Weight Initialization and Training Stability in Multi-Layer Perceptrons

Vaibhav Chourasia

December 16, 2025

1 Introduction

Training a neural network is fundamentally a problem of numerical optimization. However, unlike classical optimization problems, deep neural networks suffer from severe instability during training. Two of the most common reasons are:

- improper weight initialization,
- uncontrolled gradient propagation.

If a network is initialized poorly, learning may fail completely—even if the model architecture and dataset are correct.

2 Why Training Becomes Unstable in Deep Networks

Before discussing initialization methods, it is crucial to understand why deep networks are unstable.

2.1 The Basic Neuron Model

A neuron computes:

$$z = \sum_{i=1}^{n} w_i x_i + b$$

$$a = \phi(z)$$

where:

- x_i are inputs,

- w_i are weights,
- b is bias,
- $\phi(\cdot)$ is an activation function.

In a deep network, this computation is repeated across many layers.

2.2 What Goes Wrong as Depth Increases

Each layer transforms its input and passes it forward. If these transformations:

- shrink signals too much $\rightarrow$ information vanishes,
- amplify signals too much $\rightarrow$ values explode,

then learning becomes numerically impossible.

This instability appears both:

- in the forward pass (activations),
- in the backward pass (gradients).

1.3 Why Initialization Matters

Weight initialization determines:

- the scale of activations,
- the scale of gradients,
- whether gradients can flow backward.

Poor initialization causes:

- vanishing gradients,
- exploding gradients,
- dead neurons,
- extremely slow convergence.

Thus, initialization is not a minor detail,it is foundational.

3 Variance Propagation in Neural Networks

To understand good initialization, we study how variance propagates through layers.

3.1 Assumptions

Assume:

- inputs x_i are independent,
- $\mathbb{E}[x_i] = 0$,
- $\mathrm{Var}(x_i) = \sigma_x^2$,
- weights w_i are independent,
- $\mathbb{E}[w_i] = 0$,
- $\mathrm{Var}(w_i) = \sigma_w^2$.

These assumptions are reasonable at initialization.

3.2 Variance of Pre-Activation

The pre-activation is:

$$z = \sum_{i=1}^{n} w_i x_i$$

Since the terms are independent:

$$\mathrm{Var}(z) = \sum_{i=1}^{n} \mathrm{Var}(w_i x_i)$$

Using:

$$\mathrm{Var}(w_i x_i) = \mathrm{Var}(w_i)\mathrm{Var}(x_i)$$

we get:

$$\mathrm{Var}(z) = n\sigma_w^2 \sigma_x^2$$

3.3 Desired Stability Condition

For stable forward propagation, we want:

$$\mathrm{Var}(z) \approx \mathrm{Var}(x)$$

Thus:

$$n\sigma_w^2\sigma_x^2 \approx \sigma_x^2$$

Canceling σ_x^2:

$$n\sigma_w^2 \approx 1$$

$$\Rightarrow \sigma_w^2 \approx \frac{1}{n}$$

This simple equation is the foundation of all modern initialization methods.

4 Weight Initialization: General Principle

4.1 What Initialization Tries to Achieve

Good initialization aims to:

- preserve variance across layers,
- keep gradients stable during backpropagation,
- prevent early saturation of activation functions.

4.2 Forward vs Backward Stability

There are two constraints:

- forward pass: activations should not explode or vanish,
- backward pass: gradients should not explode or vanish.

Good initialization balances both.

4.3 Why Random Initialization Is Necessary

If all weights are initialized to zero:

- all neurons compute the same output,
- gradients are identical,
- symmetry is never broken.

Random initialization breaks symmetry and allows different neurons to learn different features.

5 Xavier (Glorot) Initialization

Xavier initialization was proposed to address instability when using sigmoid or tanh activation functions in deep networks.

5.1 Problem with Sigmoid and Tanh

Sigmoid and tanh functions saturate for large positive or negative inputs:

$$\sigma(z) \to 1 \text{ or } 0, \quad \tanh(z) \to \pm 1$$

Their derivatives approach zero in saturation regions:

$$\sigma'(z) = \sigma(z)(1 - \sigma(z)) \leq 0.25$$
$$\tanh'(z) = 1 - \tanh^2(z) \leq 1$$

If activations saturate early, gradients vanish during backpropagation.

Thus, we want:

- activations centered around zero,
- variance preserved across layers.

