

STRUCTURED AND BALANCED MULTICOMPONENT AND MULTILAYER NEURAL NETWORKS*

SHIJUN ZHANG[†], HONGKAI ZHAO[‡], YIMIN ZHONG[§], AND HAOMIN ZHOU[¶]

Abstract. In this work, we propose a balanced multicomponent and multilayer neural network (MMNN) structure to accurately and efficiently approximate functions with complex features in terms of both degrees of freedom and computational cost. The main idea is inspired by a multicomponent approach in which each component can be effectively approximated by a single-layer network, combined with a multilayer decomposition strategy to capture the complexity of the target function. Although MMNNs can be viewed as a simple modification of fully connected neural networks (FCNNs) or multilayer perceptrons (MLPs) by introducing balanced multicomponent structures, they achieve a significant reduction in training parameters, a much more efficient training process, and improved accuracy compared to FCNNs or MLPs. Extensive numerical experiments demonstrate the effectiveness of MMNNs in approximating highly oscillatory functions and their ability to automatically adapt to localized features. Our code and implementations are available at GitHub.

Key words. structured decomposition, function compositions, deep neural networks, rectified linear unit, Fourier series

MSC codes. 65Y10, 65Y20, 68W20, 68W25

DOI. 10.1137/24M1675990

1. Introduction. The key use of neural networks is to approximate an input-output relation, i.e., a mapping or a function in mathematical terms. In this work, we continue our study of numerical understanding of neural network approximation of functions from representation to learning dynamics. In our earlier study [45], we demonstrated that a one-hidden-layer (also known as a two-layer or shallow) network is essentially a “low-pass filter” when approximating a function in practice. Due to the strong correlation among the family of activation functions (parameterized by the weight and bias), such as ReLU (rectified linear unit), the Gram matrix—the element of which is the pairwise correlation (inner product) of the activation functions—has a fast spectral decay. If initialized randomly, the eigenvectors of the Gram matrix correspond to generalized Fourier modes from low frequency to high frequency ordered corresponding to decreasing eigenvalues. Due to the ill-conditioning of the representation, no matter how wide a one-hidden-layer network is, it can only learn and approximate smooth functions or sample low-frequency modes effectively and stably (with respect to noise or machine round-off errors).

*Submitted to the journal’s Machine Learning Methods for Scientific Computing section July 12, 2024; accepted for publication (in revised form) July 8, 2025; published electronically September 18, 2025.

<https://doi.org/10.1137/24M1675990>

Funding: The first author was partially supported by the start-up fund P0053092 from The Hong Kong Polytechnic University. The second author was partially supported by NSF grants DMS-2309551 and DMS-2012860. The third author was partially supported by NSF grant DMS-2309530. The fourth author was partially supported by NSF grant DMS-2307465.

[†]Corresponding author. Department of Applied Mathematics, Hong Kong Polytechnic University, 11 Yuk Choi Road, Hung Hom, Kowloon, Hong Kong SAR, China (shijun.zhang@polyu.edu.hk).

[‡]Department of Mathematics, Duke University, 120 Science Drive, Durham, NC 27708, USA (hongkai.zhao@duke.edu).

[§]Department of Mathematics and Statistics, Auburn University, 221 Parker Hall, Auburn, AL 36849, USA (yimin.zhong@auburn.edu).

[¶]School of Mathematics, Georgia Institute of Technology, 686 Cherry Street, Atlanta, GA 30332, USA (hmzhou@math.gatech.edu).

