NEURAL NETWORKS WITH TRAINABLE MATRIX ACTIVATION FUNCTIONS

Zhengqi Liu,*[1] *Shuhao Cao,*[2] *Yuwen Li,*[3] *&
***Ludmil Zikatanov*[4,*]**

[1]*Department of Mathematics, The Pennsylvania State University, University Park, PA 16802, USA*

[2]*School of Science and Engineering, University of Missouri–Kansas City, Kansas City, MO 64110, USA*

[3]*School of Mathematical Sciences, Zhejiang University, Hangzhou, Zhejiang 310058, China*

[4]*National Science Foundation, Alexandria, VA 22314, USA*

[*]Address all correspondence to: Ludmil Zikatanov, National Science Foundation, Alexandria, VA, USA, E-mail: lzikatan@nsf.gov

The training process of neural networks usually optimizes weights and bias parameters of linear transformations, while nonlinear activation functions are prespecified and fixed. This work develops a systematic approach to constructing matrix-valued activation functions whose entries are generalized from rectified linear unit (ReLU). The activation is based on matrix-vector multiplications using only scalar multiplications and comparisons. The proposed activation functions depend on parameters that are trained along with the weights and bias vectors. Neural networks based on this approach are simple and efficient and are shown to be robust in numerical experiments.

KEY WORDS: *machine learning, ReLU, ParaReLU, trainable activation*

1. INTRODUCTION

In recent decades, deep neural networks (DNNs) have achieved significant success in many fields such as computer vision and natural language processing (Otter et al., 2018; Voulodimos et al., 2018). The DNN surrogate model is constructed using recursive composition of linear transformations and nonlinear activation functions. The nonlinear activation functions are essential in providing universal approximation, and problem-appropriate choices of them are vital to the model's performance.

In the original universal approximation (Cybenko, 1989; Funahashi, 1989), sigmoid is used due to its property converging to 1 and 0 as the input goes to $\pm\infty$ and continuously changing within. However, in the modern application where the neural network layers are composed to be deeper and deeper while the whole community shifted from fp64 to fp32, sigmoid activation suffers from the "vanishing gradient" (Kolen and Kremer, 2001) during the training process, as the chain rule multiplies multiple small gradients resulting from sigmoid and eventually causes numerical underflow (You et al., 2017).

In practice, the rectified linear unit (ReLU) is one of the most popular activation functions due to its simplicity and efficiency. Moreover, it resolves the vanishing gradient problem completely, allowing a large DNN stacked with up to hundreds of layers such as the ones in He

Submitted: 10/28/24; Accepted: 10/28/24; Online: 12/4/24
https://doi/10.1615/JMachLearnModelComput.2024056966

et al. (2015a). Nevertheless, a key drawback of ReLU is the "dying ReLU" problem (Lu et al., 2020) where, if during training a neuron's ReLU activation becomes 0, then under certain circumstances it may never get activated again to output a nonzero value.

Several simple modifications have been proposed to address this problem and have achieved a certain level of success, e.g., the simple leaky ReLU, and piecewise linear unit (PLU) (Nicolae, 2018), Softplus (Glorot et al., 2011), exponential linear unit (ELU) (Clevert et al., 2016), scaled exponential linear unit (SELU) (Klambauer et al., 2017), and Gaussian error linear unit (GELU) (Hendrycks and Gimpel, 2016).

Although the aforementioned activation functions are shown to be competitive in benchmark tests, they are still fixed nonlinear functions. In a DNN structure, it is often hard to determine *a priori* the optimal activation function for a specific application. Empirically, there has been a community effort in search for a better activation (Ramachandran et al., 2017), and currently the consensus is that GELU works well in large models such as GPT (Radford et al., 2019) as it avoids the vanishing gradient, dying ReLU, and the shattered gradient problem (Balduzzi et al., 2017). In general, GELU-like activations provide a gradient in the negative regime to stop neurons "dying" while bounding how far into the negative regime activations are able to have an effect, and this allows for a better cross-layer training procedure.

However, all the activations mentioned above are *pointwise*, where this pointwiseness refers to the fact that the activations are scalar functions that only use a single componentwise value to determine its output, given a tensorial input. In this paper, we shall generalize these activation functions and introduce a mechanism to create matrix-valued activation functions, which is trainable to control the gradient magnitude in a data-adaptive fashion. The effectiveness of the proposed method is validated using function approximation examples and well-known benchmark datasets such as `MNIST` and `CIFAR-10`. There are a few classical works on adaptively tuning parameters in the training process, e.g., the parametric ReLU (He et al., 2015b). However, our adaptive matrix-valued activation functions are shown to be competitive and more robust in those experiments.

