

# Weight Initialization Methods for FFNNs with Diverse Activation Functions

Hyunwoo Lee

Korea Institute for Advanced Studys

May 28, 2025  
2025 KIAS CAINS Workshop

# Contents

- 1 Introduction
- 2 Proposed Weight Initialization for ReLU Networks
- 3 Proposed Weight Initialization for Tanh Networks
- 4 Future Work

# 1. Introduction

# Preliminary

- Let  $K$  pairs of training samples  $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^K$ , where  $\mathbf{x}_i \in \mathbb{R}^{N_x}$  is the training input and  $\mathbf{y}_i \in \mathbb{R}^{N_y}$  is its corresponding output.
- Here,  $N_x$  and  $N_y$  are the number of nodes in the input layer and output layer, respectively.

## Feedforward Neural Network (FFNN)

$$\mathbf{x}^\ell = f(\mathbf{z}^\ell) = f(\mathbf{W}^\ell \mathbf{x}^{\ell-1} + \mathbf{b}^\ell) \in \mathbb{R}^{N_\ell} \quad \text{for all } \ell = 1, \dots, L,$$

where  $\mathbf{x}^{\ell-1} \in \mathbb{R}^{N_{\ell-1}}$  is the input feature of  $\ell$ -th layer,  $\mathbf{W}^\ell \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$  is the weight matrix,  $\mathbf{b}^\ell \in \mathbb{R}^{N_\ell}$  is the bias vector for each  $\ell = 1, \dots, L$ , and  $f(\cdot)$  is an element-wise activation function.

## Weight Initialization

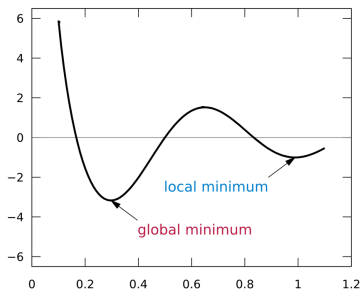
The weight initialization aims to set the initial values of  $\{\mathbf{W}^\ell\}_{\ell=1}^L$  such that training is efficient and convergence is achieved.



## Question

Why is weight initialization important for training neural networks [1, 2]?

- Speeds Up Convergence
- Improves Model Performance
- Dataset Efficiency
- Enables Training Across Various Architecture Sizes



## ReLU Activation Function

The ReLU activation function is an activation function defined as the positive part of its argument:

$$f(x) = \max(0, x)$$

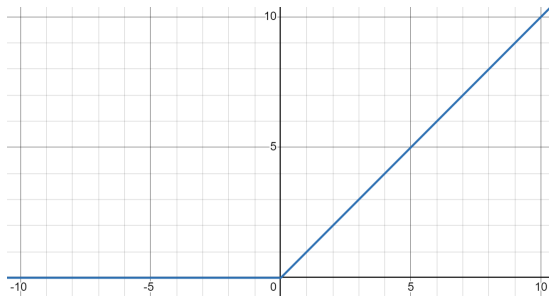


Figure 1: ReLU activation function

## Tanh Activation Function

The tanh (hyperbolic tangent) activation function is defined as:

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

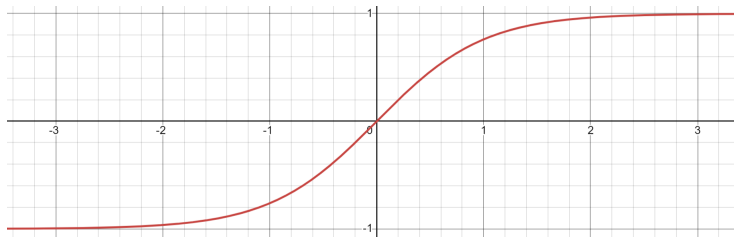


Figure 2: Tanh activation function

# Activation Function Preferences

## Neural Networks Commonly Using **ReLU**

- Feedforward Neural Networks
- Convolutional Neural Networks
- Deep Residual Networks
- Transformers

## Neural Networks Commonly Using **tanh**

- Recurrent Neural Networks
- Long Short-Term Memory Networks
- Autoencoders
- Physics Informed Neural Networks

Weight initialization depends on activation function.

# Overview of Prior and Proposed Methods

## Prior Methods on Weight Initialization for ReLU Neural Networks

- Orthogonal initialization (Saxe et al., 2014) [13]
- He Initialization (K. He et al., 2015) [12]
- Gaussian submatrix initialization (R. Burkholz et al., 2019) [11]
- Randomized asymmetric initialization (L. Lu et al., 2020) [10]
- Zero initialization (J. Zhao et al., 2022) [14]

**Proposed method 1** (H, Lee, et al., 2024) [15]

## Prior Method on Weight Initialization for Tanh Neural Networks

- Xavier Initialization (X. Glorot & Y. Bengio, 2010) [9]

**Proposed Method 2** (H, Lee, et al., 2024) [20]

## 2. Proposed Weight Initialization for ReLU Networks

# Motivation

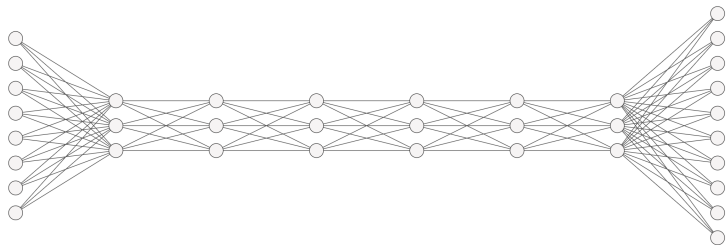


Figure 3: Example of deep and narrow FFNN with ReLU

- Crucial for finite element basis functions construction [3].
- Sparse polynomial approximation [4, 5].
- Hard to train the networks. **Why?**

# Motivation

## ReLU Neural Network

A ReLU neural network is a neural network where the activation function in each layer is the ReLU.

## Dying ReLU Problem

The dying ReLU is a kind of vanishing gradient, which refers to a problem when ReLU neurons become inactive and only output 0 for any input.

Numerous methods have been proposed to address the problem.

- Modifying network architectures [7].
- Applying various normalization techniques [8].
- **Proposing weight initialization method** [9, 10, 11].



## Xavier Normal Initialization (X. Glorot & Y. Bengio, 2010) [9]

For each  $\ell = 1, \dots, L$ ,

$$\mathbf{W}^\ell \sim \mathcal{N}(0, \text{Var}(\mathbf{W}^\ell))$$
$$\text{Var}(\mathbf{W}^\ell) = \sqrt{\frac{2}{n_{in} + n_{out}}}$$

with  $n_{in}$ ,  $n_{out}$  are the number of nodes of previous layer(input) and following layer, respectively.

- Enables stable learning in shallow networks.
- Unsuitable for nonlinear activation functions like ReLU.

## He Normal Initialization (K. He et al., 2015) [12]

For each  $\ell = 1, \dots, L$ ,

$$\mathbf{W}^\ell \sim \mathcal{N}(0, \text{Var}(\mathbf{W}^\ell))$$

$$\text{Var}(\mathbf{W}^\ell) = \sqrt{\frac{2}{n_{in}}}$$

with  $n_{in}$ ,  $n_{out}$  are the number of nodes of previous layer(input) and following layer, respectively.

- Suitable for nonlinear activation functions like ReLU.
- Perform worse with Sigmoid or Tanh than with Xavier initialization.

## Orthogonal Initialization (Saxe et al., 2014) [13]

For each  $\ell = 1, \dots, L$ , the weight matrix  $\mathbf{W}^\ell$  is initialized to be orthogonal, satisfying:

$$\mathbf{W}^{\ell T} \mathbf{W}^\ell = \mathbf{I} \quad \text{or} \quad \mathbf{W}^\ell \mathbf{W}^{\ell T} = \mathbf{I}$$

- Enhances learning speed.
- High computational cost.

## Gaussian Submatrix Initialization (R. Burkholz et al., 2019) [11]

The weight matrices  $\mathbf{W}^\ell \in \mathbb{R}^{N_{\ell-1} \times N_\ell}$ , are initially determined by a submatrix  $\mathbf{W}_0^\ell \in \mathbb{R}^{\frac{N_{\ell-1}}{2} \times \frac{N_\ell}{2}}$  as

$$\mathbf{W}^\ell = \begin{bmatrix} \mathbf{W}_0^\ell & -\mathbf{W}_0^\ell \\ -\mathbf{W}_0^\ell & \mathbf{W}_0^\ell \end{bmatrix},$$

where  $w_{0,ij}^\ell \sim \mathcal{N}(0, \sigma_{w,l}^2)$ .

- Applicable to shallow networks with arbitrary layer widths.
- Learning challenges in extremely deep and narrow ReLU networks.

