Activations, Problematic Gradients, Initialization, and Normalization

We want all values to be around zero. [-1, 1]

Outline

- Activations (output of an activation function)
- Problematic gradients (exploding and vanishing)
- The benefits of depth in neural networks
- Proper parameter initialization
- Normalization (preprocessing) of inputs and activations (a type of layer)

Recap: Optimization Techniques

- Take five minutes to draw

Purpose of Assignments

I provide most (if not all code) on assignments for 2 reasons:

$$Z = AW^T + b$$

1. This provides a nice set of working examples you can use for projects

2. It keeps assignments shorter so that you have more time for projects

You should end up spending more time reading my code and more time writing your own code.

Affine ~ linear + bias

Activation Functions, Two-Layer Network

$A^{[2]} = A^{[0]} m + b$ (Linear)

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

Without Activation Functions

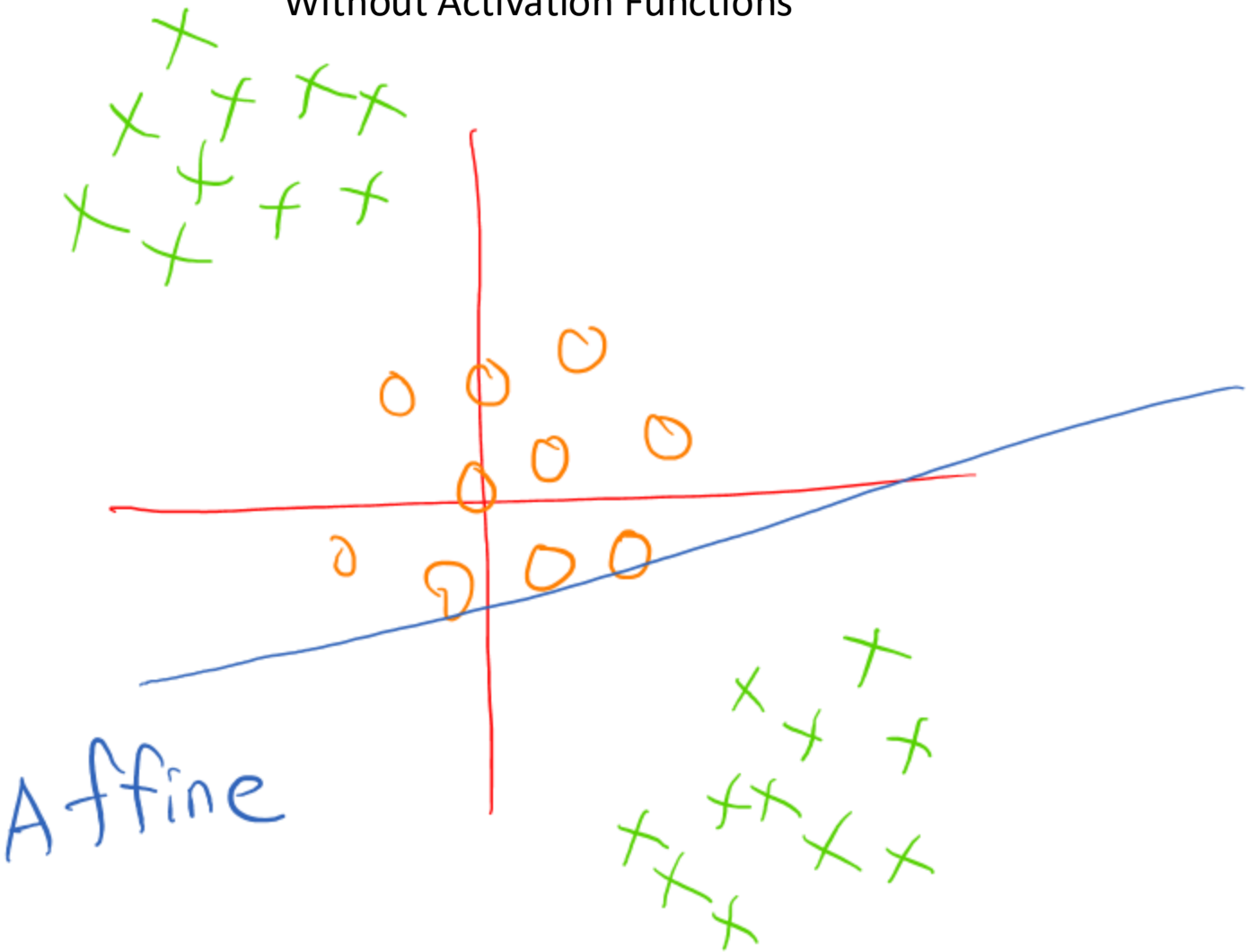
$$Z^{[1]} = A^{[0]} W^{[1]T} + b^{[1]}$$
$$A^{[1]} = Z^{[1]}$$
$$Z^{[2]} = A^{[1]} W^{[2]T} + b^{[2]}$$
$$A^{[2]} = Z^{[2]}$$
$$= A^{[1]} W^{[2]T} + b^{[2]}$$
$$= Z^{[1]} W^{[2]T} + b^{[2]}$$
$$= \left(A^{[0]} W^{[1]T} + b^{[1]}\right) W^{[2]T} +$$
$$= A^{[0]} W^{[1]T} W^{[2]T} + b^{[1]} W^{[2]T} + b^{[2]}$$