1.1 Preliminaries

For the simplicity of presentation, we consider a simple model data-fitting problem, given a training set containing $\{(x_n, f_n)\}_{n=1}^N$, where the coordinates $\{x_n\}_{n=1}^N \subset \mathbb{R}^d$ are the inputs and $\{f_n\}_{n=1}^N \subset \mathbb{R}^J$ are the output. They are implicitly related via an unknown target function $f : \mathbb{R}^d \to \mathbb{R}^J$ with the assumption that $f_n = f(x_n)$. The ReLU activation function is a piecewise linear function given by

$$\sigma(t) = \max\{t, 0\}, \quad \text{for} \quad t \in \mathbb{R}.$$

In the literature σ is acting componentwise on an input vector. In a DNN, let L be the number of layers and n_ℓ denote the number of neurons at the ℓth layer for $0 \le \ell \le L$ with $n_0 = d$ and $n_L = J$. Let $\mathcal{W} = (W_1, W_2, \ldots, W_L) \in \prod_{\ell=1}^L \mathbb{R}^{n_\ell \times n_{\ell-1}}$ denote the tuple of admissible weight matrices and $\mathcal{B} = (b_1, b_2, \ldots, b_L) \in \prod_{\ell=1}^L \mathbb{R}^{n_\ell}$ the tuple of admissible bias vectors. The ReLU DNN approximation to f at the ℓth layer is recursively defined as

$$\eta_\ell(x) := \sigma(W_\ell \eta_{\ell-1}(x) + b_\ell) \in \mathbb{R}^{n_\ell}, \quad \eta_0(x) = x \in \mathbb{R}^d. \tag{1}$$

The traditional training process for such a DNN is to find optimal $\mathcal{W}_* \in \prod_{\ell=1}^L \mathbb{R}^{n_\ell \times n_{\ell-1}}$, $\mathcal{B}_* \in \prod_{\ell=1}^L \mathbb{R}^{n_\ell}$ (and thus optimal $\eta_L = \eta_{L,\mathcal{W}_*,\mathcal{B}_*}$) such that

$$(\mathcal{W}_*, \mathcal{B}_*) = \arg\min_{\mathcal{W},\mathcal{B}} E(\mathcal{W}, \mathcal{B}), \quad \text{where} \quad E(\mathcal{W}, \mathcal{B}) = \sum_{n=1}^{N} |f_n - \eta_{L,\mathcal{W},\mathcal{B}}(x_n)|^2. \quad (2)$$

In other words, $\eta_{L,\mathcal{W}_*,\mathcal{B}_*}$ best fits the data with respect to the discrete ℓ^2-norm within the function class $\{\eta_{L,\mathcal{W},\mathcal{B}}\}$. In practice, the sum of squares norm in E could be replaced with more convenient norms.

2. TRAINABLE MATRIX-VALUED ACTIVATION FUNCTION

Having a closer look at ReLU σ, we have a simple but quite useful observation that the activation $\sigma(\xi_\ell(x))$ with $\xi_\ell := W_\ell \eta_{\ell-1} + b_\ell$ could be written as a matrix-vector multiplication $\sigma(\xi_\ell(x)) = D_\ell(\xi_\ell(x))\xi_\ell(x)$, where D_ℓ is a *diagonal* matrix-valued function mapping from $\mathbb{R}^{n_\ell}$ to $\mathbb{R}^{n_\ell \times n_\ell}$ with diagonal being $1_{\{(0,\infty)\}}(s)$, thus taking values from the discrete set $\{0, 1\}$. There is no reason to restrict on $\{0, 1\}$ and we thus look for a larger set of values over which the diagonal entries of D_ℓ are running or sampled. With a slight abuse of notation, our new DNN approximation to f is calculated using the following recurrence relation:

$$\eta_0(x) = x \in \mathbb{R}^d, \quad \xi_\ell(x) = W_\ell \eta_{\ell-1}(x) + b_\ell, \quad \eta_\ell = (D_\ell \circ \xi_\ell)\xi_\ell, \quad \ell = 1, \ldots, L. \quad (3)$$

Here each D_ℓ is diagonal and is of the form

$$D_\ell(y) = \text{diag}(\alpha_{\ell,1}(y_1), \alpha_{\ell,2}(y_2), \ldots, \alpha_{\ell,n_\ell}(y_{n_\ell})), \quad y \in \mathbb{R}^{n_\ell}, \quad (4)$$

where $\alpha_{\ell,i}(y_i)$ is a nonlinear function to be determined. Since piecewise constant functions can approximate a continuous function within arbitrarily high accuracy, we specify $\alpha_{\ell,i}$ with $1 \leq i \leq n_\ell$ as

$$\alpha_{\ell,i}(s) = \begin{cases} t_{\ell,i,0}, & s \in (-\infty, s_{\ell,i,1}], \\ t_{\ell,i,1}, & s \in (s_{\ell,i,1}, s_{\ell,i,2}], \\ \vdots & \\ t_{\ell,i,m_{\ell,i}-1}, & s \in (s_{\ell,i,m_{\ell,i}-1}, s_{\ell,i,m_{\ell,i}}], \\ t_{\ell,i,m_{\ell,i}}, & s \in (s_{\ell,i,m_{\ell,i}}, \infty), \end{cases} \quad (5)$$

