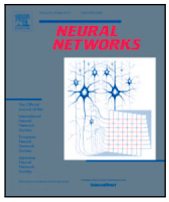
Neural Networks 176 (2024) 106362

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet



Full Length Article

# Improved weight initialization for deep and narrow feedforward neural network

Hyunwoo Lee [a], Yunho Kim [b], Seung Yeop Yang [a,c], Hayoung Choi [a,*]

[a] Department of Mathematics, Kyungpook National University, Daegu 41566, Republic of Korea
[b] Department of Mathematical Sciences, Ulsan National Institute of Science and Technology, Ulsan 44919, Republic of Korea
[c] KNU LAMP Research Center, KNU Institute of Basic Sciences, Kyungpook National University, Daegu, 41566, Republic of Korea

ARTICLE INFO

*Keywords:*
Weight initialization
Initial weight matrix
Deep learning
Feedforward neural networks
ReLU activation function

ABSTRACT

Appropriate weight initialization settings, along with the ReLU activation function, have become cornerstones of modern deep learning, enabling the training and deployment of highly effective and efficient neural network models across diverse areas of artificial intelligence. The problem of "dying ReLU," where ReLU neurons become inactive and yield zero output, presents a significant challenge in the training of deep neural networks with ReLU activation function. Theoretical research and various methods have been introduced to address the problem. However, even with these methods and research, training remains challenging for extremely deep and narrow feedforward networks with ReLU activation function. In this paper, we propose a novel weight initialization method to address this issue. We establish several properties of our initial weight matrix and demonstrate how these properties enable the effective propagation of signal vectors. Through a series of experiments and comparisons with existing methods, we demonstrate the effectiveness of the novel initialization method.

## 1. Introduction

Training neural networks have enabled dramatic advances across a wide variety of domains, notably image recognition (Krizhevsky, Sutskever, & Hinton, 2012), natural language processing (Radford, Narasimhan, Salimans, & Sutskever, 2018) and generative models (Goodfellow et al., 2014). Numerous well-known neural networks belong to the family of feedforward neural networks (FFNNs), which are used for input–output mapping. Traditionally, FFNN connection weights are optimized by the back-propagation algorithm (Rumelhart, Hinton, & Williams, 1986). In the early stages of research on FFNNs, the networks with one or a limited number of hidden layers, which are now referred to as shallow networks, were common. Consequently, extensive research was conducted to understand their properties. Notably, these networks have been demonstrated to serve as general function approximators (Cybenko, 1989; Hornik, 1991; Hornik, Stinchcombe, & White, 1989; Leshno, Lin, Pinkus, & Schocken, 1993). Deeper networks, with their multiple hidden layers, have shown enhanced performance in tasks requiring high levels of pattern recognition, such as image and speech analysis (Srivastava, Greff, & Schmidhuber, 2015). However, as the depth of FFNNs increases, the problem of the vanishing gradient becomes more pronounced (Bengio, Simard, & Frasconi, 1994). This occurs because the network weights receive updates based on the derivative of the error during training. In certain situations, the gradient becomes extremely small, making it almost impossible for weights to change, and in severe cases, it can halt the training process altogether.

The rectified linear unit (ReLU) is one of the most widely-used activation functions in the field of deep learning due to its superior training performance compared to other activation functions (Sun, Wang, & Tang, 2015). The phenomenon known as "dying ReLU" is a type of vanishing gradient issue when ReLU neurons become inactive and an output of 0 for any input (Nair & Hinton, 2010). It has been widely recognized as a major obstacle to training deep neural networks with ReLU activation function (Agarap, 2018; Trottier, Giguere, & Chaib-Draa, 2017). To address this issue, several methods have been introduced. These methods can be broadly classified into three general approaches. The first approach is to change network architectures, including the activation functions (Apicella, Donnarumma, Isgrò, & Prevete, 2021; Clevert, Unterthiner, & Hochreiter, 2015; Dubey, Singh, & Chaudhuri, 2022; Duch & Jankowski, 1999). Another approach involves various normalization techniques (Ba, Kiros, & Hinton, 2016; Ioffe & Szegedy, 2015; Salimans & Kingma, 2016). The third approach specifically is to study the weights and biases initialization with fixed network architectures (Glorot & Bengio, 2010; He, Zhang, Ren, & Sun,

* Corresponding author.
*E-mail addresses:* lhw908@knu.ac.kr (H. Lee), yunhokim@unist.ac.kr (Y. Kim), seungyeop.yang@knu.ac.kr (S.Y. Yang), hayoung.choi@knu.ac.kr (H. Choi).

https://doi.org/10.1016/j.neunet.2024.106362
Received 20 October 2023; Received in revised form 3 April 2024; Accepted 29 April 2024
Available online 3 May 2024

2015; Saxe, McClelland, & Ganguli, 2013). The third approach is the topic of our research in this paper.

Numerous papers have discussed various weight initialization methods for neural networks and emphasized their importance (Sutskever, Martens, Dahl, & Hinton, 2013). The most popular weight initializations are Xavier initialization (Glorot & Bengio, 2010) and Kaiming initialization (He et al., 2015). Both methods adjust the variance of the initial weight matrix to prevent the vanishing/exploding problem, enabling deeper networks to be trained. Saxe et al. (2013) discuss an orthogonal initialization method based on an orthonormal basis. ZerO initialization (Zhao, Schäfer, & Anandkumar, 2021) which is fully deterministic initialization has benefits in training extremely deep neural networks without batch normalization. ZerO initialization utilizes Hadamard transforms to break the training degeneracy. For more details, see the review paper (Narkhede, Bartakke, & Sutaone, 2022) and references therein.

Although deep and wide networks are popular and the most successful in practice, deep and wide networks need high computational costs to train a huge number of parameters. On the other hand, deep and narrow networks also play important roles theoretically and practically. As demonstrated in He, Li, Xu, and Zheng (2018), deep and narrow ReLU networks are essential when creating finite element basis functions. This application highlights the use of deep ReLU networks in finite element methods for solving partial differential equations. Additionally, various theoretical studies (Cai, 2022; Hanin & Sellke, 2017; Park, Yun, Lee, & Shin, 2020; Petersen & Voigtlaender, 2018; Yarotsky, 2017) exploring the expressive power of ReLU networks heavily depends on deep and narrow networks for approximating polynomials through sparse concatenations.

Weight initialization methods have been developed to prevent the dying ReLU problems in deep and narrow FFNNs with the ReLU activation function. Lu, Shin, Su, and Karniadakis (2019) provided rigorous proof that as the depth of a deep FFNNs with ReLU activation function approaches infinity, it will eventually become inactive with a certain probability. Then they propose a randomized asymmetric initialization (RAI) designed to prevent the dying ReLU problem effectively. Burkholz and Dubatovka (2019) calculated the precise joint signal output distribution for FFNNs with Gaussian weights and biases, without relying on mean field assumptions, and analyzed deviations from the mean field results. They further discussed the limitations of the standard initialization method, such as its lack of dynamical isometry, and proposed a simple alternative weight initialization method, namely, the Gaussian submatrix initialization (GSM). These studies have improved training performance in deep and narrow feedforward ReLU networks. Despite these advancements, our experiments show that existing methods did not perform well in extremely deep or narrow scenarios. To overcome the problem, this article proposes a novel weight initialization method for FFNNs with ReLU activation functions. The proposed weight initialization has several properties such as orthogonality, positive entry predominance, and fully deterministic. Furthermore, due to the properties of the proposed initial weight matrix, it effectively transmits signals even in deep and narrow FFNNs with ReLU activation.

We empirically benchmarked our proposed weight initialization method on MNIST (LeCun, Bottou, Bengio, & Haffner, 1998) and Fashion MNIST datasets comparing to other weight initialization methods such as Xavier (Glorot & Bengio, 2010), He (He et al., 2015), Orthogonal (Saxe et al., 2013), Identity, ZerO (Zhao et al., 2021), RAI (Lu et al., 2019), and GSM (Burkholz & Dubatovka, 2019). Initially, we applied our proposed weight initialization method to various dataset sizes of FFNN models using ReLU activation functions. Our method significantly improves validation accuracy in the models with no hidden layers or in narrower networks with fewer nodes, clearly outperforming other initialization methods. Moreover, various computational experiments show that the proposed method holds depth independence, width independence, and activation function independence. For depth independence, experiments were conducted on both the MNIST and Fashion MNIST datasets and tabular datasets like the Wine Quality dataset (Cortez, Cerdeira, Almeida, Matos, & Reis, 2009), and the Iris dataset (Fisher, 1988). It demonstrated that the proposed method performs well for depth independence, excelling in training deep feedforward neural networks across different numbers of layers. Also, it was shown that our method holds width independence, effectively training networks with various numbers of nodes per layer. It achieved robust validation accuracy and rapid convergence, even in network configurations that traditionally challenge other weight initialization methods. Moreover, our method demonstrated independence from activation functions in the ReLU family. The preceding experiments underscore that the proposed initialization method is more independent of network architecture.

### A. Contributions

In this paper, we propose a novel weight initialization method in extremely deep and narrow feedforward neural networks (FFNNs) with a rectified linear unit (ReLU) activation function. The main contributions of this paper are summarized as follows.

- We propose a novel weight initialization method that prevents the dying ReLU problem in extremely deep and narrow FFNNs with ReLU activation function.
- We analyze the properties of the proposed initial weight matrix. We demonstrated orthogonality and the absolute value of column sum of $\mathbf{Q}^\epsilon$ is less or equal to $\epsilon$. Furthermore, we show that $\mathbf{W}^\epsilon$ with a constant row (or column) sum. We also show that $\mathbf{W}^\epsilon\mathbf{x}$ has more positive entries.
- We conducted experiments applying our proposed method and existing methods in various scenarios.

