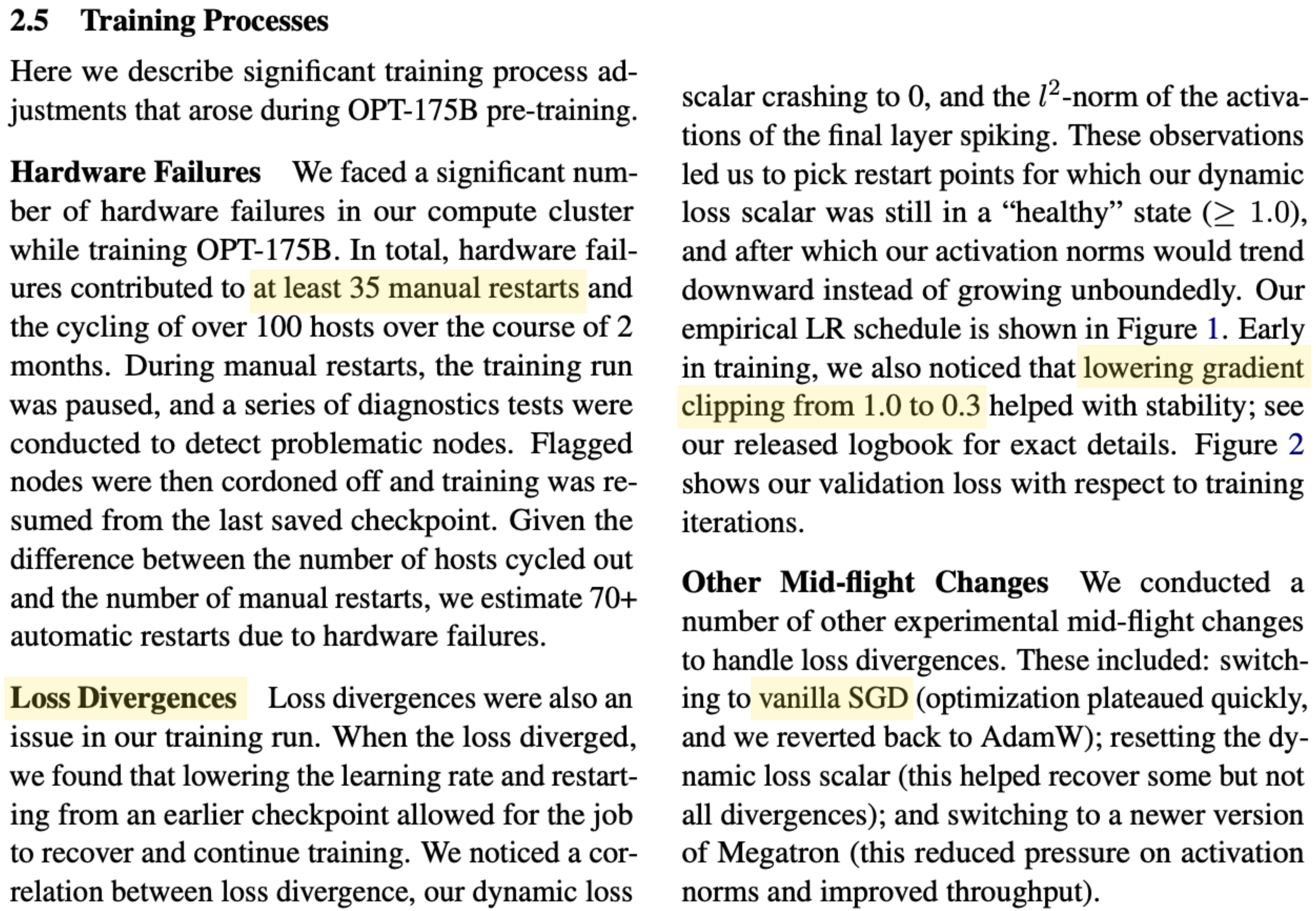
2023 Fall, Jaeho Lee

Scope

- **Monday.** Described how **Backprop** works
  - A way to run SGD efficiently
- **Today.** The success of SGD relies on how you tune them!
  - No good theory, but some rule of thumbs



Source: Meta AI, “OPT: Open Pre-trained Transformer Language Models,” 2022.



2.5 Training Processes

Here we describe significant training process adjustments that arose during OPT-175B pre-training.

Hardware Failures We faced a significant number of hardware failures in our compute cluster while training OPT-175B. In total, hardware failures contributed to at least 35 manual restarts and the cycling of over 100 hosts over the course of 2 months. During manual restarts, the training run was paused, and a series of diagnostics tests were conducted to detect problematic nodes. Flagged nodes were then cordoned off and training was resumed from the last saved checkpoint. Given the difference between the number of hosts cycled out and the number of manual restarts, we estimate 70+ automatic restarts due to hardware failures.

Loss Divergences Loss divergences were also an issue in our training run. When the loss diverged, we found that lowering the learning rate and restarting from an earlier checkpoint allowed for the job to recover and continue training. We noticed a correlation between loss divergence, our dynamic loss scalar crashing to 0, and the l^2-norm of the activations of the final layer spiking. These observations led us to pick restart points for which our dynamic loss scalar was still in a “healthy” state (≥ 1.0), and after which our activation norms would trend downward instead of growing unboundedly. Our empirical LR schedule is shown in Figure 1. Early in training, we also noticed that lowering gradient clipping from 1.0 to 0.3 helped with stability; see our released logbook for exact details. Figure 2 shows our validation loss with respect to training iterations.

Other Mid-flight Changes We conducted a number of other experimental mid-flight changes to handle loss divergences. These included: switching to vanilla SGD (optimization plateaued quickly, and we reverted back to AdamW); resetting the dynamic loss scalar (this helped recover some but not all divergences); and switching to a newer version of Megatron (this reduced pressure on activation norms and improved throughput).

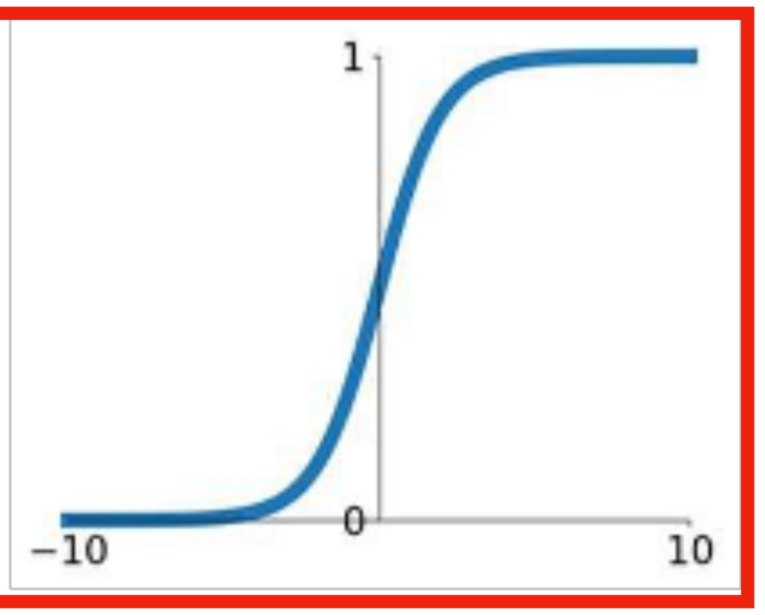
Contents

Activation function

Activation function

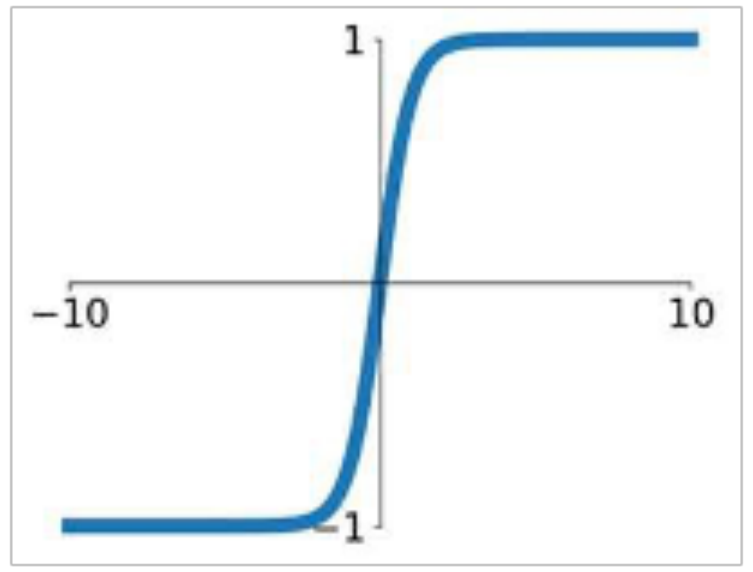
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



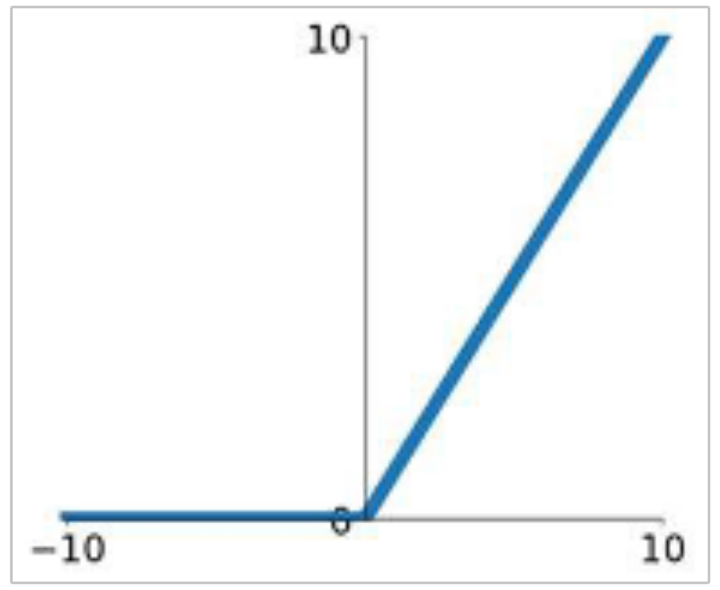
Tanh

$\tanh(x)$



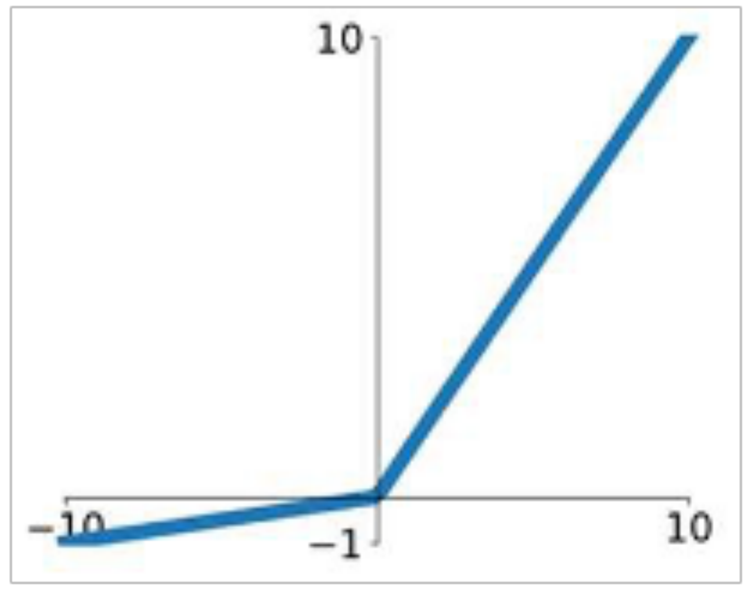
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

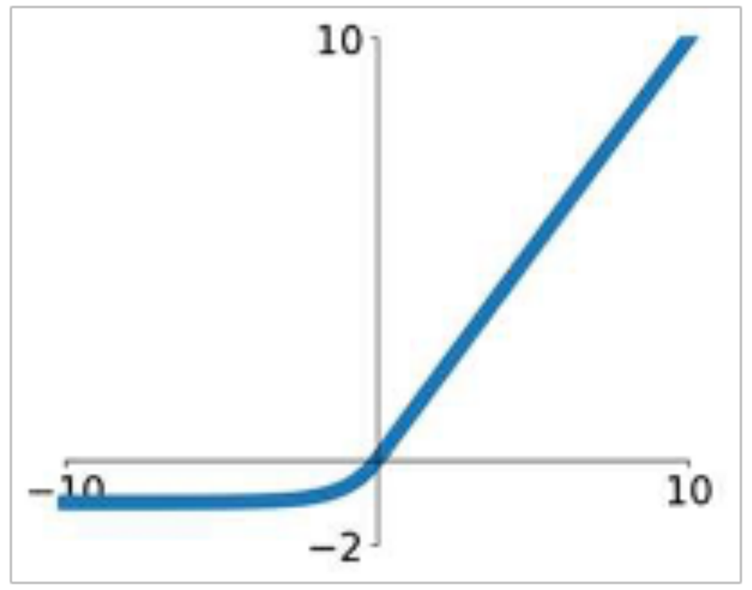


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



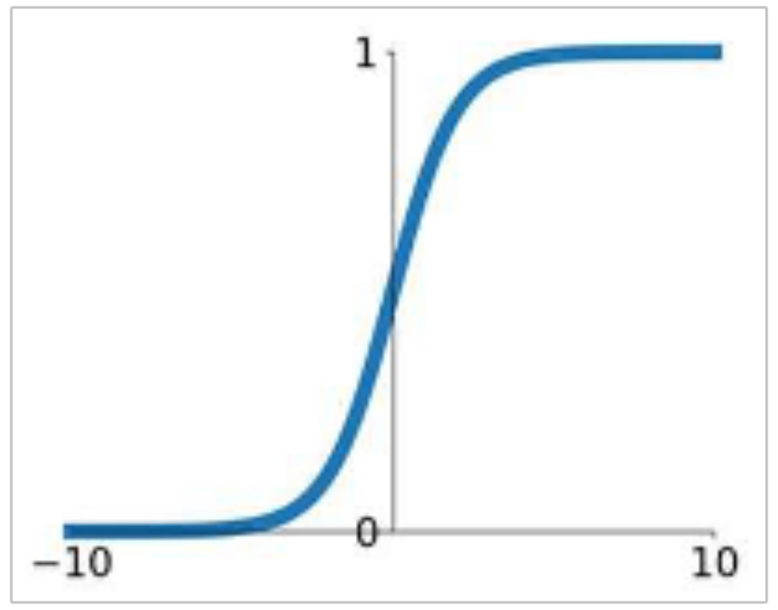
Sigmoids

- Sigmoids *were* popular!
    - Nice biological interpretation as a *firing rate* of a neuron

- Became much less popular, due to three reasons

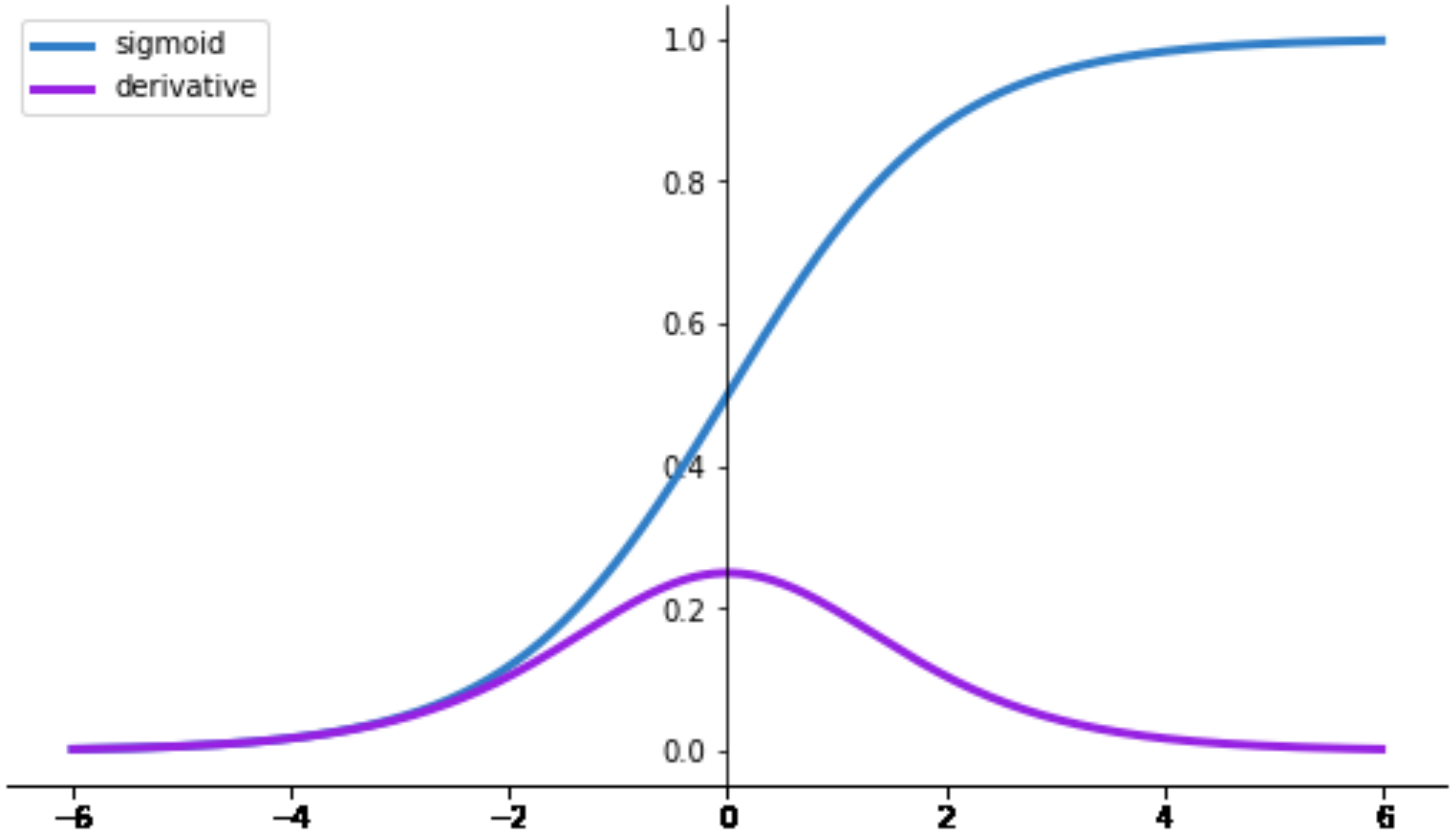
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