With Activation Functions

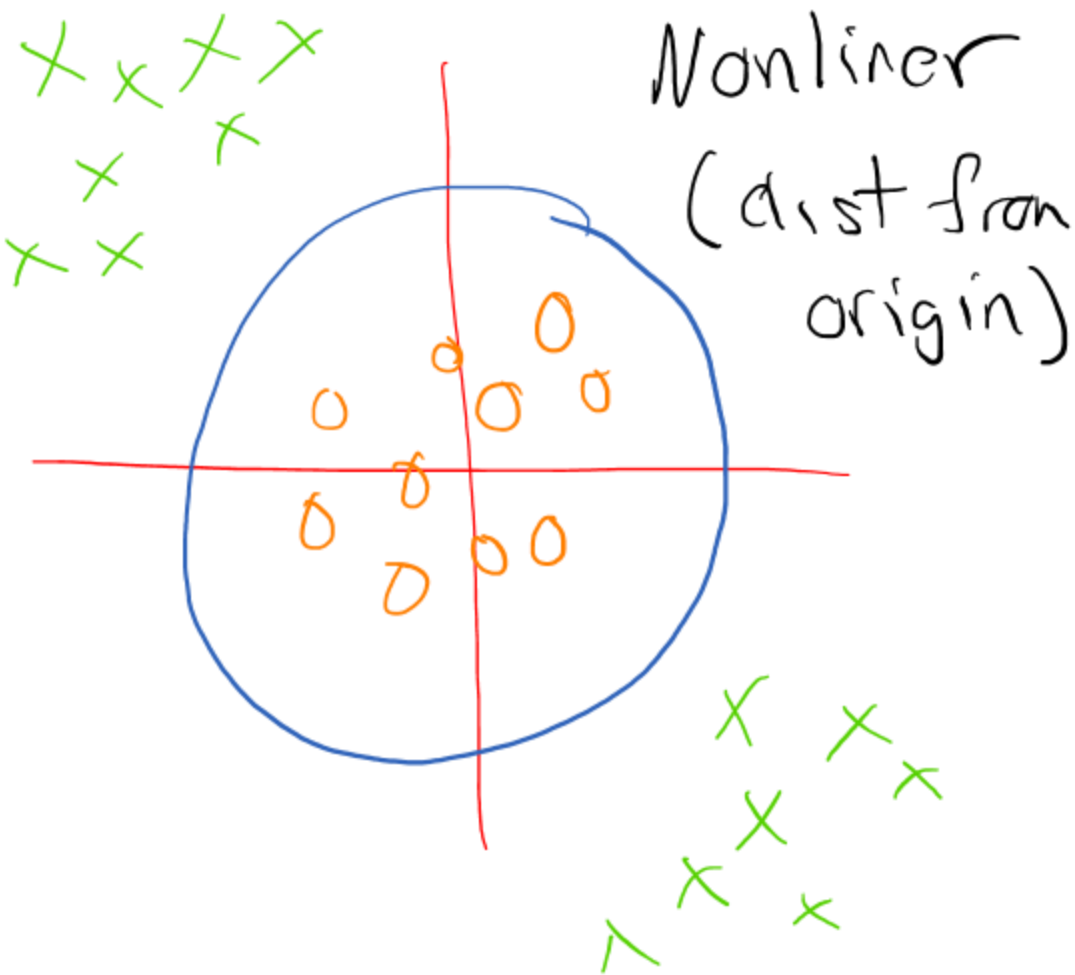
Non-Linear

$$Z^{[1]} = A^{[0]} W^{[1]T} + b^{[1]}$$
$$A^{[1]} = \sigma(Z^{[1]})$$
$$Z^{[2]} = A^{[1]} W^{[2]T} + b^{[2]}$$
$$A^{[2]} = \sigma(Z^{[2]})$$
$$= \sigma\left(A^{[1]} W^{[2]T} + b^{[2]}\right)$$
$$= \sigma\left(\sigma(Z^{[1]}) W^{[2]T} + b^{[2]}\right)$$
$$= \sigma\left(\sigma\left(A^{[0]} W^{[1]T} + b^{[1]}\right) \cdot W^{[2]T} + b^{[2]}\right)$$