In this work, we propose a balanced multicomponent and multilayer neural network (MMNN) structure based on our previous understanding of a one-hidden-layer network. First, we show that a multilayer network with a multicomponent structure, each of which can be approximated well and effectively by a one-hidden-layer network, can overcome the limitation of a shallow network by smooth decomposition and transformation. Compared to a fully connected neural network of a similar structure, our proposed MMNN is much more effective in terms of representation, training, and accuracy in approximating functions, especially for functions containing complex features, e.g., high-frequency modes. The key idea of MMNNs is to view a linear combination of activation functions as randomly parameterized basis functions, called a *component*, as a whole to represent a smooth function. Each layer has multiple components all sharing the common basis functions with different linear combinations. The number of components, called *rank*, is typically much smaller than the layer's width and increases to enhance the flexibility of decomposition when dealing with more complex functions. These components are combined and composed (through layers) in a structured and balanced way in terms of network width, rank, and depth to approximate a complicated function effectively. Another important feature we used in practice is that weights and biases inside each activation function are randomly assigned and fixed during the optimization while the linear combination weights of activation functions in each component are trained. This leads to more efficient training processes motivated by our finding that a one-hidden-layer neural network can be trained effectively to approximate a smooth function well using random basis functions. We also demonstrate interesting learning dynamics based on Adam optimizer [16], which is crucial for the successful and efficient training of MMNNs. An important remark is that a balanced and holistic approach needs to consider representation, optimization, and their interplay in an integrated manner.

The structure of this paper is as follows. Section 2 introduces and details the design of MMNNs. Section 3 provides a comparison between FCNNs and MMNNs. In section 4, we present a mathematical framework for smooth decomposition and transformation based on the MMNN architecture, showing that each component can be effectively approximated by a single-hidden-layer network. Section 5 presents extensive numerical experiments to validate our analysis and demonstrate the capability of MMNNs in approximating complex functions. Additional insights and implementation guidelines are discussed in section 6. Finally, section 7 concludes the paper with final remarks.

2. Multicomponent and multilayer neural network (MMNN). In this section, we present a novel network architecture called the multicomponent and multilayer neural network (MMNN). Let us begin with some notations. Let \mathbb{R} represent the set of real numbers. The indicator (or characteristic) function of a set A , denoted by $\mathbb{1}_A$, is a function that takes the value 1 for elements in A and 0 for elements not in A . Vectors and matrices are denoted by bold lowercase and uppercase letters, respectively. We use slicing notation for a vector $\mathbf{x} = (x_1, \dots, x_d) \in \mathbb{R}^d$, where $\mathbf{x}[n:m]$ denotes a slice of \mathbf{x} from its n th to the m th entries for any $n, m \in \{1, 2, \dots, d\}$ with $n \leq m$, and $\mathbf{x}[n]$ denotes the n th entry of \mathbf{x} . For example, if $\mathbf{x} = (x_1, x_2, x_3) \in \mathbb{R}^3$, then $(5\mathbf{x})[2:3] = (5x_2, 5x_3)$ and $(6\mathbf{x} + 1)[3] = 6x_3 + 1$. A similar notation is used for matrices. For instance, $\mathbf{A}[:, i]$ refers to the i th column of \mathbf{A} , whereas $\mathbf{A}[i, :]$ indicates the i th row of \mathbf{A} . Additionally, $\mathbf{A}[i, n:m]$ corresponds to $(\mathbf{A}[i, :])[n:m]$, extracting the entries from the n th to the m th in the i th row.

Later, we introduce the architecture of MMNNs in section 2.1. Following this, in section 2.2, we outline the learning strategy of MMNN and highlight its advantages over other methods.

2.1. Architecture of MMNNs. Here, we introduce the architecture of our multicomponent and multilayer neural network (MMNN). Each layer of the MMNN is a (shallow) neural network of the form

$$\mathbf{h}(\mathbf{x}) = \mathbf{A}\sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) + \mathbf{c}$$

to approximate a vector-valued function $\mathbf{f} : \mathbb{R}^{d_{\text{in}}} \rightarrow \mathbb{R}^{d_{\text{out}}}$, where $\mathbf{W} \in \mathbb{R}^{n \times d_{\text{in}}}$, $\mathbf{A} \in \mathbb{R}^{d_{\text{out}} \times n}$, and n is the width of this network. Here, $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ represents the activation function that can be applied elementwise to vector inputs. Throughout this paper, the activation function is chosen as ReLU unless otherwise specified. One can also write it in a more compact form,

$$(2.1) \quad \mathbf{h} = \mathbf{A}\sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) + \mathbf{c} = \tilde{\mathbf{A}} \begin{bmatrix} \sigma(\tilde{\mathbf{W}}\tilde{\mathbf{x}}) \\ 1 \end{bmatrix},$$

where

$$\tilde{\mathbf{W}} = [\mathbf{W}, \mathbf{b}], \quad \tilde{\mathbf{A}} = [\mathbf{A}, \mathbf{c}], \quad \tilde{\mathbf{x}} = \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}.$$