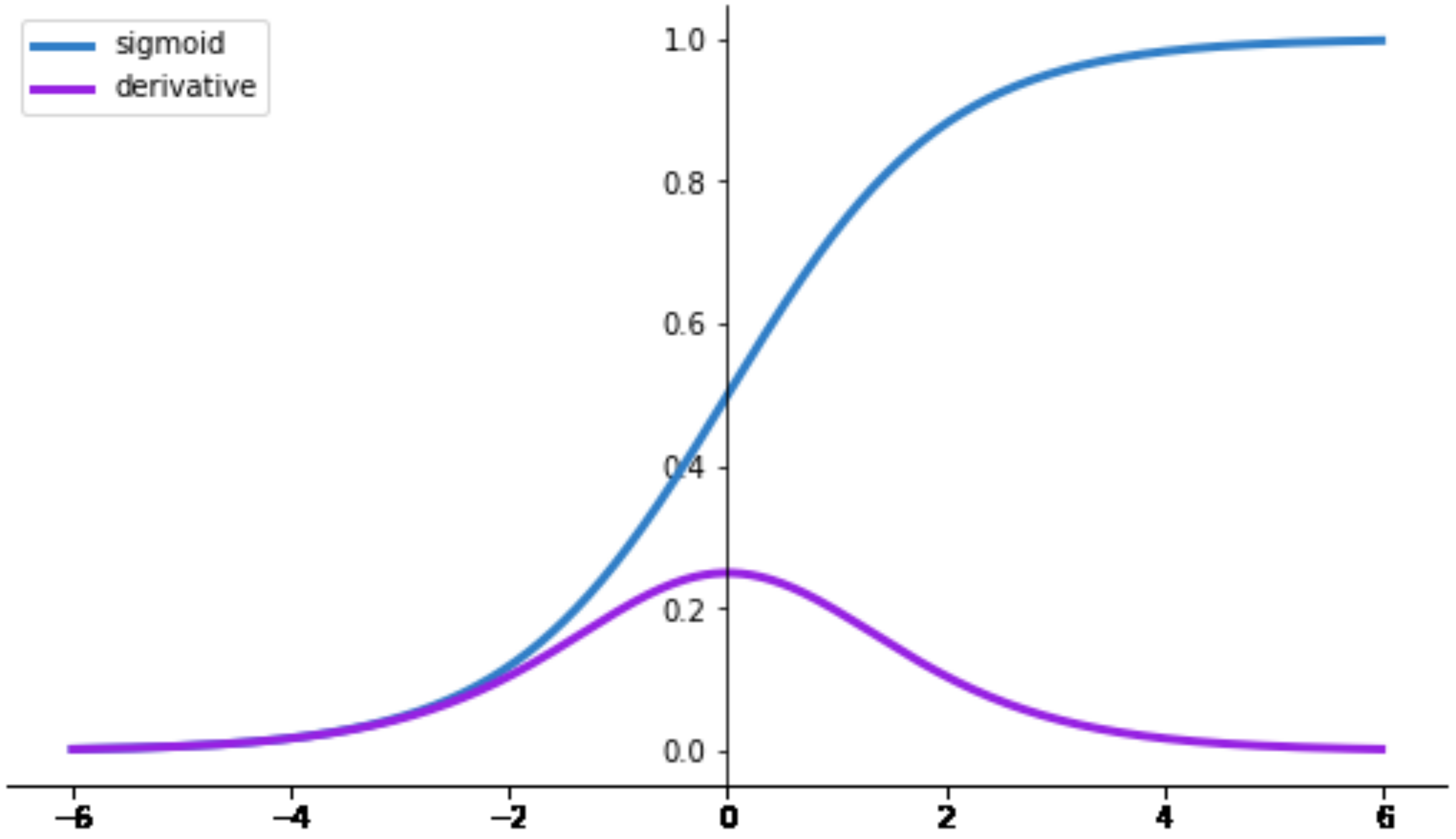
(1) Vanishing Gradients

- Suppose that $f(x) = \sigma(w \cdot x)$
    - Gradient is $\nabla_w f(x) = \sigma'(w \cdot x) \cdot x$



(1) Vanishing Gradients

- Suppose that $f(x) = \sigma(w_L \cdot \sigma(\cdots \sigma(w_1 \cdot x)\cdots))$
  - Then, $\nabla_{w_1} f(x) = \sigma'(w_L \cdot z_L) \cdot \sigma'(w_{L-1} z_{L-1}) \cdot \cdots \cdot \sigma'(w_1 \cdot x) \cdot x$



Q. What happens if $x \approx 0$?

Q. What happens if $x \gg 1$?

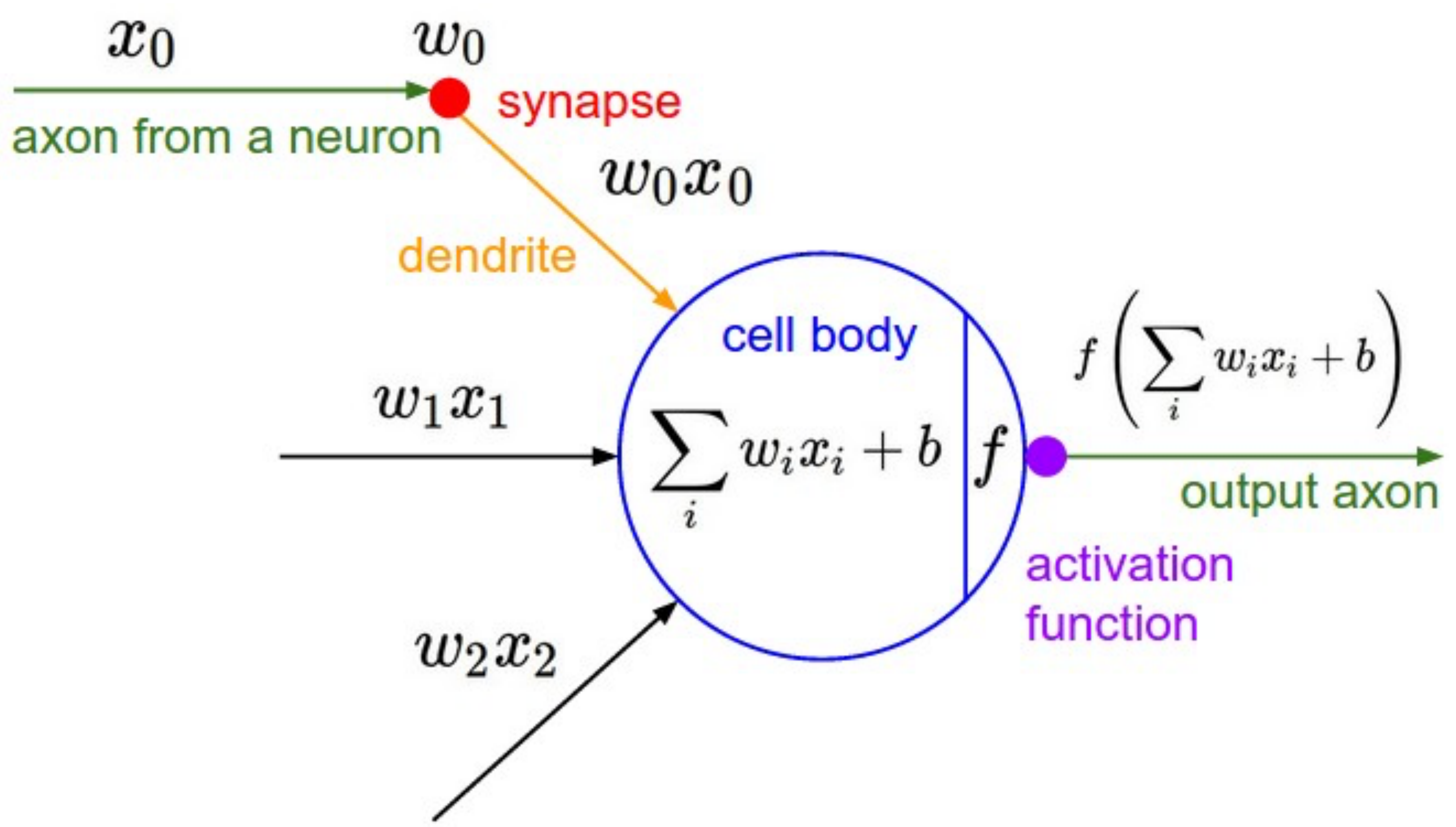
Q. What happens if $x \ll -1$?

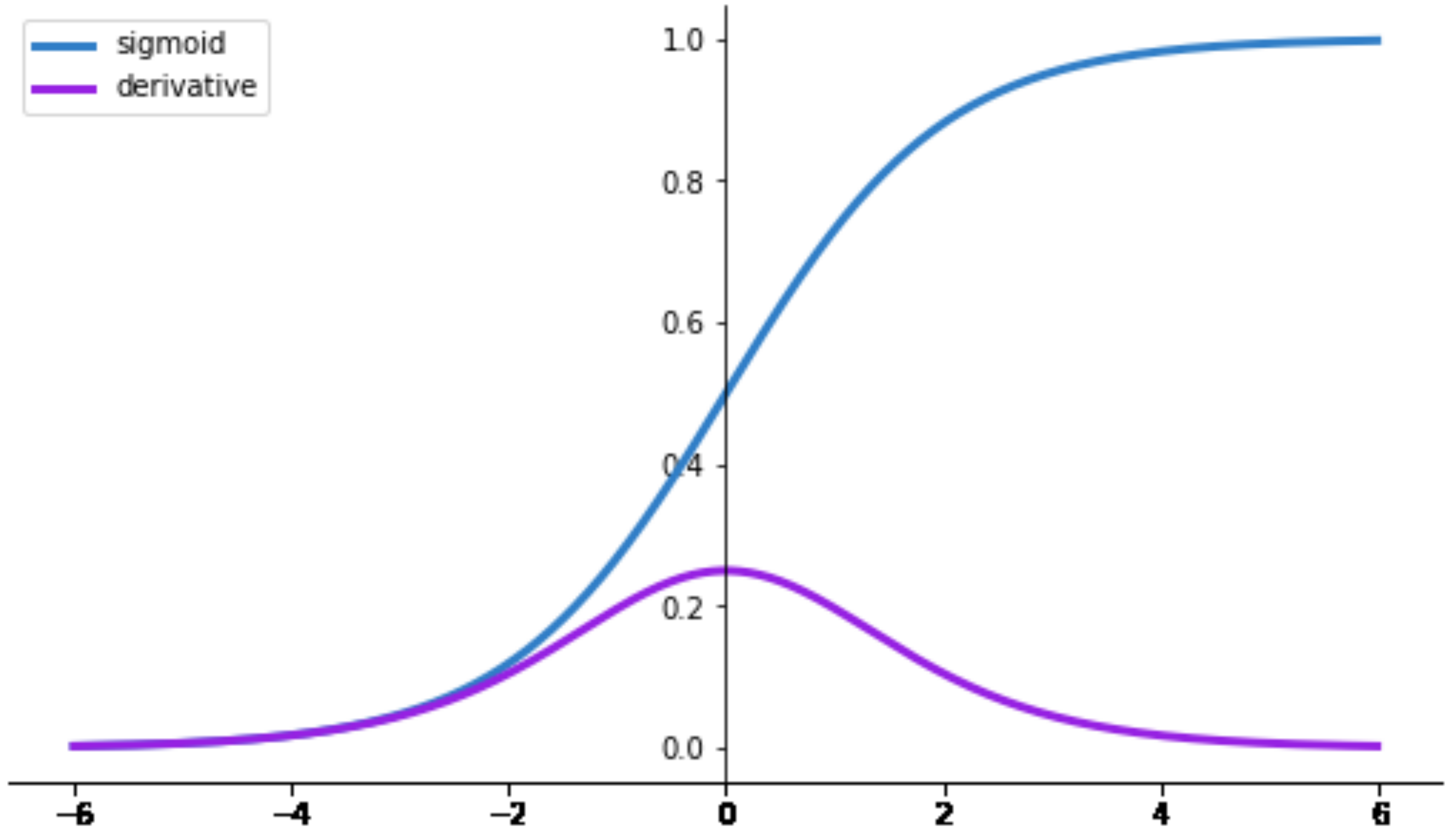
(2) Not zero-centered

- Suppose that $f(x) = \sigma\left(\mathbf{w}^\top \mathbf{x}\right)$
  - Gradients are $\nabla_{w_i} f(\mathbf{x}) = \sigma'(\mathbf{w}^\top \mathbf{x}) \cdot x_i$

