Weight Initialization and Training Stability in Multilayer Perceptrons

Naman Goel

January 7, 2026

Abstract

Weight initialization significantly influences neural network training success. Improper initialization leads to slow learning, vanishing gradients, or complete training failure. This report establishes theoretical principles for weight initialization, derives the Xavier and He initialization schemes, analyzes their theoretical justifications, and addresses vanishing and exploding gradient phenomena. The analysis demonstrates how initialization choice interacts with activation function selection to determine training dynamics.

1 Introduction

The initialization of network weights profoundly affects training trajectories and final model performance. Three critical failure modes illustrate why initialization matters: weights initialized to zero create symmetric neurons with identical behavior; weights too large produce extreme activations and gradients; weights too small produce negligible learning signals. Principled initialization schemes avoid these pitfalls by preserving signal propagation through the network.

2 The Symmetry Problem

2.1 Identical Weights Produce Identical Outputs

If all weights in a layer are initialized identically, each neuron in that layer receives identical input, computes an identical pre-activation, applies identical activation, and produces identical output. During backpropagation, all neurons receive identical error signals and compute identical gradient updates, maintaining symmetry indefinitely.

This means that despite having many neurons, the layer functions as a single neuron. Network capacity is not increased through adding neurons—only their number, not their functional complexity, increases.

2.2 Breaking Symmetry Through Randomization

Random initialization breaks symmetry, allowing different neurons to learn different features. However, the initialization distribution must be chosen carefully to maintain stable signal propagation.

3 Activation Range and Signal Propagation

3.1 Maintaining Activation Variance

For a neuron with n inputs and weights $\mathbf{w}$, the pre-activation is a weighted sum:

The variance of this weighted sum depends on input variance and weight variance. If weights are too large, the pre-activation variance becomes large, potentially saturating activation functions. If weights are too small, the pre-activation becomes negligible.

The goal is to choose weight distributions such that activations maintain reasonable magnitude throughout the network—neither saturating nor vanishing.

3.2 Variance Reduction Through Summation

When computing a sum of independent random variables, the variance of the sum is the sum of variances. If we have n independent weights and inputs, the pre-activation variance is:

$$\text{Var}(z) = n \cdot \text{Var}(w) \cdot \text{Var}(x) \tag{1}$$

To maintain constant variance as the number of inputs increases, weight variance must decrease proportionally. A natural choice is to set weight variance inversely proportional to the number of inputs.

4 Xavier Initialization

4.1 Theoretical Justification

For activation functions like sigmoid and tanh that are approximately linear around zero, maintaining constant variance through layers ensures good gradient flow. Xavier initialization (also called Glorot initialization) solves for weight variance that achieves this:

$$\text{Var}(w) = \frac{1}{n_{in}} \tag{2}$$

where n_{in} is the number of input connections to a neuron (fan-in). This leads to a standard deviation of:

$$\sigma = \sqrt{\frac{1}{n_{in}}} \tag{3}$$

Alternatively, considering both fan-in and fan-out (number of outgoing connections), a symmetric formula balances forward and backward signal propagation:

$$\text{Var}(w) = \frac{2}{n_{in} + n_{out}} \tag{4}$$

4.2 Distribution Choices

Weights can be drawn from either a Gaussian or uniform distribution with the computed variance. The uniform distribution approach samples from:

$$w \sim \text{Uniform}\left(-\sqrt{\frac{3}{n_{in}}}, \sqrt{\frac{3}{n_{in}}}\right) \tag{5}$$

This range ensures the uniform distribution has the desired variance.

4.3 Applicability to Activation Functions

Xavier initialization is derived assuming activation functions are approximately linear near zero (true for sigmoid and tanh). For these activation functions, Xavier ensures activations across layers maintain similar variance, preventing extreme values that would saturate the activation function.

4.4 Implementation

```
import torch
import torch.nn as nn
import torch.nn.init as init
import numpy as np

# Method 1: Xavier uniform initialization (PyTorch)
model = nn.Sequential(
    nn.Linear(784, 256),
    nn.Sigmoid(),
    nn.Linear(256, 10)
)

for layer in model:
    if isinstance(layer, nn.Linear):
        init.xavier_uniform_(layer.weight)
        init.zeros_(layer.bias)

# Method 2: Xavier normal initialization
for layer in model:
    if isinstance(layer, nn.Linear):
        init.xavier_normal_(layer.weight)
        init.zeros_(layer.bias)

# Method 3: Manual implementation with NumPy
def xavier_init(layer_shape):
    n_in, n_out = layer_shape
    limit = np.sqrt(6 / (n_in + n_out))
    return np.random.uniform(-limit, limit, layer_shape)
```

5 He Initialization

5.1 Deviation from Xavier for ReLU Networks

Xavier initialization assumes activation functions are centered near zero. The Rectified Linear Unit (ReLU) behaves qualitatively differently: it outputs zero for negative inputs and identity for positive inputs. Approximately half of neurons output zero, changing the statistics fundamentally.