We call each element of \mathbf{h} , i.e., $\mathbf{h}[i] = \tilde{\mathbf{A}}[i, :] \cdot [\sigma(\tilde{\mathbf{W}}\tilde{\mathbf{x}})]$ for $i = 1, 2, \dots, d_{\text{out}}$, a component. Here are a few key features of \mathbf{h} .

1. Each component is viewed as a linear combination of basis functions $\sigma(\mathbf{W}[i, :] \cdot \mathbf{x} + \mathbf{b}[i])$, $i = 1, 2, \dots, n$, which is a function in \mathbf{x} , as a whole.
2. Different components of \mathbf{h} share the same set of basis with different coefficients $\tilde{\mathbf{A}}[i, :]$.
3. Only $\tilde{\mathbf{A}}$ is trained while $\tilde{\mathbf{W}}$ is randomly assigned and fixed.
4. The output dimension d_{out} and network width n can be tuned according to the intrinsic dimension and complexity of the problem.

In comparison, each layer in a typical deep FCNN takes the form $\sigma(\tilde{\mathbf{W}}\tilde{\mathbf{x}})$, and each hidden neuron is individually a function of the input \mathbf{x} or each point $\mathbf{x} \in \mathbb{R}^{d_{\text{in}}}$ is mapped to \mathbb{R}^n , where n is the layer width. All weights $\tilde{\mathbf{W}}$ are training parameters. In MMNNs, each layer is composed of multiple components $\tilde{\mathbf{A}}\sigma(\tilde{\mathbf{W}}\tilde{\mathbf{x}})$. Each component is a linear combination of randomly parameterized hidden neurons $\sigma(\tilde{\mathbf{W}}\tilde{\mathbf{x}})$, which can be more effectively and stably trained through $\tilde{\mathbf{A}}$ as a smooth decomposition/transformation. Typically, the number of components d_{out} —the dimension of intermediate feature space—is (much) smaller than the layer width n , the number of random neurons (or basis functions) in the MMNN. On the one hand, the intermediate feature space is compressed. On the other hand, there are diverse random basis whose linear combinations can have enough representation power in the feature space.

An MMNN is a multilayer composition of \mathbf{h}_i , i.e., $\mathbf{h} : \mathbb{R}^{d_{\text{in}}} \mapsto \mathbb{R}^{d_{\text{out}}}$,

$$\mathbf{h} = \mathbf{h}_m \circ \dots \circ \mathbf{h}_2 \circ \mathbf{h}_1,$$

where each $\mathbf{h}_i : \mathbb{R}^{d_{i-1}} \mapsto \mathbb{R}^{d_i}$ is a multicomponent shallow network defined in (2.1) of width n_i , where

$$d_0 = d_{\text{in}}, \quad d_1, \dots, d_{m-1} \ll n_i, \quad d_m = d_{\text{out}}.$$

The width of this MMNN is defined as $\max\{n_i : i = 1, 2, \dots, m-1\}$, the rank as $\max\{d_i : i = 1, 2, \dots, m-1\}$, and the depth as m . MMNNs are designed to reduce the dimensionality of the intermediate feature space, thereby simplifying optimization while largely preserving the network's expressive power. To simplify, we denote a network with width w , rank r , and depth l using the compact notation (w, r, l) . See Figure 1(a) for an illustration of an MMNN of size $(4, 2, 2)$. In contrast, an FCNN ϕ can be expressed in the following composition form

$$\phi = \mathcal{L}_L \circ \sigma \circ \mathcal{L}_{L-1} \circ \dots \circ \sigma \circ \mathcal{L}_1 \circ \sigma \circ \mathcal{L}_0,$$

where \mathcal{L}_i is an affine linear map given by $\mathcal{L}_i(\mathbf{y}) = \mathbf{W}_i \cdot \mathbf{y} + \mathbf{b}_i$. Readers are referred to Figure 1(b) for an illustration and a comparison with the MMNN.

