Activation Functions and Weight Initializers for CNNs: A comparative study

Quy–Hung Xin[1]*, Duc-Tuan Doan[1]
[1]Faculty of Information Technology, University of Science-VNUHCM, Vietnam
*Corresponding author: xqhung23@apcs.fitus.edu.vn

***Abstract*—**
***Index Terms*—Activation functions, initializer, deep neural networks.**

I. Introduction

In recent years, optimizing neural networks, particularly convolutional neural networks (CNNs), has been a major focus in the deep learning (DL) field. The primary objectives include enhancing learning capacity, improving accuracy for real-world applications, and ensuring model stability during training while maintaining computational efficiency. One strategy to achieve these objectives is improving key components of CNN models, particularly activation functions (AFs) and weight initializers (WIs).

AFs introduce non-linearity into models, enabling them to learn and represent complex patterns in data. Other optimization techniques for CNNs, such as Batch Normalization [9], Skip connections [6], and Dropout [19], while they can improve model performance, often add more additional complexity to the model. In contrast, AFs play a direct role in shaping the training process and significantly influence the learning dynamics, particularly in deep architectures, where models are more prone to vanishing and exploding gradient problems [4]. In fact, the depth of CNN architectures has increased significantly in the past decade (One of the most popular CNN models, ResNet, [6] has up to 152 hidden layers in its largest variant while DenseNet [8] extends to as many as 201 hidden layers), making it important to develop AFs that maintain stable gradient flow, accelerate convergence, and enhance model efficiency.

WIs, on the other hand, determine the initial values of the weights in a neural network before training begins. A poor choice of WI can lead to common issues such as vanishing or exploding gradients, slow convergence, and unstable optimization. Moreover, the combination of AFs and WIs should be carefully considered, as it significantly impacts model performance. For instance, ReLU typically performs better with He initialization than with Xavier initialization [5]. Together, AFs and WIs play a pivotal role in determining the training efficiency, stability, and overall effectiveness of CNNs.

This study aims to conduct a comparative analysis of AF and WI pairs by evaluating their practical performance. First, we provide an analysis of popular activation functions—including ReLU, LeakyReLU, PReLU, ELU[1], GELU[7], Swish, and Mish[13]—as well as initialization methods such as Xavier, He, Orthogonal[18] and Layer-Sequential Unit-Variance (LSUV)[12]. Then, we will conduct experiments to assess the empirical effectiveness of various activation function and weight initializer pairings. Our experiments span two widely used datasets (CIFAR-10 and ImageNet100) across diverse model architectures. Our evaluation will be based on the outcome accuracy and convergence speed. Our contributions are two-fold:

- We provide a comprehensive analysis of state-of-the-art AFs and WIs. As a result, we could have a deeper understanding of the techniques behind these AFs and WIs, which help bridge the gap between observed performance and the underlying mathematical foundations, ultimately leading to more reliable and efficient models.
- Our experiment can thoroughly assess the empirical effectiveness of various AFs and WIs pairings. The results would provide insight into how the interaction between activation functions and weight initializers influences training dynamics, convergence speed, and overall model performance across different network architectures and datasets.

II. Related work

Previous work mainly focused on enhancing AFs. To address the vanishing and exploding gradient problems associated with early AFs (e.g., Sigmoid and Tanh), Nair and Hinton introduced the ReLU activation function, which has since become one of the most widely used AFs [14]. To further enhance network performance, numerous ReLU variants have been proposed. For example, Leaky ReLU [11] introduces a small negative slope to mitigate the dying ReLU problem, while PReLU [5] and ELU [1] generalize traditional ReLU by incorporating learnable parameters, improving a model's ability to learn complex representations. More recently, advanced AFs, such as Swish [16], Mish [13], and GELU [7], have been introduced, demonstrating improved gradient flow and generalization in deep networks.

Meanwhile, the impact of WIs was also explored with He et al. [5] introducing He initialization, which was shown to outperform the earlier Xavier initializer proposed by Glorot and Bengio [3] on networks using the ReLU function. In an effort to stabilize training, Mishkin and Matas introduced Linear LSUV initialization [12], examining its interaction with different AFs. However, their study was limited to traditional

AFs, such as Sigmoid, Tanh, ReLU, and PReLU, leaving the impact of modern AFs largely unexplored.

