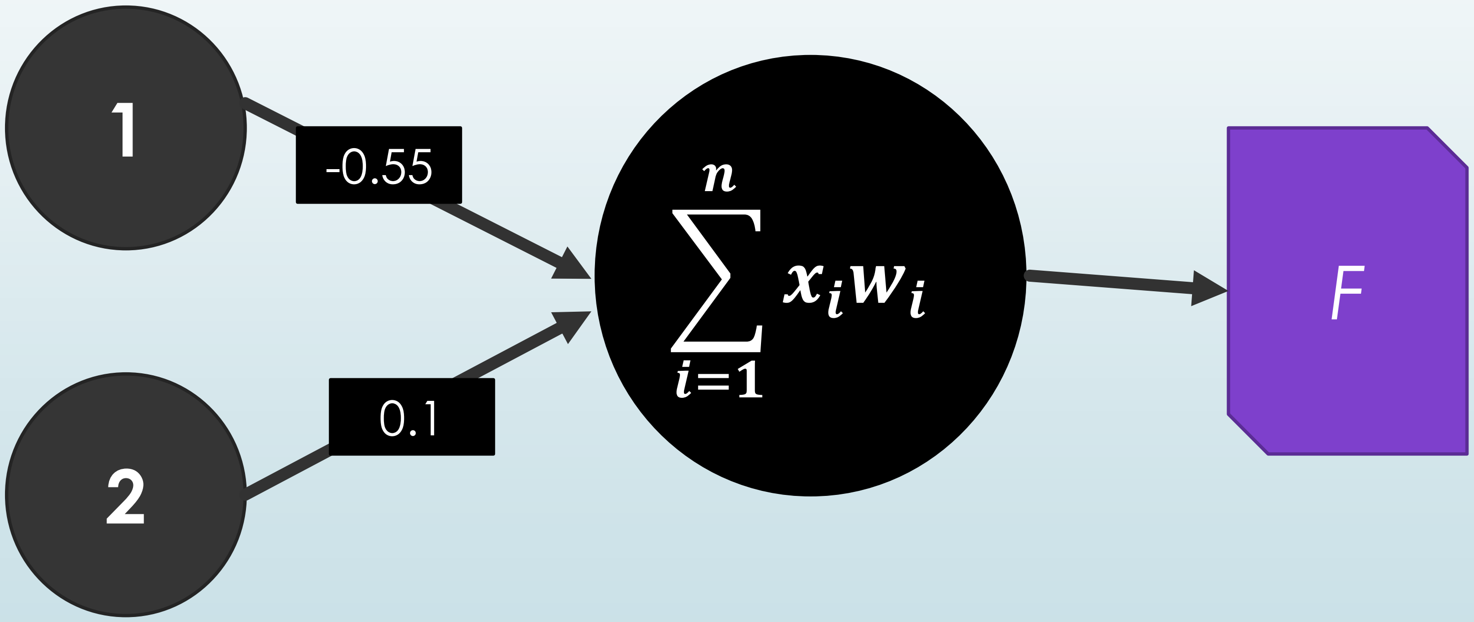
Deep Learning

Lecture 2

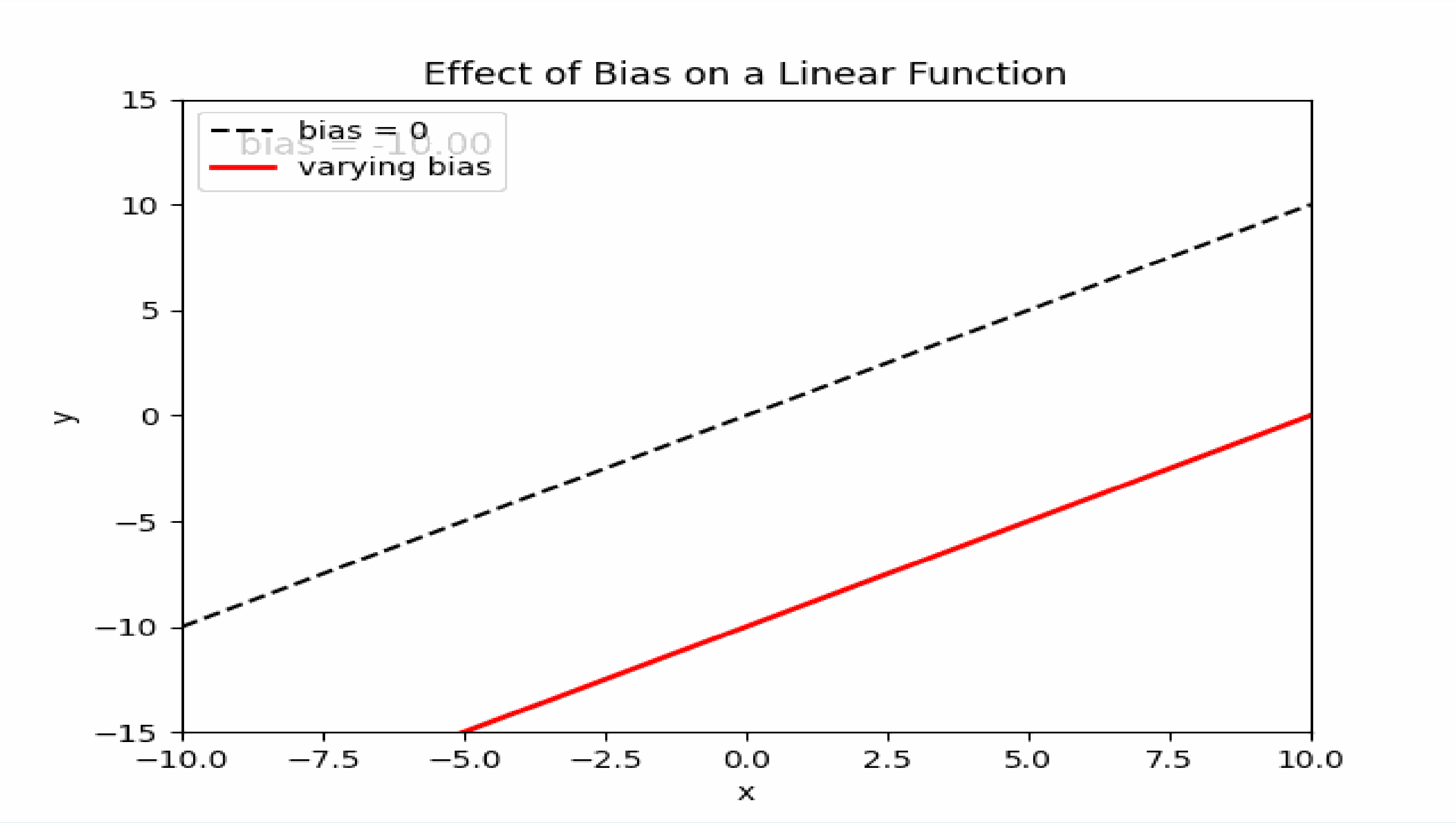
Bias in Deep Learning

- Bias is an additional parameter in a neural network that lets the model adjust its output independently of the input.
- This is equivalent to the intercept in a linear equation y=mx+b
- It allows the activation function to shift horizontally.
- It enables the network to better fit the data—even when all input features are zero.

Role of Bias



Why Bias



Effect of Bias on a Linear Function
bias = 0
varying bias
bias = -10.00
15
10
5
0
−5
−10
−15
y
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
x