## Randomized Asymmetric Initialization (L. Lu et al., 2020) [10]

Let  $\mathbf{P}_\ell$  be a probability distribution defined on  $[0, M_\ell]$  for some  $M_\ell > 0$  or  $[0, \infty)$ . Note that  $P_\ell$  is asymmetric around 0. At the first layer of  $\ell = 1$ , we employ the He initialization. For  $\ell \geq 2$ , and each  $1 \leq j \leq N_\ell$ , we initialize  $\mathbf{W}_j^\ell$  as follows:

- (i) Randomly choose  $k_j^\ell$  in  $\{1, 2, \dots, N_{\ell-1} + 1\}$ .
- (ii) Initialize  $(\mathbf{W}_j^\ell)_{-k_j^\ell} \sim \mathcal{N}(0, \sigma_\ell^2 \mathbf{I})$  and  $(\mathbf{W}_j^\ell)_{k_j^\ell} \sim P_\ell$

- Prevents the dying ReLU problem in deep networks.
- Learning challenges in extremely deep and narrow ReLU networks.

## Zero initialization (J. Zhao et al., 2022) [14]

$H_m$  is the Hadamard matrix. For  $\ell \in 1, \dots, L$

$$\mathbf{W}^\ell = \begin{cases} \mathbf{I} & \text{if } N_\ell = N_{\ell-1} \\ \mathbf{I}^* & \text{if } N_\ell < N_{\ell-1} \\ c\mathbf{I}^*H_m\mathbf{I}^*, \text{ where } m = \lceil \log_2(N_\ell) \rceil \text{ and } c = 2^{-(m-1)/2} & \text{if } N_\ell > N_{\ell-1} \end{cases}$$

where  $\mathbf{I}^*$  is a partial identity matrix.

- Enables training of extremely deep networks.
- Slow convergence in training.

## Key Properties

Our proposed weight initialization method can be characterized by key properties: **orthogonality**, **positive entry predominance**, and **fully deterministic**.

To construct a proper initial weight matrix, we find a matrix  $\mathbf{W} \in \mathbb{R}^{m \times n}$  satisfying the following conditions:

- (i) The set of all column vectors of  $\mathbf{W}$  is orthonormal;
- (ii)  $\mathbf{W}\mathbf{x}$  has more positive entries for all  $\mathbf{x} \in \mathbb{R}_+^n$ ;
- (iii)  $\mathbf{W}$  is a fully deterministic matrix.

# Proposed Method

Firstly, we define  $\mathbf{Q}_{m \times m}^\epsilon$  by the orthogonal matrix of a QR decomposition of

$$\mathbf{J}^\epsilon := \mathbf{J} + \epsilon \mathbf{I} = \begin{bmatrix} 1 + \epsilon & 1 & \cdots & 1 \\ 1 & 1 + \epsilon & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 + \epsilon \end{bmatrix}_{m \times m},$$

where  $\epsilon > 0$  is a sufficiently small.

## Proposed Weight Initialization Method (H. Lee et al., 2024) [15]

To initialize the weights of the neural networks we propose that

$$\mathbf{W}_{m \times n}^\epsilon = (\mathbf{Q}_{m \times m}^\epsilon) \mathbf{I}_{m \times n} (\mathbf{Q}_{n \times n}^\epsilon)^T.$$



# Properties of the Proposed Initial Weight Matrix 1

## Example of Our Initial Weight Matrix

For  $\epsilon = 0.1$  the proposed initial weight matrix  $\mathbf{W}_{m \times n}^\epsilon$  is computed approximately as follows.

$$\mathbf{W}_{8 \times 5}^\epsilon = \begin{bmatrix} 0.8618 & -0.1415 & -0.1413 & -0.1413 & 0.3524 \\ -0.1341 & 0.8626 & -0.1374 & -0.1374 & 0.3563 \\ -0.1342 & -0.1373 & 0.8626 & -0.1374 & 0.3563 \\ -0.1342 & -0.1373 & -0.1373 & 0.8626 & 0.3563 \\ 0.3559 & 0.3528 & 0.3528 & 0.3528 & -0.6533 \\ 0.1598 & 0.1567 & 0.1567 & 0.1567 & 0.1506 \\ 0.1598 & 0.1567 & 0.1567 & 0.1567 & 0.1506 \\ 0.1598 & 0.1567 & 0.1567 & 0.1567 & 0.1506 \end{bmatrix}.$$

The proposed weight initialization is **fully deterministic**, thus it is not dependent on randomness.

# Properties of the Proposed Initial Weight Matrix 2

## Orthogonality

Let  $\mathbf{q}_1, \dots, \mathbf{q}_m$  be the column vectors of  $\mathbf{Q}_{m \times m}^\epsilon$  and  $\hat{\mathbf{q}}_1, \dots, \hat{\mathbf{q}}_n$  be the column vectors of  $\mathbf{Q}_{n \times n}^\epsilon$ . Then it holds that

(i) if  $m = n$ ,

$$(\mathbf{W}_{m \times n}^\epsilon)^T \mathbf{W}_{m \times n}^\epsilon = \mathbf{W}_{m \times n}^\epsilon (\mathbf{W}_{m \times n}^\epsilon)^T = \mathbf{I},$$

(ii) if  $m > n$ ,

$$(\mathbf{W}_{m \times n}^\epsilon)^T \mathbf{W}_{m \times n}^\epsilon = \mathbf{I}_{n \times n},$$

$$\mathbf{W}_{m \times n}^\epsilon (\mathbf{W}_{m \times n}^\epsilon)^T = \mathbf{q}_1 \mathbf{q}_1^T + \mathbf{q}_2 \mathbf{q}_2^T + \dots + \mathbf{q}_n \mathbf{q}_n^T,$$

(iii) if  $m < n$ ,

$$\mathbf{W}_{m \times n}^\epsilon (\mathbf{W}_{m \times n}^\epsilon)^T = \mathbf{I}_{m \times m}$$

$$(\mathbf{W}_{m \times n}^\epsilon)^T \mathbf{W}_{m \times n}^\epsilon = \hat{\mathbf{q}}_1 \hat{\mathbf{q}}_1^T + \hat{\mathbf{q}}_2 \hat{\mathbf{q}}_2^T + \dots + \hat{\mathbf{q}}_m \hat{\mathbf{q}}_m^T.$$

# Properties of the Proposed Initial Weight Matrix 3

## Lemma 1.

Let  $\mathbf{q}_1, \dots, \mathbf{q}_m$  be the column vectors of  $\mathbf{Q}_{m \times m}^\epsilon$ . Then it holds that

$$\begin{aligned}\langle \mathbf{q}_1, \mathbf{1} \rangle &= \frac{m + \epsilon}{\sqrt{\epsilon^2 + 2\epsilon + m}}, \\ |\langle \mathbf{q}_j, \mathbf{1} \rangle| &\leq \epsilon \quad \text{for all } j = 2, \dots, m.\end{aligned}$$

Recall that

$$\mathbf{W}_{m \times n}^\epsilon = (\mathbf{Q}_{m \times m}^\epsilon) \mathbf{I}_{m \times n} (\mathbf{Q}_{n \times n}^\epsilon)^T.$$

## Proposition 2.

The entry sum of each column (resp. row) vector of  $\mathbf{W}_{m \times n}^\epsilon$  is almost the same.

# Properties of the Proposed Initial Weight Matrix 3

## Theorem 3.

Let  $\mathbf{W}^\epsilon \in \mathbb{R}^{N_1 \times N_x}$  with sufficiently small  $\epsilon$  be a given. Then it holds that for all  $\mathbf{x} \in \mathbb{R}^{N_x}$

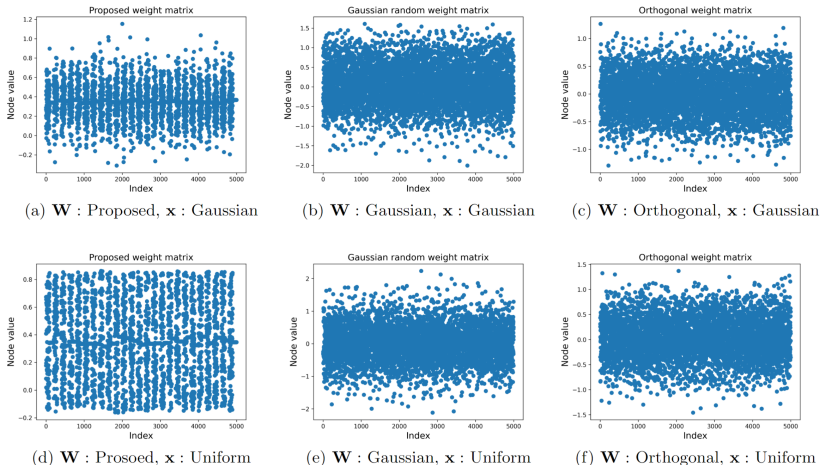
$$\frac{1}{N_x} \langle \mathbf{x}, \mathbf{1}_{N_x} \rangle \simeq \sqrt{\frac{N_1}{N_x}} \frac{1}{N_1} \langle \mathbf{W}^\epsilon \mathbf{x}, \mathbf{1}_{N_1} \rangle.$$

## Corollary 4.

Given that  $\epsilon$  is sufficiently small. Then the angle  $\theta_1$  between the one vector  $\mathbf{1}$  and  $\mathbf{x}$  in  $\mathbb{R}^{N_x}$  is nearly identical to the angle  $\theta_2$  between the one vector  $\mathbf{1}$  and  $\mathbf{W}^\epsilon \mathbf{x}$  in  $\mathbb{R}^{N_1}$ .