While previous research has extensively examined AFs and WIs, the interaction between them remains an area of ongoing exploration. Tomasz Szanda [20] investigated the performance of multiple AFs, but did not analyze their compatibility with different weight initialization strategies. More recently, Kit Wong, Rolf Dornberger, and Thomas Hanne [21] studied various AF-WI combinations in a simple feedforward neural network (FNN) [17]. However, the simplicity of their chosen datasets and models may not fully capture the potential of AF-WI interactions in real-world applications.

III. Activation Functions

AFs are an essential component of neural networks, particularly during the training process. Without AFs, a neural network would reduce to a linear combination of inputs, making it insufficient to learn complex patterns and hierarchical representations.

An efficient activation function should exhibit several key properties, including non-linearity, differentiability, computational efficiency, and smoothness for stable gradient flow [15]. Though there are many ways to classify AFs, we are interested in the two characteristics of AFs: saturating and non-saturating.

- Saturating functions approach a fixed limit as the input becomes large (positive or negative), leading to vanishing gradients that hinder training. Examples include Sigmoid and Tanh.
- Non-saturating functions, on the other hand, do not suffer from this issue and are generally preferred in deep learning due to their better gradient flow and faster convergence.

A. ReLU

The Rectified Linear Unit (ReLU) is one of the most widely used activation functions in deep neural networks. It introduces non-linearity to the network while maintaining computational efficiency. ReLU is defined as follows:

$$\mathrm{ReLU}(x) = \max(0, x) = \begin{cases} x, & \text{if } x > 0, \\ 0, & \text{if } x \leq 0. \end{cases}$$

However, ReLU has a major drawback known as the dying ReLU problem, where neurons become inactive and output only zero. This issue occurs when large negative inputs result in a zero gradient, preventing affected neurons from learning further.

B. LeakyReLU

To address the dying ReLU problem, LeakyReLU was then introduced. This function allows a small, positive gradient when the unit is inactive, helping to mitigate the vanishing gradient problem. The LeakyReLU is defined as follows:

$$\mathrm{LeakyReLU}(x) = \begin{cases} x, & \text{if } x \geq 0, \\ \alpha x, & \text{if } x < 0. \end{cases}$$

where α is a small constant, typically set to 0.01.

C. ELU

The Exponential Linear Unit, as known as ELU, was proposed by Clevert et al. [1]. The ELU is defined as follows:

$$\mathrm{ELU}(x) = \begin{cases} x, & \text{if } x > 0, \\ \alpha(e^x - 1), & \text{if } x \leq 0. \end{cases}$$

where α is the learnable parameter. And the derivative of this function is given as:

$$\text{ELU'(x)} = \begin{cases} 1, & \text{if } x > 0, \\ \alpha e^x = \mathrm{ELU}(x) + \alpha, & \text{if } x \leq 0. \end{cases}$$

Unlike ReLU, which outputs zero for negative inputs, ELU allows small negative values, helping reduce shifts and improve learning rate.

D. GELU

The Gaussian Error Linear Unit (GELU) was introduced by Hendrycks & Gimpel in 2016 [7]. GELU applies a Gaussian-based gating mechanism to decide how much input should pass through. It is defined as follows:

$$\mathrm{GELU}(x) = x \cdot \Phi(x)$$

where $\Phi(x)$ is the standard Gaussian cumulative distribution function:

$$\Phi(x) = \frac{1}{2}\left(1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$$

E. Swish

The Swish function was introduced by Ramachandran et al. (2017) [16] as a self-gated activation function. It is defined as follows:

$$\mathrm{Swish}(x) = x \cdot \mathrm{Sigmoid}(x) = \frac{x}{e^x - 1}$$

Like aforementioned functions, Swish also allows a small negative values, preventing from the Dying ReLU problem. As a result, this function can outperform ReLU in deep networks by smoothing activation behavior, making optimization easier.