When half the outputs are zero, the effective number of active inputs to the next layer is reduced. To compensate and maintain activation variance, weights must be initialized larger.

5.2 The He Initialization Scheme

He initialization adjusts weight variance to account for ReLU's asymmetry:

$$\text{Var}(w) = \frac{2}{n_{in}} \tag{6}$$

This doubles the variance compared to Xavier, accounting for approximately half the neurons being inactive. The corresponding standard deviation is:

$$\sigma = \sqrt{\frac{2}{n_{in}}} \tag{7}$$

With uniform distribution, weights are sampled from:

$$w \sim \text{Uniform}\left(-\sqrt{\frac{6}{n_{in}}}, \sqrt{\frac{6}{n_{in}}}\right) \tag{8}$$

5.3 Applicability to ReLU Variants

He initialization is specifically designed for ReLU and its variants (Leaky ReLU, ELU). For other activations with different saturation properties, empirical exploration or theoretical analysis specific to that activation is warranted.

5.4 Implementation

```
import torch.nn as nn
import torch.nn.init as init
import numpy as np

# Method 1: He uniform initialization (PyTorch)
model = nn.Sequential(
    nn.Linear(784, 256),
    nn.ReLU(),
    nn.Linear(256, 256),
    nn.ReLU(),
    nn.Linear(256, 10)
)

for layer in model:
    if isinstance(layer, nn.Linear):
        init.kaiming_uniform_(layer.weight, nonlinearity='relu')
        init.zeros_(layer.bias)

# Method 2: He normal initialization
for layer in model:
    if isinstance(layer, nn.Linear):
        init.kaiming_normal_(layer.weight, nonlinearity='relu')
        init.zeros_(layer.bias)

# Method 3: Manual NumPy implementation
def he_init(layer_shape):
    n_in, n_out = layer_shape
    limit = np.sqrt(6 / n_in)
    return np.random.uniform(-limit, limit, layer_shape)
```

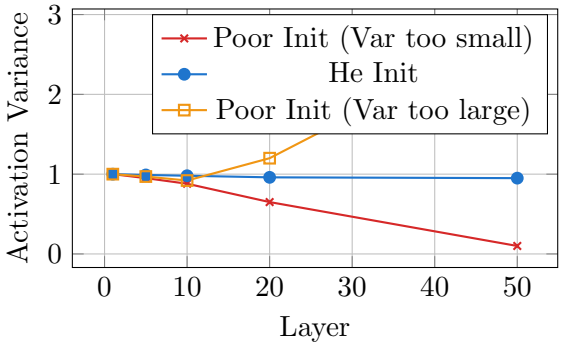

Figure 1: Activation variance through a deep network initialized with different schemes. He initialization maintains approximately constant variance, while poor initialization causes decay or explosion.

6 Vanishing Gradients

6.1 Multiplicative Composition of Gradients

In deep networks, gradients propagate backward through many layers via multiplication of local derivatives. Consider a sequence of L layers, where the gradient at each layer depends on the derivative of that layer's activation function.

For sigmoid and tanh, the gradient is bounded. The sigmoid derivative $\sigma'(z) = \sigma(z)(1 - \sigma(z))$ is at most 0.25 (achieved at $z = 0$). For tanh, the derivative $\tanh'(z) = 1 - \tanh^2(z)$ is at most 1.

When gradients flow backward through 50 layers, each multiplying by at most 0.25, the result is approximately $0.25^{50} \approx 10^{-30}$ effectively zero. Neurons in early layers receive negligible gradient signals and effectively stop learning.

6.2 Consequences for Deep Networks

This vanishing gradient problem explains empirically observed phenomena: very deep networks trained from scratch with sigmoid or tanh activations perform worse than shallow networks. The deeper layers essentially do not learn they remain near their random initialization.

6.3 Mitigation Strategies

Several approaches prevent vanishing gradients:

1. **Use ReLU:** The ReLU derivative is 1 when active (non-saturating), avoiding multiplicative decay. Only inactive neurons contribute zero, but averaging across mini-batches provides sufficient gradient signal.

2. **Careful Initialization:** Xavier/He initialization keeps activations in regions where gradients are strongest, though this alone is insufficient for very deep networks.

3. **Batch Normalization:** Resetting activation distributions prevents accumulation of extreme values that would saturate activation functions.

4. **Skip Connections:** Residual networks provide direct paths for gradients to bypass deep sequences, enabling training of very deep networks.