**$\mathbf{W}^\epsilon \mathbf{x}$  has more positive entries** for all  $\mathbf{x} \in \mathbb{R}_+^n$ .

# Properties of the Proposed Initial Weight Matrix 3



**Figure 4:** This shows its effectiveness of positive signal propagation for each weight matrices  $\mathbf{W} \in \mathbb{R}^{200 \times 100}$ . For 25 random vectors  $\mathbf{x} \in \mathbb{R}^{100}$ , the entry values of  $\mathbf{W}\mathbf{x}$  are plotted. Here, the x-axis represents the indices of all entries.

# Deep Network of Depth $\ell$

## In Deep Networks

Now we consider a deep network of depth  $\ell$  with a linear activation function and zero bias.

$$\begin{aligned}\mathbf{y} &= \mathbf{W}_{N_\ell \times N_{\ell-1}}^\epsilon \cdots \mathbf{W}_{N_1 \times N_x}^\epsilon \mathbf{x} \\ &= (\mathbf{Q}_{N_\ell \times N_\ell}^\epsilon) \mathbf{I}_{N_\ell \times N_x} (\mathbf{Q}_{N_x \times N_x}^\epsilon)^T \mathbf{x} \\ &= \mathbf{W}_{N_\ell \times N_x}^\epsilon \mathbf{x}\end{aligned}$$

It means that regardless of the network's depth,  $\mathbf{y}$  satisfies Theorem 3 and Corollary 4, provided that  $\epsilon$  is sufficiently small.

## Theorem 5.

Let  $\{\mathbf{u}_j\}_{1 \leq j \leq m}$  be defined by

$$\mathbf{u}_1 = \mathbf{1} + \epsilon \mathbf{e}_1 \in \mathbb{R}^m,$$

$$\mathbf{u}_j = \left(1 - \frac{\langle \mathbf{u}_{j-1}, \mathbf{1} + \epsilon \mathbf{e}_j \rangle}{\langle \mathbf{u}_{j-1}, \mathbf{u}_{j-1} \rangle}\right) \mathbf{u}_{j-1} + \epsilon (\mathbf{e}_j - \mathbf{e}_{j-1}) \in \mathbb{R}^m$$

for each  $j = 2, \dots, m$ .

Then  $j$ -th column vector of  $\mathbf{Q}_{m \times m}^\epsilon$  is expressed as  $\frac{1}{\|\mathbf{u}_j\|} \mathbf{u}_j$ .

- Through this theorem, the computational complexity of QR decomposition is reduced from  $\mathcal{O}(n^3)$  to  $\mathcal{O}(n^2)$ .
- $\mathbf{W}^\epsilon$  is a fully deterministic matrix.

## 2-1. Computational Results



# Comparison of $\epsilon$ Values

The figure below presents a distinct comparison of accuracy across different values of  $\epsilon$ , utilizing a small network composed of consecutively 16 and 12 hidden layers to observe the training accuracy on the MNIST dataset at each iteration. We set  $\epsilon = 0.1$  empirically.

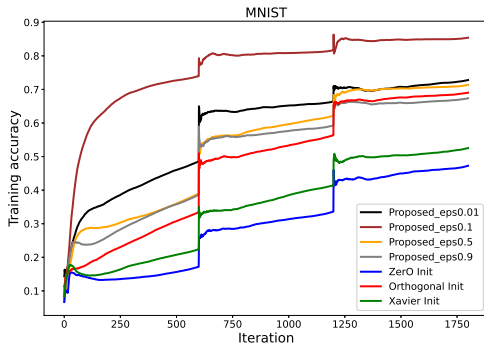


Figure 5: Comparison of  $\epsilon$  values for the MNIST dataset.

# Dataset Size Independence

Entire dataset																
	Proposed		Orthogonal		Xavier		He		Zero		Identity		RAI		GSM	
Datset	10	100	10	100	10	100	10	100	10	100	10	100	10	100	10	100
MNIST (0)	<b>88.2</b>	88.5	88.1	88.7	86.6	87.7	87.9	<b>89.1</b>	88.1	88.8	88	89.1	88	88.2	87.1	89.4
FMNIST (0)	<b>80.4</b>	<b>80.6</b>	79.6	79	79.5	78.1	79.5	80.1	78.1	80.4	78.4	80.3	79.1	80.4	75.8	80
MNIST (512)	<b>96.5</b>	<b>97.6</b>	95.8	96.3	95.8	96.5	96.4	96.6	95.1	96.5	96.2	96.5	95.9	97.5	88	89.4
FMNIST (512)	84.5	85.1	84.4	85.4	84.5	84.5	84.2	85.1	84.5	84.6	84.8	84.9	<b>84.9</b>	<b>85.2</b>	78.3	80.3
MNIST (16)	<b>92.2</b>	<b>94</b>	88	90	83.5	86	77.2	85.5	60.1	84.2	38.7	40.5	91	92.1	29	77.1
FMNIST (16)	<b>82.3</b>	<b>84.2</b>	61.1	67.3	56.4	69.2	53.3	60.7	60.1	83.1	78.1	81.2	60	78.4	34.9	37.3
4 samples per class																
	Proposed		Orthogonal		Xavier		He		Zero		Identity		RAI		GSM	
Datset	10	100	10	100	10	100	10	100	10	100	10	100	10	100	10	100
MNIST (0)	<b>55.5</b>	<b>54.1</b>	29.1	39.7	26.1	40.5	23.1	38.7	51.1	50.8	54.2	53.5	27.6	43.8	26.1	45.6
FMNIST (0)	<b>54.2</b>	<b>57.1</b>	42.7	51.4	29.5	48.1	34.2	51.1	51	50.1	52.8	56.1	36.5	53.9	33.7	51.4
MNIST (512)	<b>56.5</b>	<b>51.0</b>	49.7	50.1	44.3	45.2	46.5	48.9	22	46.3	51.9	50.8	29.9	38.8	23.3	37.2
FMNIST (512)	46.7	55.6	51	56	<b>54</b>	54.6	51	<b>56.8</b>	37.1	50.4	45.2	53.4	48.7	56.2	45.1	53.6
MNIST (16)	<b>51.2</b>	<b>52.9</b>	22.5	31.7	18.7	26.3	20	25	9.1	10.3	9.6	10.3	13.7	25.1	11.9	18.8
FMNIST (16)	<b>43.3</b>	<b>56.3</b>	23.4	24.7	18.8	17.8	20	20.8	10.8	10.7	33.3	41.5	14.9	21	10.6	26.4

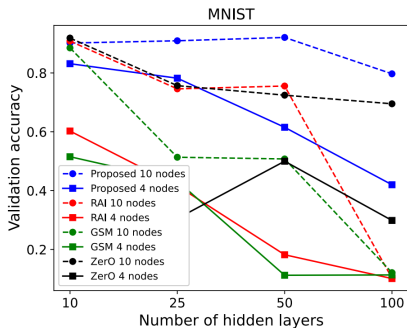
**Table 1:** This is a comparison of the validation accuracy for feedforward neural networks (FFNNs) with various weight initialization methods. Here (·) represents the number of nodes in a single hidden layer.