Without Activation Functions



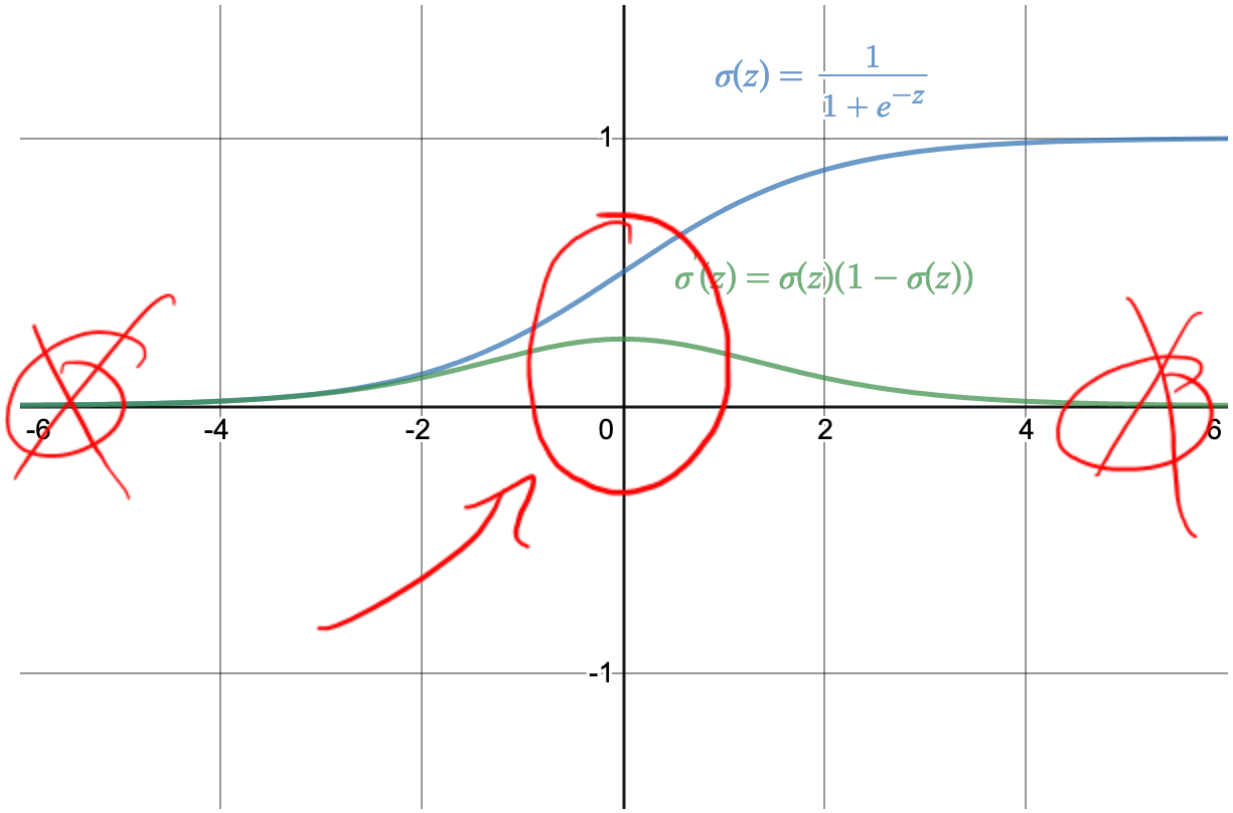
With Activation Functions