### B. Organization and notations

The remainder of this paper is organized as follows. In Section 2, we present existing weight initialization methods and introduce our proposed weight initialization method. Next, various properties of the proposed initial weight matrix are provided in Section 3. Section 4 presents simulation results. Finally, conclusions are drawn in Section 5.

**Notations**: Let $\mathbb{R}$ be the set of real numbers and $\mathbb{R}_+$ be the set of nonnegative real numbers. The standard inner product of two vectors $\mathbf{u}$ and $\mathbf{v}$ is denoted by $\langle \mathbf{u}, \mathbf{v} \rangle$, and $\|\mathbf{v}\|$ denotes the Euclidean norm. Denote the $m \times n$ matrix whose all entries are ones as $\mathbf{J}_{m\times n}$ and denote the $m \times n$ matrix with ones on the main diagonal and zeros elsewhere as $\mathbf{I}_{m\times n}$. For $m = n$ we simply denote $\mathbf{I}_m$ and $\mathbf{J}_m$ instead of $\mathbf{I}_{m\times m}$ and $\mathbf{J}_{m\times m}$, respectively. Denote $\mathbf{1}_m = [1\ 1\ \cdots\ 1]^T \in \mathbb{R}^m$. However, if the size is clear from context, we will drop $m$ from our notation for brevity. $\mathbf{e}_j$ ($j = 1, \ldots, m$) denotes the vector in $\mathbb{R}^m$ with a 1 in the $j$th coordinate and 0's elsewhere. $\mathcal{O}(\cdot)$ represents the big $O$ notation.

## 2. Methodology

Before introducing our proposed weight initialization method, we briefly give basic concepts and prior work.

### 2.1. Basic conceptions

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 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. The result $\mathbf{y}_i$ will be a vector with continuous values in the case of regression problems, a binary one-hot vector for classification problems, and so forth. An FFNN with $L$ layers performs cascaded computations of

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

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. To gain good estimation of $\mathbf{y}$ for any test sample $\mathbf{x}$, FFNNs optimization aims to find optimal solutions of network parameters $\Theta = \{\mathbf{W}^{\ell}, \mathbf{b}^{\ell}\}_{\ell=1}^{L}$. In other words, training is the process of solving the following equation:

$$\min_{\Theta} \mathcal{L}(\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{K}; \Theta),$$

where $\mathcal{L}$ is a training loss function.

The network parameters $\Theta = \{\mathbf{W}^{\ell}, \mathbf{b}^{\ell}\}_{\ell=1}^{L}$ are usually optimized using gradient descent. The gradient descent updates the network parameter with an initialization as follows: for each $t = 0, 1, 2, \dots$,

$$\mathbf{W}^{\ell}(t+1) = \mathbf{W}^{\ell}(t) - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{\ell}(t)} \quad (\ell = 1, \dots, L),$$

$$\mathbf{b}^{\ell}(t+1) = \mathbf{b}^{\ell}(t) - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{\ell}(t)} \quad (\ell = 1, \dots, L),$$

where $\eta > 0$ is the learning rate. There exist variants of gradient descents such as stochastic gradient descents (SGD), ADAM (Kingma & Ba, 2014), AdaGrad (Duchi, Hazan, & Singer, 2011), and so on.

### 2.2. Prior work

Weight initialization plays a critical role in training neural networks, significantly influencing model convergence and learning performance (Narkhede et al., 2022). Selecting an appropriate initialization method is vital for improving a model's efficiency and performance. These initialization methods affect the convergence rate and training stability of learning algorithms like gradient descent. Among well-known approaches are the Xavier and He initialization methods. These research efforts involve scaling the initial weights to maintain the variance of input and output layers or control the variance of the output layer to a desired value. They also focus on preserving the variance of gradients during training, all of which contribute to more effective and stable neural network training. However, choosing the right variance for weight initialization becomes increasingly complex, particularly with a growing number of layers. Addressing these challenges, Zhao et al. (2021) introduced ZerO, a fully deterministic initialization method. The method initializes network weights to either zeros or ones. This novel method is grounded in identity and Hadamard transforms, serving as a replacement for the traditional random weight initialization. ZerO offers numerous advantages, including the ability to train exceptionally deep networks without requiring batch normalization. The orthogonal initialization method employs an orthogonal matrix for weight initialization (Saxe et al., 2013). The method ensures that the singular values of the input–output Jacobian are approximately equal to 1. This condition, known as dynamical isometry, allows for consistent learning times that are not dependent on the depth of the neural networks. Although deep and wide networks are effective and popular, they incur high computational costs from their extensive parameters. Conversely, deep and narrow networks hold substantial theoretical and practical significance. They are essential in creating finite element basis functions, particularly in applications like solving partial differential equations, as shown in He et al. (2018). Moreover, a variety of theoretical investigations (Cai, 2022; Hanin & Sellke, 2017; Park et al., 2020; Petersen & Voigtlaender, 2018; Yarotsky, 2017) into the expressive power of ReLU networks rely heavily on deep and narrow networks to approximate polynomials efficiently through sparse concatenations. Yet, the "dying ReLU" problem remains a significant obstacle in training deep and narrow FFNNs. Lu et al. (2019) rigorously proved that as the depth of deep FFNNs with ReLU activation function tends toward infinity, it will eventually become inactive with a certain probability. They also introduced a randomized asymmetric initialization method (RAI) to effectively address the dying ReLU problem. Burkholz and Dubatovka (2019), on the other hand, calculated the precise joint signal output distribution for FFNNs with Gaussian weights and biases. Without relying on mean-field assumptions, they analyzed deviations from the mean-field results and discussed the limitations of the standard weight initialization method. They proposed an alternative weight initialization approach known as Gaussian submatrix initialization (GSM). However, the methods proposed so far have shown limited effectiveness in extremely deep and narrow FFNNs. To address this issue, we propose a new weight initialization method.

### 2.3. Proposed weight initialization method

Our proposed weight initialization method can be characterized by key properties: orthogonality, positive entry predominance, and fully deterministic. Proposition 1 establishes that the proposed initial weight matrix is orthogonal. Orthogonal weight initialization, extensively studied both theoretically and empirically, has been shown to accelerate convergence in deep linear networks through the attainment of dynamical isometry (Advani, Saxe, & Sompolinsky, 2020; Hu, Xiao, & Pennington, 2020; Saxe et al., 2013). Our method demonstrates in Proposition 2 that the initial weight matrix's entry sum of each column (resp. row) vector is almost the same. Building on this, Corollary 1 establishes that each $\mathbf{Wx}$ has more positive entries, thereby preventing the dying ReLU problem in deep networks. Finally, the proposed weight initialization is fully deterministic, thus it is not dependent on randomness.

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) $\mathbf{Wx}$ has more positive entries for all $\mathbf{x} \in \mathbb{R}^n_+$;
(iii) $\mathbf{W}$ is a fully deterministic matrix.

To obtain such a matrix we first define $\mathbf{Q}^{\epsilon}_{m\times m}$ by the orthogonal matrix of a QR decomposition of

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

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

To initialize the weights of the neural networks we propose that

$$\mathbf{W}^{\epsilon}_{m\times n} = \left(\mathbf{Q}^{\epsilon}_{m\times m}\right) \mathbf{I}_{m\times n} \left(\mathbf{Q}^{\epsilon}_{n\times n}\right)^T. \tag{1}$$

It is noteworthy that $\mathbf{W}^{\epsilon}_{m\times n}$ can be expressed as

$$\mathbf{W}^{\epsilon}_{m\times n} = \mathbf{q}_1 \hat{\mathbf{q}}_1^T + \mathbf{q}_2 \hat{\mathbf{q}}_2^T + \cdots + \mathbf{q}_s \hat{\mathbf{q}}_s^T, \tag{2}$$

where $s = \min\{m, n\}$, and $\mathbf{q}_1, \dots, \mathbf{q}_m$ are the column vectors of $\mathbf{Q}^{\epsilon}_{m\times m}$ and $\hat{\mathbf{q}}_1, \dots, \hat{\mathbf{q}}_n$ are the column vectors of $\mathbf{Q}^{\epsilon}_{n\times n}$. Note that $\mathbf{q}_1, \dots, \mathbf{q}_m$ are orthonormal vectors in $\mathbb{R}^m$ and $\hat{\mathbf{q}}_1, \dots, \hat{\mathbf{q}}_m$ are orthonormal vectors in $\mathbb{R}^n$. That is, two sets of column vectors are constructed very similarly, but they are defined in different dimensional vector spaces for $m \neq n$. Moreover, $\mathbf{q}_i \hat{\mathbf{q}}_i^T$ is a rank-one matrix for all $i$.

**Remark.** The matrix $\mathbf{J}^{\epsilon} = \mathbf{J}_m + \epsilon \mathbf{I}_m$ is positive definite, specifically, the eigenvalues consist of $\lambda_1 = m + \epsilon$ and $\lambda_2 = \epsilon$ (multiplicity is $m-1$). The corresponding eigenvector of $\lambda_1$ is $\mathbf{1}$ and the corresponding eigenvectors of $\lambda_2$ are the set of independent vectors $\{\mathbf{v}_2, \dots, \mathbf{v}_m\}$ such that $\mathbf{1} \perp \mathbf{v}_i$ for all $i = 2, \dots, m$. For more details on the matrix $\mathbf{J}^{\epsilon}$, see the paper B. (2021), Choi, Kim, Lee, and Lim (2020).

We first give the proposed initial weight matrix $\mathbf{W}^{\epsilon}_{m\times n}$ for small values $m, n$ (see Fig. 1).