where $m_{\ell,i}$ is a positive integer and $\{t_{\ell,i,j}\}_{j=0}^{m_{\ell,i}}$ and $\{s_{\ell,i,j}\}_{j=1}^{m_{\ell,i}}$ are constants. We may suppress the indices ℓ, i in $\alpha_{\ell,i}$, $m_{\ell,i}$, $t_{\ell,i,j}$, $s_{\ell,i,j}$ and write them as α, m, t_j, s_j when those quantities are uniform across layers and neurons. If $m = 1$, $s_1 = 0$, $t_0 = 0$, $t_1 = 1$, then the DNN [Eq. (3)] is exactly the ReLU DNN. If $m = 1$, $s_1 = 0$, $t_1 = 1$ and t_0 is a fixed small negative number, Eq. (3) reduces to the DNN based on leaky ReLU. If $m = 2$, $s_1 = 0$, $s_2 = 1$, $t_0 = t_2 = 0$, $t_1 = 1$, then $\alpha = \alpha_{\ell,i}$ actually represents a discontinuous activation function.

In our case, we shall fix some parameters from $\cup_{\ell=1}^{L} \cup_{i=1}^{n_\ell} \{t_{\ell,i,j}\}_{j=0}^{m_{\ell,i}}$ and $\cup_{\ell=1}^{L} \cup_{i=1}^{n_\ell} \{s_{\ell,i,j}\}_{j=1}^{m_{\ell,i}}$ and let the rest of them vary in the training process. When the diagonal cutoffs are fixed while making the slopes learnable, this replicates the layerwise adaptive rate scaling in You et al. (2017). Heuristically speaking, the activation functions among different layers in the resulting DNN may adapt to the target function f. Since the nonparametric ReLU and a 1-parameter leaky ReLU are special cases of the new activation functions, the proposed DNN with the new activations theoretically should bear at least the same approximation property. If the optimization problem is solved exactly, in practice the training error should be no worse than before. In the following, the activation function in Eq. (5) with trainable parameters in Eq. (3) is named as the "trainable matrix-valued activation function (TMAF)."

Starting from the diagonal activation D_ℓ, one can step further to construct more general activation matrices. First we note that D_ℓ could be viewed as a nonlinear operator $T_\ell : [C(\mathbb{R}^d)]^{n_\ell} \to [C(\mathbb{R}^d)]^{n_\ell}$, where

$$[T_\ell(g)](x) = D_\ell(g(x))g(x), \quad g \in [C(\mathbb{R}^d)]^{n_\ell}, \quad x \in \mathbb{R}^d.$$

With this observation, one can parametrize a trainable nonlinear activation *operator* determined by more general matrices, e.g., the following tridiagonal operator:

$$[T_\ell(g)](x) = \begin{pmatrix} \alpha_{\ell,1} & \beta_{\ell,2} & 0 & \cdots & 0 \\ \gamma_{\ell,1} & \alpha_{\ell,2} & \beta_{\ell,3} & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \alpha_{\ell,n_\ell-1} & \beta_{\ell,n_\ell} \\ 0 & 0 & \cdots & \gamma_{\ell,n_\ell-1} & \alpha_{\ell,n_\ell} \end{pmatrix} g(x), \quad x \in \mathbb{R}^d. \tag{6}$$

The diagonal $\{\alpha_{\ell,i}\}$ is given in Eq. (5) while the off-diagonals $\beta_{\ell,i}$, $\gamma_{\ell,i}$ are piecewise constant functions in the ith coordinate y_i of $y \in \mathbb{R}^{n_\ell}$ defined in a fashion similar to $\alpha_{\ell,i}$. Theoretically speaking, even trainable full matrix activation is possible despite potentially increased training cost. In summary, the corresponding DNN based on trainable nonlinear activation operators $\{T_\ell\}_{\ell=1}^L$ reads

$$\eta_0(x) = x \in \mathbb{R}^d, \quad \xi_\ell(x) = W_\ell \eta_{\ell-1}(x) + b_\ell, \quad \eta_\ell := T_\ell(\xi_\ell), \quad \ell = 1, \dots, L. \tag{7}$$

The evaluations of D_ℓ and T_ℓ are cheap because they require only scalar multiplications and comparisons. When calling a general-purpose package such as PyTorch in the training process, it is observed that the computational time of D_ℓ and T_ℓ is comparable to the classical ReLU.

Remark 1. Our observation also applies to an activation function σ other than ReLU. For example, we may rescale $\sigma(x)$ to obtain $\sigma(\omega_{i,\ell}x)$ for constants $\{\omega_{i,\ell}\}$ varying layer by layer and neuron by neuron. Then $\sigma(\omega_{i,\ell}x)$ are used to form a matrix activation function and a TMAF DNN, where $\{\omega_{i,\ell}\}$ are trained according to given data and are adapted to the target function. This observation may be useful for specific applications.

Remark 2. We also note that the diagonal activation with a fixed bandwidth bears similarity with the piecewise-linear activation after the convolution layer in a so-called alternating direction method of multipliers (ADMM)-net (Yang et al., 2017), yet is designed with a fundamentally different philosophy. In Yang et al. (2017), an activation matrix with the size of the filter is composed of the convolution operation. If one views the convolution operation as a diagonal matrix-valued operator, for a $p \times p$ convolution filter, the activation is a $n^2 \times n^2$ diagonal matrix with the bandwidth being p^2. The nonlinearity in ADMM couples neighboring pixels in an image, while here TMAF performs a nonlinear mixing in the channel dimension.

