Weight Initialization in Neural Networks

Weight initialization is a critical step in training neural networks. Improper initialization can lead to issues like **vanishing gradients** or **exploding gradients**, making the network hard or slow to converge. Below, I explain **what not to do**, common problems, and the best methods like **Xavier Initialization** and **He Initialization**.

What Not to Do in Weight Initialization

1. **Zero Initialization**

- **Description**: Initializing all weights to `0`.
- **Problem**:
  - If all weights are the same (e.g., zero), every neuron in a layer will compute the same gradient during backpropagation. This causes the network to lose its ability to learn diverse features (called **symmetry breaking failure**).
  - **Result**: Network fails to converge.

---

2. **Non-Zero Same Constant Values**

- **Description**: Initializing all weights to a constant value (e.g., 0.5) and biases to another constant.
- **Problem**:
  - Like zero initialization, this also leads to **symmetry breaking failure**, as all neurons in a layer behave identically.

- **Result**: Slower convergence and suboptimal results.

---

3. **Random Initialization with Small Values**

- **Description**: Randomly initializing weights with very small values close to zero (e.g., $\mathrm{N}(0, 0.01)$).
- **Problem**:
  - Small weights cause outputs of neurons to be small.
  - This can result in the **vanishing gradient problem**: Gradients shrink during backpropagation, making updates negligible for earlier layers.
  - **Result**: The network learns very slowly or stops learning.

---

4. **Random Initialization with Large Values**

- **Description**: Randomly initializing weights with large values (e.g., $\mathrm{N}(0, 10)$).
- **Problem**:
  - Large weights amplify the neuron outputs exponentially.
  - This can result in the **exploding gradient problem**: Gradients grow exponentially during backpropagation, causing instability or NaN values.
  - **Result**: The network diverges during training.

Best Methods for Weight Initialization

The goal of proper weight initialization is to maintain a balance between vanishing and exploding gradients by scaling weights according to the number of inputs to a neuron. Two widely used methods are **Xavier Initialization** and **He Initialization**.

---

1. Xavier Initialization (Glorot Initialization)

- **Idea**:
  - Designed to maintain the variance of activations across layers.
  - Suitable for **sigmoid** and **tanh** activations.
- **Formulas**:
  - **Uniform Distribution**:

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

  - **Normal Distribution**:

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

- **Explanation**:
  - n_{in}: Number of inputs to the layer.
  - n_{out}: Number of outputs from the layer.

- **Use Case**:
  - Works well for **sigmoid** and **tanh** activations because it keeps activations in a range where gradients don't vanish or explode.

---

2. He Initialization

- **Idea**:
  - A variant of Xavier Initialization, but with higher variance to account for **ReLU** and its variants.
  - ReLU activations output 0 for negative inputs, reducing the effective number of neurons. He Initialization adjusts for this by increasing the variance.
- **Formulas**:
  - **Uniform Distribution**:

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

  - **Normal Distribution**:

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

- **Explanation**:
  - n_{in}: Number of inputs to the layer.
- **Use Case**:

- Works well for **ReLU** and its variants like Leaky ReLU, ELU, etc., where gradients need slightly higher variance to propagate effectively.

Which is Better?

Criteria	Xavier Initialization	He Initialization
Best for Activations	Sigmoid, tanh	ReLU, Leaky ReLU, ELU
Gradient Propagation	Balanced gradients	Enhanced gradients for ReLU
Divergence Risk	Low	Low
Default Choice	For tanh, sigmoid networks	For ReLU-based networks

Summary:

- Use **Xavier Initialization** for non-linear activation functions like **sigmoid** and **tanh**.
- Use **He Initialization** for activations like **ReLU** and its variants.

Graph: Normal Distribution of Weights

Below is a conceptual representation of how weights are distributed under different initialization schemes.

- **Small Values**: Cluster tightly around zero → Vanishing gradients.

- **Large Values**: Spread widely → Exploding gradients.
- **Xavier**: Balanced variance → Ensures stable gradient flow for sigmoid/tanh.
- **He**: Slightly larger variance → Ideal for ReLU activations.