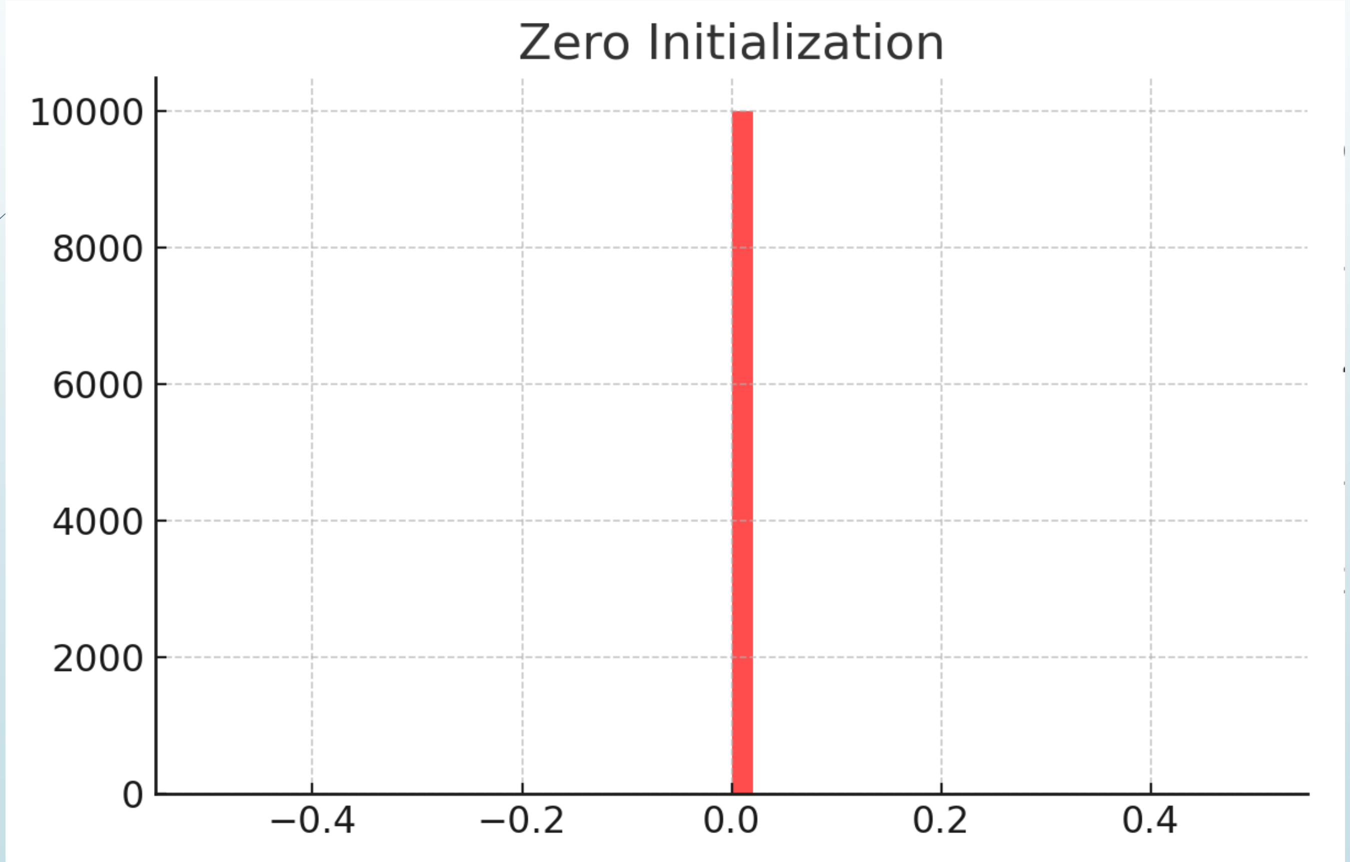
Weights Initialization in Deep Learning

1. Zero Initialization
2. Random Initialization
3. Xavier (Glorot) Initialization
4. He Initialization

Zero Initialization

- Setting all weights to zero causes all neurons to learn the same features, making them redundant
- This prevents the network from learning properly.

Zero Initialization



Random Initialization

- Small random values (e.g., Gaussian or Uniform distribution) are assigned to break symmetry.
- Weights are assigned from a uniform distribution between a range, typically:
- W = np.random.uniform(-0.1, 0.1, size=(layer_size, prev_layer_size))

Random Initialization

A Simple 3-Layer Neural Network

Assume we have a network with:

- **Input layer (5 neurons)**
- **Hidden layer (4 neurons)**
- **Output layer (3 neurons)**

Random Initialization

We initialize weights for:

1. **Hidden Layer**: Weights connect 5 input neurons to 4 hidden neurons → **Shape (4, 5)**
2. **Output Layer**: Weights connect 4 hidden neurons to 3 output neurons → **Shape (3, 4)**

Random Initialization

Weight Matrix Shape: (4, 5)