# Dataset Size Independence

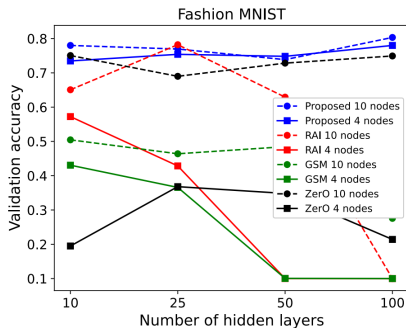
2 samples per class																
	Proposed		Orthogonal		Xavier		He		Zero		Identity		RAI		GSM	
Datset	10	100	10	100	10	100	10	100	10	100	10	100	10	100	10	100
MNIST (0)	46.4	46.5	23.7	31.6	19.6	30.1	20.7	28.7	42.6	43.3	44.1	43.6	26.3	37.8	21.2	34.5
FMNIST (0)	49.1	50.3	38.3	43.3	31.4	41.7	27.5	38.6	43	46.1	45.5	42.7	36	40	38.2	44.7
MNIST (512)	39.7	37.1	33.8	36.3	32.7	33.1	39.3	39.1	27.2	33.4	45.9	45.4	38.5	42.4	37.7	41.1
FMNIST (512)	46.8	46.2	45	48.4	43.4	44.7	42.4	51.2	34.7	43.8	44.2	47.8	38.8	39.9	40.1	42.6
MNIST (16)	44.3	41.5	19.7	23.6	16.6	21.6	19.3	22.2	10.1	11	9.6	9.5	11.2	22.8	12.5	22.6
FMNIST (16)	43.8	47.9	22.1	26.1	18.6	20	19.4	22.7	9.9	10.5	29.1	39.6	24	26.6	13.4	21.9
1 samples per class																
	Proposed		Orthogonal		Xavier		He		Zero		Identity		RAI		GSM	
Datset	10	100	10	100	10	100	10	100	10	100	10	100	10	100	10	100
MNIST (0)	37.1	38.2	20.6	23.9	29.1	36.9	25.4	33.8	9.7	33	12.6	12.4	19.1	23.6	23.1	27.4
FMNIST (0)	43.5	39.4	30.3	33.7	27.4	30.8	24.6	35.8	9.7	25.8	40.7	40.6	18	33.7	34.5	40.6
MNIST (512)	36.1	34.9	28.2	27.7	31.2	32.3	27	27.4	22.2	29.8	39.2	40.3	32.5	29.3	31.6	36.6
FMNIST (512)	39.2	37.4	36.7	34.7	38.5	37.6	36.1	35	31.7	37	0.3	3.4	39	37.2	35.2	36
MNIST (16)	33.5	34.2	16.5	19.4	14.3	16.8	14.3	19.9	10.6	11.6	10	9.8	18.1	22.6	18.4	19.8
FMNIST (16)	35	34.2	18.7	22.9	16.1	16.8	19.8	22.8	10.3	10.5	7	7.2	13.7	19.7	15.9	21.9

**Table 2:** This is a comparison of the validation accuracy for feedforward neural networks (FFNNs) with various weight initialization methods. Best results are marked in bold.

# Depth Independence



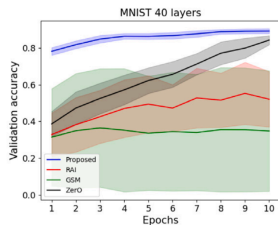
(a) MNIST



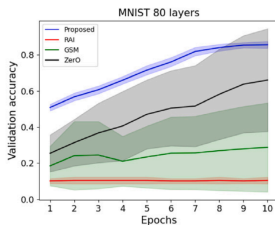
(b) Fashion MNIST

**Figure 6:** Validation accuracy for FFNNs with ReLU activation is presented across varying depths. (a) and (b) investigate networks where all hidden layers maintain the same dimension.

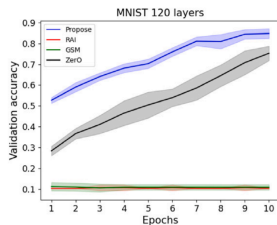
# Depth Independence



(c) MNIST 40 hidden layers



(d) MNIST 80 hidden layers



(e) MNIST 120 hidden layers

**Figure 7:** Validation accuracy for FFNNs with ReLU activation is presented across varying depths. (c), (d), and (e) investigate networks consisting of a layer with 10 nodes and a layer with 6 nodes, repeated throughout the structure.

# Width Independence

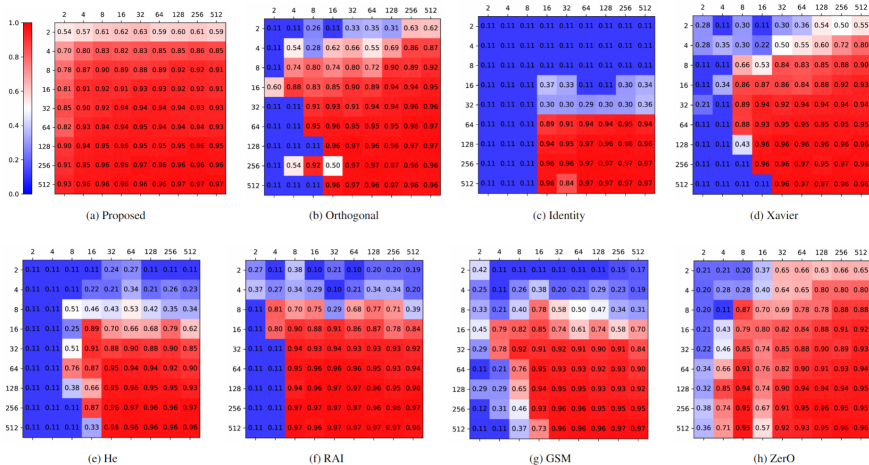


Figure 8: The y-axis (resp. x-axis) presents the number of nodes in the first (resp. second) hidden layer. Each is trained on MNIST dataset for 10 epochs.

# Width Independence

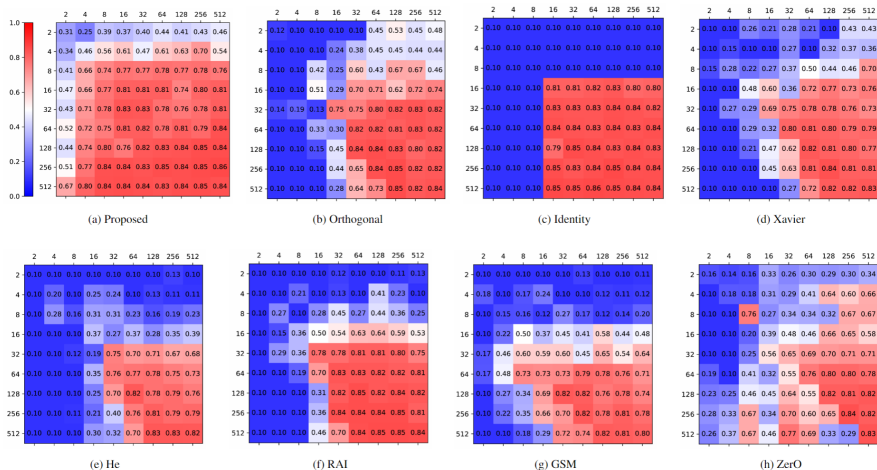


Figure 9: The y-axis (resp. x-axis) presents the number of nodes in the first (resp. second) hidden layer. Each is trained on FMNIST dataset for 1 epoch.

# Activation Function Independence

Dataset	Proposed		Orthogonal		Xavier		He		Zero		Identity		RAI		GSM	
	M	F	M	F	M	F	M	F	M	F	M	F	M	F	M	F
Tanh	11.1	10.0	14.3	9.9	11.0	9.9	11.7	9.9	10.3	9.9	<b>27.0</b>	9.9	16.1	10.0	12.3	<b>13.6</b>
Sigmoid	11.3	10.0	10.3	10.0	10.3	9.9	11.3	9.9	10.3	10.0	10.2	9.9	10.2	9.9	10.3	10.0
Selu	<b>38.3</b>	33.3	11.7	9.9	10.2	9.9	9.8	9.9	33.0	<b>34.5</b>	10.4	9.9	10.2	9.9	12.0	11.0
Gelu	<b>83.6</b>	<b>68.1</b>	65.5	10.0	11.2	10.0	11.3	10.9	76.2	65	11.3	10.0	11.0	34.0	13.1	34.4
Relu	<b>86.7</b>	<b>76.5</b>	11.3	10.0	11.3	10.1	11.3	9.9	82.9	69.4	11.3	10.0	11.3	10.0	11.3	8.6

**Table 3:** A validation accuracy is presented for FFNNs with various activation functions. The FFNN comprises 120 hidden layers with a layer of 10 nodes and a layer of 6 nodes repeated 60 times each. Each is trained on MNIST (M) and FMNIST (F) datasets for 10 epochs. The best results are marked in bold.



### 3. Proposed Weight Initialization for Tanh Networks

## Tanh Neural Network

A tanh neural network is a neural network where the activation function in each layer is the hyperbolic tangent function ( $\tanh$ ).

## Xavier Normal initialization (X. Glorot & Y. Bengio, 2010) [9]

For each  $\ell = 1, \dots, L$ ,

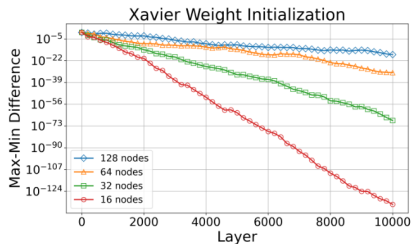
$$\mathbf{W}^\ell \sim \mathcal{N}(0, \text{Var}(\mathbf{W}^\ell))$$
$$\text{Var}(\mathbf{W}^\ell) = \sqrt{\frac{2}{n_{in} + n_{out}}}$$

with  $n_{in}$ ,  $n_{out}$  are the number of nodes of previous layer(input) and following layer, respectively.