F. Mish

The Mish activation function was introduced by Misra (2019) [13] as an improvement of Swish and ReLU AFs. Its expression is defined as:

$$\mathrm{Mish}(x) = x \cdot \mathrm{Tanh}\,\mathrm{Softplus}(x)$$

where

$$\mathrm{Softplus}(x) = \ln(1 + e^x)$$

Mish provides smooth gradients, leading to better feature representation in deep networks. In practice, Mish achieves a better accuracy compared to ReLU and Swish in image classification tasks. However, the increased complexity of its formula results in higher computational cost, making it less efficient than ReLU for real-time applications or resource-constrained environments.

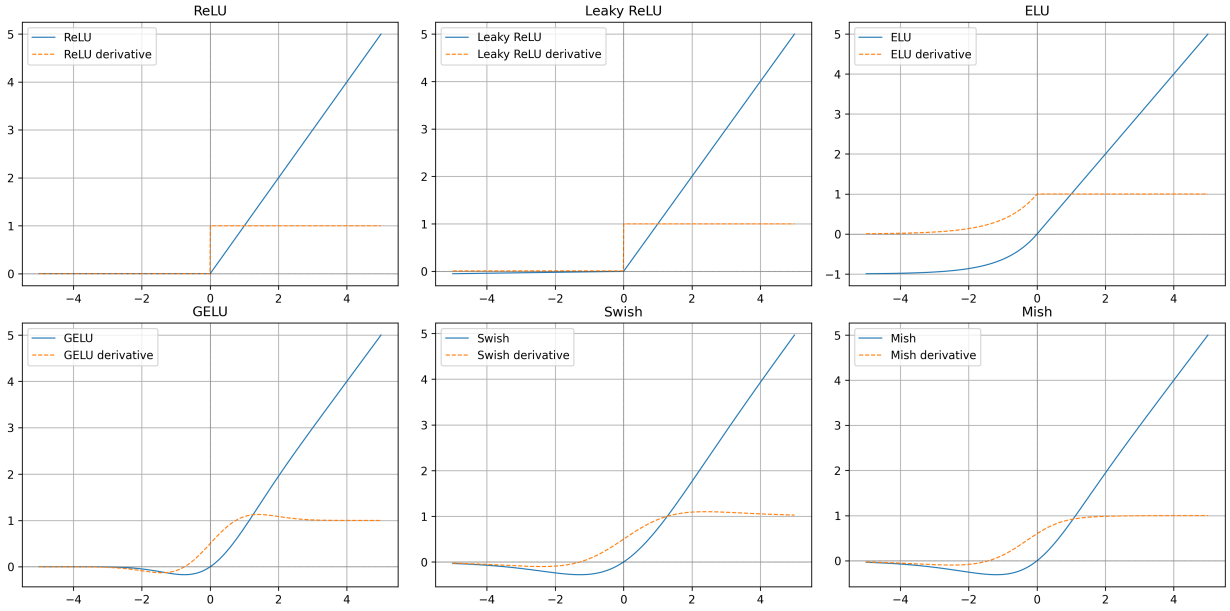

Fig. 1. Graph of Activation Functions and their Derivatives

IV. Weight Initializers

WIs also play a crucial role in neural networks by determining the initial values of parameters before training. The choice of WI directly impacts gradient flow, convergence speed, and overall model stability. [2]

The naive approaches are to initialize all weights to a constant value (e.g., zero) and assign them random small values. However, these methods often result in poor model performance, leading to issues such as symmetry problems or unstable gradient propagation. Thus, more advanced initialization techniques should be used to ensure effective learning and stability.

A. Xavior Initialization

Proposed by Xavier Glorot and Yoshua Bengio [3], Xavier initialization was one of the earliest weight initialization methods designed to improve the stability of deep neural networks. It remains widely used today due to its ability to maintain variance across different layers during both the forward and backward pass, reducing the risk of vanishing and exploding gradients.