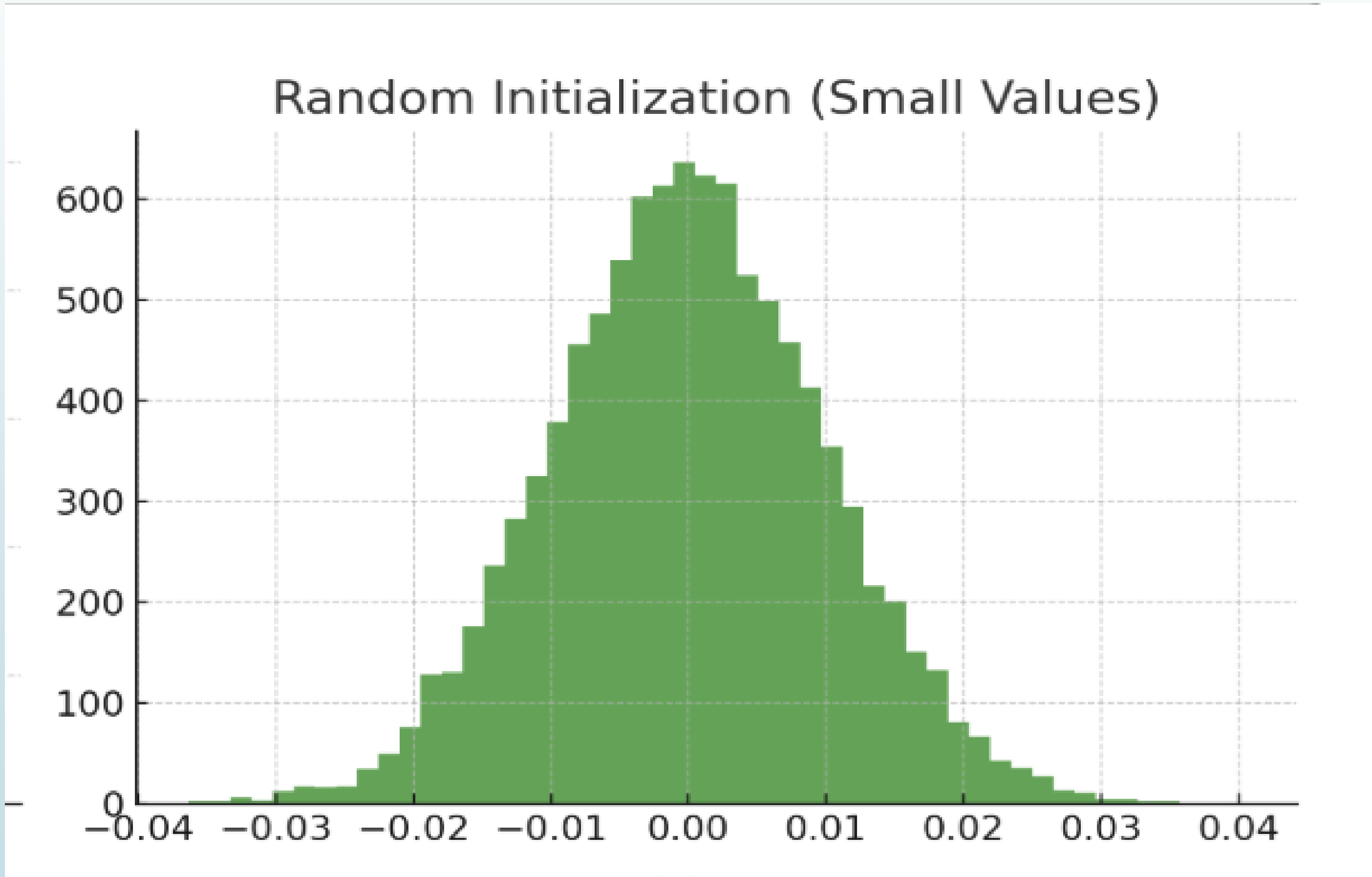
[[0.01 -0.02 0.005 0.01 -0.01]

[-0.005 0.03 -0.02 0.02 -0.01]

[0.01 -0.01 0.02 -0.005 0.01]

[-0.02 0.015 -0.01 0.03 -0.02]]