**Example 1.** For $\epsilon = 0.01$ the proposed initial weight matrix is computed approximately as follows.

$$\mathbf{W}^{\epsilon}_{3\times 2} = \begin{bmatrix} -0.0829 & 0.9097 \\ 0.9081 & -0.0993 \\ 0.4106 & 0.4032 \end{bmatrix},$$

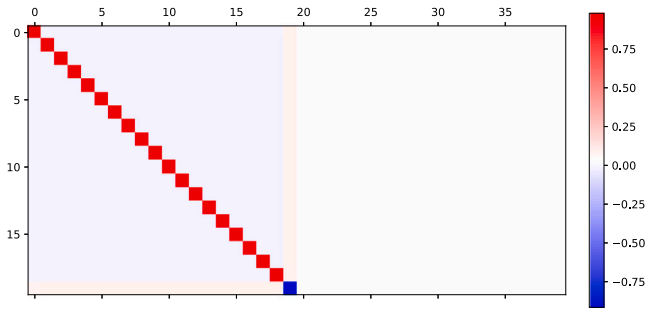



**Fig. 1.** A proposed initial weight matrix $\mathbf{W}^{\epsilon}_{20\times 40}$ is shown via heatmap ($\epsilon = 0.01$). There exists a certain pattern of values for entries of $\mathbf{W}^{\epsilon}_{20\times 40}$.

$$\mathbf{W}^{\epsilon}_{4\times 3} = \begin{bmatrix} 0.6241 & -0.3762 & 0.6213 \\ -0.3754 & 0.6242 & 0.6217 \\ 0.6213 & 0.6209 & -0.3816 \\ 0.2890 & 0.2887 & 0.2862 \end{bmatrix}.$$

**Example 2.** For $\epsilon_1 = 0.0001$ and $\epsilon_2 = 0.1$ the proposed initial weight matrix $\mathbf{W}^{\epsilon}_{m\times n}$ is computed approximately as follows.

$$\mathbf{W}^{\epsilon_1}_{8\times 5} = \begin{bmatrix} 0.8581 & -0.1419 & -0.1419 & -0.1419 & 0.3581 \\ -0.1419 & 0.8581 & -0.1419 & -0.1419 & 0.3581 \\ -0.1419 & -0.1419 & 0.8581 & -0.1419 & 0.3581 \\ -0.1419 & -0.1419 & -0.1419 & 0.8581 & 0.3581 \\ 0.3581 & 0.3581 & 0.3581 & 0.3581 & -0.6419 \\ 0.1581 & 0.1581 & 0.1581 & 0.1581 & 0.1581 \\ 0.1581 & 0.1581 & 0.1581 & 0.1581 & 0.1581 \\ 0.1581 & 0.1581 & 0.1581 & 0.1581 & 0.1581 \end{bmatrix},$$

$$\mathbf{W}^{\epsilon_2}_{8\times 5} = \begin{bmatrix} 0.8618 & -0.1415 & -0.1413 & -0.1413 & 0.3524 \\ -0.1341 & 0.8626 & -0.1374 & -0.1374 & 0.3563 \\ -0.1342 & -0.1373 & 0.8626 & -0.1374 & 0.3563 \\ -0.1342 & -0.1373 & -0.1373 & 0.8626 & 0.3563 \\ 0.3559 & 0.3528 & 0.3528 & 0.3528 & -0.6533 \\ 0.1598 & 0.1567 & 0.1567 & 0.1567 & 0.1506 \\ 0.1598 & 0.1567 & 0.1567 & 0.1567 & 0.1506 \\ 0.1598 & 0.1567 & 0.1567 & 0.1567 & 0.1506 \end{bmatrix}.$$

## 3. Properties of the proposed initial weight matrix

This section presents several key properties of the proposed initial weight matrix, accompanied by rigorous proofs. Initially, Proposition 1 establishes the orthogonality of the proposed initial weight matrix $\mathbf{W}^{\epsilon}$. Furthermore, Theorem 1 introduces an algorithm designed to reduce the computational complexity of $\mathbf{W}^{\epsilon}$. Proposition 2 demonstrates that the column sums and row sums of $\mathbf{W}^{\epsilon}$ are nearly identical. Building on this, Corollary 1 shows that $\mathbf{W}^{\epsilon}\mathbf{x}$ has more positive entries for any vector $\mathbf{x}$ with positive entries.

**Proposition 1.** *Let $\mathbf{q}_1, \ldots, \mathbf{q}_m$ be the column vectors of $\mathbf{Q}^{\epsilon}_{m\times m}$ and $\hat{\mathbf{q}}_1, \ldots, \hat{\mathbf{q}}_n$ be the column vectors of $\mathbf{Q}^{\epsilon}_{n\times n}$. Then it holds that*

*(i) if $m = n$,*

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

*(ii) if $m > n$,*

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

*(iii) if $m < n$,*

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

*(iv) $(\mathbf{W}^{\epsilon}_{m\times n})^T = \mathbf{W}^{\epsilon}_{n\times m}$ for all $m, n$.*

It is easy to verify it. The proof is left to the reader.

Now, we introduce an algorithm that can reduce the computational complexity of calculating $\mathbf{W}^{\epsilon}$. Recall that for given vectors $\mathbf{u}$ and $\mathbf{v}$, the vector projection of $\mathbf{v}$ onto $\mathbf{u}$ is defined as

$$\mathrm{proj}_{\mathbf{u}}\mathbf{v} := \frac{\langle \mathbf{u}, \mathbf{v}\rangle}{\langle \mathbf{u}, \mathbf{u}\rangle}\mathbf{u}.$$

QR decomposition is performed as follows. For a $n \times n$ matrix $\mathbf{A} = [\mathbf{a}_1 \cdots \mathbf{a}_n]$, the QR decomposition is defined as

$$\mathbf{A} = \mathbf{QR} \quad (\mathbf{Q}\text{: orthogonal matrix, } \mathbf{R}\text{: upper triangular matrix}),$$

where

$$\mathbf{Q} = \underbrace{\begin{bmatrix} | & | & & | \\ \mathbf{q}_1 & \mathbf{q}_2 & \cdots & \mathbf{q}_n \\ | & | & & | \end{bmatrix}}_{\text{orthogonal matrix}},$$

$$\mathbf{R} = \underbrace{\begin{bmatrix} \mathbf{q}_1^T\cdot\mathbf{a}_1 & \mathbf{q}_1^T\cdot\mathbf{a}_2 & \cdots & \mathbf{q}_1^T\cdot\mathbf{a}_n \\ 0 & \mathbf{q}_2^T\cdot\mathbf{a}_2 & \cdots & \mathbf{q}_2^T\cdot\mathbf{a}_n \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{q}_n^T\cdot\mathbf{a}_n \end{bmatrix}}_{\text{upper triangular matrix}}.$$

Here, the matrices $\mathbf{Q}$ and $\mathbf{R}$ are generated by the Gram–Schmidt process for the full column rank matrix $\mathbf{A} = [\mathbf{a}_1 \cdots \mathbf{a}_n]$.

$$\mathbf{u}_1 = \mathbf{a}_1, \qquad \mathbf{q}_1 = \frac{\mathbf{u}_1}{\|\mathbf{u}_1\|},$$
$$\mathbf{u}_2 = \mathbf{a}_2 - \mathrm{proj}_{\mathbf{u}_1}\mathbf{a}_2, \qquad \mathbf{q}_2 = \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|},$$
$$\mathbf{u}_3 = \mathbf{a}_3 - \mathrm{proj}_{\mathbf{u}_1}\mathbf{a}_3 - \mathrm{proj}_{\mathbf{u}_2}\mathbf{a}_3, \qquad \mathbf{q}_3 = \frac{\mathbf{u}_3}{\|\mathbf{u}_3\|},$$
$$\vdots$$
$$\mathbf{u}_n = \mathbf{a}_n - \sum_{j=1}^{n-1}\mathrm{proj}_{\mathbf{u}_j}\mathbf{a}_n, \qquad \mathbf{q}_n = \frac{\mathbf{u}_n}{\|\mathbf{u}_n\|}.$$

From the Gram–Schmidt process, we have the following iteration to construct $\mathbf{Q}^{\epsilon}_{m\times m}$.

**Theorem 1.** *Let $\{\mathbf{u}_j\}_{1\le j\le m}$ 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 $j = 2, \ldots, m$. Then $j$th column vector of $\mathbf{Q}^{\epsilon}_{m\times m}$ is expressed as $\frac{1}{\|\mathbf{u}_j\|}\mathbf{u}_j$.*

**Proof.** Let $\mathbf{a}_j$ be the $j$th column vector of $\mathbf{J}^{\epsilon}$, i.e.,

$$\mathbf{J}^{\epsilon} = \begin{bmatrix} \mathbf{a}_1 & \ldots & \mathbf{a}_n \end{bmatrix} = \left[\begin{bmatrix} 1+\epsilon \\ 1 \\ \vdots \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1+\epsilon \\ \vdots \\ 1 \end{bmatrix}, \cdots, \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1+\epsilon \end{bmatrix}\right].$$

Now we apply the Gram–Schmidt process to matrix $\mathbf{J}^{\epsilon}$. Then the 1st orthogonal vector is given as

$$\mathbf{u}_1 = \mathbf{a}_1 = \begin{bmatrix} 1+\epsilon \\ 1 \\ \vdots \\ 1 \end{bmatrix}.$$