7 Exploding Gradients

7.1 Exponential Growth of Gradient Magnitude

The inverse problem occurs when gradient factors exceed one in magnitude. If each local gradient derivative is 2, then multiplying 50 such factors produces $2^{50} \approx 10^{15}$. Parameters receive enormous update signals, potentially diverging to infinity.

7.2 Manifestations During Training

Exploding gradients exhibit characteristic signatures: training loss suddenly jumps to NaN or infinity, weights become extremely large, and training crashes. These problems typically occur within the first few iterations when large gradients have not yet pushed parameters toward better optima.

7.3 Prevention Strategies

1. **Proper Initialization:** Xavier/He initialization prevents initial weights from being too large, reducing gradient magnitudes early in training.

2. **Gradient Clipping:** If gradient norm exceeds a threshold, scale all gradients proportionally to the threshold. This bounds update magnitudes while preserving direction.

3. **L2 Regularization:** Penalizing large weights discourages them from growing unboundedly.

4. **Reduced Learning Rate:** Smaller learning rates reduce update magnitudes, providing robustness to large gradients.

8 Interaction Between Initialization and Activation Functions

8.1 Sigmoid and Tanh: Xavier-Based

For sigmoid and tanh, Xavier initialization maintains activations in the region where the activation function is approximately linear. This maximizes the effective gradient and prevents saturation.

Using improper initialization with sigmoid/tanh can cause activations to cluster near 0 or 1, where gradients vanish. Xavier directly addresses this by controlling activation variance.

8.2 ReLU: He-Based

For ReLU, initialization should account for the dead neuron problem: neurons that receive only negative inputs output zero permanently. He initialization provides stronger initial signals to prevent premature death.

Moreover, with ReLU, the activation function is not saturating in the traditional sense; hence Xavier's derivation (assuming approximate linearity everywhere) is not directly applicable.

9 Initialization in Modern Architectures

9.1 Batch Normalization Effects

With batch normalization, dependence on initialization is reduced. Batch norm resets activation distributions to have zero mean and unit variance every layer, substantially reducing sensitivity to initialization scheme.

However, good initialization still benefits training: reduces early training instability and may improve final generalization. Even with batch norm, using appropriate initialization (He for ReLU networks) is recommended.

9.2 Transfer Learning and Pre-training

When using pre-trained networks, the initialization of original training is irrelevant—weights have already converged to good values. New layers (task-specific heads) still require proper initialization, typically using He or Xavier depending on activation functions employed.

10 Practical Initialization Strategy

10.1 Diagnostic Checks

When implementing training, verify initialization correctness through:

1. Compute activation statistics per layer after initialization: mean should be near 0, standard deviation should be reasonable (typically 0.5-2)
2. Check that different neurons have different initial outputs (symmetry broken)
3. Verify training begins: loss should decrease initially from initialization value
4. Monitor gradient magnitude per layer: should be comparable across layers, not rapidly increasing or decreasing

Activation	**Scheme**	**Formula**	**PyTorch Function**
Sigmoid/Tanh	Xavier uniform	$w \sim U(-\sqrt{3/n_{in}}, \sqrt{3/n_{in}})$	xavier_uniform_
Sigmoid/Tanh	Xavier normal	$w \sim N(0, 1/n_{in})$	xavier_normal_
ReLU	He uniform	$w \sim U(-\sqrt{6/n_{in}}, \sqrt{6/n_{in}})$	kaiming_uniform_
ReLU	He normal	$w \sim N(0, 2/n_{in})$	kaiming_normal_

Table 1: Initialization schemes by activation function.

11 Conclusion

Weight initialization profoundly affects neural network training. The principles governing initialization stem from variance preservation through layers: weights must be scaled inversely to the number of inputs to prevent signal explosion or extinction. Xavier initialization achieves this for sigmoid/tanh networks, while He initialization accounts for ReLU's asymmetry. Proper initialization is particularly critical for deep networks trained without batch normalization. When combined with appropriate architecture choices (skip connections, batch norm) and training strategies (gradient clipping), proper initialization enables successful training of very deep networks.

References

- Glorot, X., Bengio, Y. (2010). Understanding the difficulty of training deep feed-forward neural networks. AISTATS.
- He, K., Zhang, X., Ren, S., Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. ICCV.
- LeCun, Y., Bottou, L., Orban, G. B., Muller, K. R. (1998). Efficient backprop. In Neural networks: Tricks of the trade. Springer.
- Saxe, A. M., McClelland, J. L., Ganguli, S. (2013). Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. arXiv preprint arXiv:1312.6120.
- Goodfellow, I., Bengio, Y., Courville, A. (2016). Deep learning. MIT press.