3. NUMERICAL RESULTS

In this section, we demonstrate the feasibility and efficiency of TMAF by comparing it with the traditional ReLU-type activation functions. In principle, all parameters in Eq. (5) are allowed to be trained while we shall fix the intervals in Eq. (5) and only let function values $\{t_j\}$ vary for simplicity in the following. In each experiment, we use the same neural network structure, as well as the same learning rates, stochastic gradient descent (SGD) optimization, and number of epochs (NE) (SGD iterations). In particular, the learning rate 1e–4 is used for epochs 1 to NE$/2$ and 1e–5 is used for epochs NE$/2+1$ to NE.

3.1 Function Approximation (Regression) Problem

For the first class of examples we use the ℓ^2-loss function as defined in Eq. (2). For the classification problems we consider the *cross-entropy* that is widely used as a loss function in classification models. The cross-entropy is defined using a training set which consists of p images, each with N pixels. Thus, we have a matrix $Z \in \mathbb{R}^{N\times p}$ and each column corresponds to an image with N pixels. Each image belongs to a fixed class c_j from the set of image classes $\{c_k\}_{k=1}^{p}$, where $c_j \in \{1,\dots,M\}$. The network structure maps $Z \in \mathbb{R}^{N\times p}$ to $X \in \mathbb{R}^{M\times p}$, and each column x_j of X is an output of the network evaluation at the corresponding column z_j of Z. More precisely,

$$Z = (z_1,\dots,z_p), \quad X = (x_1,\dots,x_p), \quad c_j = \text{class}(z_j),$$

$$x_j := \eta_{L,\mathcal{W},\mathcal{B}}(z_j), \quad x_j \in \mathbb{R}^M, \quad z_j \in \mathbb{R}^N, \quad j = 1,\dots,p.$$

The cross-entropy loss function then is defined by

$$\mathcal{C}(\mathcal{W},\mathcal{B}) = \sum_{k=1}^{p} -\log\left(\frac{\exp(x_{c_k,k})}{\sum_{j=1}^{M}\exp(x_{j,k})}\right),$$

$$(\mathcal{W}_*,\mathcal{B}_*) = \arg\min_{\mathcal{W},\mathcal{B}}\mathcal{C}(\mathcal{W},\mathcal{B}).$$

To evaluate the loss function at a given image $z \in \mathbb{R}^N$, we first evaluate the network at z with the given $(\mathcal{W},\mathcal{B}) = (\mathcal{W}_*,\mathcal{B}_*)$. We then define the class $c(z)$ of z and the loss loss(z) at z as follows:

$$\text{c}(z) = \arg\min_{1\le j\le M}\{\exp(a_j)\}, \quad \text{where} \quad a = \eta_{L,\mathcal{W}_*,\mathcal{B}_*}(z),$$

$$\text{loss}(z) = -\log\left(\frac{\exp(a_{c(z)})}{\sum_{j=1}^{M}\exp(a_j)}\right).$$

3.1.1 Approximation of a Smooth Function

As our first example, we use neural networks to approximate

$$f(x_1,\cdots,x_n) = \sin(\pi x_1 + \cdots + \pi x_n), \quad x_k \in [-2,2], \quad k = 1,\dots,n.$$

The training datasets are 20,000 input-output data pairs where the input data are randomly sampled from the hypercube $[-2,2]^n$. The networks [Eqs. (1) and (3)] have single or double hidden layers with 20 neurons per layer. For TMAF D_ℓ in Eq. (4), the function $\alpha = \alpha_{\ell,i}$ in Eq. (5) uses intervals $(-\infty,-5)$, $(-5+k,-4+k]$, $(5,\infty)$, $0 \le k \le 9$. The approximation results are shown in Table 1 and Figs. 1–3. It is observed that TMAF is the most accurate activation approach. Moreover, the parametric ReLU does not approximate $\sin(\pi x_1 + \ldots + \pi x_6)$ well; see Fig. 3(b).

TABLE 1: Approximation errors for $\sin(\pi x_1 + \cdots + \pi x_n)$ by neural networks

	Approximation Error							
	Single Hidden Layer				**Two Hidden Layers**			
n	1	2	3	4	5	6	7	8
ReLU	0.089	0.34	0.39	0.41	0.14	0.21	0.25	0.31
TMAF	0.015	0.016	0.13	0.18	0.07	0.105	0.153	0.17