Next, the 2nd orthogonal vector is constructed as

$$\mathbf{u}_2 = \mathbf{a}_2 - \mathrm{proj}_{\mathbf{u}_1}\mathbf{a}_2$$

$$
\begin{aligned}
&= \mathbf{a}_2 - \frac{\langle \mathbf{u}_1, \mathbf{a}_2 \rangle}{\langle \mathbf{u}_1, \mathbf{u}_1 \rangle} \mathbf{u}_1 \\
&= \mathbf{a}_1 + \epsilon(\mathbf{e}_2 - \mathbf{e}_1) - \frac{\langle \mathbf{u}_1, \mathbf{a}_2 \rangle}{\langle \mathbf{u}_1, \mathbf{u}_1 \rangle} \mathbf{u}_1 \\
&= \left(1 - \frac{\langle \mathbf{u}_1, \mathbf{a}_2 \rangle}{\langle \mathbf{u}_1, \mathbf{u}_1 \rangle}\right) \mathbf{u}_1 + \epsilon(\mathbf{e}_2 - \mathbf{e}_1).
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{u}_k &= \mathbf{a}_k - \sum_{j=1}^{k-1} \mathrm{proj}_{\mathbf{u}_j} \mathbf{a}_n \\
&= \mathbf{a}_k - \sum_{j=1}^{k-1} \frac{\langle \mathbf{u}_j, \mathbf{a}_k \rangle}{\langle \mathbf{u}_j, \mathbf{u}_j \rangle} \mathbf{u}_j \\
&= \mathbf{a}_{k-1} + \epsilon(\mathbf{e}_k - \mathbf{e}_{k-1}) - \sum_{j=1}^{k-1} \frac{\langle \mathbf{u}_j, \mathbf{a}_k \rangle}{\langle \mathbf{u}_j, \mathbf{u}_j \rangle} \mathbf{u}_j \\
&= \mathbf{a}_{k-1} - \sum_{j=1}^{k-2} \frac{\langle \mathbf{u}_j, \mathbf{a}_k \rangle}{\langle \mathbf{u}_j, \mathbf{u}_j \rangle} \mathbf{u}_j - \frac{\langle \mathbf{u}_{k-1}, \mathbf{a}_k \rangle}{\langle \mathbf{u}_{k-1}, \mathbf{u}_{k-1} \rangle} \mathbf{u}_{k-1} + \epsilon(\mathbf{e}_k - \mathbf{e}_{k-1}) \\
&= \mathbf{a}_{k-1} - \sum_{j=1}^{k-2} \frac{\langle \mathbf{u}_j, \mathbf{a}_{k-1} \rangle}{\langle \mathbf{u}_j, \mathbf{u}_j \rangle} \mathbf{u}_j - \frac{\langle \mathbf{u}_{k-1}, \mathbf{a}_k \rangle}{\langle \mathbf{u}_{k-1}, \mathbf{u}_{k-1} \rangle} \mathbf{u}_{k-1} + \epsilon(\mathbf{e}_k - \mathbf{e}_{k-1}) \\
&= \mathbf{u}_{k-1} - \frac{\langle \mathbf{u}_{k-1}, \mathbf{a}_k \rangle}{\langle \mathbf{u}_{k-1}, \mathbf{u}_{k-1} \rangle} \mathbf{u}_{k-1} + \epsilon(\mathbf{e}_k - \mathbf{e}_{k-1}).
\end{aligned}
$$

The last second equality holds from the fact that all $i$th ($i \geq j$) entries of $\mathbf{u}_{j-1}$ are identical. □

The iteration in Theorem 1 can reduce the computational complexity of $\mathbf{W}^\epsilon$. Through this theorem, the computational complexity of QR decomposition is reduced from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$. Furthermore, since the proposed initialization method is fully deterministic, it is sufficient to compute the matrix only once for a given epsilon $\epsilon$ and dimension of the matrix $m, n$, allowing it reused. Before demonstrating that the proposed initial weights have nearly equal column sums and row sums, we first establish the properties of $\mathbf{Q}^\epsilon$.

Next, we prove the bound on the column sum of $\mathbf{Q}^\epsilon$ in Lemma 2, and from Lemma 2, we demonstrate Proposition 2: The entry sum of each column (resp. row) vector of $\mathbf{W}^\epsilon$ is almost same.

**Lemma 1.** *Let $\mathbf{a}_j$ be the $j$th column vector of $\mathbf{J}^\epsilon$ for $j = 1, \ldots, m$. Then*

$$
\left\langle \frac{\mathbf{a}_j}{\|\mathbf{a}_j\|}, \frac{\mathbf{1}}{\|\mathbf{1}\|} \right\rangle = \frac{m+\epsilon}{\sqrt{m}\sqrt{m+2\epsilon+\epsilon^2}} = 1 - \frac{m-1}{2m^2}\epsilon^2 + \mathcal{O}(\epsilon^3).
$$

*Furthermore, if $\epsilon \to 0$, then*

$$
\left\langle \frac{\mathbf{a}_j}{\|\mathbf{a}_j\|}, \frac{\mathbf{1}}{\|\mathbf{1}\|} \right\rangle \to 1,
$$

*provided that $m$ is fixed.*

**Lemma 2.** *Let $\mathbf{q}_1, \ldots, \mathbf{q}_m$ be the column vectors of $\mathbf{Q}^\epsilon_{m\times m}$. Then it holds that*

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

**Proof.** By QR decomposition we have

$$
\mathbf{J}^\epsilon = \mathbf{J} + \epsilon \mathbf{I} = \mathbf{Q}^\epsilon \mathbf{R}^\epsilon, \tag{3}
$$

where $\mathbf{Q}^\epsilon$ is the orthogonal matrix and $\mathbf{R}^\epsilon$ is the upper triangular matrix. By multiplying $(\mathbf{Q}^\epsilon)^T$ and $\mathbf{1}$ on both sides, it follows that

$$
(\mathbf{Q}^\epsilon)^T (\mathbf{J} + \epsilon \mathbf{I}) \mathbf{1} = \mathbf{R}^\epsilon \mathbf{1}.
$$

Let $q_{ij}$ be the entry in $i$th row and $j$th column of $\mathbf{Q}^\epsilon_{m\times m}$ and let $\mathbf{q}_1, \ldots, \mathbf{q}_m$ are the column vectors of $\mathbf{Q}^\epsilon_{m\times m}$. Thus we have

$$
(m+\epsilon)(\mathbf{Q}^\epsilon)^T \mathbf{1} = \begin{bmatrix} \langle \mathbf{q}_1, \mathbf{v}_1 \rangle \\ \langle \mathbf{q}_2, \mathbf{v}_2 \rangle \\ \vdots \\ \langle \mathbf{q}_m, \mathbf{v}_m \rangle \end{bmatrix}, \tag{4}
$$

where for each $k = 1, \ldots, m$

$$
\mathbf{v}_k = \begin{bmatrix} m-k+1 \\ m-k+1 \\ \vdots \\ m-k+1 \\ m-k+1+\epsilon \\ \vdots \\ m-k+1+\epsilon \end{bmatrix}.
$$

Let $S_j$ be the sum of all entries of $\mathbf{q}_j$. Then we have that for all $j = 2, 3, \ldots, m$

$$
\begin{aligned}
S_j &= \frac{1}{m+\epsilon} \langle \mathbf{q}_j, \mathbf{v}_j \rangle \\
&= \frac{1}{m+\epsilon}((m-j+1)q_{1j} + \cdots + (m-j+1+\epsilon)q_{jj} \\
&\quad + \cdots + (m-j+1+\epsilon)q_{mj}) \\
&= \frac{m-j+1+\epsilon}{m+\epsilon} S_j - \frac{\epsilon}{m+\epsilon} \left(q_{1j} + \cdots + q_{j-1,j}\right),
\end{aligned}
$$

implying that

$$
\left(1 - \frac{m-j+1+\epsilon}{m+\epsilon}\right) S_j = -\frac{\epsilon}{m+\epsilon} \left(q_{1j} + \cdots + q_{j-1,j}\right).
$$

Thus it follows that for all $j = 2, 3, \ldots, m$

$$
\begin{aligned}
\left|S_j\right| &= \frac{\epsilon}{j-1} \left|q_{1j} + \cdots + q_{j-1,j}\right| \\
&\leq \frac{\epsilon}{j-1} \left(|q_{1j}| + \cdots + |q_{j-1,j}|\right) \\
&\leq \frac{\epsilon}{\sqrt{j-1}} \leq \epsilon.
\end{aligned} \tag{5}
$$

The first inequality and the second inequality hold from the triangle inequality and the Cauchy–Schwarz inequality, respectively. □

**Proposition 2.** *The entry sum of each column (resp. row) vector of $\mathbf{W}^\epsilon_{m\times n}$ is almost the same.*

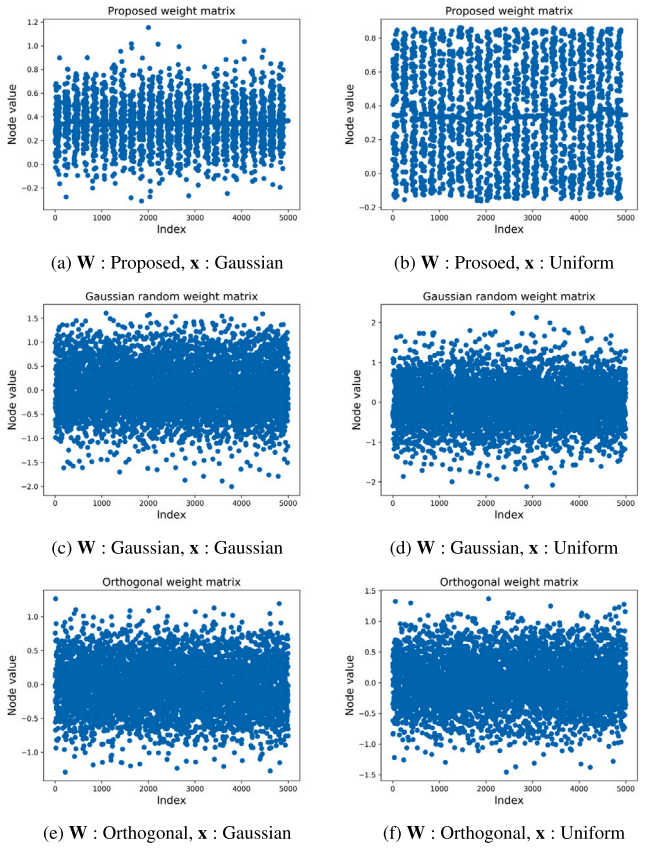
**Proof.** Let $\mathbf{c}_j$ be $j$th column vector of $\mathbf{W}^\epsilon = \mathbf{W}^\epsilon_{m\times n}$ for each $j = 1, \ldots, n$. Then by (2) the sum of all entries of $j$th column vector can be expressed as