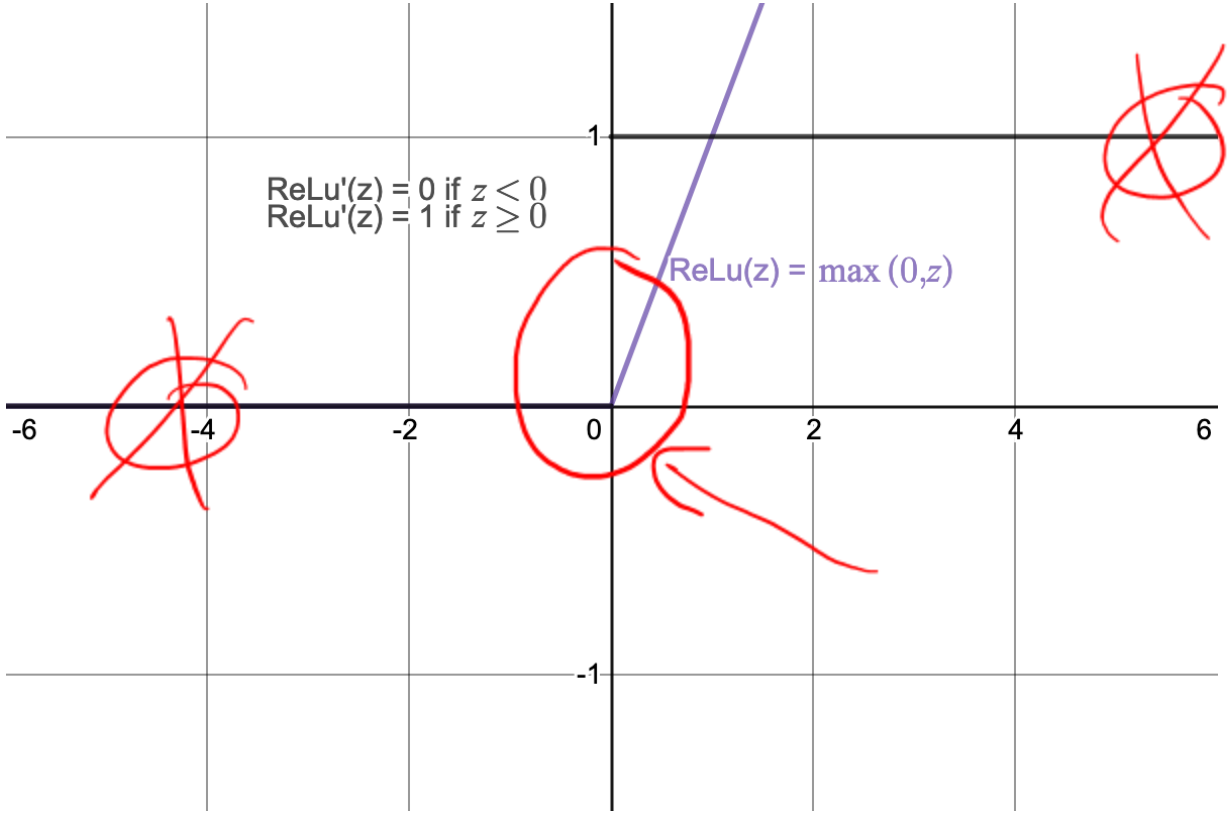
Sigmoid and ReLU Activation Functions

Most interesting around 0 : [-1, 1]

