Activations, Problematic Gradients, Initialization, and Normalization

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:

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.

Activation Functions, Two-Layer Network

Without Activation Functions

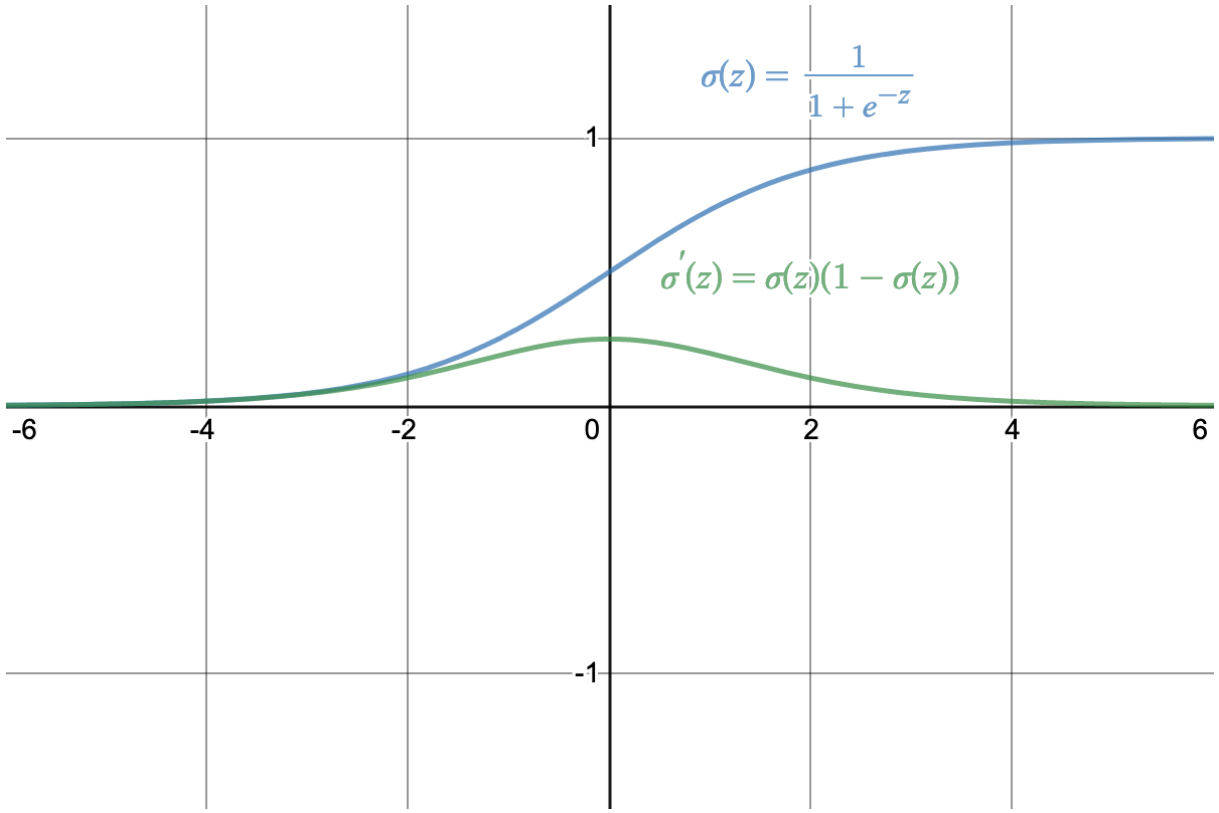
$$Z^{[1]} = A^{[0]}W^{[1]T} + b^{[1]}$$
$$A^{[1]} = Z^{[1]}$$
$$Z^{[2]} = A^{[1]}W^{[2]T} + b^{[2]}$$
$$A^{[2]} = Z^{[2]}$$