$$
\mathbf{c}_j^T \mathbf{1} = \mathbf{e}_j^T (\mathbf{W}^\epsilon)^T \mathbf{1} = \mathbf{e}_j^T \left(\hat{\mathbf{q}}_1 \mathbf{q}_1^T + \cdots + \hat{\mathbf{q}}_s \mathbf{q}_s^T\right) \mathbf{1},
$$

where $s = \min\{m, n\}$. So, it follows that for each $j = 1, \ldots, n$

$$
\begin{aligned}
\left|\mathbf{c}_j^T \mathbf{1} - \mathbf{e}_j^T \hat{\mathbf{q}}_1 \mathbf{q}_1^T \mathbf{1}\right| &= \left|\mathbf{e}_j^T (\mathbf{W}^\epsilon)^T \mathbf{1} - \mathbf{e}_j^T \hat{\mathbf{q}}_1 \mathbf{q}_1^T \mathbf{1}\right| \\
&= \left|\mathbf{e}_j^T \left(\hat{\mathbf{q}}_2 \mathbf{q}_2^T + \cdots + \hat{\mathbf{q}}_s \mathbf{q}_s^T\right) \mathbf{1}\right| \\
&\leq \left|\mathbf{e}_j^T \hat{\mathbf{q}}_2\right| \left|\mathbf{q}_2^T \mathbf{1}\right| + \cdots + \left|\mathbf{e}_j^T \hat{\mathbf{q}}_s\right| \left|\mathbf{q}_s^T \mathbf{1}\right| \\
&= \left|\hat{q}_{j2}\right| \left|\langle \mathbf{q}_2, \mathbf{1} \rangle\right| + \cdots + \left|\hat{q}_{js}\right| \left|\langle \mathbf{q}_s, \mathbf{1} \rangle\right| \\
&\leq \left|\hat{q}_{j2}\right| \frac{\epsilon}{\sqrt{1}} + \cdots + \left|\hat{q}_{js}\right| \frac{\epsilon}{\sqrt{s-1}} \\
&\leq \epsilon \sqrt{\frac{1}{1} + \cdots + \frac{1}{s-1}} \sqrt{\left|\hat{q}_{j2}\right|^2 + \cdots + \left|\hat{q}_{js}\right|^2} \\
&= \epsilon \sqrt{H_{s-1}} \approx \epsilon \sqrt{\log(s-1)},
\end{aligned}
$$

where $\hat{q}_{ij}$ is the $(i, j)$-entry of $\mathbf{Q}^\epsilon_{n\times n}$ and $H_k$ is the $k$th harmonic number. The last equality holds from the fact that $\hat{\mathbf{q}}_1, \ldots, \hat{\mathbf{q}}_n$ are the column vectors of the orthogonal matrix $\mathbf{Q}^\epsilon_{n\times n}$, and the last inequality holds from the Cauchy–Schwarz inequality and by (5) the second last inequality holds.




(a) $\mathbf{W}$ : Proposed, $\mathbf{x}$ : Gaussian (b) $\mathbf{W}$ : Prosoed, $\mathbf{x}$ : Uniform

(c) $\mathbf{W}$ : Gaussian, $\mathbf{x}$ : Gaussian (d) $\mathbf{W}$ : Gaussian, $\mathbf{x}$ : Uniform

(e) $\mathbf{W}$ : Orthogonal, $\mathbf{x}$ : Gaussian (f) $\mathbf{W}$ : Orthogonal, $\mathbf{x}$ : Uniform

**Fig. 2.** This shows its effectiveness of positive signal propagation for each weight matrices $\mathbf{W} \in \mathbb{R}^{200\times100}$. For 25 random vectors $\mathbf{x} \in \mathbb{R}^{100}$, the entry values of $\mathbf{Wx}$ are plotted. Here, the $x$-axis represents the indices of all entries.

In a similar way, one can check that for each $i = 1, \dots, m$ it holds that

$$\left|\mathbf{r}_i^T\mathbf{1} - \mathbf{e}_i^T\mathbf{q}_1\hat{\mathbf{q}}_1^T\mathbf{1}\right| \le \epsilon\sqrt{H_{s-1}},$$

where $\mathbf{r}_i$ be the $i$th row vector of $\mathbf{W}^\epsilon$ for each $i = 1, \dots, m$. □

Now, we have established that the entry sum of each column vector of $\mathbf{W}^\epsilon_{m\times n}$ is almost the same. Note that the entry sum of a given column vector can be expressed as the inner product of the column vector and $\mathbf{1}$. It means that the column vectors of $\mathbf{W}^\epsilon_{m\times n}$ are located in the vicinity of a plane that forms a specific angle with the vector $\mathbf{1}$.

The challenges associated with training deep neural networks using ReLU arise from the limitation that negative signals cannot propagate through the network. Our proposed weight matrix can make positive signals propagate through the network. Firstly, let us give experimental results for positive signal propagation with the proposed weight matrix and a Gaussian random matrix, orthogonal matrix. We set $\mathbf{W} \in \mathbb{R}^{200\times100}$ as our proposed initial weight matrix with $\epsilon = 0.1$, a Gaussian random matrix with a mean of 0 a standard deviation of 0.1, and an orthogonal matrix. The positive signals $\mathbf{x} \in \mathbb{R}^{100}$ are generated from (i) normal distribution $\mathcal{N}(0.5, 0.25^2)$ and (ii) uniform distribution $\mathcal{U}_{[0,1]}$. For a random vector $\mathbf{x} \in \mathbb{R}^{100}$, the entry values of $\mathbf{Wx}$ are computed over 25 times for each weight matrices. The computational results confirm that our proposed weight initialization method consistently yields a higher number of positive entries in $\mathbf{Wx}$ than other matrices, demonstrating its effectiveness in facilitating signal propagation in networks with ReLU activation (see Fig. 2).

Now, we provide a theoretical analysis demonstrating that $\mathbf{W}^\epsilon\mathbf{x}$ contains a higher number of positive entries.

**Theorem 2.** *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 $\mathbf{x} \in \mathbb{R}^{N_x}$*

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

**Proof.** Since the proposed weight matrix $\mathbf{W}^\epsilon \in \mathbb{R}^{N_1\times N_x}$ holds orthogonality, it holds that

$$\begin{aligned}\frac{1}{N_x}\left\langle \mathbf{x}, \mathbf{1}_{N_x}\right\rangle &= \frac{1}{N_x}\left\langle \mathbf{W}^\epsilon\mathbf{1}_{N_x}, \mathbf{W}^\epsilon\mathbf{x}\right\rangle \\ &= \frac{1}{N_x}\left\langle (\mathbf{q}_1\hat{\mathbf{q}}_1^T + \mathbf{q}_2\hat{\mathbf{q}}_2^T + \cdots + \mathbf{q}_s\hat{\mathbf{q}}_s^T)\mathbf{1}_{N_x}, \mathbf{W}^\epsilon\mathbf{x}\right\rangle \\ &= \frac{1}{N_x}\left\langle \mathbf{q}_1\hat{\mathbf{q}}_1^T\mathbf{1}_{N_x}, \mathbf{W}^\epsilon\mathbf{x}\right\rangle + \frac{1}{N_x}\sum_{i=2}^{s}\left\langle \mathbf{q}_i\hat{\mathbf{q}}_i^T\mathbf{1}_{N_x}, \mathbf{W}^\epsilon\mathbf{x}\right\rangle \\ &\simeq \frac{1}{N_x}\left\langle \mathbf{q}_1\hat{\mathbf{q}}_1^T\mathbf{1}_{N_x}, \mathbf{W}^\epsilon\mathbf{x}\right\rangle,\end{aligned}$$

where $s = \min\{N_x, N_1\}$. The last approximate equality holds from that

$$\begin{aligned}\left|\frac{1}{N_x}\sum_{i=2}^{s}\left\langle \mathbf{q}_i\hat{\mathbf{q}}_i^T\mathbf{1}_{N_x}, \mathbf{W}^\epsilon\mathbf{x}\right\rangle\right| &\le \frac{\epsilon\sqrt{H_{s-1}}}{N_x}\left\langle \mathbf{1}_{N_1}, \mathbf{W}^\epsilon\mathbf{x}\right\rangle \\ &= \frac{\epsilon\sqrt{H_{s-1}}}{N_x}\sum_{j=1}^{N_1}\left\langle \mathbf{r}_j, \mathbf{x}\right\rangle \\ &\le \frac{\epsilon\sqrt{H_{s-1}}}{N_x}\sum_{j=1}^{N_1}\|\mathbf{x}\|,\end{aligned}$$

where $\mathbf{r}_j$ be the $j$th row vector of $\mathbf{W}^\epsilon$ for each $j = 1, \dots, N_1$. The first inequality holds from Proposition 2, and the last inequality holds from orthogonality and Cauchy–Schwarz inequality. Then by Proposition 2 it follows that

$$\begin{aligned}\frac{1}{N_x}\left\langle \mathbf{q}_1\hat{\mathbf{q}}_1^T\mathbf{1}_{N_x}, \mathbf{W}^\epsilon\mathbf{x}\right\rangle &= C\left(N_1\frac{1}{N_1}\left\langle \mathbf{1}_{N_1}, \mathbf{W}^\epsilon\mathbf{x}\right\rangle + (\epsilon^2 + \epsilon N_x)\left\langle \mathbf{e}_1, \mathbf{W}^\epsilon\mathbf{x}\right\rangle\right) \\ &\simeq \sqrt{\frac{N_1}{N_x}}\frac{1}{N_1}\left\langle \mathbf{1}_{N_1}, \mathbf{W}^\epsilon\mathbf{x}\right\rangle,\end{aligned}$$

where $C = \dfrac{N_x + \epsilon}{N_x\sqrt{\epsilon^2 + 2\epsilon + N_1}\sqrt{\epsilon^2 + 2\epsilon + N_x}}$. □