positive

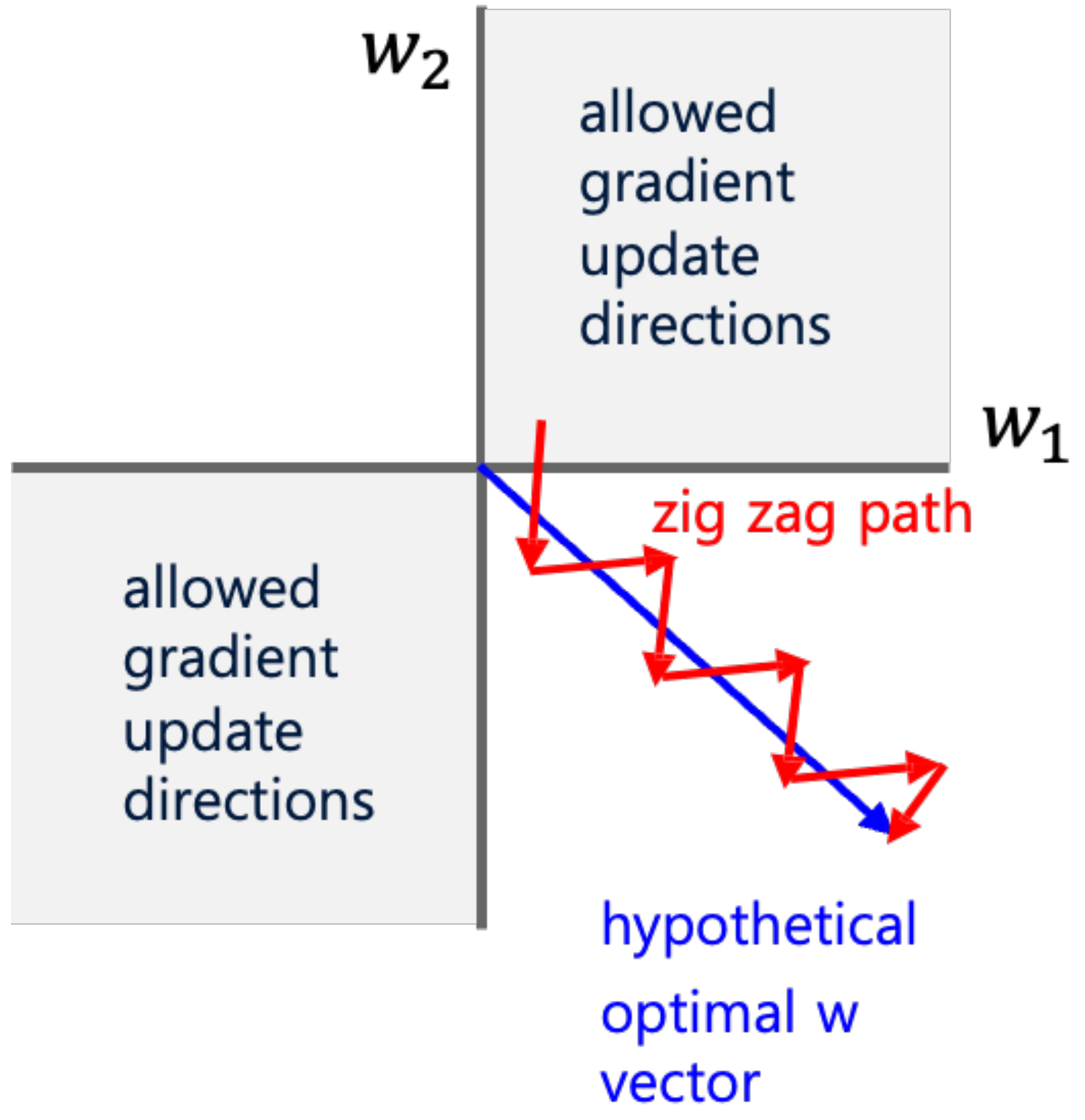
positive, if also sigmoid outputs





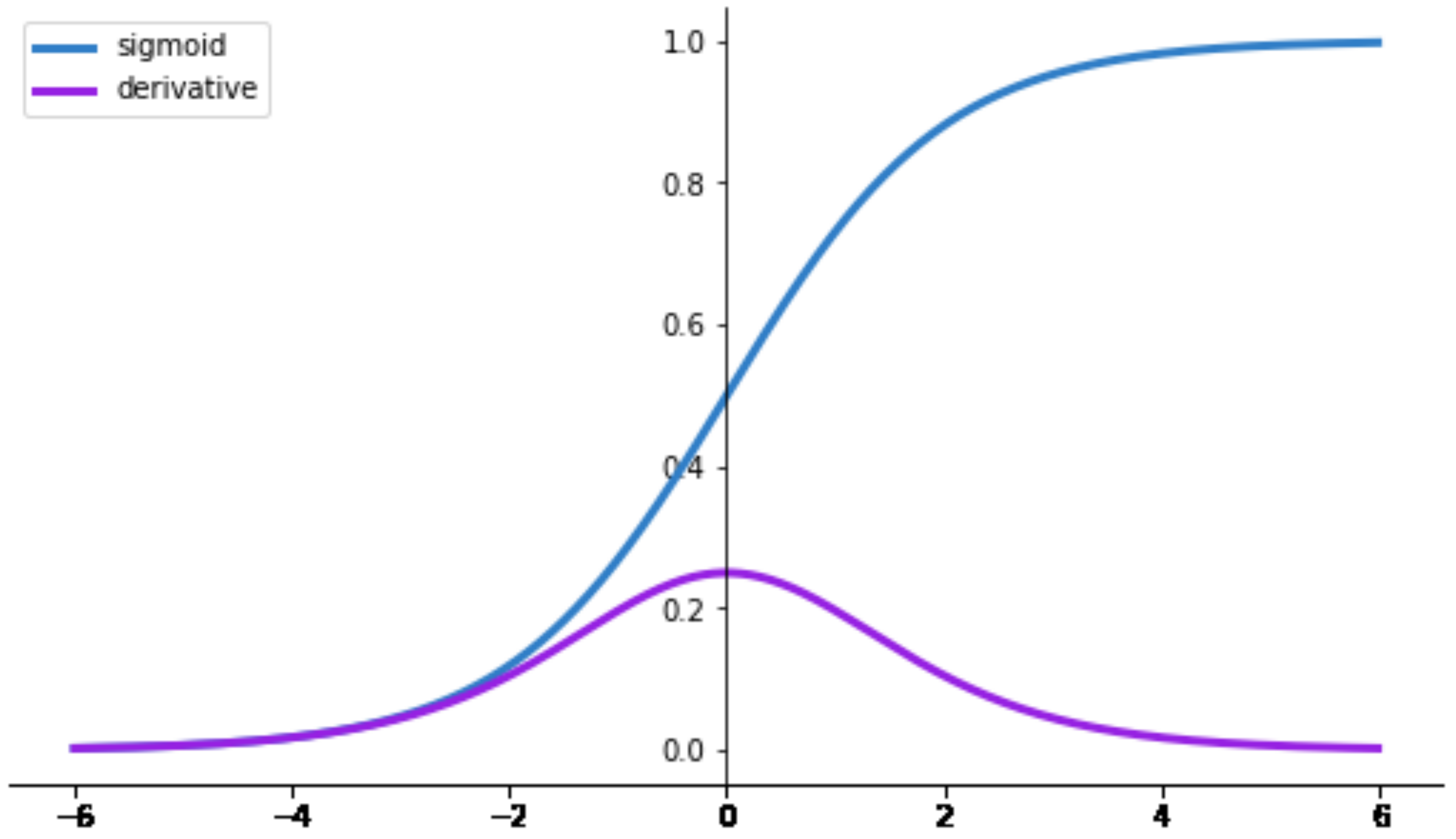
(2) Not zero-centered

- Suppose that $f(x) = \sigma\left(\mathbf{w}^\top \mathbf{x}\right)$
  - Gradients are $\nabla_{w_i} f(\mathbf{x}) = \sigma'(\mathbf{w}^\top \mathbf{x}) \cdot x_i$
  - This means loss gradients are always all-positive or all-negative
    - Results in undesirable zigzag paths
    - Mitigated when x_i are normalized!


w_2
allowed gradient update directions
w_1
zig zag path
allowed gradient update directions
hypothetical optimal w vector

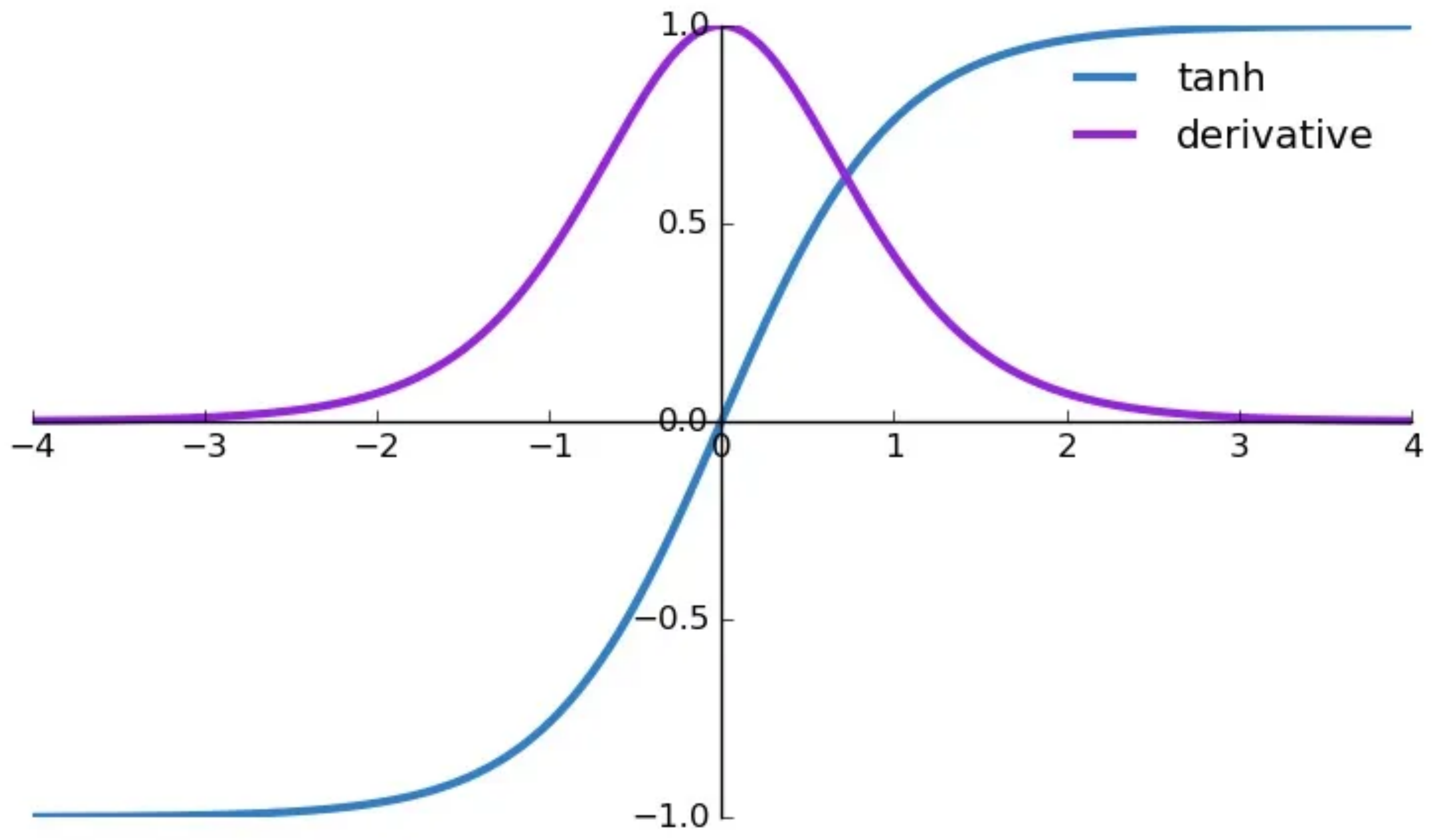
(3) Efficiency

- **Inference.** Need to compute $\sigma(t) = 1/(1 + \exp(-t))$
- **Training.** Need to compute $\sigma'(t) = \sigma(t)(1 - \sigma(t))$
  Need to store the computed outputs for backprop.



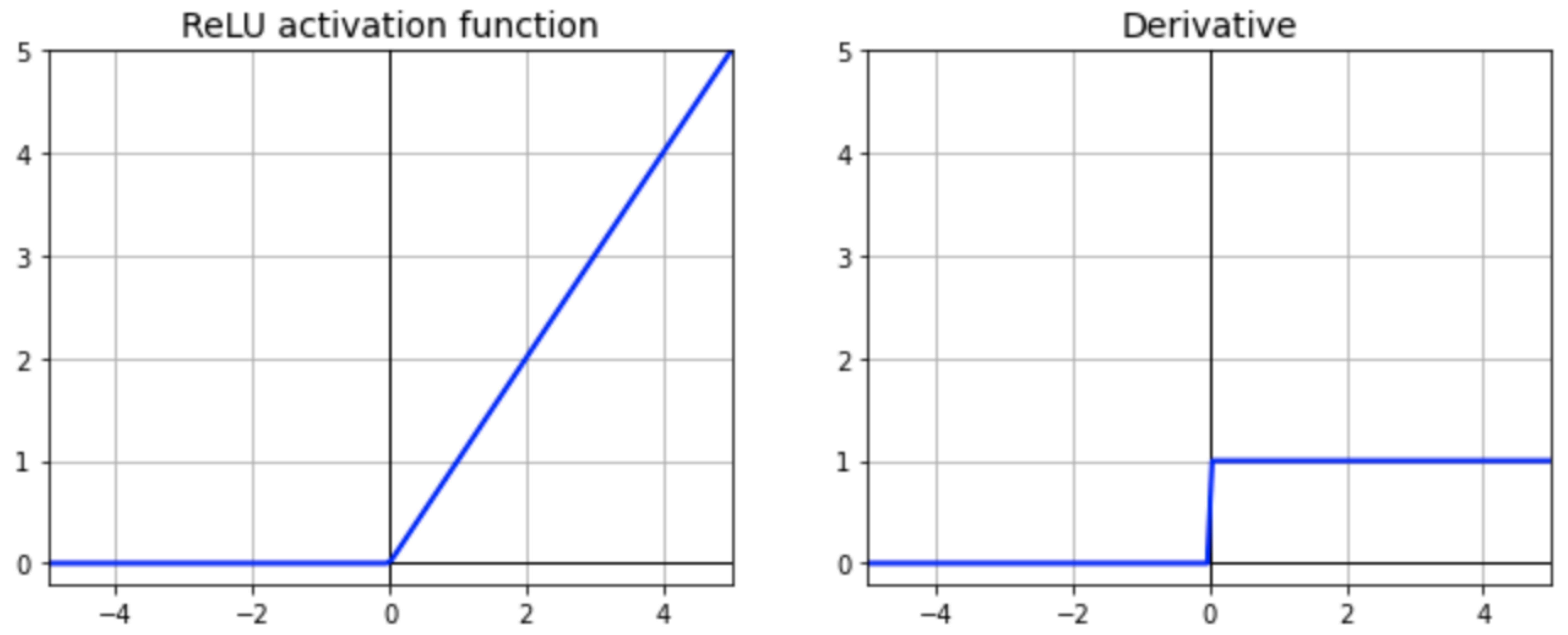
Tanh

- ✅ Zero-centered!
- ❌ also experiences vanishing gradient.
- ❌ computationally inefficient

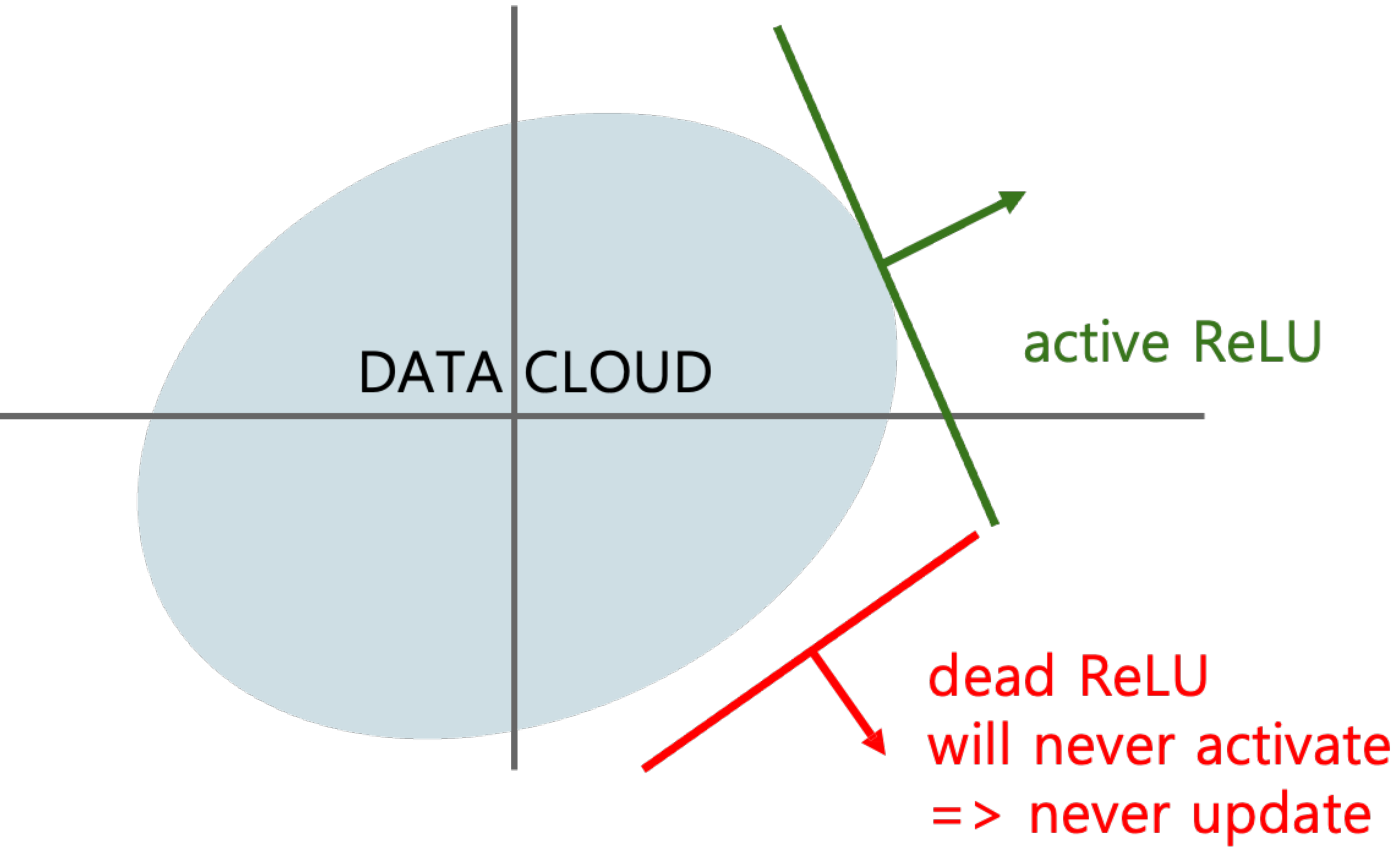


ReLU

- ✅ Computationally efficient!
- ✅ Converges faster in practice (e.g., 6x)
- ❌ Still not zero-centered. (normalize!)
- ❌ “Dead neurons”