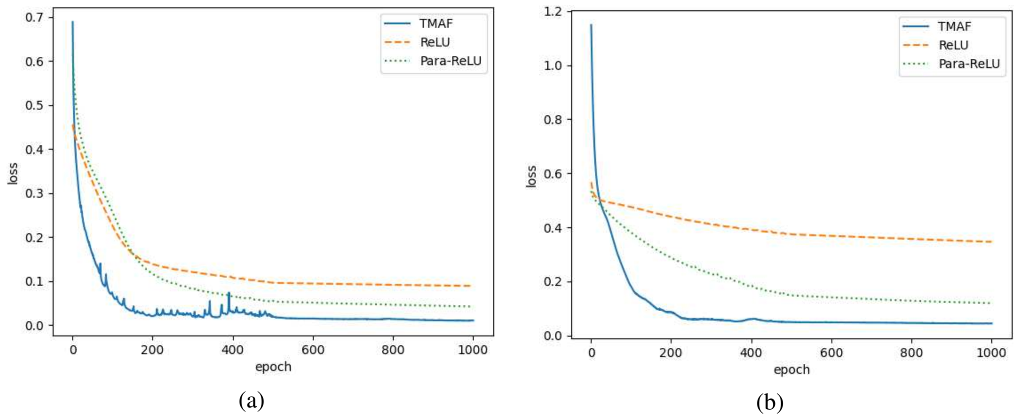


FIG. 1: Training errors for $\sin(\pi x_1 + \cdots + \pi x_n)$, single hidden layer: (a) $n = 1$ and (b) $n = 2$

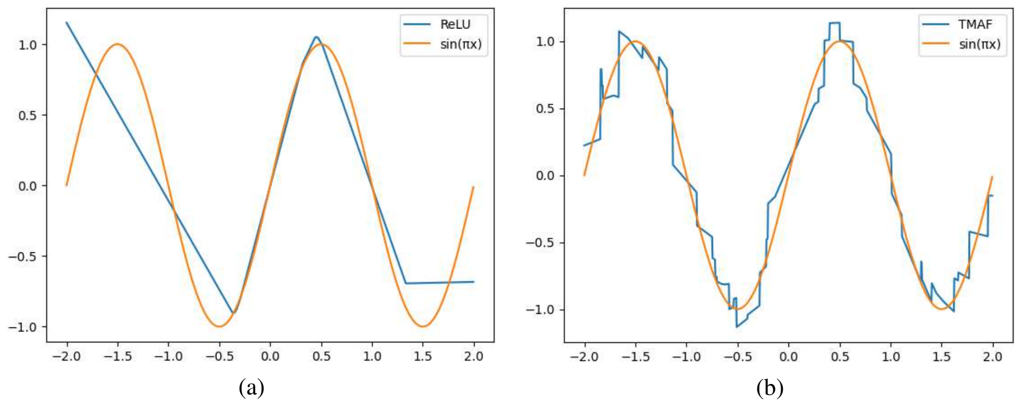


FIG. 2: Neural network approximations to $\sin(\pi x)$, single hidden layer: (a) ReLU and (b) TMAF

3.1.2 Approximation of an Oscillatory Multifrequency Function

The next example is of approximating the following function having high- and low-frequency components

$$f(x) = \sin(100\pi x) + \cos(50\pi x) + \sin(\pi x), \tag{8}$$

see Fig. 4 for an illustration. The function in Eq. (8) is notoriously difficult to capture by numerical methods in scientific computing. In the context of approximation using NNs, it is observed

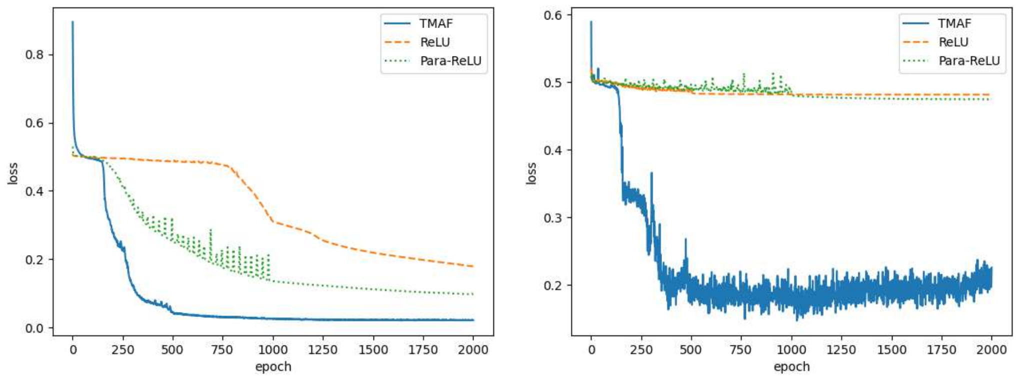


FIG. 3: Training errors for $\sin(\pi x_1 + \cdots + \pi x_n)$, two hidden layers: (a) $n = 5$ and (b) $n = 6$