**Corollary 1.** *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 $\mathbb{R}^{N_1}$.*

**Proof.** Note that the orthogonality of $\mathbf{W}^\epsilon$ implies that $\|\mathbf{W}^\epsilon\mathbf{x}\| = \|\mathbf{x}\|$. Therefore, by Theorem 2 one can see that

$$\cos\theta_2 = \frac{\left\langle \mathbf{W}^\epsilon\mathbf{x}, \mathbf{1}_{N_1}\right\rangle}{\|\mathbf{W}^\epsilon\mathbf{x}\|\left\|\mathbf{1}_{N_1}\right\|} \simeq \sqrt{\frac{N_x}{N_1}}\frac{\left\langle \mathbf{x}, \mathbf{1}_{N_x}\right\rangle}{\|\mathbf{x}\|\left\|\mathbf{1}_{N_1}\right\|} = \cos\theta_1. \quad \square\ \square$$

Theorem 2 states that the average of $\mathbf{x}$ is linearly preserved in $\mathbf{W}^\epsilon\mathbf{x}$, where the proportion is almost $\sqrt{\frac{N_1}{N_x}}$. As demonstrated in Fig. 2, the average of $\mathbf{W}^\epsilon\mathbf{x}$ is approximately 0.35, which is almost equal to the product of $\mathbf{x}$'s average of 0.5 and $\sqrt{\frac{N_1}{N_x}} = \sqrt{\frac{100}{200}} \approx 0.7$. Corollary 1 states that the angle between $\mathbf{x}$ and the one vector is preserved in the angle between $\mathbf{W}^\epsilon\mathbf{x}$ and the one vector. If the entries of $\mathbf{x}$ are all positive, then the angle between the one vector and $\mathbf{x}$ is acute. Corollary 1 implies that the orthant in which $\mathbf{W}^\epsilon\mathbf{x}$ resides will predominantly be composed of positive values.

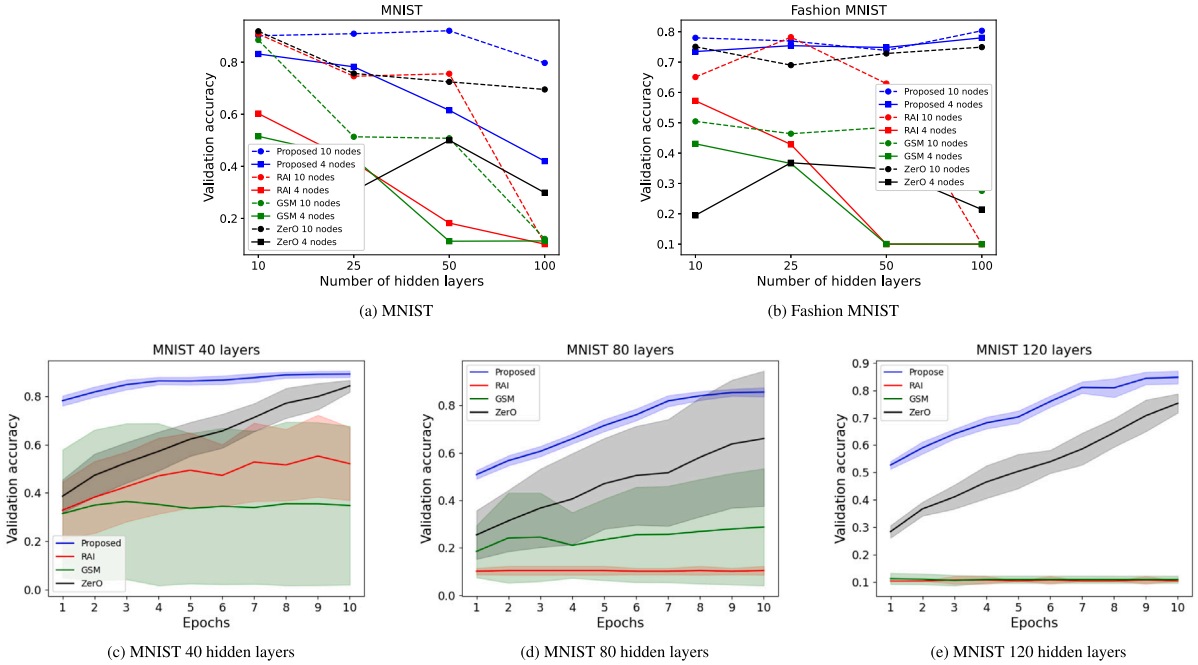
Now we consider a deep network of depth $\ell$ with linear activation function and zero bias. According to the definition of the proposed initial weight matrix, the following equation is satisfied.

$$\mathbf{y} = \mathbf{W}^\epsilon_{N_\ell\times N_{\ell-1}}\cdots\mathbf{W}^\epsilon_{N_1\times N_x}\mathbf{x} = \mathbf{W}^\epsilon_{N_\ell\times N_x}\mathbf{x}.$$

It means that regardless of the network's depth, $\mathbf{y}$ satisfies Theorem 2 and Corollary 1, provided that $\epsilon$ is sufficiently small.

**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. The simulations are performed with datasets MNIST and FMNIST over 10 epochs and 100 epochs. Best results are marked in bold.

| 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) | **88.2** | 88.5 | 88.1 | 88.7 | 86.6 | 87.7 | 87.9 | **89.1** | 88.1 | 88.8 | 88 | 89.1 | 88 | 88.2 | 87.1 | 89.4 |
| FMNIST (0) | **80.4** | **80.6** | 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) | **96.5** | **97.6** | 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 | **84.9** | **85.2** | 78.3 | 80.3 |
| MNIST (16) | **92.2** | **94** | 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) | **82.3** | **84.2** | 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) | **55.5** | **54.1** | 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) | **54.2** | **57.1** | 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) | **56.5** | **51.0** | 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 | **54** | 54.6 | 51 | **56.8** | 37.1 | 50.4 | 45.2 | 53.4 | 48.7 | 56.2 | 45.1 | 53.6 |
| MNIST (16) | **51.2** | **52.9** | 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) | **43.3** | **56.3** | 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 |
| 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 |




(a) MNIST (b) Fashion MNIST

(c) MNIST 40 hidden layers (d) MNIST 80 hidden layers (e) MNIST 120 hidden layers

**Fig. 3.** 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. (c), (d), and (e) investigate networks consisting of a layer with 10 nodes and a layer with 6 nodes, repeated throughout the structure.

## 4. Experimental results

In Section 4.1, we describe the experimental environment, benchmark datasets, and the methods compared in this paper. In Section 4.2, we introduce existing methods that effectively prevent the dying ReLU problem in deep and narrow FFNNs with ReLU activation function, and we present the settings used for each method in our experiments. Section 4.3 presents the results from experiments across diverse dataset sizes and network architectures. In Section 4.4, we experiment to investigate the trainability of networks at various depths. In Section 4.5, we conduct experiments to determine the feasibility of training on extremely narrow networks. Finally, Section 4.6 investigates the trainability of networks at various activation functions.

### 4.1. Experimental settings

Section 4 analyzed a range of weight initialization methods via a comprehensive experimental approach. We referred to our method as "Proposed". As other methods, we used Xavier initialization, He initialization, ZerO initialization, Identity initialization, Orthogonal initialization, RAI, and GSM. To assess the effectiveness of the proposed weight initialization method, we conducted experiments on MNIST, Fashion MNIST, Wine Quality dataset, and Iris. And 15% of the total dataset served as the validation dataset. We trained the network using cross-entropy loss implemented the neural network in Python with Tensorflow, and trained the neural network on a computer with GPU RTX TITAN. And we used Adam optimizer with a learning rate of 0.001 and a batch size of 100. Empirically setting $\epsilon = 0.1$, we configured all activation function parameters to their TensorFlow default values. For each dataset, we maintained consistent hyperparameter settings for all experiments. Each experiment was repeated ten times with random seeds.

### 4.2. Prior weight initialization method for FFNNs

We briefly review Randomized Asymmetric Initialization (RAI) (Lu et al., 2019) and Gaussian Submatrix Initialization (GSM) (Burkholz & Dubatovka, 2019), both of which are designed to enhance the training of deep and narrow feedforward neural networks. And we also present the settings for each of them. The RAI weight initialization method is a technique used to address the "dying ReLU" problem in deep FFNNs. It involves the creation of an initialization matrix with random values drawn from an asymmetric probability distribution. In this paper, we employed Beta(2,1) probability distribution. And standard deviation of the weight matrix was $-\frac{2\sqrt{2}}{3\sqrt{\pi}} + \sqrt{1+\frac{8}{9\pi}} \approx 0.6007$ adopting a setting similar to that in Lu et al. (2019). The GSM is a weight initialization method for ReLU layers that ensures perfect dynamical isometry. In this paper, we constructed the submatrix using the He initialization method.