5.2 Forward Variance Constraint

From earlier derivation:

$$\text{Var}(z^{(l)}) = n_{\text{in}} \sigma_w^2 \text{Var}(a^{(l-1)})$$

To keep:

$$\text{Var}(z^{(l)}) \approx \text{Var}(a^{(l-1)})$$

we require:

$$n_{\text{in}} \sigma_w^2 = 1$$

5.3 Backward Variance Constraint

During backpropagation:

$$\delta^{(l-1)} = (W^{(l)})^T \delta^{(l)} \odot \phi'(z^{(l-1)})$$

Ignoring activation derivatives temporarily:

$$\text{Var}(\delta^{(l-1)}) = n_{\text{out}} \sigma_w^2 \text{Var}(\delta^{(l)})$$

To preserve gradient variance:

$$n_{\text{out}} \sigma_w^2 = 1$$

5.4 Balancing Forward and Backward Conditions

We now have two conditions:

$$\sigma_w^2 = \frac{1}{n_{\text{in}}}, \quad \sigma_w^2 = \frac{1}{n_{\text{out}}}$$

Xavier proposed averaging them:

$$\sigma_w^2 = \frac{2}{n_{\text{in}} + n_{\text{out}}}$$

5.5 Xavier Initialization Formula

Weights are sampled as:

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

or uniformly:

$$W \sim \text{Uniform}\left(-\sqrt{\frac{6}{n_{\text{in}} + n_{\text{out}}}}, \sqrt{\frac{6}{n_{\text{in}} + n_{\text{out}}}}\right)$$

5.6 When to Use Xavier Initialization

Xavier works best for:

- sigmoid activation,
- tanh activation,
- shallow or moderately deep networks.

However, it is not ideal for ReLU networks.

6 He (Kaiming) Initialization

He initialization was designed specifically for ReLU-based networks.

6.1 Why Xavier Fails for ReLU

ReLU activation:

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

For zero-mean inputs, approximately half the activations are zero.

Thus:

$$\mathrm{Var}(a^{(l)}) = \frac{1}{2}\mathrm{Var}(z^{(l)})$$

This variance drop accumulates across layers if not corrected.

6.2 Forward Variance Derivation for ReLU

From variance propagation:

$$\mathrm{Var}(z^{(l)}) = n_{\text{in}}\sigma_w^2\mathrm{Var}(a^{(l-1)})$$

After ReLU:

$$\mathrm{Var}(a^{(l)}) = \frac{1}{2}n_{\text{in}}\sigma_w^2\mathrm{Var}(a^{(l-1)})$$

For stability:

$$\frac{1}{2}n_{\text{in}}\sigma_w^2 = 1$$

$$\Rightarrow \sigma_w^2 = \frac{2}{n_{\text{in}}}$$

6.3 He Initialization Formula

Weights are initialized as:

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

or:

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

6.4 Why He Initialization Works

He initialization:

- compensates for ReLU zeroing,
- preserves activation variance,
- stabilizes gradients,
- enables very deep networks.

6.5 Summary: Xavier vs He

Activation	Initialization	Variance
Sigmoid	Xavier	$\frac{2}{n_{\text{in}}+n_{\text{out}}}$
Tanh	Xavier	$\frac{2}{n_{\text{in}}+n_{\text{out}}}$
ReLU	He	$\frac{2}{n_{\text{in}}}$

7 Vanishing Gradient Problem

7.1 Origin of the Problem

Backpropagation computes gradients using repeated application of the chain rule.

For a deep network with L layers:

$$\frac{\partial L}{\partial W^{(l)}} = \frac{\partial L}{\partial a^{(L)}} \prod_{k=l+1}^{L} \frac{\partial a^{(k)}}{\partial z^{(k)}} \frac{\partial z^{(k)}}{\partial a^{(k-1)}}$$

This is a product of many terms.

7.2 Role of Activation Derivatives

For sigmoid activation:

$$\sigma'(z) = \sigma(z)(1 - \sigma(z)) \leq 0.25$$

For tanh activation:

$$\tanh'(z) = 1 - \tanh^2(z) \leq 1$$

Thus each layer contributes a factor ≤ 1.

7.3 Exponential Decay of Gradients

Let each layer contribute an average factor $c < 1$.

Then for early layers:

$$\left\| \frac{\partial L}{\partial W^{(l)}} \right\| \approx c^{L-l}$$

As depth increases:

$$c^{L-l} \to 0$$

Hence gradients vanish.

7.4 Consequences

- Early layers learn extremely slowly
- Network behaves like a shallow model
- Training stagnates despite low loss at top layers

7.5 Mathematical Interpretation

The Jacobian of each layer has singular values < 1. Repeated multiplication shrinks gradient magnitude exponentially.

8 Exploding Gradient Problem

8.1 Mathematical Cause

If weight matrices have large norms:

$$\|W^{(k)}\| > 1$$

Then during backpropagation:

$$\prod_{k=l+1}^{L} \|W^{(k)}\| \to \infty$$

8.2 Gradient Explosion

Gradients grow exponentially:

$$\left\| \frac{\partial L}{\partial W^{(l)}} \right\| \to \infty$$

8.3 Consequences

- Extremely large weight updates
- Loss becomes NaN or Inf
- Training diverges

8.4 Why Initialization Matters

Improper initialization causes:

$$\text{Var}(W) \gg \frac{1}{n_{\text{in}}}$$

leading to unstable gradient propagation.

9 Dying ReLU Problem

9.1 Definition

ReLU activation:

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

Derivative:

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

9.2 Mechanism of Death

If a neuron's input becomes negative:

$$z \leq 0 \Rightarrow a = 0 \Rightarrow \frac{\partial a}{\partial z} = 0$$

Then:

$$\frac{\partial L}{\partial W} = 0$$

The neuron stops updating forever.

