Weight Initialization and Training Stability

Idant Srivastava

December 2025

Contents

1 Weight Initialization **2**
1.1 Concept 2

2 Xavier (Glorot) Initialization **3**
2.1 Concept 3
2.2 Mathematical Foundation 3

3 He Initialization **4**
3.1 Concept 4
3.2 Mathematical Foundation 4

4 Activation Functions and Training Stability **5**
4.1 ReLU 5
4.2 Dying ReLU Problem 5

5 Gradient Clipping **7**
5.1 Concept 7
5.2 Mathematical Foundation 7

6 Softmax **8**
6.1 Concept 8
6.2 Numerical Stability 8

Chapter 1

Weight Initialization

1.1 Concept

Weight initialization determines the starting point of optimization. If weights are initialized improperly:

- Very small weights lead to vanishing activations and gradients.
- Very large weights lead to exploding activations and gradients.
- Poor variance propagation across layers slows downs the learning.

The central goal of initialization schemes is to preserve the variance of activations and gradients across layers, ensuring stable forward and backward propagation.

For a layer with n_{in} inputs and n_{out} outputs, we want to maintain the variance of activations across layers. The variance of the output is:

$$Var(y) = n_{in} Var(w) Var(x)$$

To maintain $ar(y) = Var(x)$, we need: $Var(w) = \frac{1}{n_{in}}$

Chapter 2

Xavier (Glorot) Initialization

2.1 Concept

Xavier initialization is designed for symmetric activation functions such as tanh or sigmoid, where outputs are roughly centered around zero. It balances variance between both forward and backward passes.

2.2 Mathematical Foundation

Weights are sampled from:

- Uniform:
$$W \sim \mathcal{U}\left(-\sqrt{\frac{6}{n_{in}+n_{out}}}, \sqrt{\frac{6}{n_{in}+n_{out}}}\right)$$
- Normal:
$$W \sim \mathcal{N}\left(0, \frac{2}{n_{in}+n_{out}}\right)$$

where:
n_{in}: number of input neurons
n_{out}: number of output neurons

Chapter 3

He Initialization

3.1 Concept

He initialization is specifically designed for ReLU-based networks, where approximately half of the activations are zero due to the rectification operation. He initialization sets the variance of weights such that the expected variance of activations remains constant across layers when using ReLU.

3.2 Mathematical Foundation

Weights are sampled as:

$$W \sim \mathcal{N}\left(0, \frac{2}{n_{in}}\right)$$

$$W \sim \mathcal{U}\left(-\sqrt{\frac{6}{n_{in}}}, \sqrt{\frac{6}{n_{in}}}\right)$$

Chapter 4

Activation Functions and Training Stability

4.1 ReLU

$$\text{ReLU}(z) = \max(0, z)$$

Advantages

- Avoids saturation
- Efficient gradient propagation

Derivative

$$\frac{d}{dz}\text{ReLU}(z) = \begin{cases} 1 & z > 0 \\ 0 & z \leq 0 \end{cases}$$

4.2 Dying ReLU Problem

The dying ReLU problem refers to a failure in neural networks that use the ReLU activation function, where a significant number of neurons become permanently inactive during training.

If a neuron's input (z) becomes negative for all inputs, its output is always zero. the neuron receives no gradient updates during backpropagation. As a result, its weights stop updating, and the neuron effectively "dies."

This problem is commonly caused by:

- Poor weight initialization
- Large learning rates causing weights to shift into negative regions
- Deep networks without normalization

Dying ReLU reduces model capacity and can significantly degrade performance. Common remedies include:

- He initialization
- Leaky ReLU
- Smaller learning rates
  Leaky ReLU:

$$\text{LeakyReLU(z)} = max(\alpha z, z), \alpha \ll 1$$

Chapter 5

Gradient Clipping

5.1 Concept

Gradient clipping is a training stabilization technique used in neural networks to prevent exploding gradients during backpropagation. When gradients become excessively large, they can cause unstable parameter updates and lead to divergence of the training process. Gradient clipping addresses this issue by limiting the magnitude of gradients, either by capping individual gradient values or by rescaling the entire gradient vector when its norm exceeds a predefined threshold. By constraining gradient updates, gradient clipping enables more stable and reliable training, particularly in deep networks and recurrent neural networks.

5.2 Mathematical Foundation

Given gradient vector (g):

$$g \leftarrow g \cdot \text{clip}(g, -c, c)$$

where c is a predefined threshold.

This prevents large parameter updates while preserving gradient direction.

Chapter 6

Softmax

6.1 Concept

Softmax is an activation function commonly used in the output layer of multi-class classification neural networks. It converts a vector of raw scores (logits) into a probability distribution by exponentiating each score and normalizing by the sum of exponentials, ensuring that all output values lie between 0 and 1 and sum to 1. This probabilistic interpretation allows the model to represent confidence over multiple classes and is typically paired with categorical cross-entropy loss for stable and efficient training.

6.2 Numerical Stability

For multi-class classification:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}$$

To avoid overflow:

$$\text{softmax}(z_i) = \frac{e^{z_i - \max(z)}}{\sum_{j} e^{z_j - \max(z)}}$$

Conclusion

Weight initialization and training stability are foundational to successful MLP training. Xavier and He initialization schemes ensure controlled variance propagation, while appropriate activation functions such as ReLU enable efficient gradient flow. Addressing issues like dying ReLU, exploding gradients, and numerical instability through gradient clipping and stable softmax formulations further enhances convergence. Together, these techniques form the backbone of modern deep learning practice.