### 4.3. Experiments in various settings

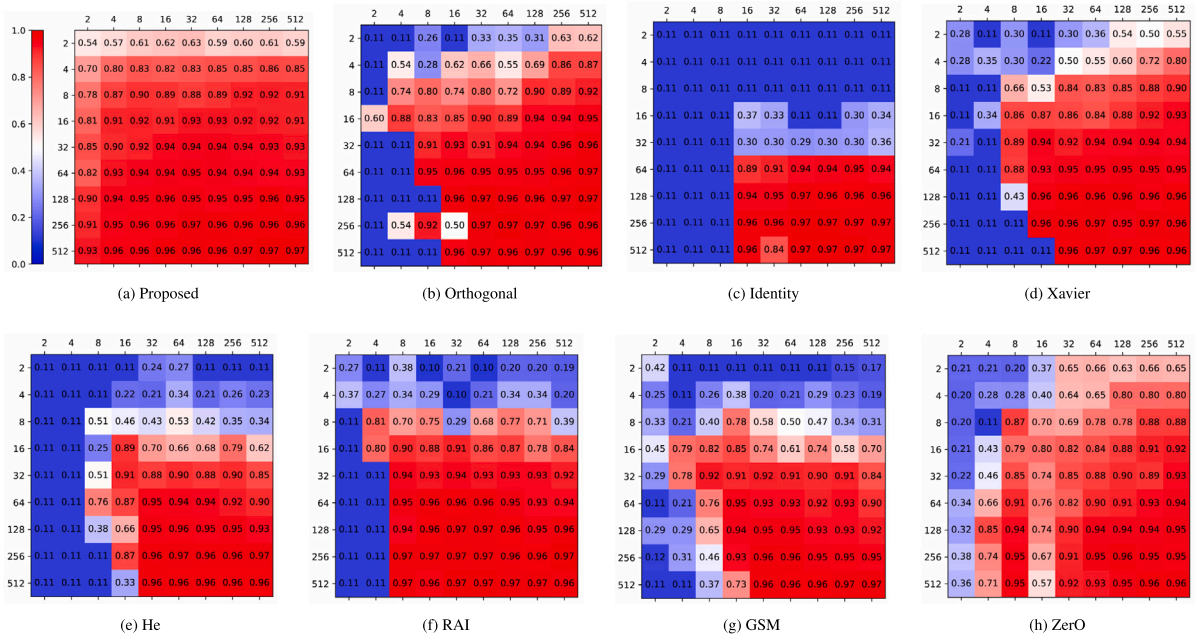
In this section, we conducted experiments using the MNIST and Fashion MNIST datasets. We compared various initialization methods while varying the dataset size with FFNNs with ReLU activation function. As shown in Table 1, we measured the validation accuracy in various settings at 10 and 100 epochs. The term '$k$ samples per class' indicates that each class consists of $k$ number of samples. We conducted experiments for $1, 2, 4$ samples per class and the entire dataset. In the table, (0), (512), and (16) denote the number of nodes in a single hidden layer of FFNNs with ReLU activation function. In detail, (0) signifies an FFNN without hidden layers; (512) corresponds to one with a single hidden layer of 512 nodes; and (16), one with a single hidden layer of 16 nodes. With no hidden layers, the proposed method consistently achieved higher validation accuracy at 10 epochs, irrespective of the dataset size. Both identity initialization and Zero initialization also demonstrated high validation accuracy. Zero initialization is an identity matrix when the number of rows in the weight matrix is less than the number of columns. These three weight initialization methods outperformed random weight initialization in networks without hidden layers. Furthermore, even for small values of $k$, all three methods exhibited good performance. When $k$ equals 1, the decline in accuracy observed for the proposed method on the FMNIST dataset at 100 epochs was attributed to overfitting, possibly due to excessively rapid convergence. An FFNN with 512 nodes can be considered a wide FFNN, whereas an FFNN with 16 nodes is relatively narrow. We conducted comparative experiments on these two FFNNs to assess how independent our proposed method is regarding the number of nodes. Generally, in narrow networks with fewer nodes, learning is less effective compared to wider networks with a larger number of nodes. However, the proposed weight initialization exhibited significantly higher validation accuracy even with 16 nodes, surpassing other weight initialization methods. Contrastingly, in networks with 512 nodes, it exhibited validation accuracy similar to other weight initialization methods. The reason is that in the narrow network, the dying ReLU problem is particularly detrimental to network training. To demonstrate that the proposed method is independent of both network depth and width, more diverse experiments are needed (Section 4.4 and Section 4.5).

### 4.4. Depth independent

In this section, we applied the proposed weight initialization method to investigate its effectiveness in training deep FFNNs with the ReLU activation function. We compared the proposed initialization method with the RAI, GSM, and ZerO methods – previously studied and proven to perform well in training deep ReLU neural networks – using the MNIST and Fashion MNIST datasets. This experiment drew inspiration from the methods described in Burkholz and Dubatovka (2019), Lu et al. (2019), Zhao et al. (2021).

The experiments were divided into two main parts: one where the hidden layers had the same dimensionality, and the other where they had varying dimensionality. We made this division because networks in practice often exhibit varying dimensionalities across their layers. Fig. 3(a), (b) shows that the $x$-axis represents the number of hidden layers, while the $y$-axis represents the validation accuracy measured at 10 epochs. The label 10 (resp. 4) nodes indicate that all hidden layers have 10 (resp. 4) nodes. The experimental results of the MNIST dataset indicated that our proposed method demonstrated high validation accuracy, independent of the number of hidden layers and nodes. Specifically, for configurations with 10 nodes per layer, we observed that RAI and GSM were effective up to 50 hidden layers, after which they experienced difficulty in training as the number of layers increased to 100. ZerO initialization performed reasonably well, but when compared to our proposed method, it consistently yielded lower validation accuracy across various numbers of hidden layers. In scenarios where the network had only four nodes per layer, most initialization methods struggled due to the narrow network architecture. However, our proposed initialization method stood out by successfully enabling training even with 100 hidden layers. On the Fashion MNIST dataset, the experimental results also indicated that our proposed method demonstrated high validation accuracy, independent of the number of hidden layers and nodes. In contrast, zero initialization achieved high validation accuracy with 10 nodes per layer but faced instability in narrower networks. Furthermore, RAI and GSM struggled to train networks with 50 or 100 hidden layers effectively.

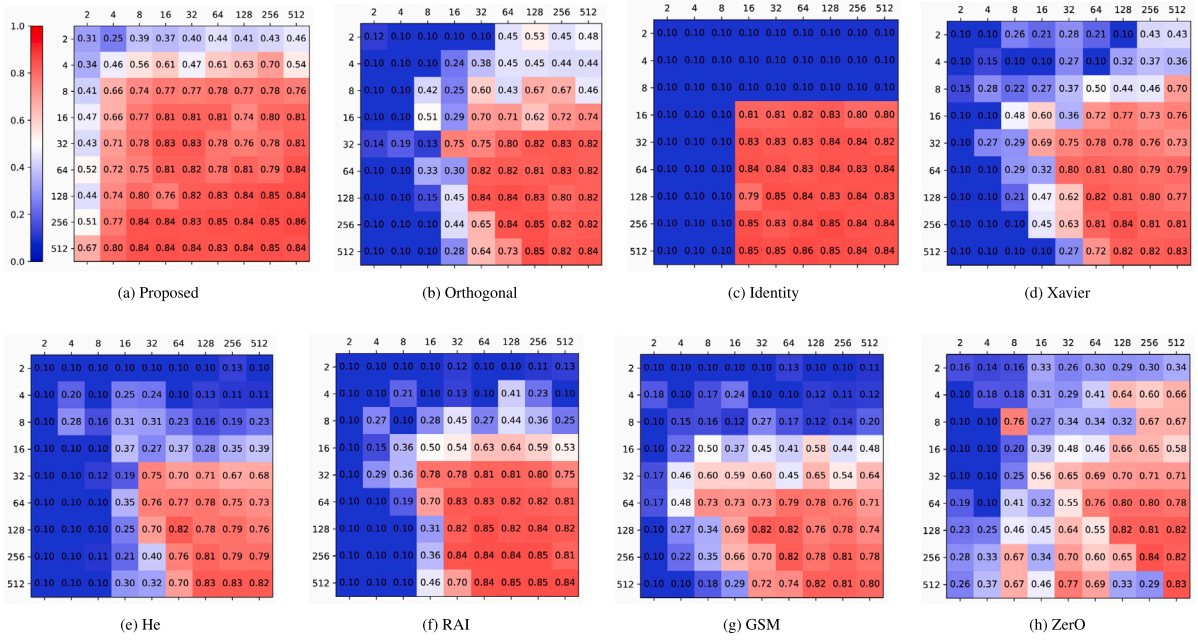
In the experiments involving hidden layers with varying numbers of nodes, the results are depicted in Fig. 3(c),(d), and (e). The network architecture consisted of a layer with 10 nodes and a layer with 6 nodes, repeated throughout the structure. For instance, a network with 20 hidden layers comprised 10 node layers and 6 node layers repeated 10 times. When the network had 40 hidden layers, the validation accuracy of RAI and GSM across epochs exhibited significant variability. In



(a) Proposed (b) Orthogonal (c) Identity (d) Xavier

(e) He (f) RAI (g) GSM (h) ZerO

**Fig. 4.** A validation accuracy is presented for FFNNs with two hidden layers and ReLU activation function. 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.



(a) Proposed (b) Orthogonal (c) Identity (d) Xavier

(e) He (f) RAI (g) GSM (h) ZerO

**Fig. 5.** A validation accuracy is presented for FFNNs with two hidden layers and ReLU activation function. 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.

contrast, our proposed method maintained stable validation accuracy during training. This trend continued as the number of hidden layers increased to 80, and it became evident that when the network comprised 120 hidden layers, only our proposed method and zero initialization managed to facilitate successful learning. Furthermore, we conducted simulations on two types of tabular data, each with fewer than 100 features: the Wine Quality Dataset (Cortez et al., 2009) and the Iris dataset (Fisher, 1988). In further experiments, we compared the proposed initialization method with the RAI, ZerO, He, and Orthogonal initialization methods. We trained on the Wine Quality Dataset (resp. the Iris Dataset) using an FFNN configured with ReLU activation, comprising layers of 10 nodes and layers of 6 nodes, this configuration being repeated 60 (resp. 100) times. The experimental results indicated that the proposed initialization method achieved higher validation accuracy compared to other methods across both the Wine Quality and Iris datasets. For the Wine Quality dataset at 200 epochs, the proposed method's validation accuracy (58%) surpassed those of ZerO (50%), Orthogonal (40%), RAI (40%), and He (40%). In the case of the Iris dataset, the proposed method rapidly achieved high accuracy by the 10th epoch and maintained it. Proposed (94%), ZerO (63%), Orthogonal (30%), RAI (38%), and He (38%) are validation accuracies at 100 epochs for the respective methods. In summary, our method

**Table 2**
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. Best results are marked in bold.

| 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 | **27.0** | 9.9 | 16.1 | 10.0 | 12.3 | **13.6** |
| 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 | **38.3** | 33.3 | 11.7 | 9.9 | 10.2 | 9.9 | 9.8 | 9.9 | 33.0 | **34.5** | 10.4 | 9.9 | 10.2 | 9.9 | 12.0 | 11.0 |
| Gelu | **83.6** | **68.1** | 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 | **86.7** | **76.5** | 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 |

demonstrated depth independence by achieving higher validation accuracy in deep ReLU neural networks compared to other initialization methods.