With Activation Functions

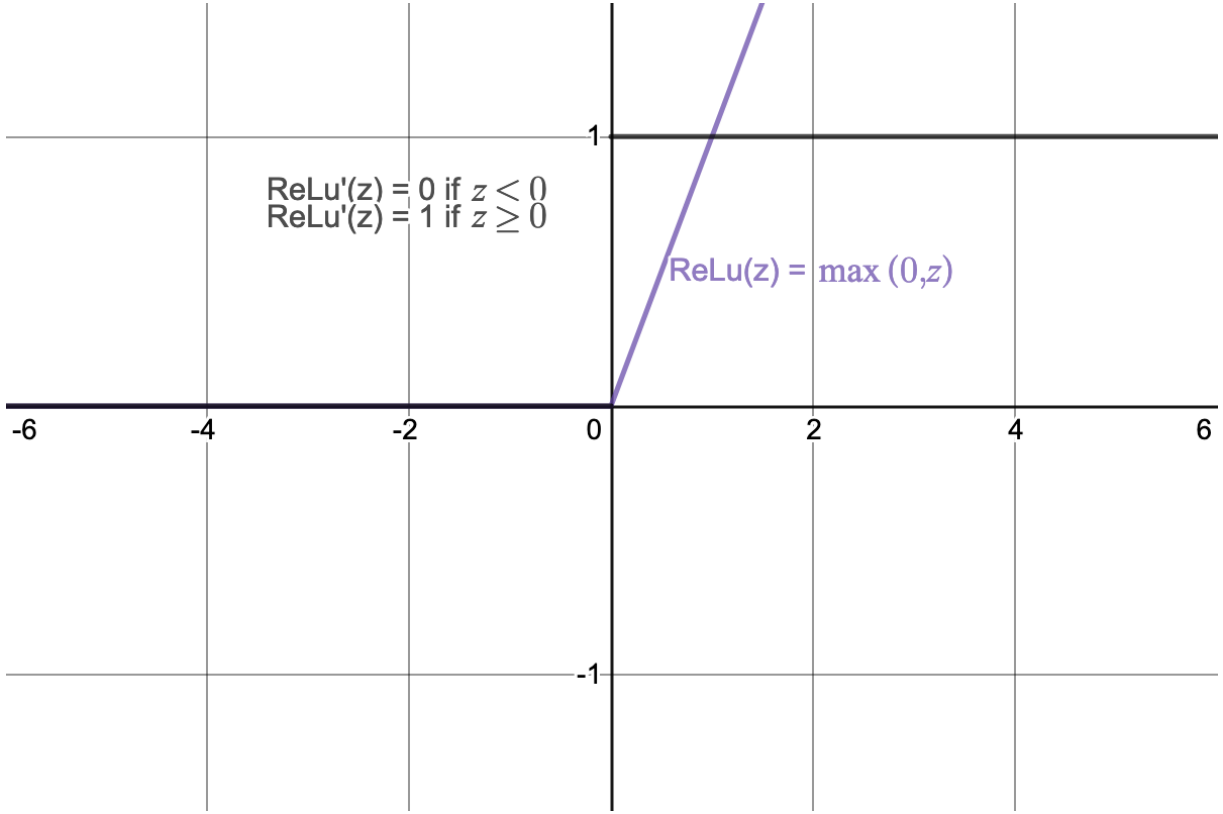
$$Z^{[1]} = A^{[0]}W^{[1]T} + b^{[1]}$$
$$A^{[1]} = \sigma\left(Z^{[1]}\right)$$
$$Z^{[2]} = A^{[1]}W^{[2]T} + b^{[2]}$$
$$A^{[2]} = \sigma(Z^{[2]})$$

Sigmoid and ReLU Activation Functions

Sigmoid



ReLU



Problematic Gradients, Three-Layer Network

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

Exploding

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

Vanishing

Within ½ MSE and Sigmoids

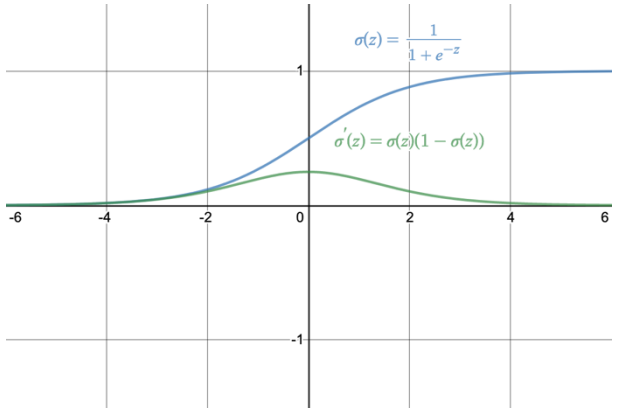
Fixes for Problematic Gradients

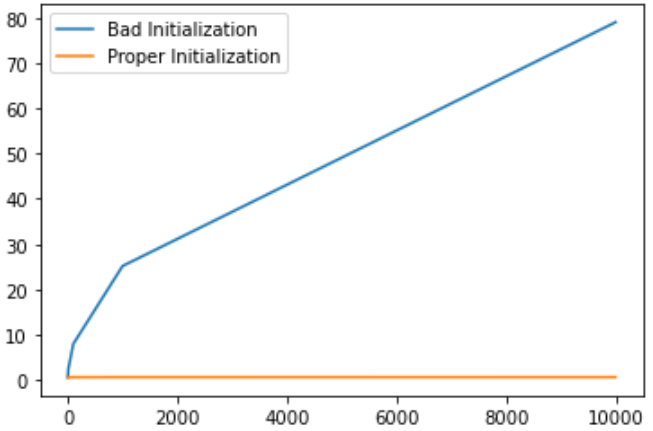
- Shallower networks
- Gradient clipping
- Clever initialization
- Batch normalization
- Better activation functions
- Residual connections

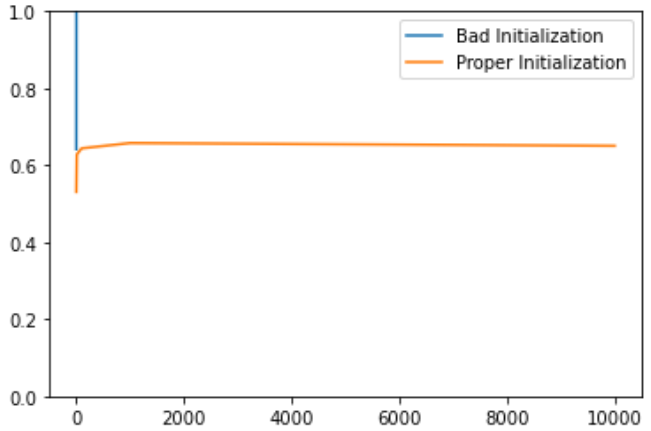
Depth vs Width

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

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

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