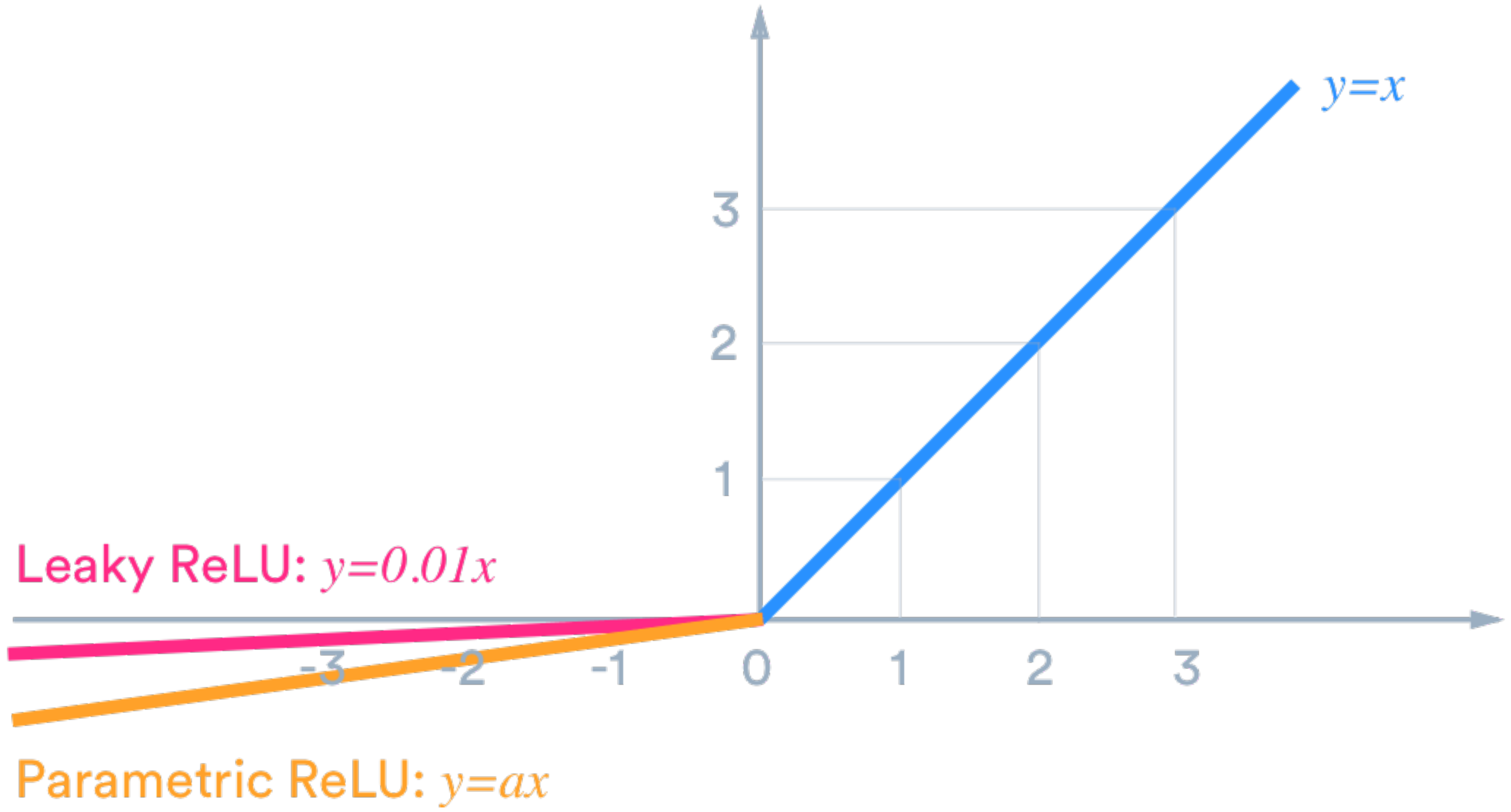
Dying ReLU



- Some neurons never activate
  - Sometimes up to 40%
- **A trick.** Use small but positive biases (e.g., 0.01)

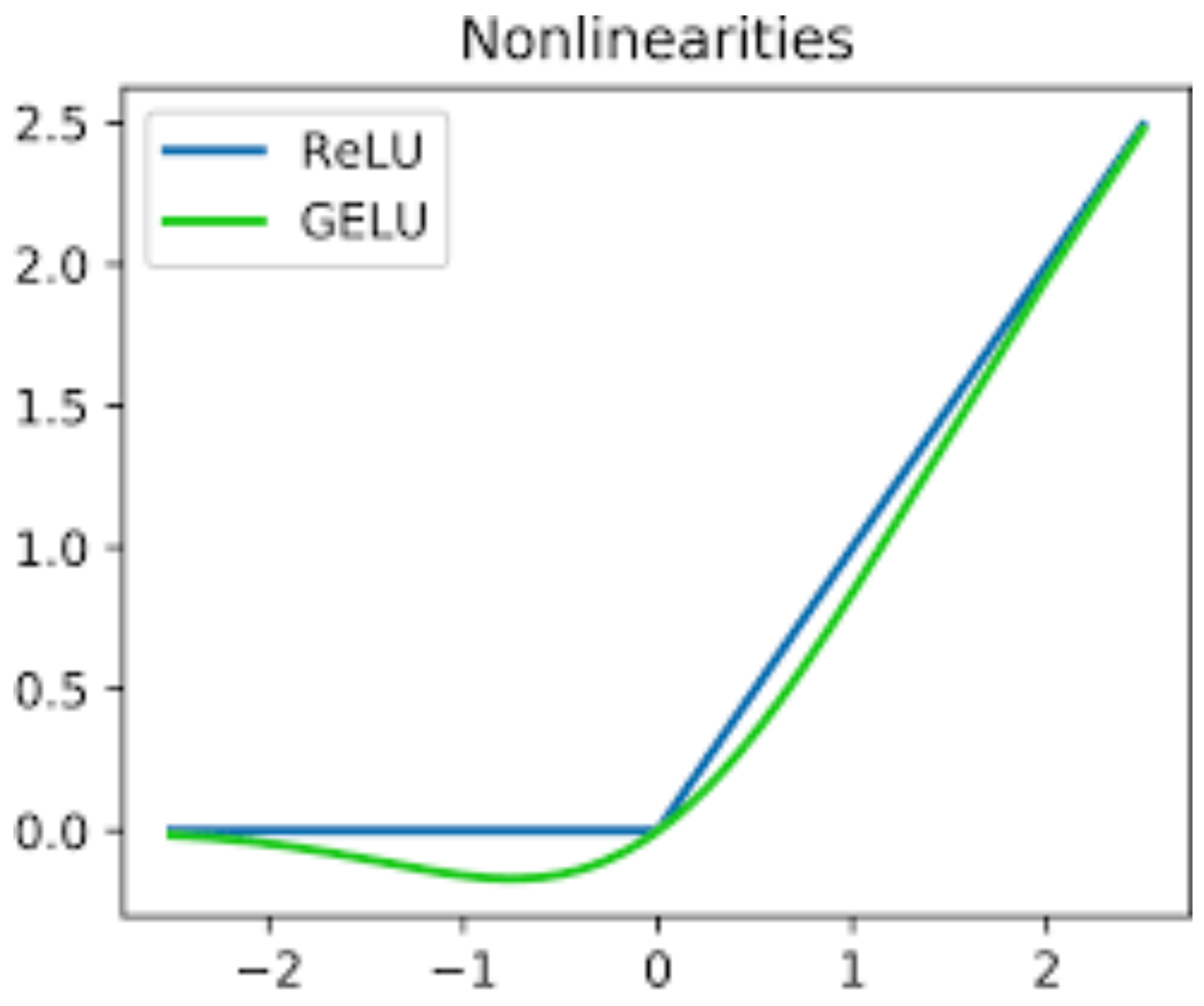
Leaky ReLU / Parametric ReLU

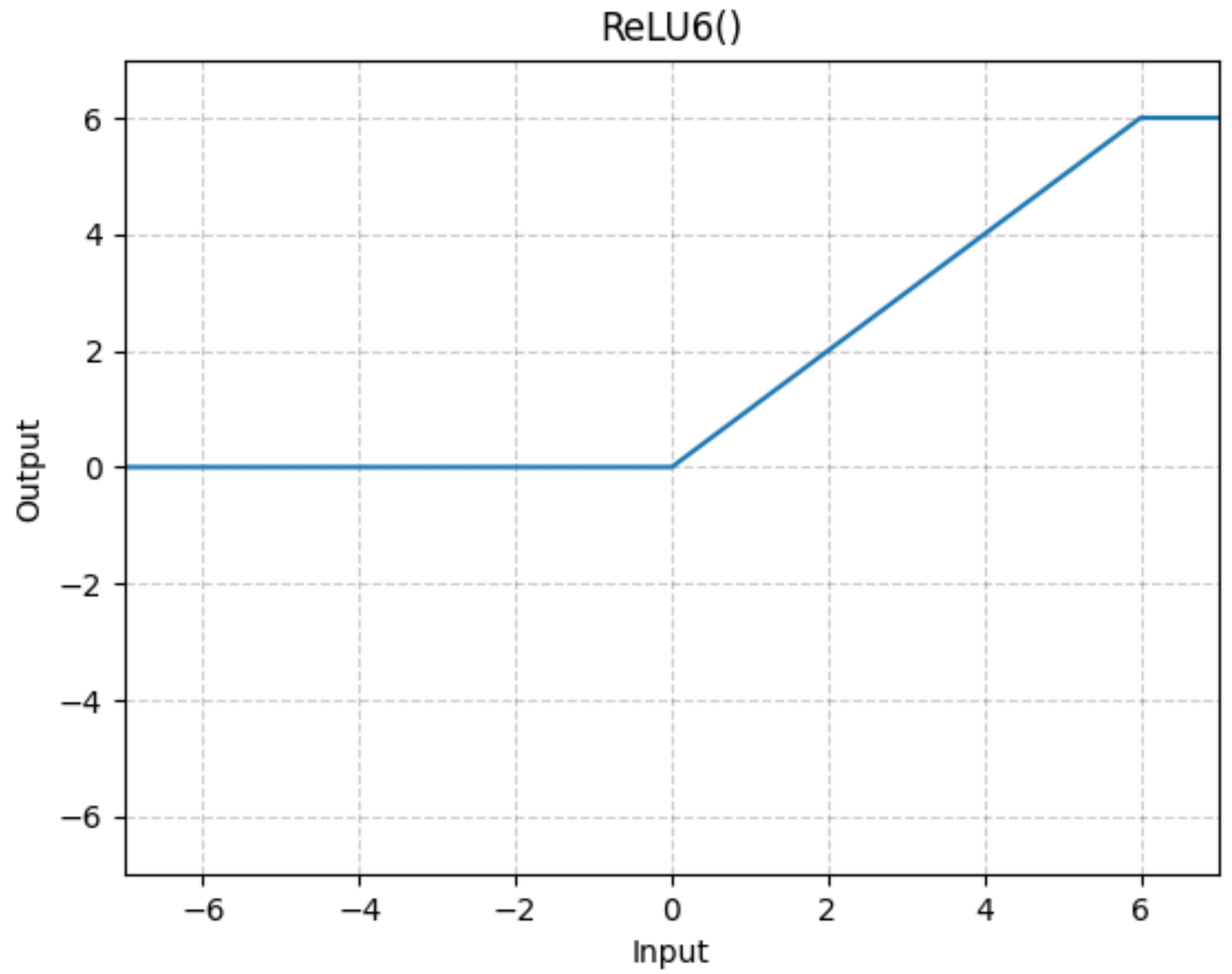
- ✅ No dead neurons!



Modern Choices

- Practitioners who train giant models love GeLU / SwiGLU
- Quantization people loves ReLU6

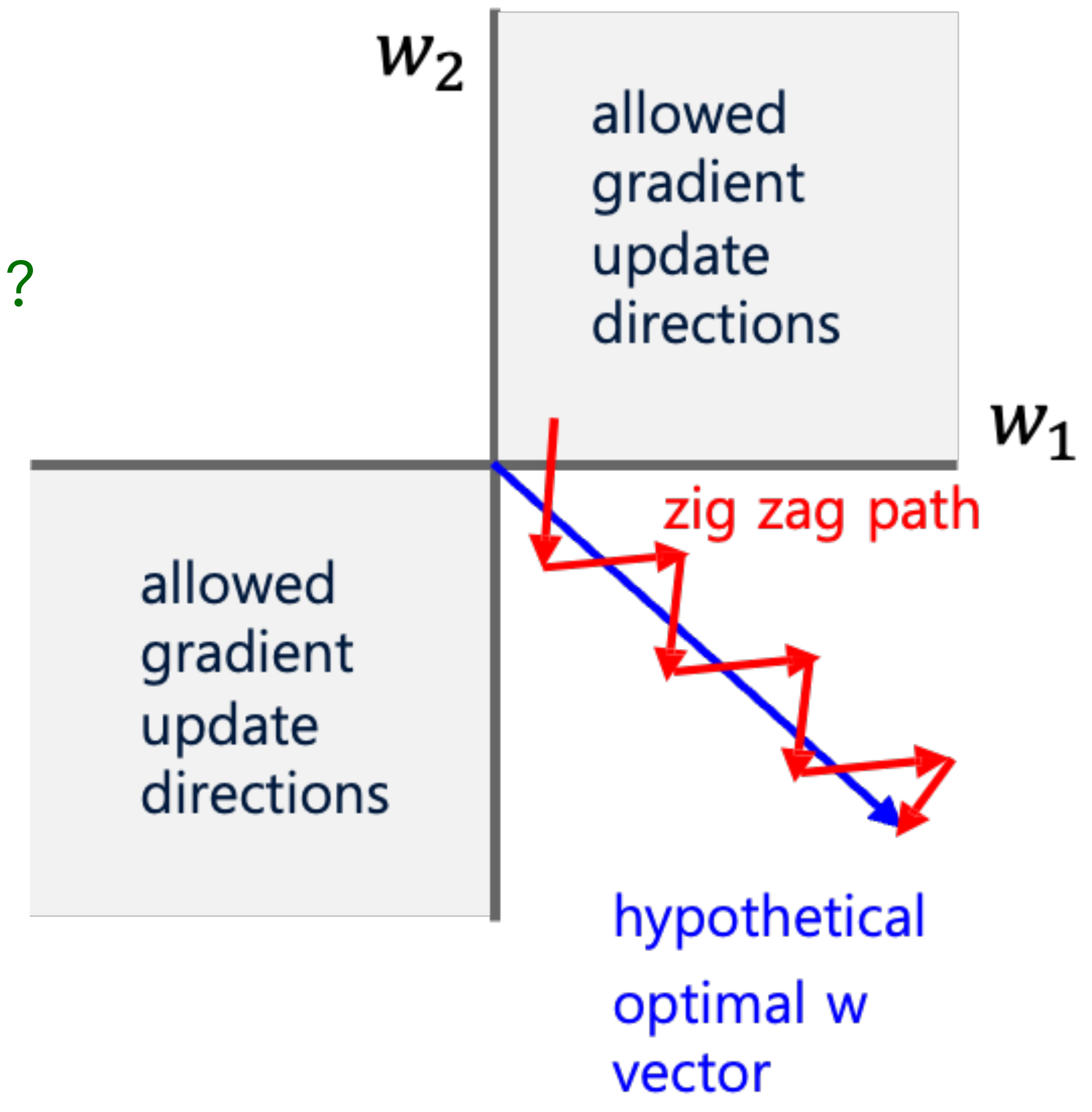




Data Preprocessing

Recall

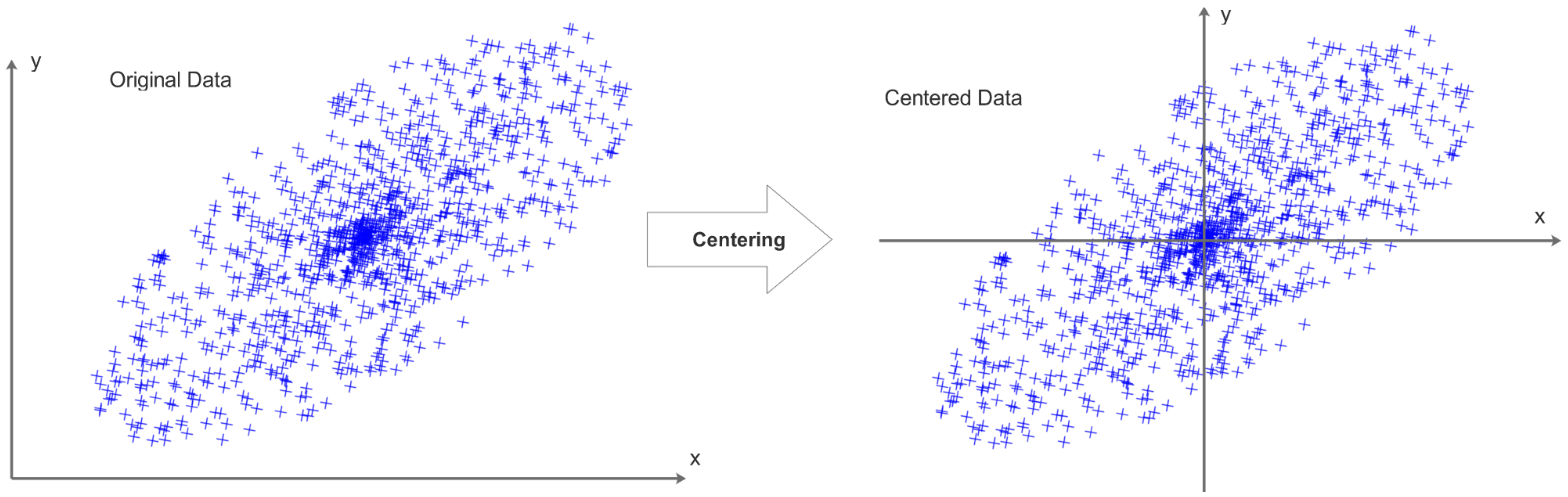
- Recall that we have “zigzags” when the neuron inputs are all-positive.
- Suppose that $f(x) = \sigma\left(\mathbf{w}^\top \mathbf{x}\right)$
    - Gradients are $\nabla_{w_i} f(\mathbf{x}) = \underbrace{\sigma'(\mathbf{w}^\top \mathbf{x})}_{\text{positive}} \cdot \underbrace{x_i}_{\text{positive...?}}$
- **Idea.** Force data to have different signs



w_2
allowed gradient update directions
w_1
zig zag path
allowed gradient update directions
hypothetical optimal w vector