There are two variants of Xavier initialization:

- Xavior Uniform Initialization:

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

- Xavior Normal Initialization:

$$W \sim \mathcal{N}\left(0, \frac{1}{n_{\text{in}}+n_{\text{out}}}\right)$$

where n_{in} and n_{out} represent the number of input and output neurons in the layer, respectively. By scaling the initial weights appropriately, Xavier initialization helps preserve stable gradient flow, enabling more efficient training of deep networks.

However, it should be noted that Xavier initialization was originally designed for AFs like Sigmoid and Tanh, which were widely used at the time of its proposal. These functions are approximately linear near zero and symmetric about the origin. Xavier initialization assumes that the variance of activations is preserved across layers under these conditions. However, for non-saturating AFs, such as ReLU, these assumptions no longer hold. For instance, ReLU produces only non-negative outputs, breaking the zero-mean assumption and leading to a mean shift in activations. As a result, Xavier initialization often leads to suboptimal weight scaling for ReLU-based networks, especially in deep networks [10].

B. He initialization

In 2015, He et al. introduced an alternative WI method that addressed the limitations of Glorot and Bengio's (Xavier) initialization [5]. Their method accounts for the non-linearity of AFs, making it one of the most widely used initialization techniques in deep learning.

Similarly, there are two variants of He initialization:

- He uniform initialization:

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

- He normal initialization:

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

where n_{in} represents the number of inputs.

Despite its effectiveness for modern non-saturating AFs, a model may still suffer from issues such as dying ReLU neurons and vanishing gradients in very deep networks. To

address these challenges, several approaches have been proposed, including more sophisticated WI techniques.

C. Orthogonal initialization

Orthogonal initialization was reintroduced by Saxe et al. for recurrent neural networks (RNNs) [18]. However, it has also been shown to improve performance in deep networks by preserving the variance of activations across layers.

The initialization process consists of the following steps:

1) Generate a random matrix $A \in \mathbb{R}^{m \times n}$, where each element follows a normal distribution:
$$A_{ij} \sim \mathcal{N}(0, 1)$$
where m and n denote the input and output dimensions of the layer.
2) Compute the QR decomposition of A:
$$A = QR$$
where Q is an orthogonal matrix and R is an upper triangular matrix.
3) Adjust the sign of Q to maintain consistency:
$$W = Q \cdot \text{sign}(\text{diag}(R))$$
4) Apply a scaling factor γ to adjust the variance:
$$W' = \gamma W$$
where the gain γ is defined as:
$$\gamma = \begin{cases} 1, & \text{for Sigmoid and Tanh activations} \\ \sqrt{2}, & \text{for ReLU and its variants} \end{cases}$$

Orthogonal takes into account different AFs through the scaling steps. However, while it also helps stabilize very deep neural networks, it comes at a higher computational cost compared to He initialization. Due to this limitation, its use is generally not recommended for shallow networks where simpler initializations (e.g., He or Xavier) also demonstrate similar performance.

D. LSUV initialization

Layer-Sequential Unit-Variance (LSUV) initialization was introduced by Mishkin & Matas (2015) [12]. It is a data-driven method which pre-trains the weights based on the data.

1) Initialize the weight matrix W using He initialization:
2) Perform a forward pass with a mini-batch of input data.
3) Compute the variance σ_l^2 of the activations for layer l:
$$\sigma_l^2 = \frac{1}{N} \sum_{i=1}^{N} (h_i^{(l)})^2$$
where N is the batch size and $h_i^{(l)}$ are the activations at layer l.
4) Adjust the weight matrix to normalize the variance:
$$W^{(l)} \leftarrow \frac{W^{(l)}}{\sqrt{\sigma_l^2}}$$
5) Repeat steps 2–4 for each layer until $|\sigma_l^2 - 1| < \epsilon$ (e.g., $\epsilon = 0.01$).

LSUV dynamically adjusts weights to maintain unit-variance activations before training begins, making it a powerful alternative to He initialization for deep networks. However, its extra computational cost means it is best suited for very deep architectures.