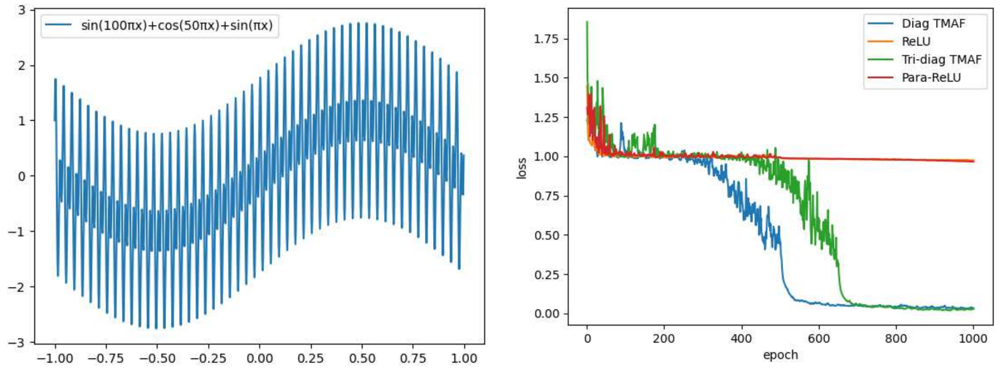


FIG. 4: (a) Plot of $f(x) = \sin(100\pi x) + \cos(50\pi x) + \sin(\pi x)$, exact oscillating function, and (b) training loss comparison

in Hong et al. (2022) that ReLU-based NN cannot resolve the high-frequency oscillatory feature of this function at all. The training datasets are 20,000 input-output data pairs where the input data are randomly sampled from the interval $[-1, 1]$. We test the diagonal TMAF [Eq. (4)] and the function $\alpha = \alpha_{\ell,i}$ [Eq. (5)] uses intervals $(-\infty, -5)$, $(-5 + kh, -5 + (k + 1)h]$, $(5, \infty)$ with $h = 0.1$, $0 \leq k \leq 99$. We also consider the tridiagonal TMAF [Eq. (6)], where $\{\alpha_{\ell,i}\}$ is the same as the diagonal TMAF, $\{\beta_{\ell,i}\}$ and $\{\gamma_{\ell,i}\}$ are all piecewise constants based on intervals $(-\infty, -5 + \underline{h})$, $(-5 + kh + \underline{h}, -5 + (k + 1)h + \underline{h}]$, $(5 + \underline{h}, \infty)$ and $(-\infty, -5 + 2\underline{h})$, $(-5 + kh + 2\underline{h}, -5 + (k + 1)h + 2\underline{h}]$, $(5 + 2\underline{h}, \infty)$ with $\underline{h} = 0.1/3$, $0 \leq k \leq 99$, respectively. Numerical results can be found in Figs. 4 and 5 and Table 2.

For this challenging problem, we note that the diagonal TMAF and tridiagonal TMAF produce high-quality approximations while ReLU and parametric ReLU are not able to approximate the highly oscillating function within reasonable accuracy. It is observed from Fig. 5 that ReLU actually approximates the low-frequency part of Eq. (8). To capture the high-frequency, part ReLU clearly has to use more neurons and thus much more weight and bias parameters. On the other hand, increasing the number of intervals in TMAF only leads to a few more training parameters.

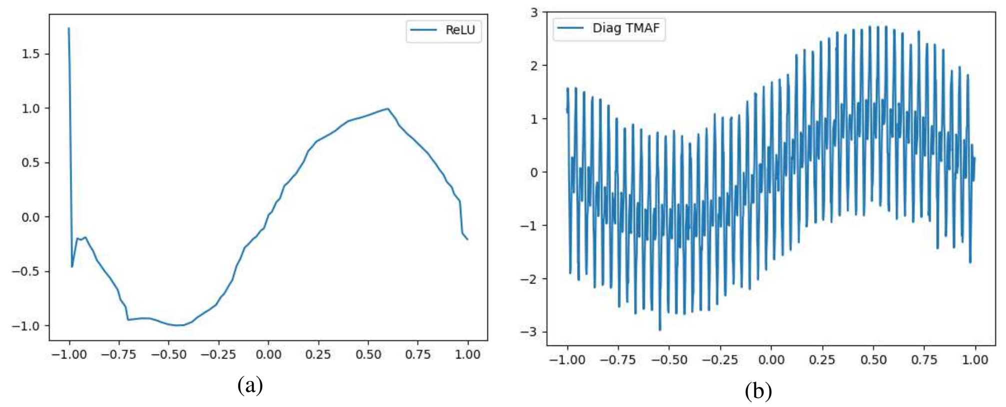


FIG. 5: Approximations to $f(x) = \sin(100\pi x) + \cos(50\pi x) + \sin(\pi x)$ by neural networks: (a) ReLU approximation and (b) TMAF approximation

TABLE 2: Error comparison for $f(x) = \sin(100\pi x) + \cos(50\pi x) + \sin(\pi x)$

	Error
RELU	0.97
Diag-TMAF	0.033
Tri-diag TMAF	0.029

3.2 Classification Problem of MNIST and CIFAR-10 Datasets

We now test TMAF by classifying images in the `MNIST` and `CIFAR-10` datasets. For TMAF D_ℓ in Eq. (4), the function $\alpha = \alpha_{\ell,i}$ [Eq. (5)] uses intervals $(-\infty, -5)$, $(-5+k, -4+k]$, $(5, \infty)$ with $0 \leq k \leq 9$.