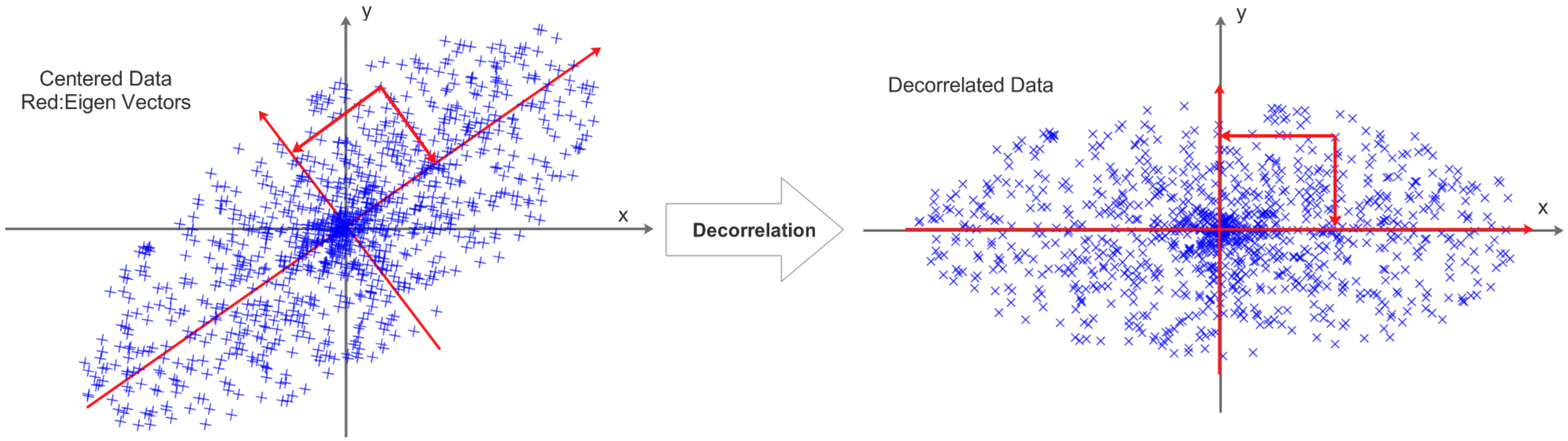
Preprocessing

- **Centering.** Makes the data have zero-mean.



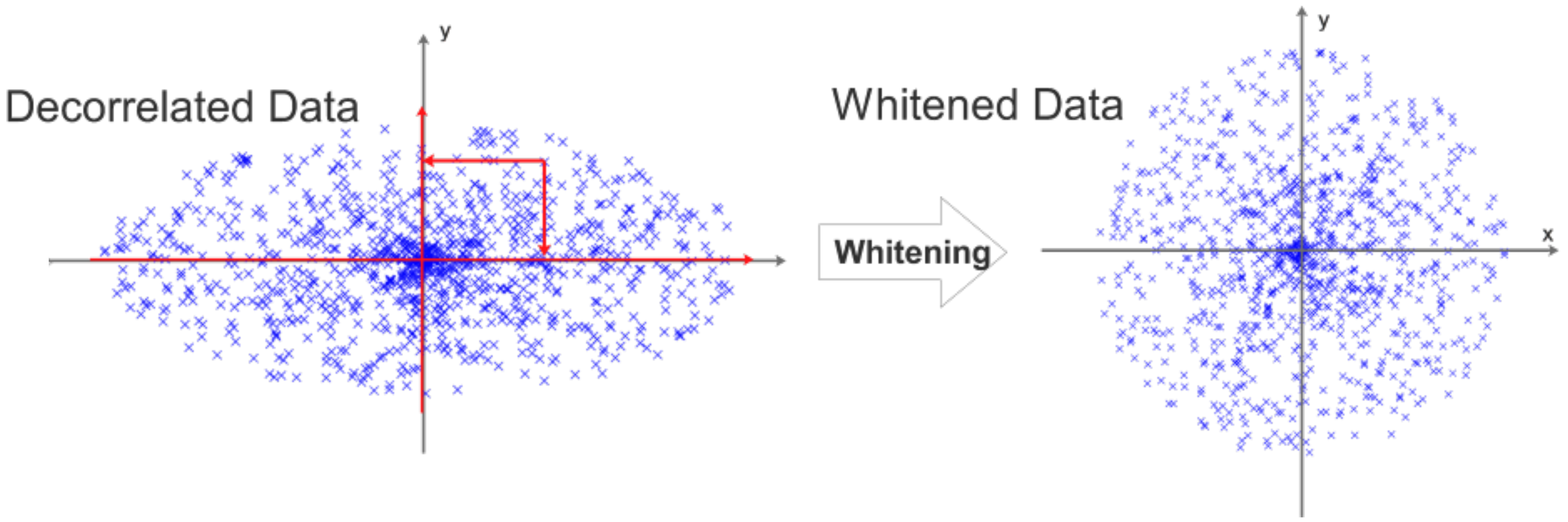
Preprocessing

- **Decorrelate.** Makes the axes have no correlation. (many cases, skipped—e.g., image)



Preprocessing

- **Whitening.** Makes the each dimension have unit variance / range. (also often skipped)



Remarks

- In some cases, also perform dimensionality reduction.

- In many practical cases (esp. images), only perform **centering**
  - For CIFAR-10…
    - AlexNet -> subtract the mean image ([32,32,3] tensor)
    - VGG -> subtract the mean along RGB channels (3-dim)

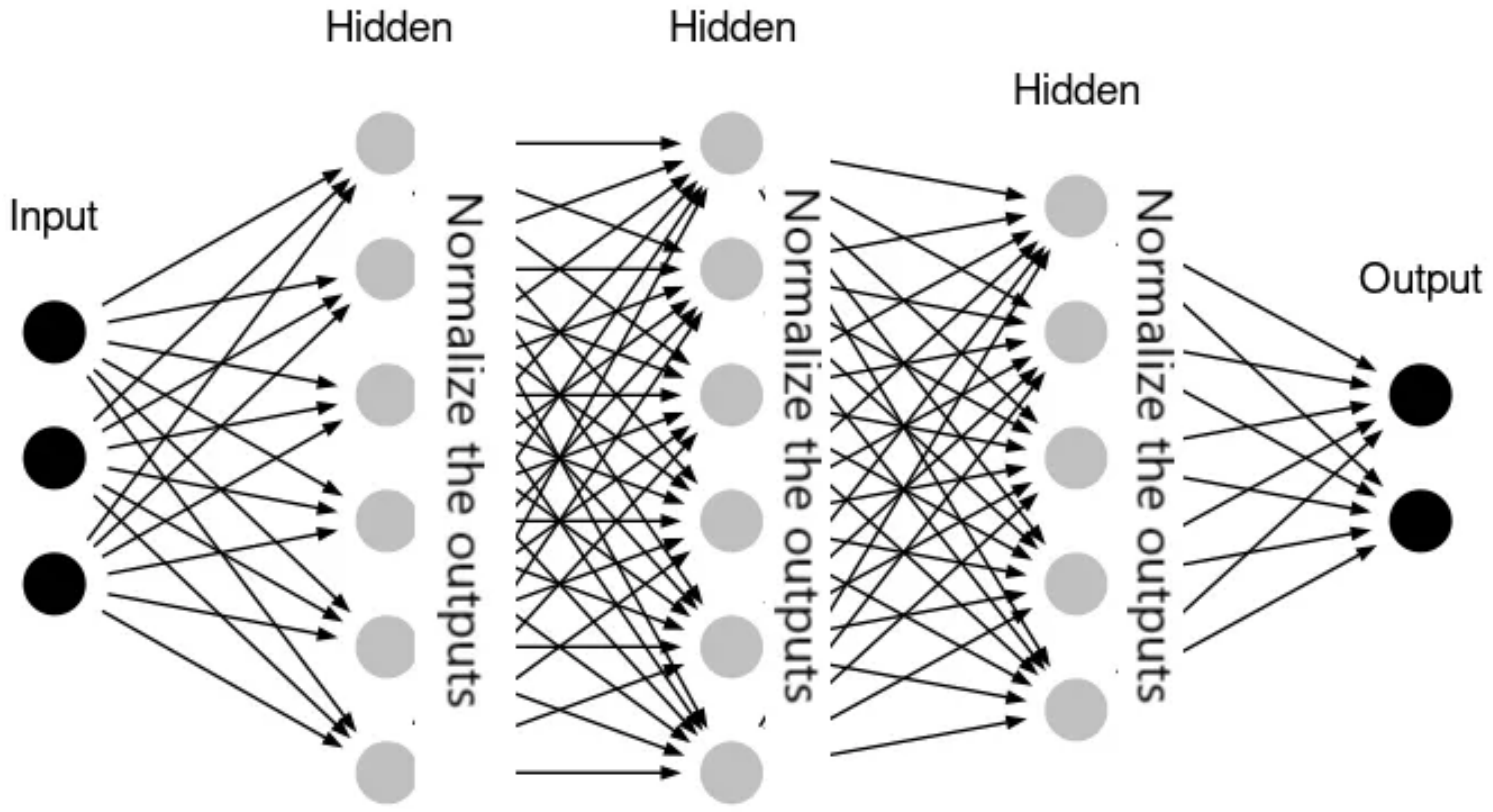
Batch Normalization

Idea

- Performing centering + scaling, but in intermediate layers!

zero-mean

unit variance



Idea

- Consider a batch of activations at some layer: $\mathbf{z}_1, \ldots, \mathbf{z}_B$ (B: batch size)
  - Each activation has d channels: $\mathbf{z}_i = \left(\mathbf{z}_i^{(1)}, \ldots, \mathbf{z}_i^{(d)}\right)$

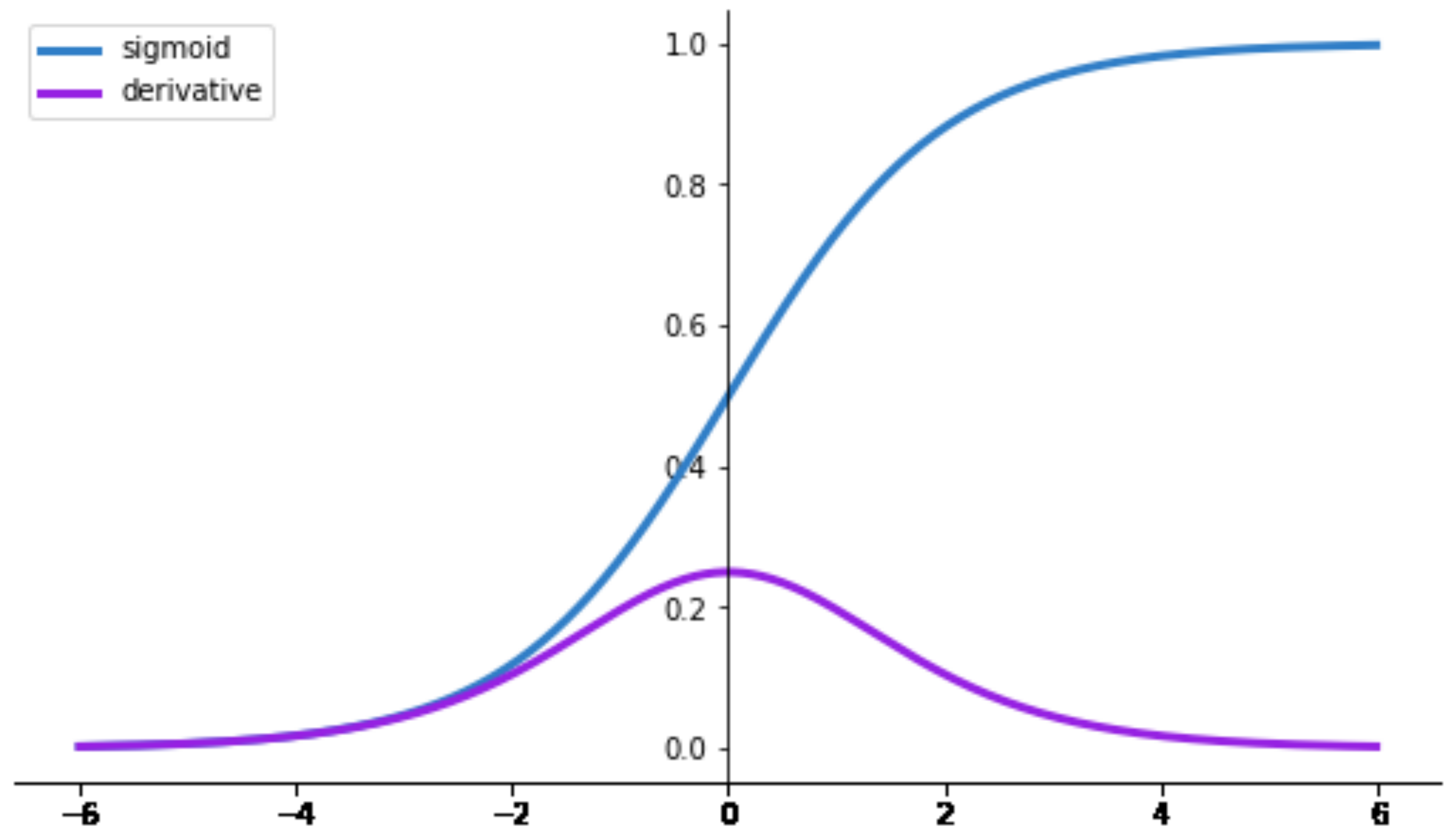
- For each dimension, apply:

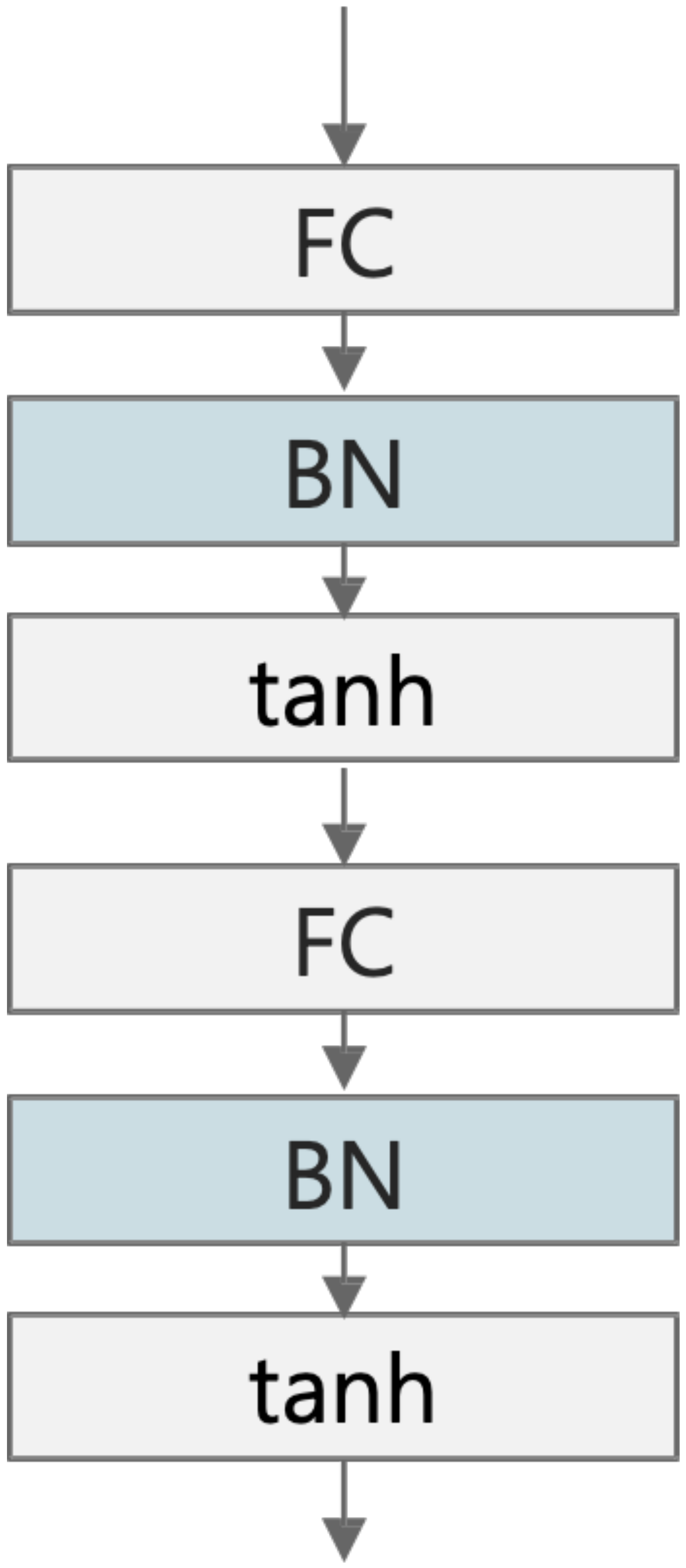
$$\hat{\mathbf{z}}^{(j)} = \frac{\hat{\mathbf{z}}^{(j)} - \mathbb{E}[\mathbf{z}^{(j)}]}{\sqrt{\mathbf{Var}(\mathbf{z}^{(j)})}}$$

  - This is a *differentiable* function!

Where to normalize?

- Mostly placed at...
  - after each Conv/FC layers
  - before activation
- But doing BNs at all layers may be harmful—
  - Normalizing pre-sigmoids puts it in a linear region.





Cure

- We actually add one more linear operation:

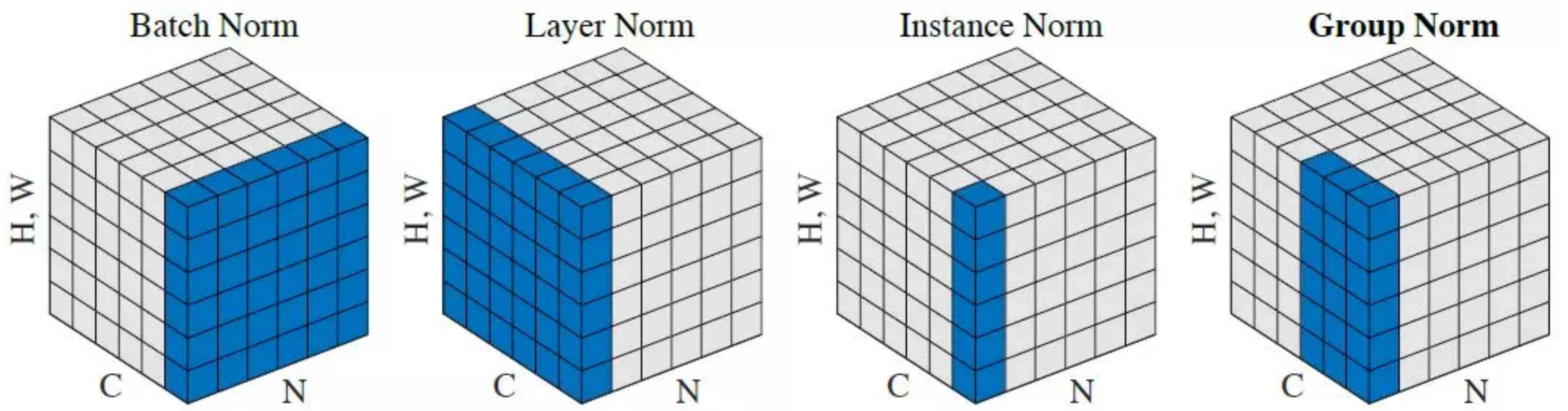
$$\hat{\mathbf{y}}^{(j)} = \gamma^{(j)}\hat{\mathbf{x}}^{(j)} + \beta^{(j)}, \qquad j \in [d]$$

(this can even be identity mappings!)

- **At Inference.** Don't really take data as a batch…
  - **Cure.** Takes a running average of mean/var during training, and use these values at test time.
    (these can usually be merged into FC/Conv layers)

Effect

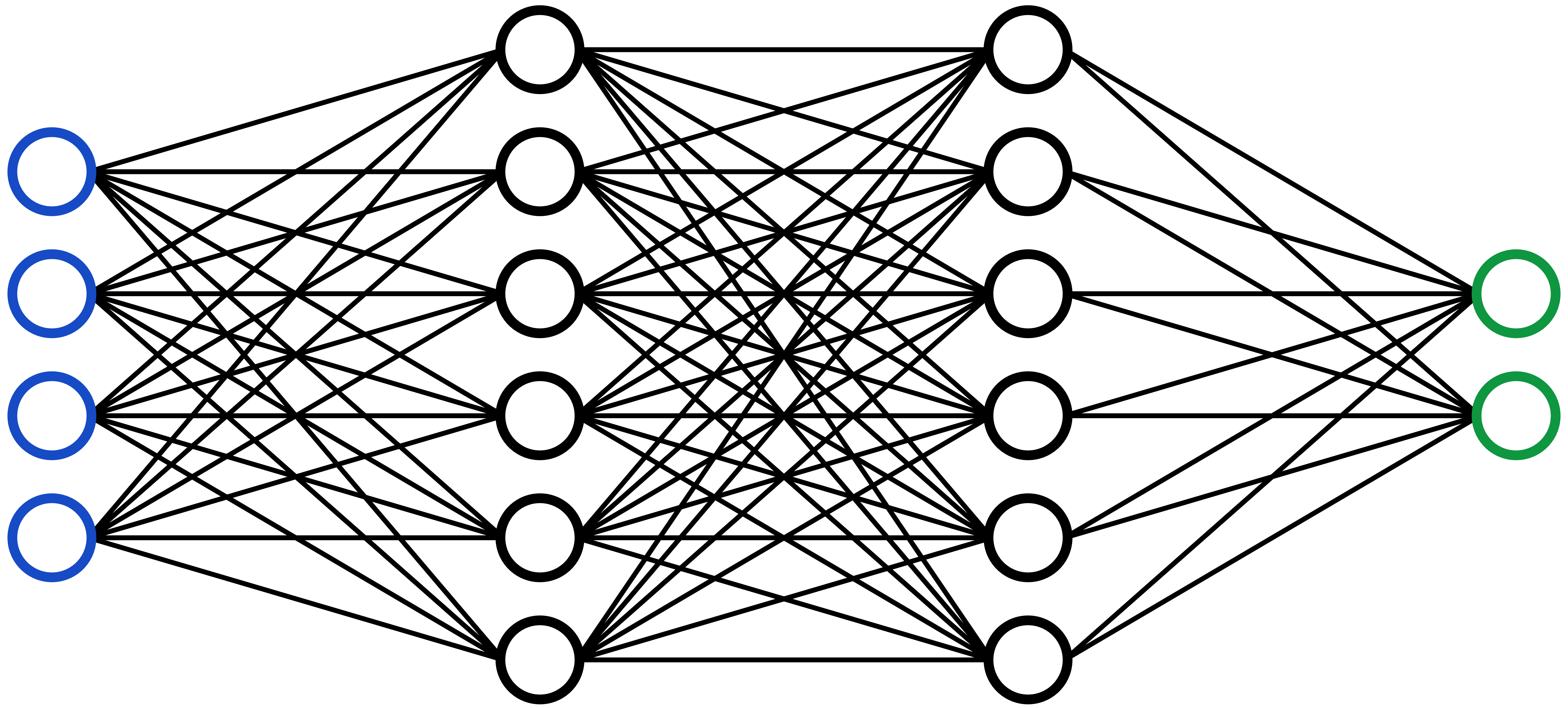
- ✅ Improves the gradient flow during the training.
- ✅ Allows higher learning rates.
- ✅ Reduces the initialization sensitivity.
- ❌ Often, undesired side effects and instability…



Weight Initialization

Question

- What happens if all weights are initialized to the same constant?



Random initialization

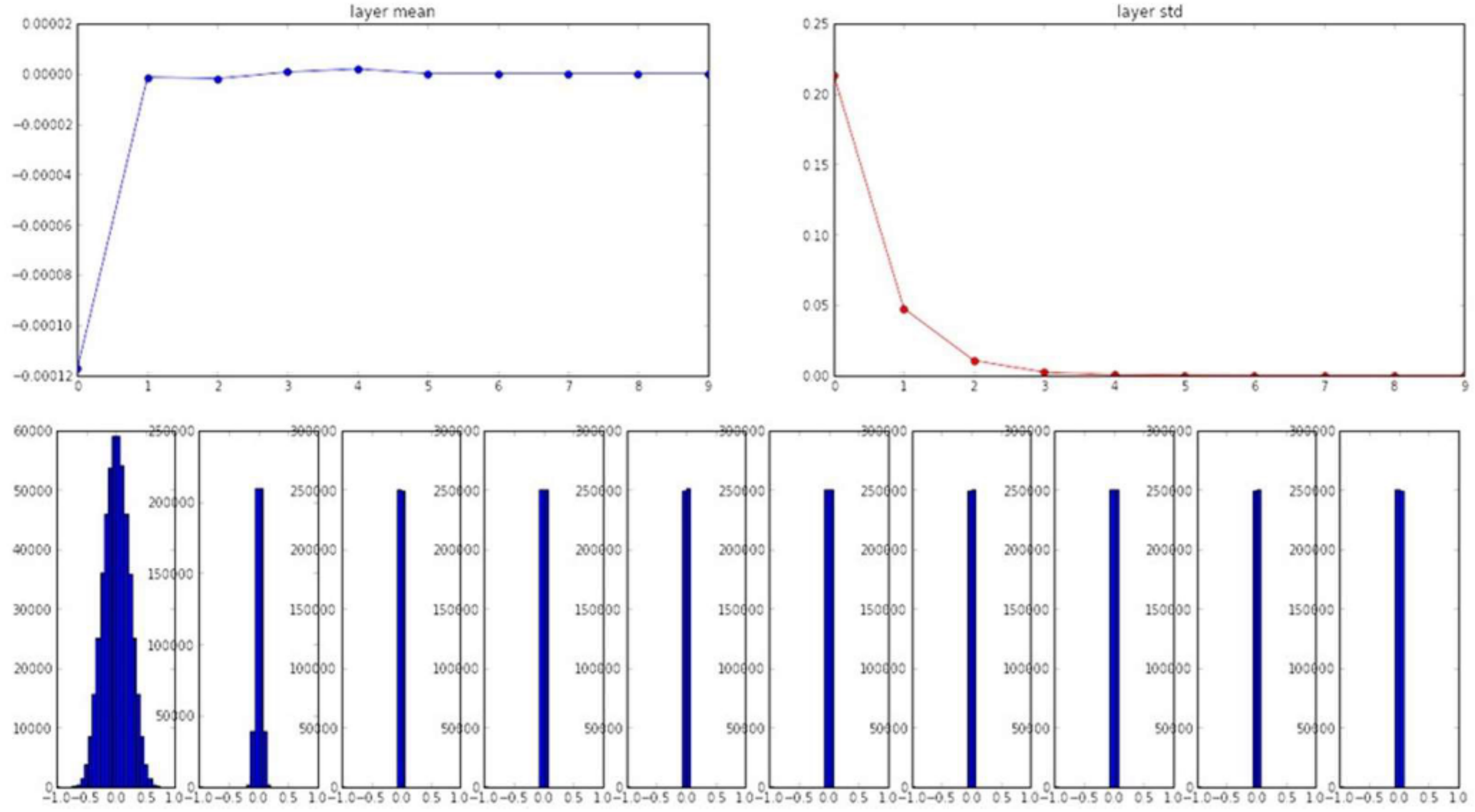
- **First Idea.** Initialize all weights with $w \sim N(0, 0.1^2)$

- **Problem.** Works okay for shallow nets, but not for deeper models…
    - _Example_. 10-layer network with
        - 500 neurons in each layer
        - tanh nonlinearities

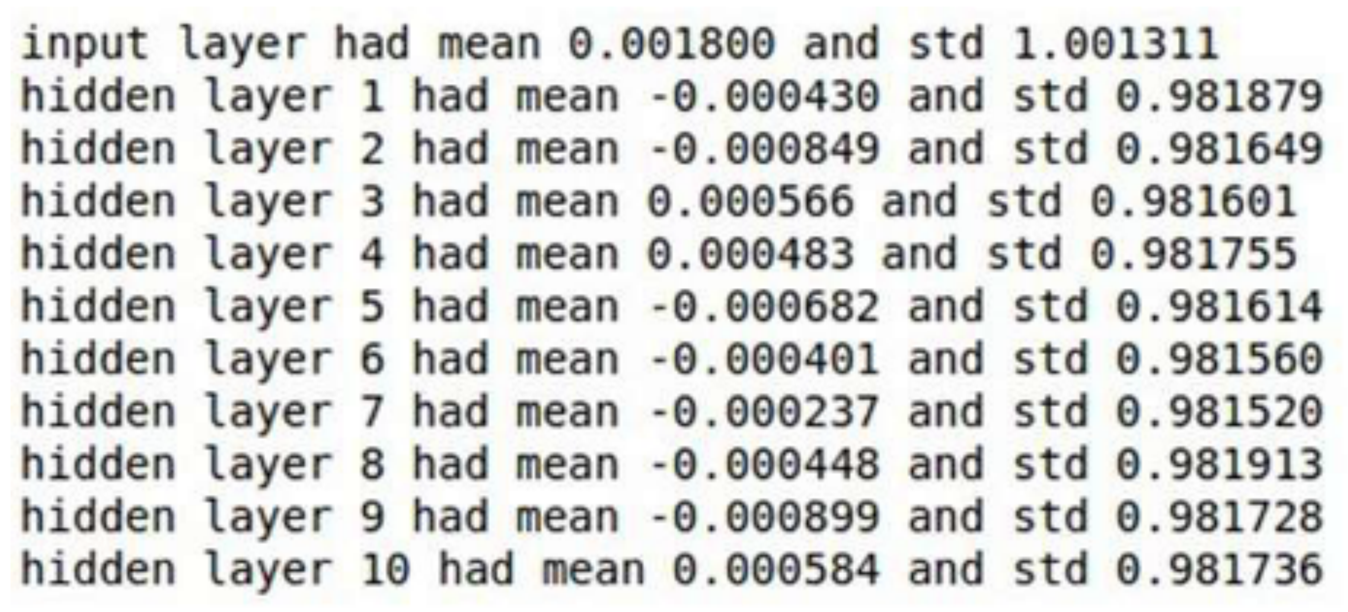
All activations become zero...

```
input layer had mean 0.000927 and std 0.998388
hidden layer 1 had mean -0.000117 and std 0.213081
hidden layer 2 had mean -0.000001 and std 0.047551
hidden layer 3 had mean -0.000002 and std 0.010630
hidden layer 4 had mean 0.000001 and std 0.002378
hidden layer 5 had mean 0.000002 and std 0.000532
hidden layer 6 had mean -0.000000 and std 0.000119
hidden layer 7 had mean 0.000000 and std 0.000026
hidden layer 8 had mean -0.000000 and std 0.000006
hidden layer 9 had mean 0.000000 and std 0.000001
hidden layer 10 had mean -0.000000 and std 0.000000
```

Question. If activations are zero, what would the gradients be?