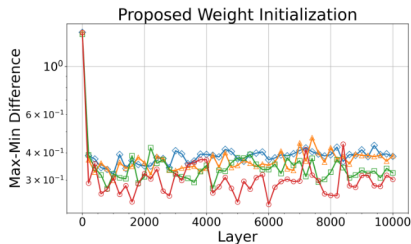
# Motivation

## Zero-Centered Activation Problem

When using **Xavier initialization** in tanh neural networks, the outputs of deeper layers tend to converge toward values extremely close to zero.



(a) Xavier Initialization



(b) Proposed Initialization

**Figure 10:** Difference between maximum and minimum activation values at each layer when propagating 3,000 input data through a 10,000-layer tanh FFNN, using Xavier initialization (Left) and the proposed initialization (Right).

## Motivation

- This makes training difficult in deep tanh neural networks, forcing the network to learn in shallower ones [16].
- The expressivity of neural networks exponentially increases with depth [17, 18].
- The use of tanh neural networks has surged with the rise of Physics-Informed Neural Networks (PINNs) [19].

We propose a **robust initialization** for tanh neural networks that enables training across various network sizes.

# The Derivation of the Proposed Method

## Question

How to effectively propagate signals to deeper layers in a tanh network?

## Simplified analysis of signal propagation in tanh FFNNs

Given an arbitrary input vector  $\mathbf{x} = (x_1, \dots, x_n)$ , the first layer activation  $\mathbf{x}^1 = \tanh(\mathbf{W}^1 \mathbf{x})$  can be expressed component-wise as:

$$x_i^1 = \tanh \left( w_{i1}^1 x_1 + \dots + w_{in}^1 x_n \right) = \tanh \left( \left( w_{ii}^1 + \sum_{\substack{j=1 \\ j \neq i}}^n \frac{w_{ij}^1 x_j}{x_i} \right) x_i \right).$$

For the  $k + 1$ -th layer,  $i = 1, \dots, n$ , this expression can be generalized as:

$$x_i^{k+1} = \tanh \left( a_i^{k+1} x_i^k \right), \text{ where } a_i^{k+1} = w_{ii}^{k+1} + \sum_{\substack{j=1 \\ j \neq i}}^n \frac{w_{ij}^{k+1} x_j^k}{x_i^k}.$$

# The Derivation of the Proposed Method

## Lemma 6.

For a fixed  $a > 0$  define the function  $\phi_a : \mathbb{R} \rightarrow \mathbb{R}$  given as

$$\phi_a(x) := \tanh(ax).$$

Then, there exists a fixed point  $x^*$ . Furthermore,

- (1) if  $0 < a \leq 1$ , then  $\phi$  has a unique fixed point  $x^* = 0$ .
- (2) if  $a > 1$ , then  $\phi$  has three distinct fixed points:  $x^* = -\xi_a, 0, \xi_a$  such that  $\xi_a > 0$ .

# The Derivation of the Proposed Method

## Lemma 7.

*For a given initial value  $x_0 > 0$  define*

$$x_{n+1} = \phi_a(x_n), \quad n = 0, 1, 2, \dots$$

*Then  $\{x_n\}_{n=1}^{\infty}$  converges regardless of the positive initial value  $x_0 > 0$ .*

*Moreover,*

- (1) if  $0 < a \leq 1$ , then  $x_n \rightarrow 0$  as  $n \rightarrow \infty$ .*
- (2) if  $a > 1$ , then  $x_n \rightarrow \xi_a$  as  $n \rightarrow \infty$ .*

# The Derivation of the Proposed Method

## Proposition 8.

*Let  $\{a_n\}_{n=1}^{\infty}$  be a positive real sequence, i.e.,  $a_n > 0$  for all  $n \in \mathbb{N}$ , such that only finitely many elements are greater than 1. Suppose that  $\{\Phi_m\}_{m=1}^{\infty}$  is a sequence of functions defined as for each  $m \in \mathbb{N}$*

$$\Phi_m = \phi_{a_m} \circ \phi_{a_{m-1}} \circ \cdots \circ \phi_{a_1}.$$

*Then for any  $x \in \mathbb{R}$*

$$\lim_{m \rightarrow \infty} \Phi_m(x) = 0.$$

Therefore, to ensure that the initial weights satisfy the following conditions:

- (i)  $a_i^k$  remains close to 1.
- (ii)  $a_i^k \leq 1$  does not hold for all  $N \leq k \leq L$ .



# The Derivation of the Proposed Method

## Proposed Weight Initialization Method (H. Lee et al., 2024) [20]

For each  $\ell = 1, \dots, L$ ,

$$\mathbf{W}^\ell = \mathbf{D}^\ell + \mathbf{Z}^\ell \in \mathbb{R}^{N_\ell \times N_{\ell-1}},$$

$$\mathbf{D}_{i,j}^\ell = \begin{cases} 1, & \text{if } i \equiv j \pmod{N_{\ell-1}}, \\ 0, & \text{otherwise} \end{cases}$$

$$\mathbf{Z}^\ell \sim \mathcal{N}(0, \sigma_z^2),$$

Then  $a_i^{k+1}$  follows the distribution:

$$a_i^{k+1} \sim \mathcal{N}\left(1, \sigma_z^2 + \sigma_z^2 \sum_{\substack{j=1 \\ j \neq i}}^n \left(\frac{x_j^k}{x_i^k}\right)^2\right). \quad (1)$$

## Corollary 9.

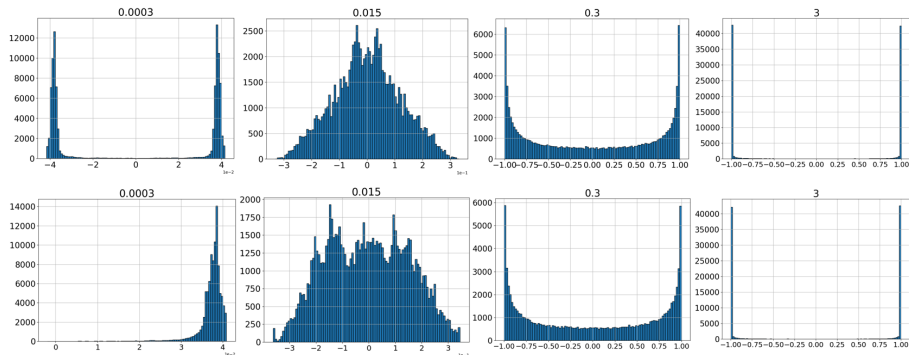
*Let  $\epsilon > 0$  be given. Suppose that  $\{a_n\}_{n=1}^{\infty}$  be a positive real sequence such that only finitely many elements are lower than  $1 + \epsilon$ . Then for any  $x \in \mathbb{R} \setminus \{0\}$*

$$\lim_{m \rightarrow \infty} |\Phi_m(x)| \geq \xi_{1+\epsilon}$$

- By Corollary 9, a too large  $\sigma_z$  causes activation saturation.
- By Equation (1), a too small  $\sigma_z$  reduces activation values as the layer depth increases.

Therefore, we experimentally found an optimal  $\sigma_z = \alpha / \sqrt{N^{\ell-1}}$ , with  $\alpha = 0.085$ , that is neither too large nor too small.

# Preventing Activation Saturation with $\sigma_z$ Tuning



**Figure 11:** The activation values in the 1000<sup>th</sup> layer, with 32 nodes per hidden layer, were analyzed using the proposed weight initialization method with  $\sigma_z$  values of 0.0003, 0.015, 0.3, and 3. The **upper row** shows results for 3000 input samples drawn from a standard normal distribution, while the **lower row** presents results for samples drawn from a Beta distribution with parameters  $a = 2.0$  and  $b = 5.0$ .