Sigmoid



ReLU



Inputs in [-1, 1] range

Problematic Gradients, Three-Layer Network

$$\frac{\partial L}{\partial W^{[1]}} = (A^{[3]} - Y)A^{[3]}(1 - A^{[3]})W^{[3]}A^{[2]}(1 - A^{[2]})W^{[2]}A^{[1]}(1 - A^{[1]})X$$

$$A^{[1]} = g(Z^{[1]})$$

Within ½ MSE and Sigmoids

$$Z^{[1]} = A^{[0]}W^{[1]T} + b^{[1]}$$

Exploding

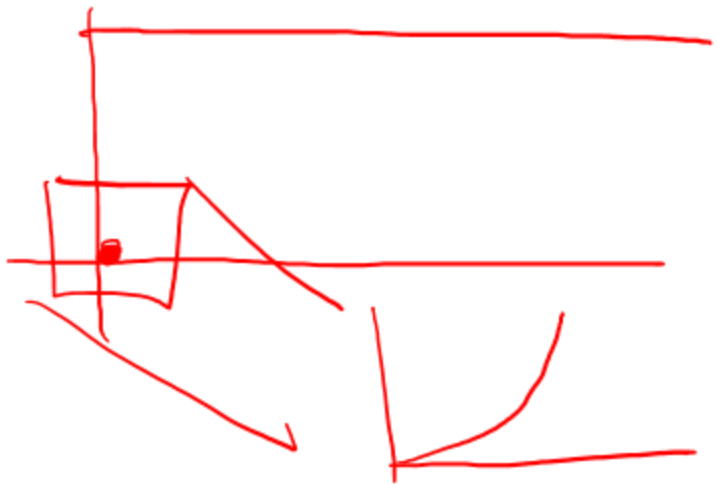
$$W^{[1]} := W^{[1]} - \eta \frac{\partial L}{\partial W^{[1]}}$$