For the `MNIST` set, we implement single and double layer fully connected networks [Eqs. (1) and (3)] with 10 neurons per layer (except at the first layer $n_0 = 764$), and ReLU or diagonal TMAF [Eq. (4)] activation. Numerical results are shown in Figs. 6 and 7 and Table 3. We note

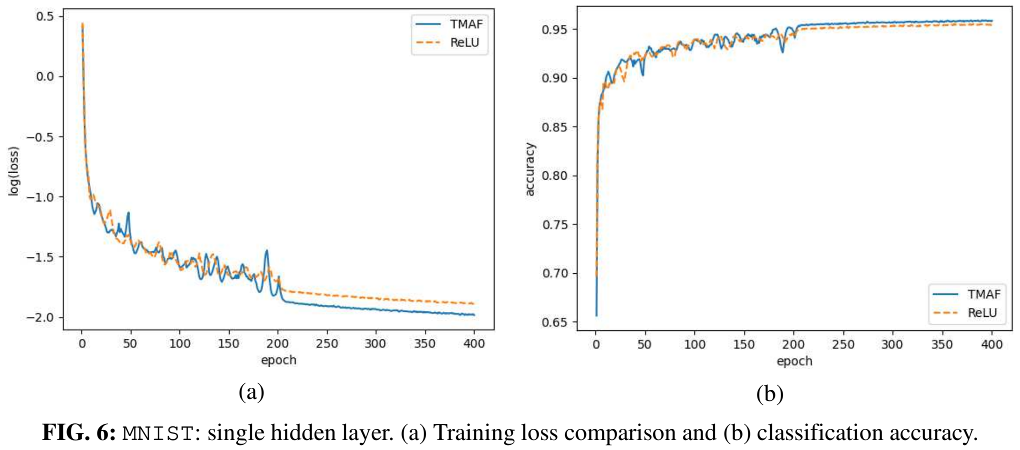
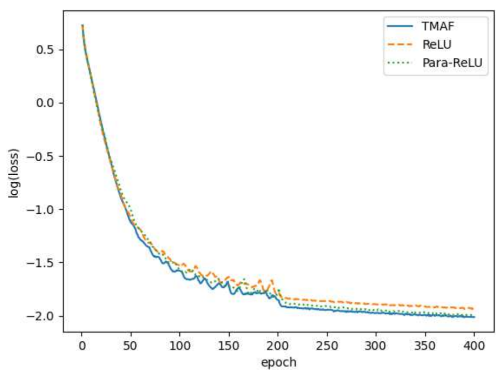
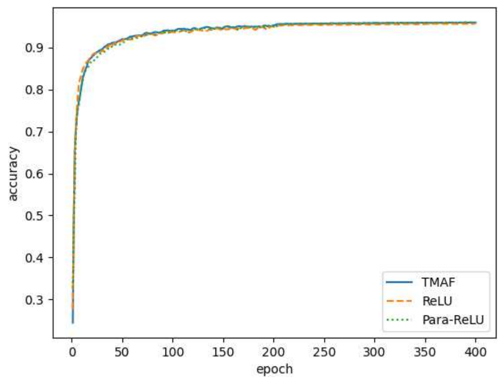


FIG. 6: `MNIST`: single hidden layer. (a) Training loss comparison and (b) classification accuracy.



(a)



(b)

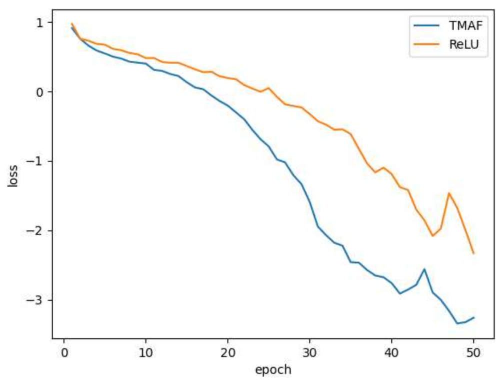
FIG. 7: `MNIST`: two hidden layers. (a) Training loss and (b) classification accuracy

TABLE 3: Evaluation accuracy for the `MNIST` and `CIFAR-10`

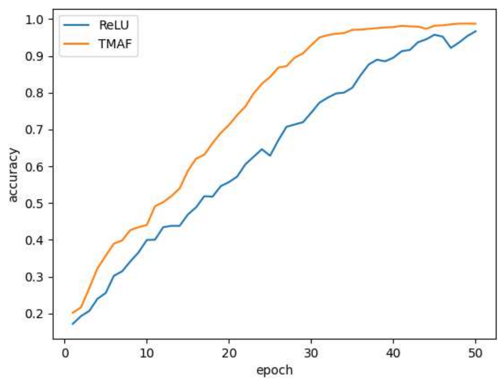
Dataset	Evaluation Accuracy	
	ReLU	TMAF
`MNIST` (1 hidden layer)	86.1%	92.1%
`MNIST` (2 hidden layers)	91.8%	92.2%
`CIFAR-10` (Resnet18)	92.8%	93.2%

that the TMAF with a single hidden layer ensures higher evaluation accuracy than ReLU; see Table 3.