## 3-1. Computational Results

# Width Independence In Classification Task

Dataset	Method	2 Nodes		8 Nodes		32 Nodes		128 Nodes		512 Nodes	
		Accuracy	Loss	Accuracy	Loss	Accuracy	Loss	Accuracy	Loss	Accuracy	Loss
MNIST	Xavier	49.78	1.632	68	0.958	91.67	0.277	95.45	0.154	<u>97.35</u>	0.087
	Proposed	<b>62.82</b>	<b>1.185</b>	<b>77.95</b>	<b>0.706</b>	<b>92.51</b>	<b>0.255</b>	<b>96.12</b>	<b>0.134</b>	<b><u>97.96</u></b>	<b>0.067</b>
FMNIST	Xavier	42.89	1.559	68.55	0.890	81.03	0.533	86.20	0.389	<u>88.28</u>	0.331
	Proposed	<b>51.65</b>	<b>1.324</b>	<b>71.31</b>	<b>0.777</b>	<b>83.06</b>	<b>0.475</b>	<b>87.12</b>	<b>0.359</b>	<b><u>88.59</u></b>	<b>0.323</b>
CIFAR-10	Xavier	32.82	1.921	43.51	1.608	48.62	1.473	47.58	1.510	<u>51.71</u>	1.369
	Proposed	<b>38.16</b>	<b>1.780</b>	<b>47.04</b>	<b>1.505</b>	<b>48.80</b>	<b>1.463</b>	<b>48.51</b>	<b>1.471</b>	<b><u>52.21</u></b>	<b>1.359</b>
CIFAR-100	Xavier	10.87	4.065	18.53	3.619	23.71	3.301	<u>23.83</u>	3.324	17.72	3.672
	Proposed	<b>15.22</b>	<b>3.818</b>	<b>23.07</b>	<b>3.350</b>	<b><u>24.93</u></b>	<b>3.237</b>	<b>24.91</b>	<b>3.240</b>	<b>22.80</b>	<b>3.435</b>

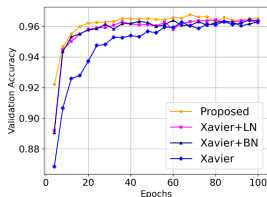
**Table 4:** Validation accuracy and loss are presented for FFNNs with varying numbers of nodes (2, 8, 32, 128, 512), each with 20 hidden layers using tanh activation function. All models were trained for 20 epochs, and the highest average accuracy and lowest average loss, computed from 10 runs, are presented.

# Depth Independence in Classification Task

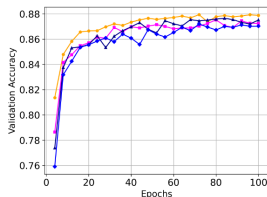
Dataset	Method	3 Layers		10 Layers		50 Layers		100 Layers	
		Accuracy	Loss	Accuracy	Loss	Accuracy	Loss	Accuracy	Loss
MNIST	Xavier	95.98	0.130	96.55	0.112	<u>96.57</u>	0.123	94.08	0.194
	Proposed	<b>96.32</b>	<b>0.123</b>	<u><b>97.04</b></u>	<b>0.102</b>	<b>96.72</b>	<b>0.109</b>	<b>96.06</b>	<b>0.132</b>
FMNIST	Xavier	85.91	0.401	<u>88.73</u>	0.319	87.72	0.344	83.41	0.463
	Proposed	<b>86.51</b>	<b>0.379</b>	<u><b>89.42</b></u>	<b>0.305</b>	<b>88.51</b>	<b>0.324</b>	<b>86.01</b>	<b>0.382</b>
CIFAR-10	Xavier	42.91	1.643	<u>48.39</u>	1.468	47.87	1.474	46.71	1.503
	Proposed	<b>45.05</b>	<b>1.588</b>	<b>48.41</b>	<b>1.458</b>	<b>48.71</b>	<b>1.461</b>	<b>48.96</b>	<b>1.437</b>
CIFAR-100	Xavier	19.10	3.628	22.73	3.400	<u>24.27</u>	3.283	20.32	3.515
	Proposed	<b>19.30</b>	<b>3.609</b>	<b>23.83</b>	<b>3.309</b>	<u><b>25.07</b></u>	<b>3.190</b>	<b>24.41</b>	<b>3.234</b>

**Table 5:** Validation accuracy and loss are presented for FFNNs with varying numbers of layers (3, 10, 50, 100), each with 64 number of nodes using the tanh activation function. All models were trained for 40 epochs, and the highest average accuracy and lowest average loss, computed from 10 runs, are presented.

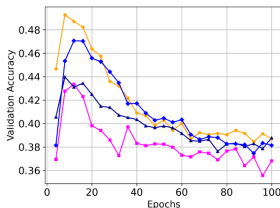
# Normalization in Classification Task



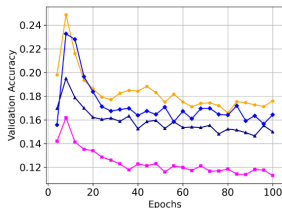
(a) MNIST



(b) Fashion MNIST



(c) CIFAR10



(d) CIFAR100

**Table 6:** Validation accuracy for a tanh FFNN with 50 hidden layers (32 nodes each).

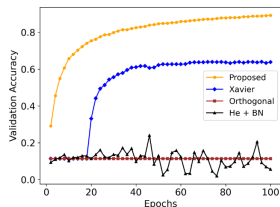
# Dataset Efficiency in Classification Task

Dataset	Method	10		20		30		50		100	
		Accuracy	Loss	Accuracy	Loss	Accuracy	Loss	Accuracy	Loss	Accuracy	Loss
MNIST	Xavier	31.13	2.281	35.03	2.078	45.05	1.771	58.45	1.227	64.02	1.139
	Xavier + BN	22.46	2.267	33.73	2.053	37.13	2.042	39.78	1.944	57.51	1.464
	Xavier + LN	28.52	2.411	41.54	1.796	41.94	1.886	54.97	1.362	65.11	1.093
	Proposed	<b>37.32</b>	<b>2.204</b>	<b>46.79</b>	<b>1.656</b>	<b>48.60</b>	<b>1.645</b>	<b>61.54</b>	<b>1.131</b>	<b>68.44</b>	<b>1.043</b>
FMNIST	Xavier	36.16	2.320	41.69	1.814	53.86	1.459	64.53	1.140	63.58	1.048
	Xavier + BN	35.44	<b>2.136</b>	38.58	1.925	40.16	1.819	53.93	1.728	59.78	1.237
	Xavier + LN	34.94	2.362	37.90	1.793	53.27	1.470	59.50	1.198	62.01	1.073
	Proposed	<b>37.31</b>	2.217	<b>49.25</b>	<b>1.651</b>	<b>55.19</b>	<b>1.372</b>	<b>66.14</b>	<b>1.057</b>	<b>67.58</b>	<b>0.914</b>

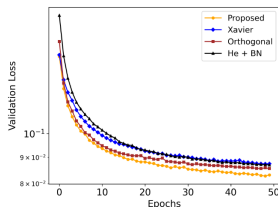
**Table 7:** Validation accuracy and loss for a 10-layer FFNN (64 nodes per layer) trained on datasets of sizes 10, 20, 30, 50, and 100. Results show the highest average accuracy and lowest average loss over 5 runs after 100 epochs.



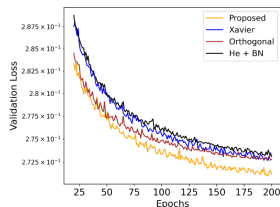
# Non-uniform Hidden Layer Dimensions



(a) FFNN (MNIST)



(b) Autoencoder (MNIST)



(c) Autoencoder (FMNIST)

**Table 8:** **(a)** Validation loss for an FFNN with alternating hidden layers of 16 and 4 nodes, repeated 50 times, comparing four methods: Tanh with Xavier initialization, Tanh with the proposed initialization, ReLU with He initialization + BN, and ReLU with orthogonal initialization. **(b)** Validation loss for an autoencoder with encoder-decoder layers of 512, 256, 128, and 64 units, comparing the same four methods. **(c)** Same as (b), but on the FMNIST dataset.

A **Physics-Informed Neural Network (PINN)** [19] integrates physical laws, such as PDEs, into the training process to ensure the model adheres to these constraints. This approach allows PINNs to solve scientific problems efficiently, even with limited data.

## Total Loss Function

The total loss  $\mathcal{L}_{\text{total}}$  is a weighted sum of all components:

$$\mathcal{L}_{\text{total}} = \lambda_1 \mathcal{L}_{\text{data}} + \lambda_2 \mathcal{L}_{\text{physics}} + \lambda_3 \mathcal{L}_{\text{boundary}} + \lambda_4 \mathcal{L}_{\text{initial}}$$

where  $\lambda_1, \lambda_2, \lambda_3, \lambda_4$  are weights to balance each term.