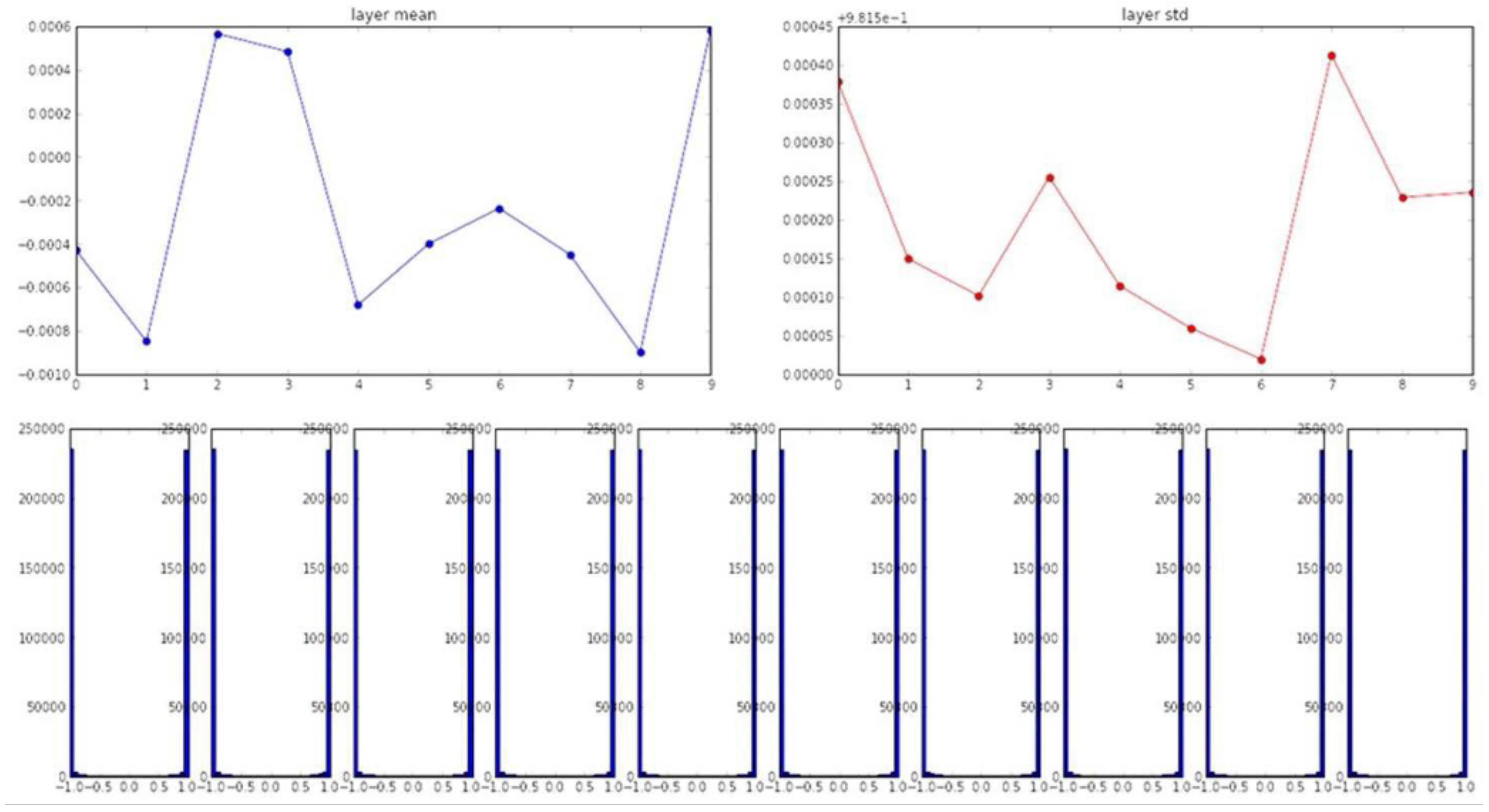


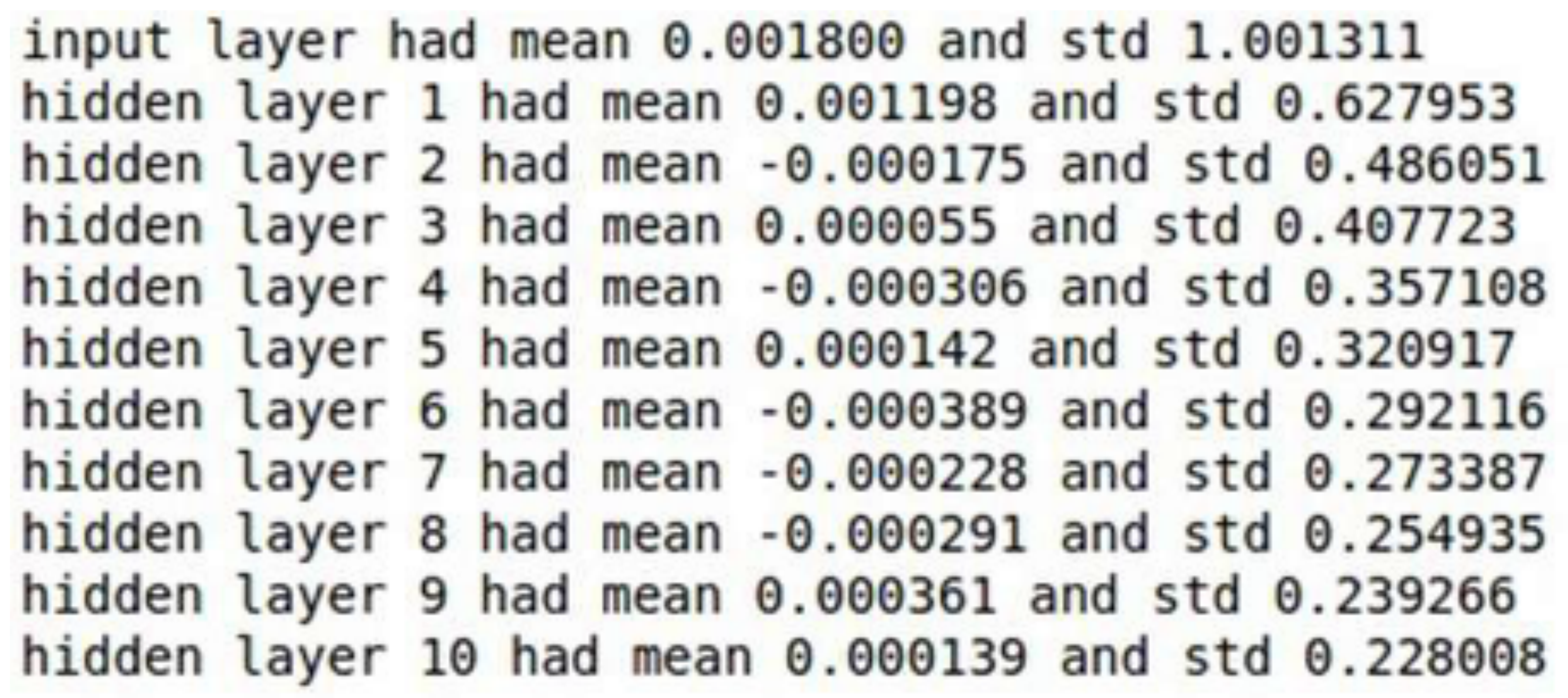
If weights are $\sim N(0,10^2)$



```
input layer had mean 0.001800 and std 1.001311
hidden layer 1 had mean -0.000430 and std 0.981879
hidden layer 2 had mean -0.000849 and std 0.981649
hidden layer 3 had mean 0.000566 and std 0.981601
hidden layer 4 had mean 0.000483 and std 0.981755
hidden layer 5 had mean -0.000682 and std 0.981614
hidden layer 6 had mean -0.000401 and std 0.981560
hidden layer 7 had mean -0.000237 and std 0.981520
hidden layer 8 had mean -0.000448 and std 0.981913
hidden layer 9 had mean -0.000899 and std 0.981728
hidden layer 10 had mean 0.000584 and std 0.981736
```

Neurons are saturated, making gradients near-zero!



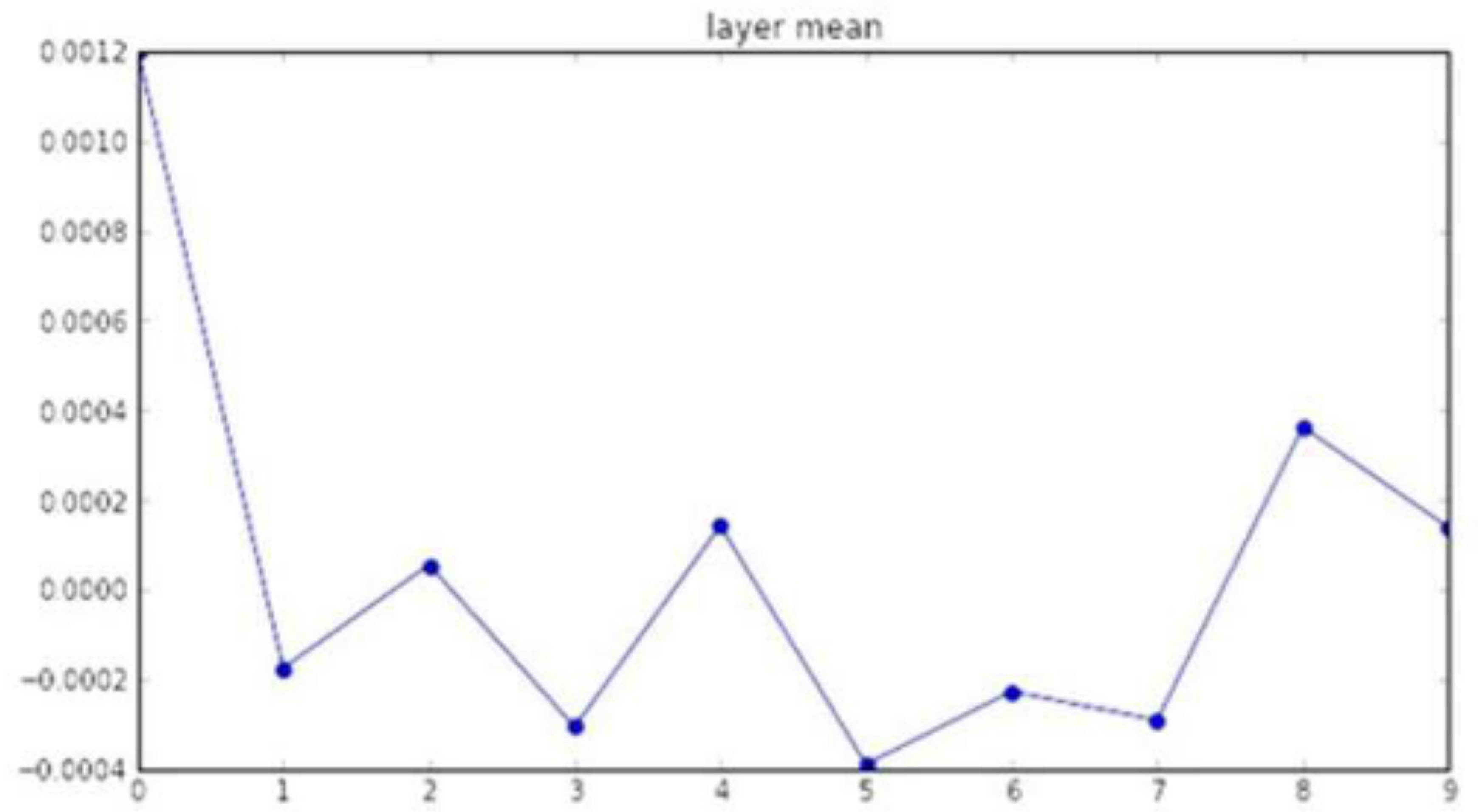


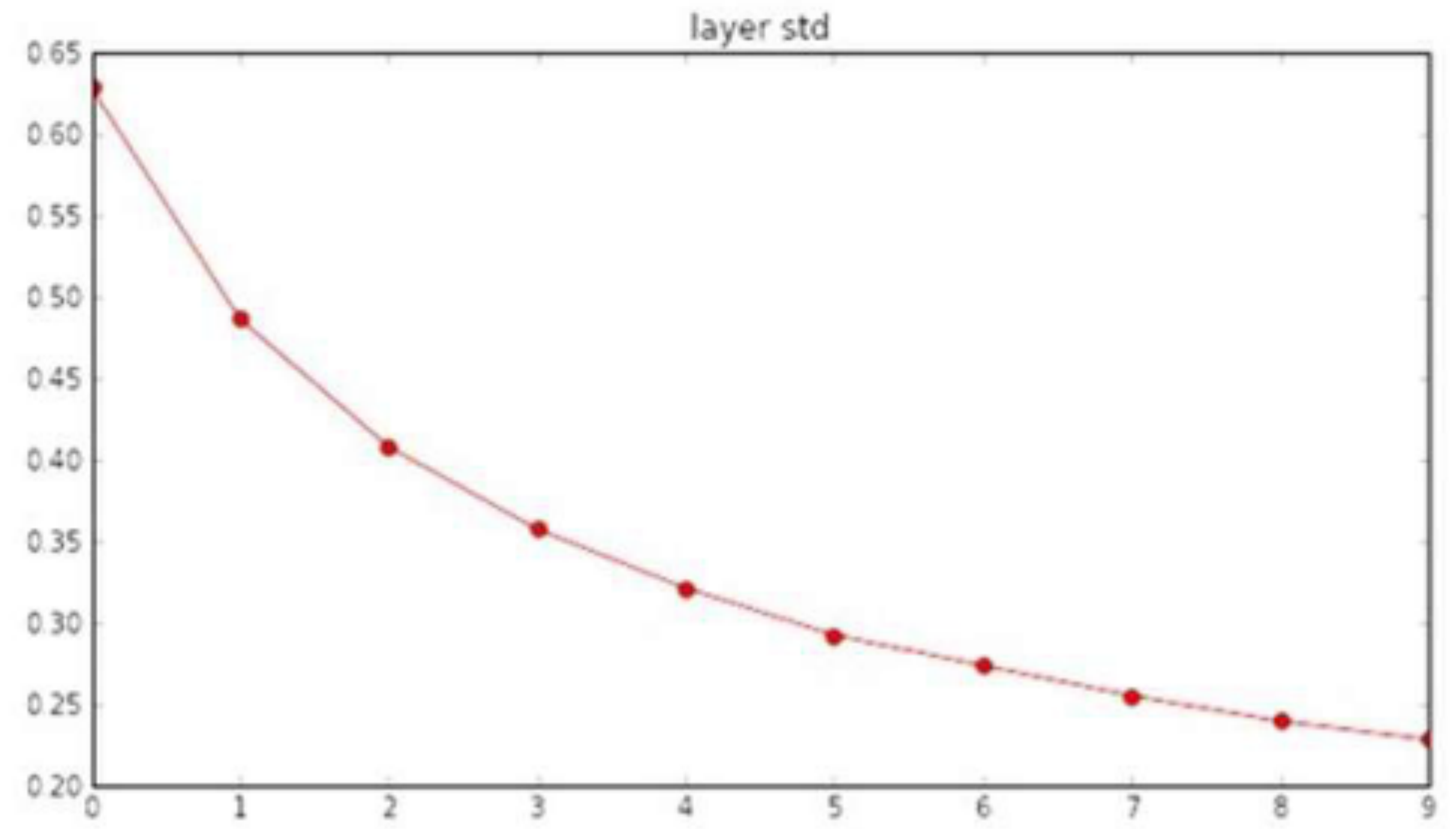
```
input layer had mean 0.001800 and std 1.001311
hidden layer 1 had mean 0.001198 and std 0.627953
hidden layer 2 had mean -0.000175 and std 0.486051
hidden layer 3 had mean 0.000055 and std 0.407723
hidden layer 4 had mean -0.000306 and std 0.357108
hidden layer 5 had mean 0.000142 and std 0.320917
hidden layer 6 had mean -0.000389 and std 0.292116
hidden layer 7 had mean -0.000228 and std 0.273387
hidden layer 8 had mean -0.000291 and std 0.254935
hidden layer 9 had mean 0.000361 and std 0.239266
hidden layer 10 had mean 0.000139 and std 0.228008
```