For the `CIFAR-10` dataset, we use the `ResNet18` network structure with 18 layers and number of neurons provided by He et al. (2015a). The activation functions are still ReLU and the diagonal TMAF [Eq. (4)]. Numerical results are presented in Fig. 8 and Table 3. Those



(a)



(b)

FIG. 8: Comparison between ReLU and TMAF for `CIFAR-10`: (a) training loss and (b) classification accuracy

parameters given in Paszke et al. (2017) are already tuned well with respect to ReLU. Nevertheless, TMAF still produces smaller errors in the training process and returns better classification results in the evaluation stage.

It is possible to improve the performance of TMAF applied to those benchmark datasets. The key point is to select suitable intervals in $\alpha_{\ell,i}$ to optimize the performance. A simple strategy is to let those intervals in Eq. (5) be varying and adjusted in the training process, which will be investigated in our future research.

ACKNOWLEDGMENTS

The research of Ludmil Zikatanov is based upon work supported by and while serving at the National Science Foundation. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

The research of Shuhao Cao is supported partially by the National Science Foundation award DMS-2309778.

REFERENCES

Balduzzi, D., Frean, M., Leary, L., Lewis, J., Ma, K.W.-D., and McWilliams, B., The Shattered Gradients Problem: If Resnets Are the Answer, Then What Is the Question?, in *Int. Conf. on Machine Learning*, Sydney, Australia, pp. 342–350, 2017.

Clevert, D., Unterthiner, T., and Hochreiter, S., Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs), in *4th Int. Conf. on Learning Representations, ICLR 2016*, San Juan, Puerto Rico, May 2–4, 2016.

Cybenko, G., Approximation by Superpositions of a Sigmoidal Function, *Math. Control Signals Syst.*, vol. **2**, no. 4, pp. 303–314, 1989.

Funahashi, K.-I., On the Approximate Realization of Continuous Mappings by Neural Networks, *Neural Networks*, vol. **2**, no. 3, pp. 183–192, 1989.

Glorot, X., Bordes, A., and Bengio, Y., Deep Sparse Rectifier Neural Networks, in *Proc. of the Fourteenth Int. Conf. on Artificial Intelligence and Statistics, AISTATS 2011*, Fort Lauderdale, FL, pp. 315–323, April 11–13, 2011.

He, K., Zhang, X., Ren, S., and Sun, J., Deep Residual Learning for Image Recognition, arXiv Preprint arXiv: 1512.03385, 2015a.

He, K., Zhang, X., Ren, S., and Sun, J., Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, in *2015 IEEE Int. Conf. on Computer Vision (ICCV)*, Los Alamitos, CA, pp. 1026–1034, 2015b.

Hendrycks, D. and Gimpel, K., Bridging Nonlinearities and Stochastic Regularizers with Gaussian Error Linear Units, arXiv Preprint arXiv: 1606.08415, 2016.

Hong, Q., Siegel, J.W., Tan, Q., and Xu, J., On the Activation Function Dependence of the Spectral Bias of Neural Networks, arXiv Preprint arXiv:2208.04924, 2022.

Klambauer, G., Unterthiner, T., Mayr, A., and Hochreiter, S., Self-Normalizing Neural Networks, in *Advances in Neural Information Processing Systems 30: Annual Conf. on Neural Information Processing Systems 2017*, Long Beach, CA, pp. 971–980, December 4–9, 2017.

Kolen, J.F. and Kremer, S.C., Gradient Flow in Recurrent Nets: The Difficulty of Learning Long-Term Dependencies, *A Field Guide to Dynamical Recurrent Networks*, Hoboken, NJ: Wiley, pp. 237–243, 2001.

Lu, L., Shin, Y., Su, Y., and Karniadakis, G.E., Dying ReLU and Initialization: Theory and Numerical Examples, *Commun. Comput. Phys.*, vol. **28**, no. 5, pp. 1671–1706, 2020.

Nicolae, A., PLU: The Piecewise Linear Unit Activation Function, arXiv Preprint arXiv: 1809.09534, 2018.

Otter, D.W., Medina, J.R., and Kalita, J.K., A Survey of the Usages of Deep Learning in Natural Language Processing, arXiv Preprint arXiv: 1807.10854, 2018.

Paszke, A., Gross, S., and Chintal, S., PyTorch, accessed from https://github.com/pytorch/pytorch, 2017.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I., Language Models are Unsupervised Multitask Learners, *OpenAI Blog*, vol. **1**, no. 8, p. 9, 2019.

Ramachandran, P., Zoph, B., and Le, Q.V., Searching for Activation Functions, arXiv Preprint arXiv:1710.05941, 2017.

Voulodimos, A., Doulamis, N., Doulamis, A., and Protopapadakis, E., Deep Learning for Computer Vision: A Brief Review, *Comput. Intell. Neurosci.*, vol. **2018**, pp. 1–13, 2018.

Yang, Y., Sun, J., Li, H., and Xu, Z., ADMM-Net: A Deep Learning Approach for Compressive Sensing MRI, arXiv Preprint arXiv:1705.06869, 2017.

You, Y., Gitman, I., and Ginsburg, B., Large Batch Training of Convolutional Networks, arXiv Preprint arXiv:1708.03888, 2017.