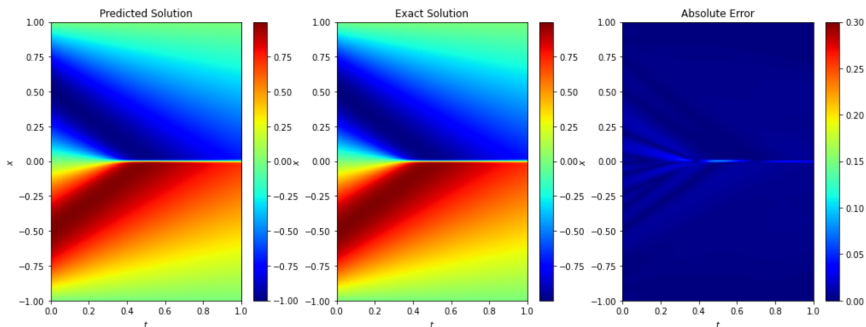
# Physics-Informed Neural Networks

## Example: Burgers Equation

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2}, \quad x \in [-1, 1], \quad t \in [0, 1]$$

with the Dirichlet boundary conditions and initial conditions

$$u(-1, t) = u(1, t) = 0, \quad u(x, 0) = -\sin(\pi x).$$



# Network Size Independence in PINN

Allen-Cahn (16 Nodes)	5	10	20	30	40	50	60	80
Xavier	9.58e-04	8.16e-04	<u>7.61e-04</u>	1.06e-03	1.1e-03	1.24e-03	3.55e-03	1.81e-03
Xavier + BN	1.42e-03	8.17e-04	8.56e-04	<u>7.07e-04</u>	7.77e-04	8.87e-04	9.11e-04	2.15e-03
Xavier + LN	6.29e-01	1.77e-03	6.98e-04	1.27e-03	1.82e-03	6.65e-01	3.29e-01	5.86e-01
Proposed	<b>9.21e-04</b>	<b>7.29e-04</b>	<b>5.76e-04</b>	<b>5.29e-04</b>	<b>5.37e-04</b>	<b>4.03e-04</b>	<b>4.73e-04</b>	<b>5.77e-04</b>
Burgers (16 Nodes)	5	10	20	30	40	50	60	80
Xavier	<u>6.97e-03</u>	1.11e-02	7.9e-03	9.71e-03	2.45e-02	2.65e-02	6.5e-02	5.71e-02
Xavier + BN	<u>8.07e-03</u>	7.72e-03	<u>6.24e-03</u>	1.70e-02	1.50e-02	1.85e-02	2.91e-02	6.84e-02
Xavier + LN	3.89e-02	1.88e-02	9.48e-03	<u>9.28e-03</u>	2.46e-02	3.30e-02	6.91e-02	4.42e-02
Proposed	<b>6.19e-03</b>	<b>5.08e-03</b>	<b>5.28e-03</b>	<b>9.31e-04</b>	<b>3.56e-03</b>	<b>8.27e-04</b>	<b>3.43e-04</b>	<b>2.05e-03</b>
Diffusion (16 Nodes)	5	10	20	30	40	50	60	80
Xavier	<u>2.52e-03</u>	4.82e-03	9.69e-03	1.33e-02	2.08e-02	1.50e-02	2.92e-02	7.24e-02
Xavier + BN	<u>2.89e-03</u>	5.77e-03	1.05e-02	9.65e-03	2.76e-02	1.07e-02	9.07e-03	1.43e-02
Xavier + LN	<u>1.72e-03</u>	6.10e-03	8.04e-03	9.48e-03	2.14e-02	7.59e-03	2.05e-02	2.21e-02
Proposed	<b>9.14e-04</b>	<b>2.59e-03</b>	<b>2.40e-03</b>	<b>1.01e-03</b>	<b>1.97e-03</b>	<b>1.21e-03</b>	<b>1.12e-03</b>	<b>1.91e-03</b>
Poisson (16 Nodes)	5	10	20	30	40	50	60	80
Xavier	<u>1.52e-02</u>	2.87e-02	1.28e-01	9.82e-02	1.15e-01	1.37e-01	1.82e-01	2.55e-01
Xavier + BN	<u>1.62e-02</u>	2.02e-02	8.72e-02	1.12e-01	2.45e-01	9.85e-02	1.00e-01	1.34e-01
Xavier + LN	5.39e-01	<u>4.40e-02</u>	1.34e-01	3.91	2.52e+02	2.58	9.79e+02	nan
Proposed	<b>1.37e-02</b>	<b>1.70e-02</b>	<b>4.62e-02</b>	<b>2.43e-02</b>	<b>3.75e-02</b>	<b>4.03e-02</b>	<b>6.07e-02</b>	<b>6.01e-02</b>

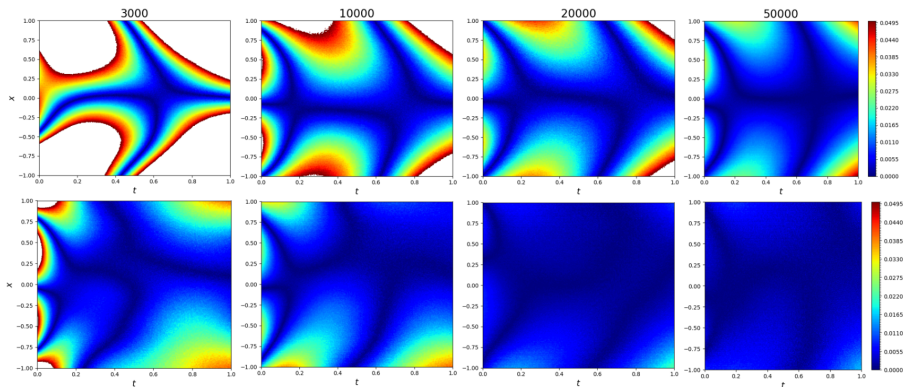
Figure 13: A PINN loss is presented for FFNNs with varying numbers of layers (5, 10, 20, 30, 40, 50, 60, 80) using the tanh activation function.

# Network Size Independence in PINN

Allen-Cahn (32 Nodes)	5	10	20	30	40	50	60	80
Xavier	3.13e-01	5.03e-02	3.64e-03	<u>2.37e-03</u>	4.03e-03	5.27e-03	1.73e-02	6.94e-01
Xavier + BN	4.05e-01	8.85e-04	8.41e-04	7.82e-04	9.97e-04	<u>6.80e-04</u>	9.34e-04	6.94e-01
Xavier + LN	3.31e-01	2.10e-03	<u>5.99e-04</u>	6.71e-04	1.49e-03	1.29e-03	3.31e-02	6.93e-01
Proposed	<b>1.04e-03</b>	<b>6.92e-04</b>	<b>5.34e-04</b>	<b>4.26e-04</b>	<b>3.31e-04</b>	<b>3.52e-04</b>	<b>3.85e-04</b>	<b>5.96e-04</b>
Burgers (32 Nodes)	5	10	20	30	40	50	60	80
Xavier	1.12e-02	3.53e-03	2.72e-03	<u>1.81e-03</u>	7.60e-03	8.56e-03	9.86e-03	1.66e-01
Xavier + BN	5.88e-03	<b>1.04e-03</b>	1.79e-03	2.80e-03	5.95e-03	3.66e-02	6.60e-02	1.66e-01
Xavier + LN	4.31e-02	1.21e-02	<u>1.88e-03</u>	7.22e-03	5.54e-03	8.46e-03	9.04e-03	4.86e-02
Proposed	<b>4.14e-03</b>	4.11e-03	<b>1.58e-03</b>	<b>1.29e-03</b>	<b>7.96e-04</b>	<b>5.85e-04</b>	<b>9.80e-04</b>	<b>1.47e-03</b>
Diffusion (32 Nodes)	5	10	20	30	40	50	60	80
Xavier	<u>1.69e-03</u>	6.85e-03	7.63e-03	4.50e-03	8.98e-03	5.67e-03	6.33e-01	1.59
Xavier + BN	<u>1.68e-03</u>	2.66e-03	1.08e-02	6.00e-03	8.58e-03	6.60e-03	5.66e-02	1.69e+02
Xavier + LN	<u>8.16e-04</u>	2.85e-03	8.46e-03	4.57e-03	9.40e-03	1.04e-02	2.42e-01	1.67e+02
Proposed	<b>2.89e-04</b>	<b>8.03e-04</b>	<b>5.25e-04</b>	<b>5.07e-04</b>	<b>5.33e-04</b>	<b>6.17e-04</b>	<b>9.80e-04</b>	<b>1.53e-03</b>
Poisson (32 Nodes)	5	10	20	30	40	50	60	80
Xavier	<u>1.09e-02</u>	1.33e-02	3.13e-02	7.69e-02	6.72e-02	8.90e-02	9.68e+02	1.46e+02
Xavier + BN	<u>1.14e-02</u>	1.47e-02	2.68e-02	3.55e-02	8.25e-02	8.97e-02	4.50e-02	7.75e-01
Xavier + LN	2.36e-02	<u>2.18e-02</u>	3.07e-02	3.85e-01	1.40	4.69	2.60	6.14
Proposed	<b>9.63e-03</b>	<b>8.29e-03</b>	<b>1.41e-02</b>	<b>1.88e-02</b>	<b>1.65e-02</b>	<b>1.85e-02</b>	<b>1.73e-02</b>	<b>3.59e-02</b>