V. Experimental design

In this section, we will conduct experiments to evaluate the combinations of AFs and WIs. We are interested in the following questions:

RQ1. *Is there an optimal combination of AF and WI for CNNs?*

RQ2. *How do different pairings of AF and WI influence the CNNs' performance?*

A. Experimental setup

Our experiments are conducted on Google Colab, utilizing an NVIDIA T4 GPU for accelerated training. We employ Python and PyTorch as the primary deep learning framework for implementing and evaluating the models. To evaluate the models' performance, we will apply accuracy, loss, and convergence speed as main metrics.

Previous research [5] [10] showed that He initialization consistently outperforms Xavier initialization for non-saturating activation functions such as ReLU, LeakyReLU, PReLU, and ELU. Given this, we exclude Xavier initialization for these activations to reduce computational overhead. Instead, our study focuses on testing different pairings of AFs and WIs that are more relevant for modern deep networks.

B. CIFAR-10 Dataset

CIFAR-10 is a widely used benchmark dataset for image classification in deep learning and computer vision. It consists of 60,000 32x32 color images with 10 classes. CIFAR-10 is a medium-scale dataset that is suitable for evaluating deep learning models while remaining computationally efficient.

For our experiments on CIFAR-10, we adopt ResNet-18 [6] as the base model. However, we modify the network by systematically replacing:

- The default AFs used between layers.
- The WI scheme for the entire model.

We employ Cross-Entropy Loss as the objective function and Adam optimizer for training, as they are widely used in deep learning for classification tasks.

Provide a model summary here

C. ImageNet-100 dataset

ImageNet-100 is a subset of the widely used ImageNet-1K dataset, containing 100 classes instead of the original 1,000 classes. It retains the diversity and complexity of ImageNet while significantly reducing computational costs, making it an effective benchmark for evaluating deep learning models in resource-constrained environments.

Compared to CIFAR-10, ImageNet-100 presents a more complex classification task. Thus, we adopt ResNet-152 [6] and DenseNet-121 [8] as our baseline models. Similar to our CIFAR-10 experiments, we systematically modify these architectures by replacing the AFs and WI.

Similarly, for training, we also use Cross-Entropy Loss as the objective function and Adam optimizer.

Provide model summary here

VI. Results

This section gives the results of the performance of combination of AFs and WIs aforementions with different datasets and model architecture–including...

A. Results for CIFAR-10

Provide tables and figures here:
Write about the results

B. Results for ImageNet-100

Provide tables and figures here
Write about the results

VII. Dicussion

VIII. Conclusion

References

[1] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[2] Chitra Desai and C Desai. Impact of weight initialization techniques on neural network efficiency and performance: A case study with mnist dataset. *International Journal Of Engineering And Computer Science*, 13(04), 2024.

[3] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[4] Boris Hanin. Which neural net architectures give rise to exploding and vanishing gradients? *Advances in neural information processing systems*, 31, 2018.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[7] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[8] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[9] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[10] Siddharth Krishna Kumar. On weight initialization in deep neural networks. *arXiv preprint arXiv:1704.08863*, 2017.

[11] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Atlanta, GA, 2013.

[12] Dmytro Mishkin and Jiri Matas. All you need is a good init. *arXiv preprint arXiv:1511.06422*, 2015.

[13] Diganta Misra. Mish: A self regularized non-monotonic activation function. *arXiv preprint arXiv:1908.08681*, 2019.

[14] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[15] Gaurav Kumar Pandey and Sumit Srivastava. Resnet-18 comparative analysis of various activation functions for image classification. In *2023 International Conference on Inventive Computation Technologies (ICICT)*, pages 595–601. IEEE, 2023.

[16] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Swish: a self-gated activation function. *arXiv: Neural and Evolutionary Computing*, 2017.

[17] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[18] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

[19] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[20] Tomasz Szandała. Review and comparison of commonly used activation functions for deep neural networks. *Bio-inspired neurocomputing*, pages 203–224, 2021.

[21] Kit Wong, Rolf Dornberger, and Thomas Hanne. An analysis of weight initialization methods in connection with different activation functions for feedforward neural networks. *Evolutionary Intelligence*, 17(3):2081–2089, 2024.