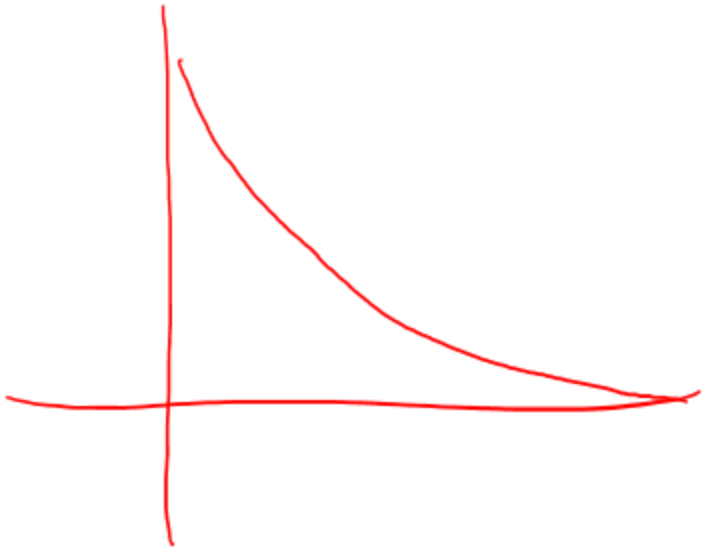
Vanishing

$$10 := 10 - 0.1 \cdot 1000$$

$$-90 := -90 - 0.1 \cdot 2300$$

$$0.01 := 0.01 - 0.1 \cdot 0.1$$

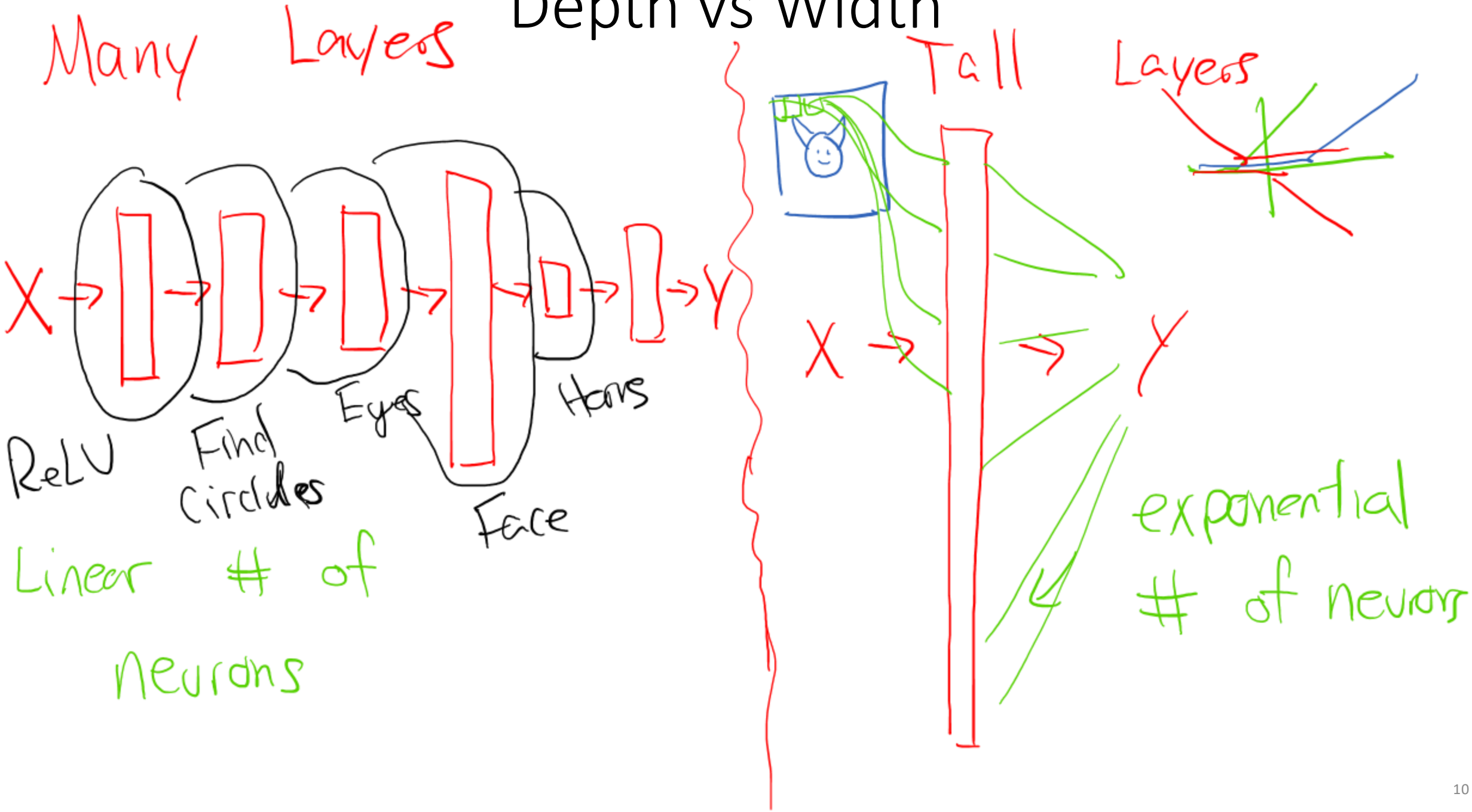




Fixes for Problematic Gradients

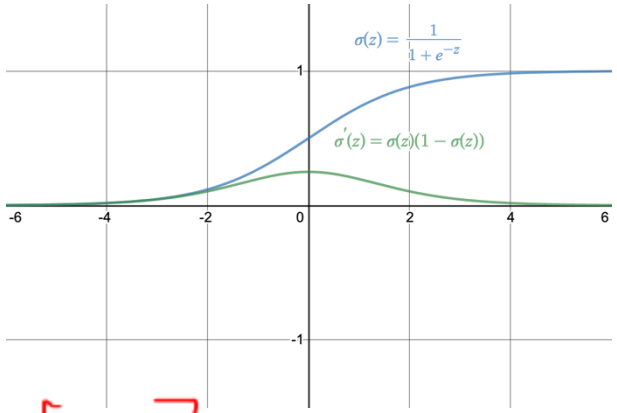
- Shallower networks
  - ↳ bad results
- Gradient clipping
  - ↳ not used in practice
- Clever initialization
  - ↳ easy + very effective
- Batch normalization
  - ↳ works pretty well for most problems
- Better activation functions
  - ↳ use ReLU over sigmoid
- Residual connections
  - ↳ we'll talk about them in a few weeks