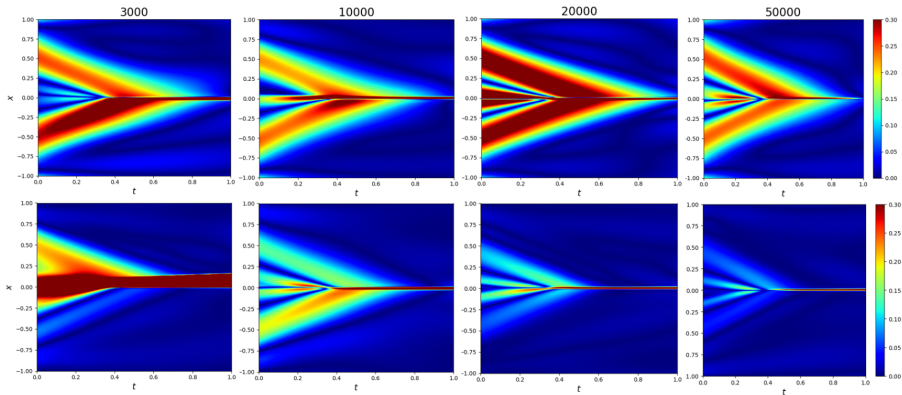
Figure 14: A PINN loss is presented for FFNNs with varying numbers of layers (5, 10, 20, 30, 40, 50, 60, 80) using the tanh activation function.

# Dataset Efficiency in PINN



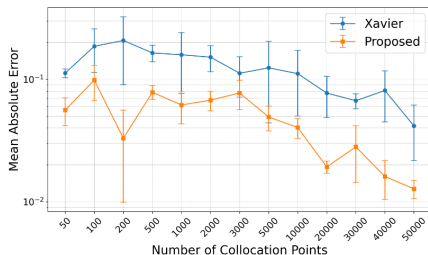
**Figure 15:** Absolute error between the exact solution and the PINN-predicted solution for the Diffusion equation with varying numbers of collocation points (3000, 10000, 20000, 50000) using **(upper row)** Xavier and **(lower row)** the proposed initialization. The FFNN has 30 hidden layers (32 nodes each) and is trained for 300 iterations using Adam followed by 300 iterations using L-BFGS.

# Dataset Efficiency in PINN

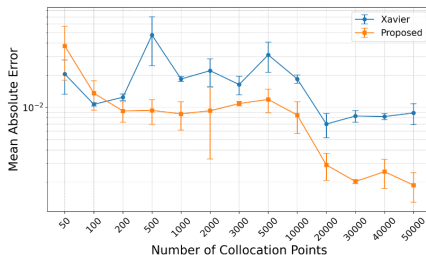


**Figure 16:** Absolute error between the exact solution and the PINN-predicted solution for the Burgers' equation with varying numbers of collocation points (3000, 10000, 20000, 50000) using **(upper row)** Xavier and **(lower row)** the proposed initialization. The FFNN has 30 hidden layers (32 nodes each) and is trained for 300 iterations using Adam followed by 300 iterations using L-BFGS.

# Dataset Efficiency in PINN



(a) Burgers Equation



(b) Diffusion Equation

**Figure 17:** Mean absolute error between the exact solution and PINN-predicted solution with varying numbers of collocation points. The FFNN has 30 hidden layers (32 nodes each) and is trained for 300 iterations using Adam followed by 300 iterations using L-BFGS. The results are averaged over 5 experiments.



## 4. Future Works

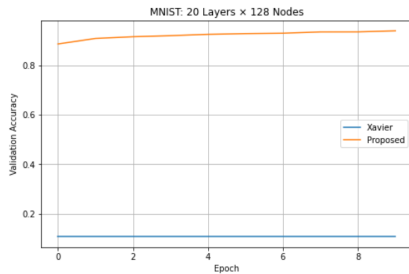
## Major Topic

Proposed a method to determine an appropriate weight initialization for a given activation function.

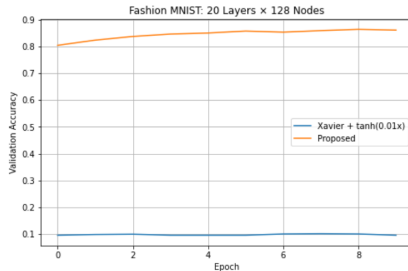
## Activation function properties

- 1  $f \in C^1(\mathbb{R})$
- 2  $f(-x) = -f(x)$ , hence  $f(0) = 0$
- 3  $\lim_{x \rightarrow \pm\infty} f(x) = \pm L$  for some  $L < \infty$
- 4  $f'(x) > 0 \quad \forall x \in \mathbb{R}$

When using an activation function that satisfies certain conditions



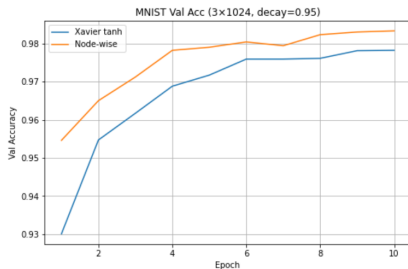
(a) MNIST



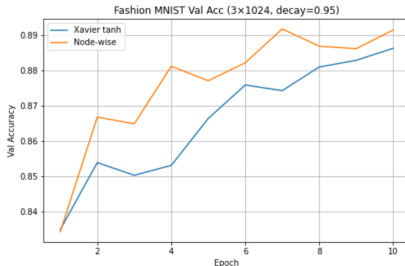
(b) Fashion MNIST

**Figure 18:** Validation accuracy of an FFNN with 3 hidden layers of 1024 nodes each.

Training was conducted under the optimal learning rate setting.



(a) MNIST



(b) Fashion MNIST

**Figure 19:** Validation accuracy of a feedforward network with 20 hidden layers of 128 nodes each, using an activation function satisfying the proposed conditions.

# References I

- [1] Narkhede, Meenal V., Prashant P. Bartakke, and Mukul S. Sutaone. (2022) A review on weight initialization strategies for neural networks." Artificial intelligence review 55.1: 291-322.
- [2] Kumar, Siddharth Krishna. (2017) On weight initialization in deep neural networks. arXiv preprint arXiv:1704.08863 (2017).
- [3] He, J., Li, L., Xu, J., & Zheng, C. (2018). ReLU deep neural networks and linear finite elements. arXiv preprint arXiv:1807.03973.
- [4] Hanin, B., & Sellke, M. Approximating continuous functions by ReLU nets of minimal width (2018). arXiv preprint arXiv:1710.11278.
- [5] Yarotsky, D. (2017). Error bounds for approximations with deep ReLU networks. Neural Networks, 94, 103-114.

- [6] A. T. Puig, A. Wiesel, G. Fleury, and A. O. Hero, "Multidimensional shrinkage-thresholding operator and group LASSO penalties," *IEEE Signal Process. Lett.*, vol. 18, no. 6, pp. 363–366, Jun. 2011.
- [7] Apicella, A., Donnarumma, F., Isgrò, F., & Prevete, R. (2021). A survey on modern trainable activation functions. *Neural Networks*, 138, 14-32.
- [8] Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.
- [9] Glorot, X., & Bengio, Y. (2010, March). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (pp. 249-256). *JMLR Workshop and Conference Proceedings*.

# References III

- [10] Lu, L., Shin, Y., Su, Y., & Karniadakis, G. E. (2019). Dying relu and initialization: Theory and numerical examples. arXiv preprint arXiv:1903.06733.
- [11] Burkholz, R., & Dubatovka, A. (2019). Initialization of relus for dynamical isometry. Advances in Neural Information Processing Systems, 32.
- [12] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE international conference on computer vision (pp. 1026-1034).
- [13] 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.

# References IV

- [14] Zhao, J., Schäfer, F., & Anandkumar, A. (2021). Zero initialization: Initializing neural networks with only zeros and ones. arXiv preprint arXiv:2110.12661.
- [15] Lee, H., Kim, Y., Yang, S., & Choi, H. (2024). Improved weight initialization for deep and narrow feedforward neural network. arXiv preprint arXiv:2311.03733.
- [16] Rathore, Pratik, et al. (2024) Challenges in training PINNs: A loss landscape perspective. ICML2024.
- [17] Poole, Ben, et al. (2016) Exponential expressivity in deep neural networks through transient chaos.” Advances in neural information processing systems 29.
- [18] Raghu, Maithra, et al. (2017) On the expressive power of deep neural networks.” International conference on machine learning. PMLR.



- [19] Karniadakis, George Em, et al. (2021) Physics-informed machine learning. Nature Reviews Physics 3.6.
- [20] Lee, H., Choi, H., & Kim, H. (2024). Robust Weight Initialization for Tanh Neural Networks with Fixed Point Analysis. arXiv preprint arXiv:2410.02242.

Thank you for your attention!