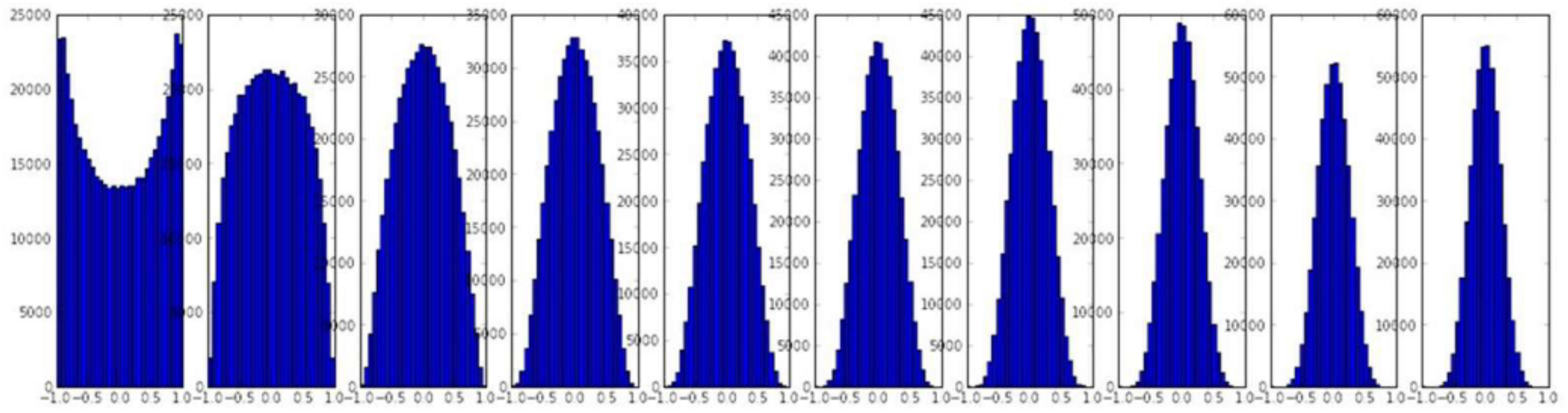
Xavier Init. Draw weights from $N(0,\sigma^2)$,

where $$\sigma = \sqrt{\frac{2}{\text{(input dim)} + \text{(output dim)}}}$$

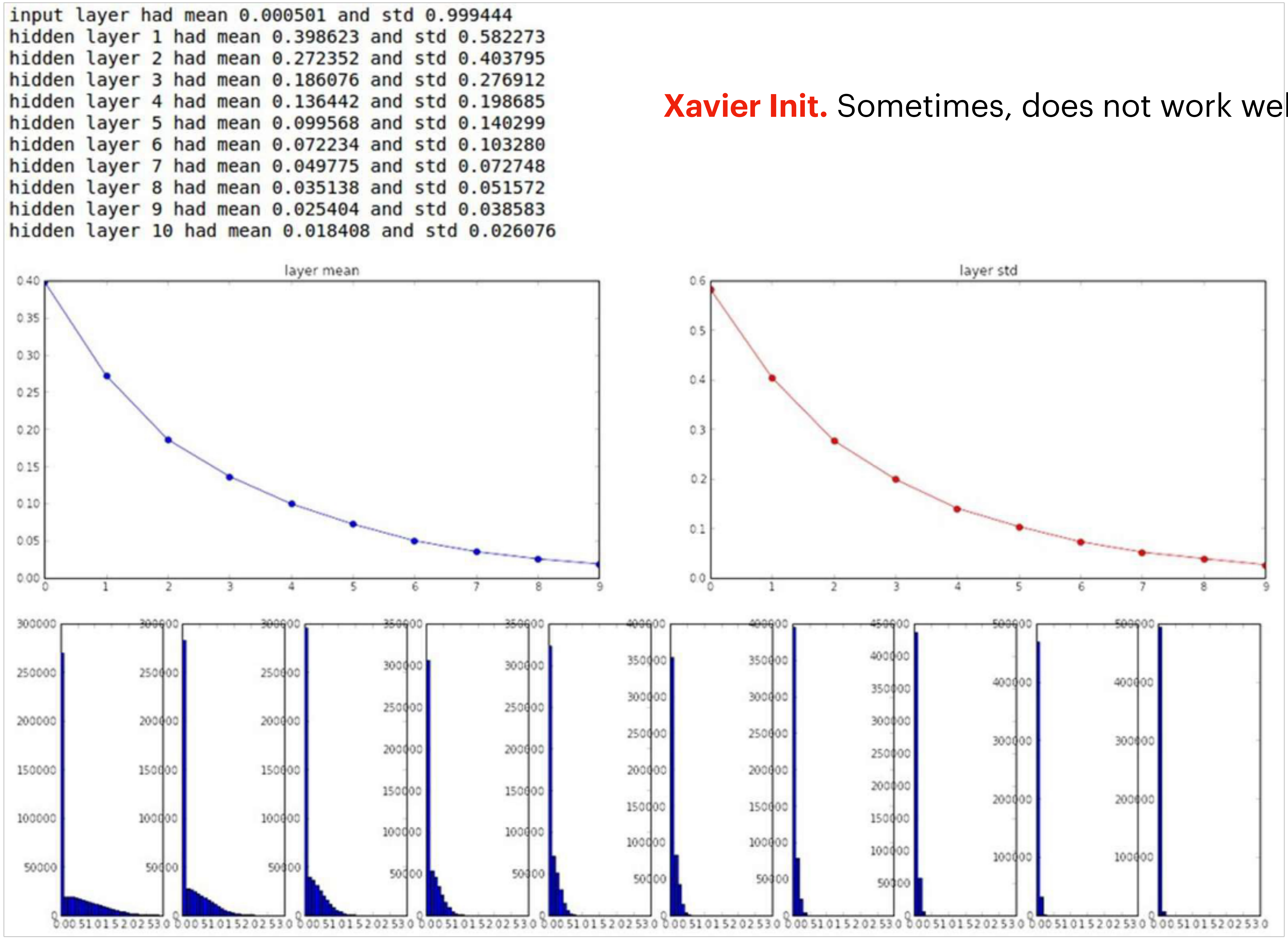
(assumes linear activation for mathematical derivation)







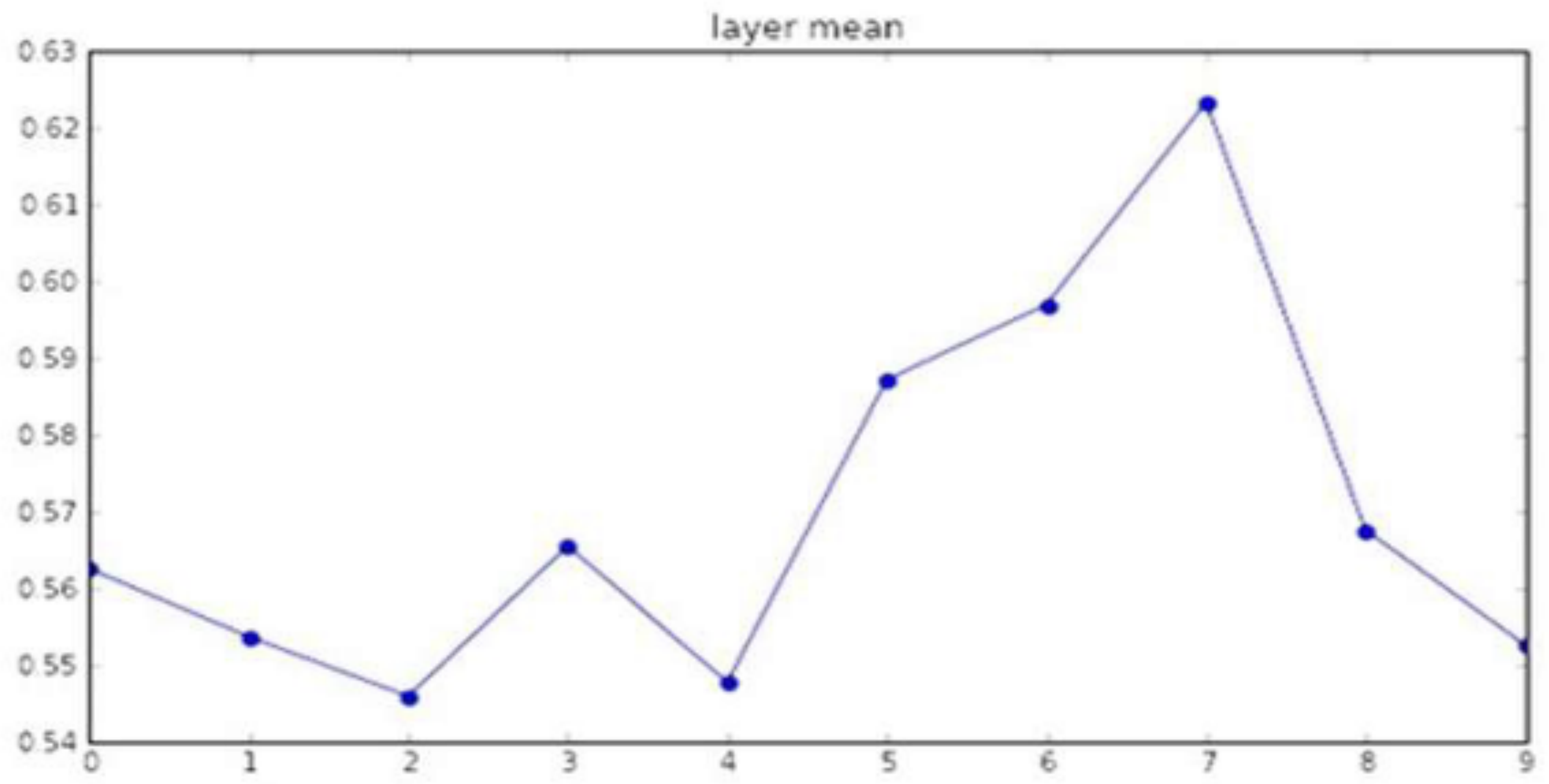
Xavier Init. Sometimes, does not work well with ReLU!

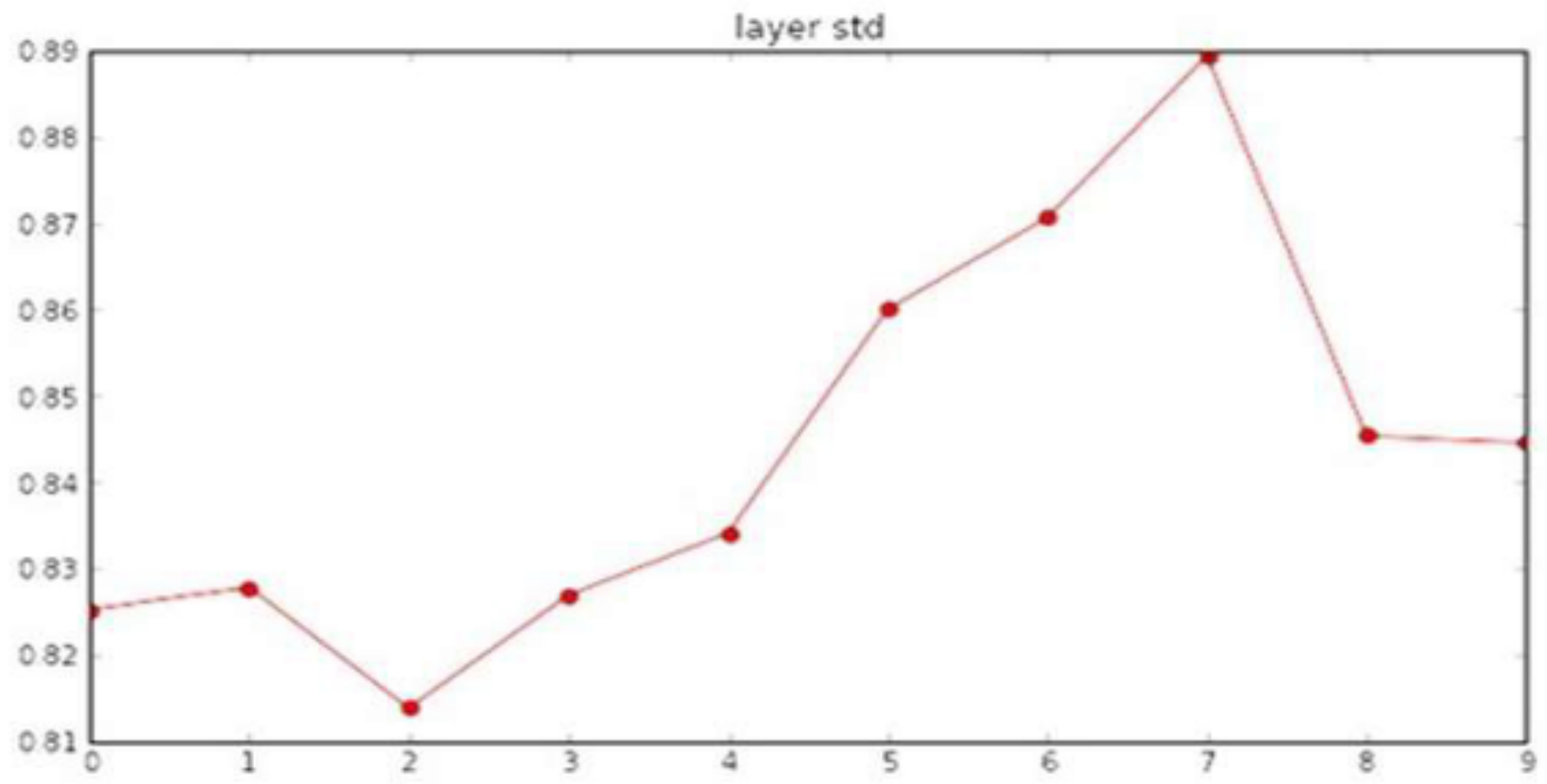


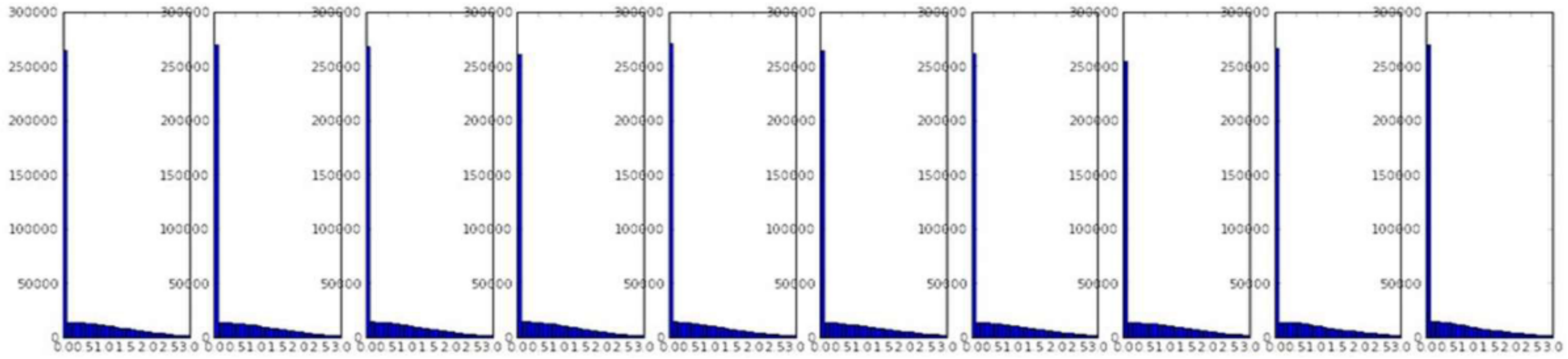
```
input layer had mean 0.000501 and std 0.999444
hidden layer 1 had mean 0.398623 and std 0.582273
hidden layer 2 had mean 0.272352 and std 0.403795
hidden layer 3 had mean 0.186076 and std 0.276912
hidden layer 4 had mean 0.136442 and std 0.198685
hidden layer 5 had mean 0.099568 and std 0.140299
hidden layer 6 had mean 0.072234 and std 0.103280
hidden layer 7 had mean 0.049775 and std 0.072748
hidden layer 8 had mean 0.035138 and std 0.051572
hidden layer 9 had mean 0.025404 and std 0.038583
hidden layer 10 had mean 0.018408 and std 0.026076
```

He init. Draw weights from $\sigma = \sqrt{\dfrac{2}{\text{(input dim)}}}$

```
input layer had mean 0.000501 and std 0.999444
hidden layer 1 had mean 0.562488 and std 0.825232
hidden layer 2 had mean 0.553614 and std 0.827835
hidden layer 3 had mean 0.545867 and std 0.813855
hidden layer 4 had mean 0.565396 and std 0.826902
hidden layer 5 had mean 0.547678 and std 0.834092
hidden layer 6 had mean 0.587103 and std 0.860035
hidden layer 7 had mean 0.596867 and std 0.870610
hidden layer 8 had mean 0.623214 and std 0.889348
hidden layer 9 had mean 0.567498 and std 0.845357
hidden layer 10 had mean 0.552531 and std 0.844523
```







Remarks

- There are many research on how to initialize
  - Has been mostly okay with BNs, but BNs are getting faded away…
  - Many unmentioned:
    - Orthogonal initialization
    - Identity initialization
    - Zero initialization…

Cheers

- *Next up.* Part 2