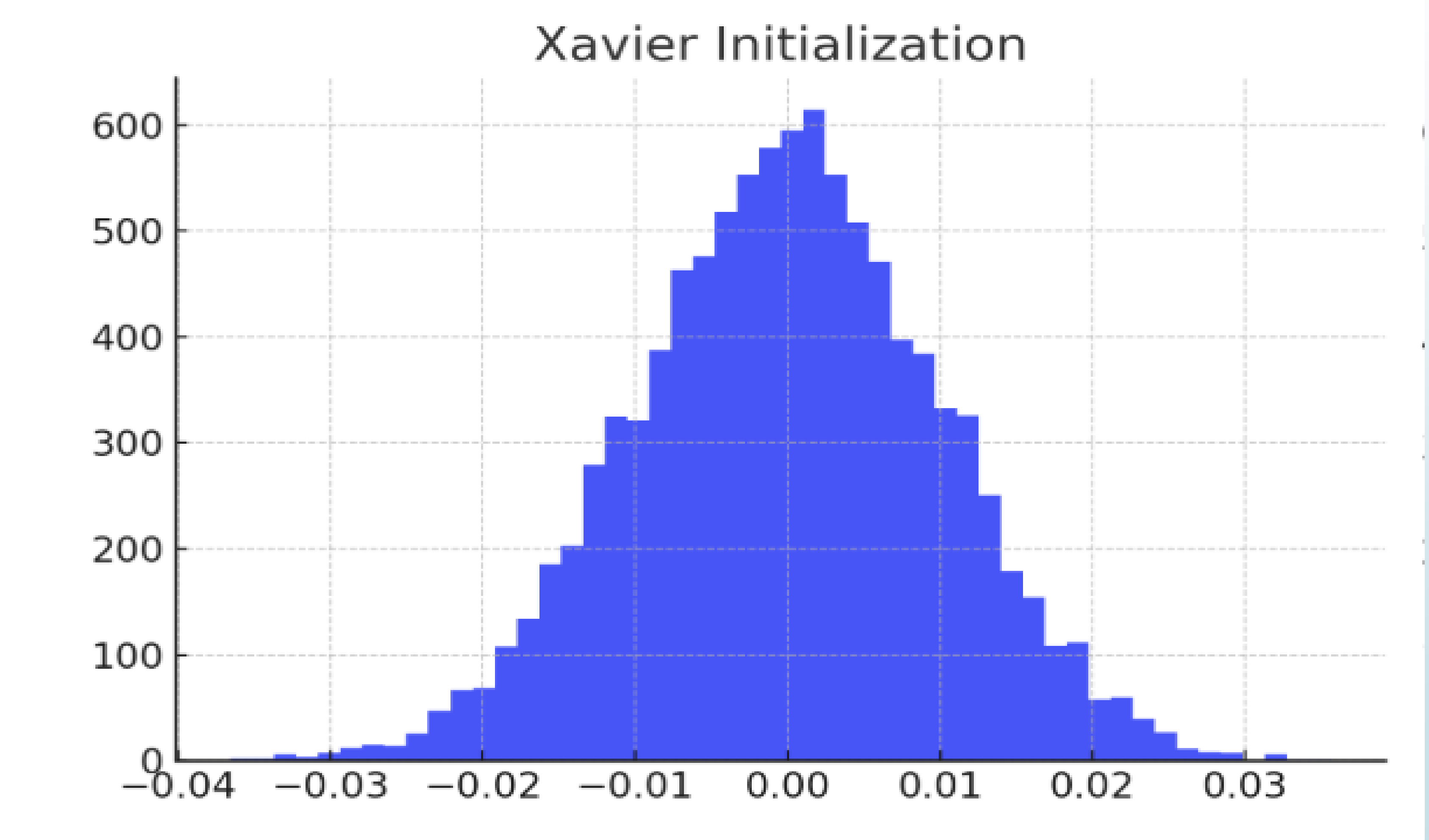
Random Initialization



Xavier (Glorot) Initialization

- Used for **sigmoid** and **tanh** activation functions.
- Weights are drawn from a normal or uniform distribution with variance controlled by the number of inputs and outputs.
- **Low variance** → Small weights → Vanishing gradients (slow learning).
- **High variance** → Large weights → Exploding gradients (unstable learning).
- Techniques like **Xavier** and **He initialization** set variance based on **the number of neurons** in the layer to ensure stable learning.
- W = np.random.randn(n_out, n_in) * np.sqrt(2 / (n_in + n_out))