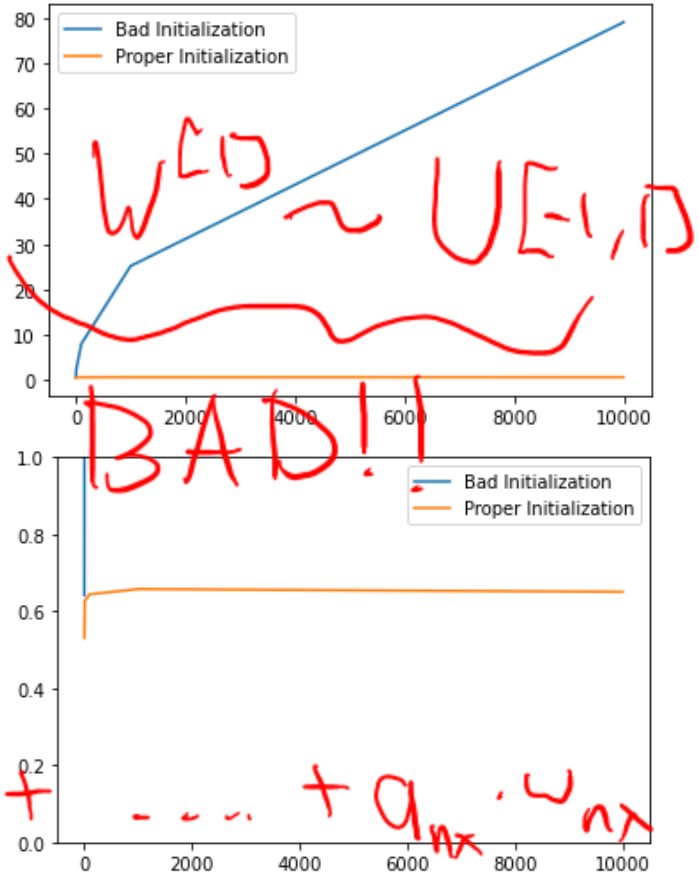
Depth vs Width



Proper Initialization

- Activation functions produce “useful” output around 0
- $Z^{[1]} = A^{[0]}W^{[1]T} + b^{[1]}$
- $A^{[1]} = \sigma(Z^{[1]})$





PyTorch Kaiming He Initialization

$W = \text{torch.randn}(10, 5) \cdot 0.01$

torch.nn.init.kaiming_uniform_(*tensor*, *a=0*, *mode='fan_in'*, *nonlinearity='leaky_relu'*) [SOURCE]

Fills the input *Tensor* with values according to the method described in *Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification* - He, K. et al. (2015), using a uniform distribution. The resulting tensor will have values sampled from $\mathcal{U}(-\text{bound}, \text{bound})$ where

$$\text{bound} = \text{gain} \times \sqrt{\frac{3}{\text{fan\_mode}}}$$

```
gain = calculate_gain(nonlinearity, a)
std = gain / math.sqrt(fan)
bound = math.sqrt(3.0) * std  # Calculate uniform bounds from standard deviation
with torch.no_grad():
    return tensor.uniform_(-bound, bound)
```