### 4.5. Width independent

In this section, we employed the proposed weight initialization method to assess its effectiveness in training feedforward neural networks (FFNNs) with ReLU activation function, emphasizing its independence from network width. We created FFNNs with ReLU activation function, each consisting of two hidden layers. As shown in Figs. 4 and 5, the $y$-axis represents the number of nodes in the first hidden layer, while the $x$-axis represents the number of nodes in the second hidden layer. The values within the heatmap correspond to the validation accuracy, trained for 10 epochs on the MNIST dataset, where the accuracy is determined based on the respective numbers of nodes in the $x$ and $y$ dimensions. The proposed method achieved a validation accuracy of 54.3% when the number of nodes in each hidden layer was set to 2. In contrast, other methods exhibited the lowest validation accuracy of only 10%. Our proposed method demonstrated independence from the number of nodes, effectively enabling the training of narrow feedforward neural networks with ReLU activation function. We also recorded the validation accuracy at 1 epoch of training on the FMNIST dataset to assess the convergence speed for various network sizes. The proposed method achieved the lowest validation accuracy of 25% among the tested architectures when the first hidden layer had 2 nodes, and the second hidden layer had 4 nodes. In contrast, the lowest accuracy recorded by all other methods was 10%. In summary, our proposed method demonstrated independence from the number of nodes in FFNNs with ReLU activation function and converged faster compared to other methods.

### 4.6. Activation function independent

Finally, we employed the proposed weight initialization method to assess its effectiveness in training feedforward neural networks, emphasizing its independence from the activation function. Table 2 illustrates the validation accuracy of tanh, sigmoid, ReLU (Nair & Hinton, 2010), GeLU (Hendrycks & Gimpel, 2016), and SeLU (Klambauer, Unterthiner, Mayr, & Hochreiter, 2017) on the MNIST and FMNIST datasets at 10 epochs. Here, GeLU and SeLU were set to their default settings in TensorFlow. A feedforward neural network was constructed following the same configuration used in the depth independence experiment in Section 4.4, comprising 120 hidden layers, with a layer of 10 nodes and a layer of 6 nodes, repeated 60 times each. Notably, for activation functions within the ReLU family, our method outperformed other weight initialization strategies. In particular, with the GeLU activation function, our proposed method achieved a validation accuracy of 68.1% on FMNIST, and with ReLU, it reached an accuracy of 76.5%, demonstrating higher accuracy compared to other weight initialization methods. Also, the proposed method showed high validation accuracy on the MNIST dataset. With the GeLU activation function, the proposed weight initialization method achieved an accuracy of 83.6%, and with ReLU, it achieved an accuracy of 86.7% (see Table 2).

## 5. Conclusion

In this work, we propose a novel weight initialization method and provide several properties for the proposed initial weight matrix. We demonstrated the proposed matrix holds orthogonality. Moreover, it was shown that the proposed initial matrix has constant row (or column) sum. Also, we demonstrate that our weight initialization method ensures efficient signal transmission even in extremely deep and narrow feedforward ReLU neural networks. Experimental results demonstrate that the network performs well regardless of whether it is deep or narrow, and even when there is a significant difference in the number of nodes between hidden layers.

## CRediT authorship contribution statement

**Hyunwoo Lee:** Writing – original draft, Conceptualization, Formal analysis, Writing – review & editing. **Yunho Kim:** Formal analysis, Writing – original draft. **Seung Yeop Yang:** Formal analysis, Writing – original draft. **Hayoung Choi:** Conceptualization, Formal analysis, Writing – original draft, Writing – review & editing.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

Data will be made available on request.

## Acknowledgments

The authors wish to express their gratitude to the anonymous referees for their careful reading of the manuscript and their helpful suggestions. This work of H. Lee, Y. Kim, S. Y. Yang, and H. Choi was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2022R1A5A1033624). The work of S. Y. Yang was supported by Learning & Academic research institution for Master's · Ph.D. students, and Postdocs (LAMP) Program of the National Research Foundation of Korea (NRF) grant funded by the Ministry of Education, Korea (No. RS2023-00301914).

## References

Advani, M. S., Saxe, A. M., & Sompolinsky, H. (2020). High-dimensional dynamics of generalization error in neural networks. *Neural Networks*, *132*, 428–446.

Agarap, A. F. (2018). Deep learning using rectified linear units. arXiv preprint arXiv:1803.08375.

Apicella, A., Donnarumma, F., Isgrò, F., & Prevete, R. (2021). A survey on modern trainable activation functions. *Neural Networks*, *138*, 14–32.

B., O'Neill (2021). The double-constant matrix, centering matrix and equicorrelation matrix: Theory and applications. arXiv preprint arXiv:2109.05814.

Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. arXiv preprint arXiv:1607.06450.

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*(2), 157–166.

Burkholz, R., & Dubatovka, A. (2019). Initialization of relus for dynamical isometry. *Advances in Neural Information Processing Systems*, *32*.

Cai, Y. (2022). Achieve the minimum width of neural networks for universal approximation. In *International conference on learning representations*.

Choi, H., Kim, S., Lee, H., & Lim, Y. (2020). Matrix extremal problems and shift invariant means. *Linear Algebra and its Applications*, *587*, 166–194.

Clevert, D. A., Unterthiner, T., & Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (ELUS). In *International conference on learning representation*.

Cortez, P., Cerdeira, A., Almeida, F., Matos, T., & Reis, J. (2009). Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, *47*(4), 547–553.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, *2*(4), 303–314.

Dubey, S. R., Singh, S. K., & Chaudhuri, B. B. (2022). Activation functions in deep learning: A comprehensive survey and benchmark. *Neurocomputing*.

Duch, W., & Jankowski, N. (1999). Survey of neural transfer functions. *Neural Computing Surveys*, *2*(1), 163–212.

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, *12*(7).

Fisher, R. A. (1988). Iris. http://dx.doi.org/10.24432/C56C76, UCI Machine Learning Repository.

Glorot, X., & Bengio, Y. (2010). 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.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., & ...Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*: *vol. 27*.

Hanin, B., & Sellke, M. (2017). Approximating continuous functions by ReLU nets of minimal width. arXiv preprint arXiv:1710.11278.

He, J., Li, L., Xu, J., & Zheng, C. (2018). ReLU deep neural networks and linear finite elements. *Journal of Computational Mathematics*, *38*(3), 502–527.

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).

Hendrycks, D., & Gimpel, K. (2016). Gaussian error linear units (GELUS). arXiv preprint arXiv:1606.08415.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks*, *4*(2), 251–257.

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, *2*(5), 359–366.

Hu, W., Xiao, L., & Pennington, J. (2020). Provable benefit of orthogonal initialization in optimizing deep linear networks. In *International Conference on Learning Representations*.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning* (pp. 448–456). pmlr.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

Klambauer, G., Unterthiner, T., Mayr, A., & Hochreiter, S. (2017). Self-normalizing neural networks. In *Advances in neural information processing systems*: *vol. 30*.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*: *vol. 25*.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

Leshno, M., Lin, V. Y., Pinkus, A., & Schocken, S. (1993). Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, *6*(6), 861–867.

Lu, L., Shin, Y., Su, Y., & Karniadakis, G. E. (2019). Dying ReLU and initialization: Theory and numerical examples. *Communications in Computational Physics*, *28*(5), 1671–1706.

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th international conference on machine learning* (pp. 807–814).

Narkhede, M. V., Bartakke, P. P., & Sutaone, M. S. (2022). A review on weight initialization strategies for neural networks. *Artificial Intelligence Review*, *55*(1), 291–322.

Park, S., Yun, C., Lee, J., & Shin, J. (2020). Minimum width for universal approximation. In *International conference on learning representations*.

Petersen, P., & Voigtlaender, F. (2018). Optimal approximation of piecewise smooth functions using deep ReLU neural networks. *Neural Networks*, *108*, 296–330.

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536.

Salimans, T., & Kingma, D. P. (2016). Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in neural information processing systems*: *vol. 29*.

Saxe, A. M., McClelland, J. L., & Ganguli, S. (2013). Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *International conference on learning representations*.

Srivastava, R. K., Greff, K., & Schmidhuber, J. (2015). Training very deep networks. In *Advances in neural information processing systems*: *vol. 28*.

Sun, Y., Wang, X., & Tang, X. (2015). Deeply learned face representations are sparse, selective, and robust. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2892–2900).

Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *International conference on machine learning* (pp. 1139–1147). PMLR.

Trottier, L., Giguere, P., & Chaib-Draa, B. (2017). Parametric exponential linear unit for deep convolutional neural networks. In *2017 16th IEEE international conference on machine learning and applications* (pp. 207–214). IEEE.

Yarotsky, D. (2017). Error bounds for approximations with deep ReLU networks. *Neural Networks*, *94*, 103–114.

Zhao, J., Schäfer, F., & Anandkumar, A. (2021). ZerO initialization: Initializing neural networks with only zeros and ones. *Transactions on Machine Learning Research*.