9.3 Why It Happens

- Large negative bias
- Large learning rate
- Poor initialization

9.4 Mathematical Lock-In

Once:

$$z^{(l)} < 0 \ \forall \text{ samples}$$

then:

$$\forall t, \quad W_t^{(l)} = W_{t-1}^{(l)}$$

Neuron is permanently dead.

9.5 Solutions

- He initialization
- Smaller learning rate
- Leaky ReLU

Leaky ReLU:

$$\text{LeakyReLU}(z) = \begin{cases} z & z > 0 \\ \alpha z & z \leq 0 \end{cases}$$

Derivative:

$$\text{LeakyReLU}'(z) = \begin{cases} 1 & z > 0 \\ \alpha & z \leq 0 \end{cases}$$

-

10 Gradient Clipping

10.1 Motivation

To prevent exploding gradients while preserving direction.

10.2 Norm-Based Clipping

Let gradient vector be g.

If:

$$\|g\| > \tau$$

Then rescale:

$$g \leftarrow \tau \frac{g}{\|g\|}$$

10.3 Effect on Training

- Stabilizes training
- Prevents NaNs
- Preserves gradient direction

10.4 Mathematical Interpretation

Clipping projects gradients onto a hypersphere of radius τ.

This bounds update magnitude:

$$\|\Delta\theta\| \leq \alpha\tau$$

10.5 When to Use Gradient Clipping

- Very deep networks
- Recurrent networks
- Unstable loss behavior

11 Choice of Activation Functions and Training Stability

Activation functions are the second major factor (after initialization) that determines whether training is stable or not.

11.1 Why Activation Functions Matter

Activation functions control:

- non-linearity,
- gradient magnitude,
- signal propagation,
- saturation behavior.

Even with perfect initialization, a poor activation choice can cause vanishing or exploding gradients.

11.2 Linear Activation and Its Limitation

Linear activation:

$$\phi(z) = z$$

Derivative:

$$\phi'(z) = 1$$

While gradients do not vanish, stacking linear layers results in:

$$f(x) = W_L W_{L-1} \cdots W_1 x$$

which is still a linear function.

Thus:

- no expressive power,
- cannot model complex functions.

11.3 Sigmoid Activation

Sigmoid:

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

Derivative:

$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$

Problems:

- derivative ≤ 0.25,
- saturates for large $|z|$,
- gradients vanish exponentially.

Thus sigmoid is unsuitable for deep networks.

11.4 Tanh Activation

Tanh:

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

Derivative:

$$\tanh'(z) = 1 - \tanh^2(z)$$

Advantages over sigmoid:

- zero-centered output,
- larger gradients near zero.

Still suffers from vanishing gradients in deep networks.

11.5 ReLU Family

ReLU:

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

Derivative:

$$\mathrm{ReLU}'(z) = \begin{cases} 1 & z > 0 \\ 0 & z \leq 0 \end{cases}$$

Advantages:

- avoids vanishing gradients,
- sparse activations,
- computationally efficient.

Requires He initialization for stability.

11.6 Activation Choice Summary

Activation	Gradient Stability	Recommended Init
Sigmoid	Poor	Xavier
Tanh	Moderate	Xavier
ReLU	Good	He
Leaky ReLU	Good	He

12 Softmax Function

Softmax is used in the output layer for multi-class classification.

12.1 Definition

For logits $z \in \mathbb{R}^K$:

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

Properties:

- outputs probabilities,
- $\sum_i p_i = 1$,
- $0 \leq p_i \leq 1$.

12.2 Why Softmax Is Needed

Using raw logits does not give probabilistic interpretation. Softmax converts arbitrary real values into a probability distribution.

12.3 Numerical Instability Problem

Direct computation may overflow:

$$e^{z_i} \to \infty \quad \text{if } z_i \gg 0$$

12.4 Stable Softmax Derivation

Subtract maximum logit:

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

Since:

$$\max(z_i - \max(z)) = 0$$

exponentials remain bounded.

12.5 Jacobian of Softmax

Let:

$$p_i = \text{softmax}(z_i)$$

Derivative:

$$\frac{\partial p_i}{\partial z_j} = \begin{cases} p_i(1 - p_i) & i = j \\ -p_i p_j & i \neq j \end{cases}$$

This can be written compactly:

$$J = \text{diag}(p) - pp^T$$

12.6 Softmax and Gradient Flow

Softmax gradients are well-behaved when combined with cross-entropy loss. This pairing avoids vanishing gradients at the output layer.

13 Conclusion

Training stability in multi-layer perceptrons is not accidental,it is the result of carefully designed mathematical principles.

In this report, we showed how:

- variance propagation explains why naive initialization fails,
- Xavier and He initialization preserve signal and gradient magnitude,
- activation functions directly control gradient flow,
- ReLU enables deep training but introduces dying neurons,
- vanishing and exploding gradients arise from repeated Jacobian multiplication,
- gradient clipping stabilizes optimization,
- softmax provides probabilistic outputs while remaining numerically stable.