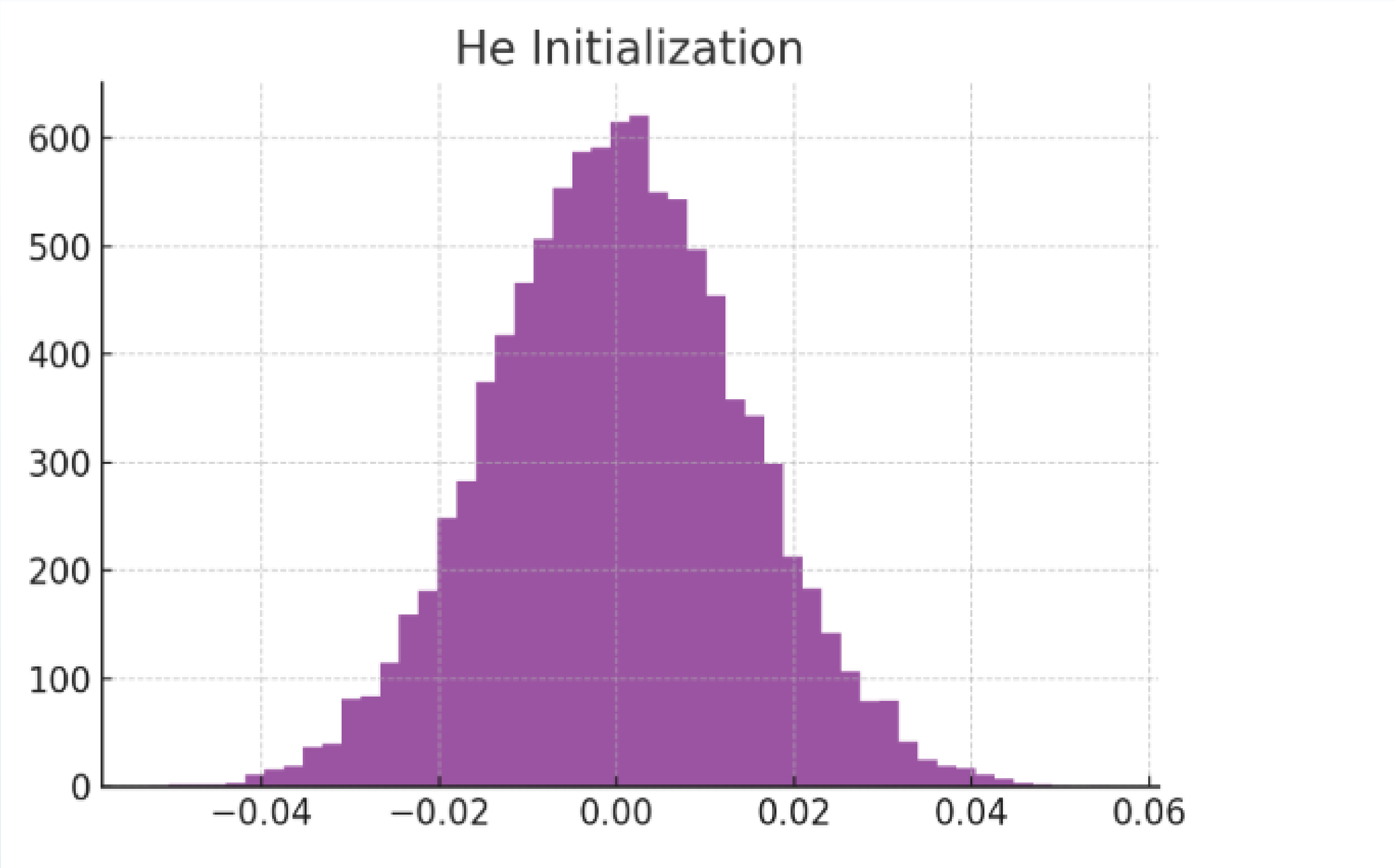
Xavier (Glorot) Initialization



He Initialization

- Used for ReLU and Leaky ReLU activation functions.
- Helps prevent the vanishing gradient problem.
- W = np.random.randn(n_out, n_in) * np.sqrt(2 / n_in)