Batch Normalization

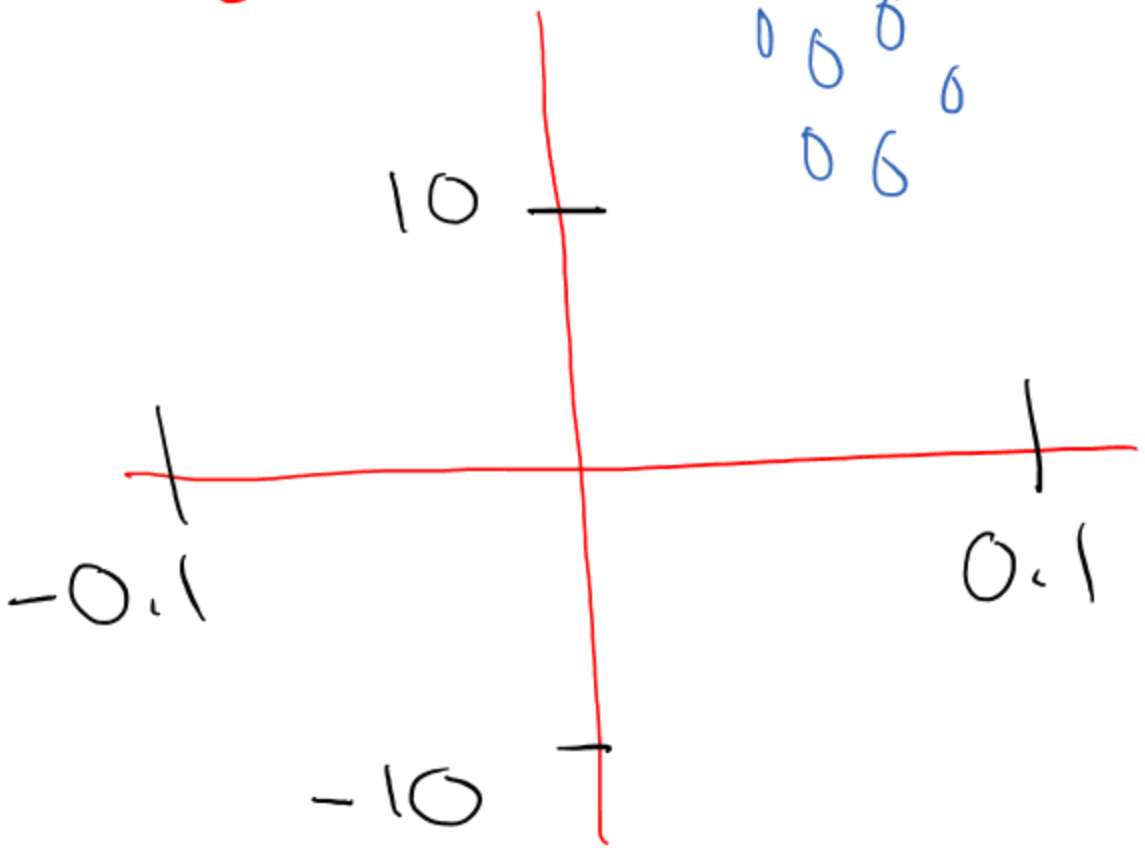
- What is normalization?
  - Adjusting values to a different (common) scale.
  - For example, the standard-score normalization: $\bar{x} = \frac{x-\mu}{\sigma}$
- Normalize with respect to what?

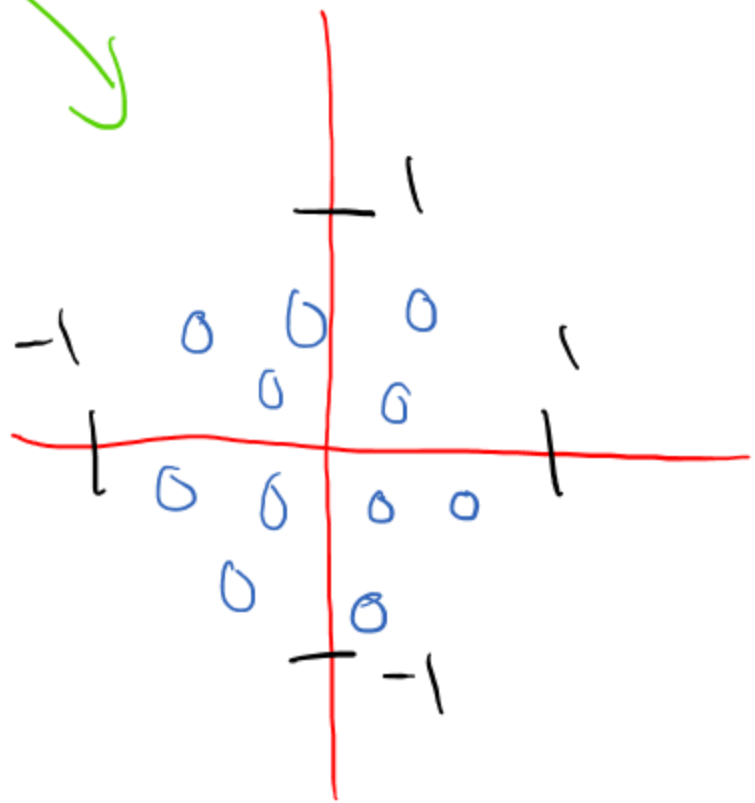
Batch Normalization

$$z^{[1]} = A^{[0]} W^{T} + b^{[1]}$$

$$A^{[1]} = g(z^{[1]})$$

Batch Normalization





Summary

- Activation functions behave nicely with inputs around zero
- Gradients behave nicely with values around zero (but not too small)
- Depth is good for extracting/finding complex features
- You should
  - Normalize input features
  - Initialize parameters properly
  - Prefer deeper over wider networks
  - Try/consider batch normalization