He Initialization



Leaky Relu

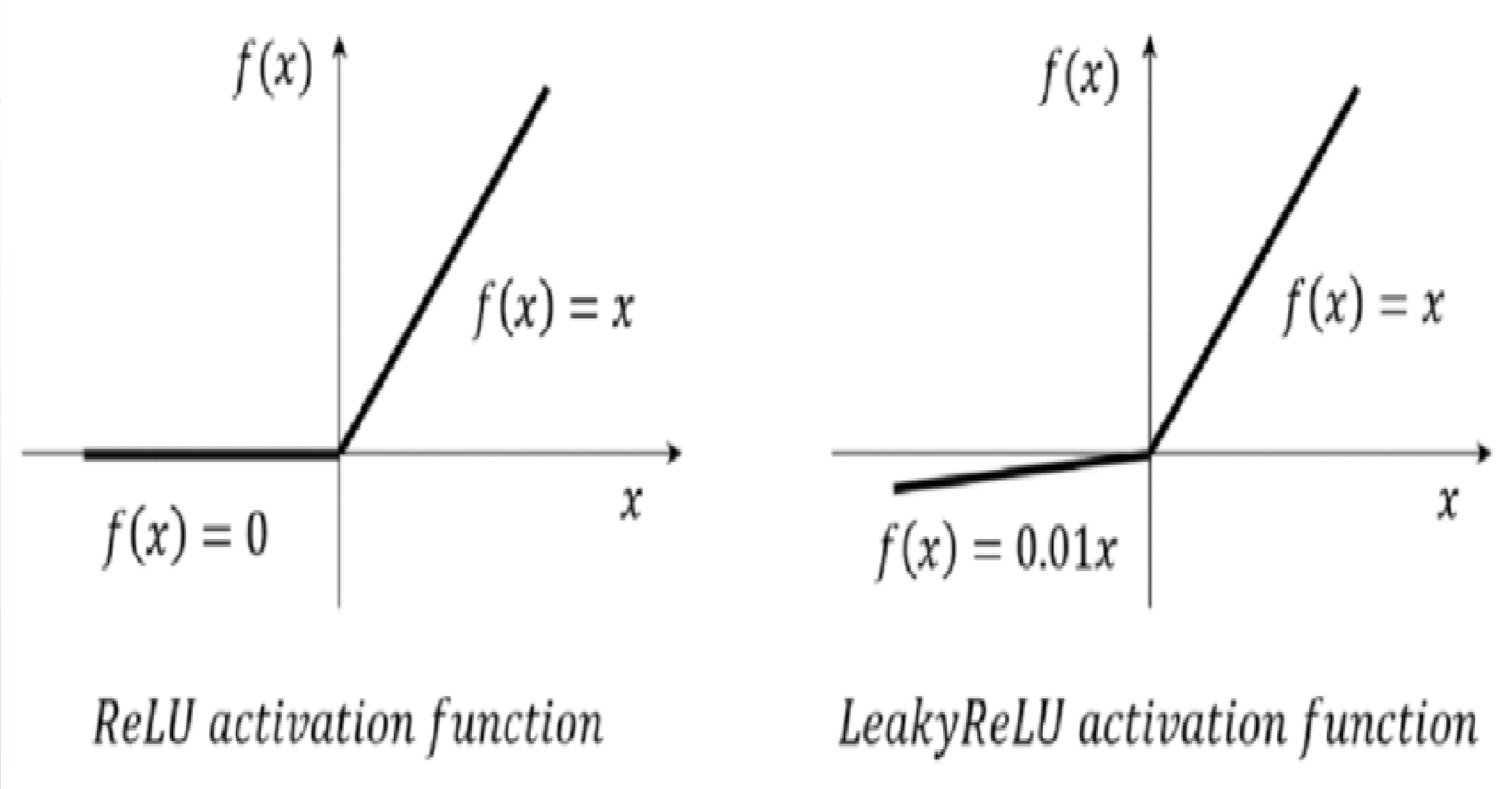
- Leaky ReLU is an **activation function** used in neural networks to solve the **dying ReLU problem** (when neurons output zero and stop learning).
- It introduces a small **negative slope** for negative inputs instead of setting them to zero.

Leaky Relu

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha x, & \text{if } x \leq 0 \end{cases}$$

Where α is a small positive constant (e.g., 0.01) that allows a small slope in the negative region.

Leaky Relu



https://www.researchgate.net/publication/358306930_cardiGAN_A_Generative_Adversarial_Network_Model_for_Design_and_Discovery_of_Multi_Principal_Element_Alloys/figures?lo=1&utm_source=google&utm_medium=organic