We clarify the structural differences between FCNNs and MMNNs, omitting bias terms for notational simplicity. In MMNNs, each layer is defined by a composition

$$\mathbf{h} : \mathbb{R}^r \rightarrow \mathbb{R}^r, \quad \mathbf{h}(\mathbf{x}) = \mathbf{A}\sigma(\mathbf{W}\mathbf{x}),$$

while in standard FCNNs, layers are typically written as

$$\mathbf{h} : \mathbb{R}^n \rightarrow \mathbb{R}^n, \quad \mathbf{h}(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x}),$$

where $r \ll n$, and r denotes the MMNN rank, an internal dimensionality that helps regulate network complexity. The core principle behind MMNNs is to limit the dimensionality of intermediate space, which helps streamline the optimization process while maintaining the model's expressive power.

One might suggest that the product $\mathbf{A}_j \mathbf{W}_{j+1}$ in MMNNs can be collapsed into a single matrix, making MMNNs a special case of FCNNs. However, this overlooks two critical distinctions.

1. **Asymmetric Parameter Roles:** In MMNNs, \mathbf{W} is randomly initialized and fixed while \mathbf{A} is learnable. This asymmetry is central to our representation and training strategy and cannot be replicated by simply reinterpreting MMNNs as FCNNs with a low rank factorization of the weight matrix \mathbf{W} which needs to be learned fully posing a challenging task for the optimization.
2. **Architectural Interventions:** In practice, modern networks often employ techniques, such as batch normalization, dropout, and residual connections, which fundamentally alter the layer-wise composition. For instance,
 - with batch normalization layers B_i s:

$$\mathbf{h}_m \circ B_{m-1} \circ \mathbf{h}_{m-1} \circ \dots \circ B_1 \circ \mathbf{h}_1;$$

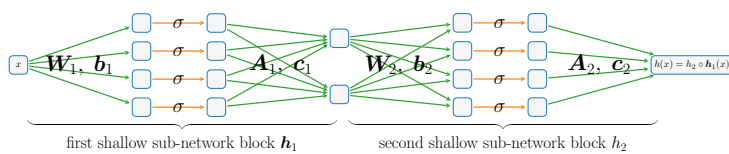
- with dropout layers D_i s:

$$\mathbf{h}_m \circ D_{m-1} \circ \mathbf{h}_{m-1} \circ \dots \circ D_1 \circ \mathbf{h}_1;$$

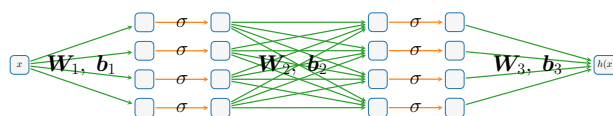
- with residual connections:

$$\mathbf{h}_m \circ (\mathbf{I} + \mathbf{h}_{m-1}) \circ \dots \circ (\mathbf{I} + \mathbf{h}_2) \circ \mathbf{h}_1.$$

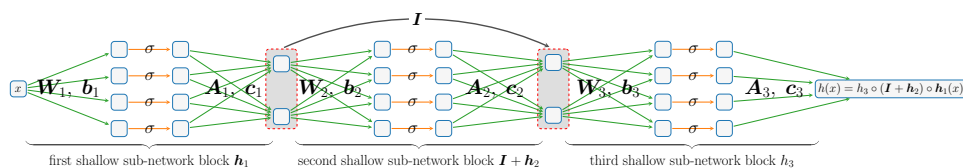
In these scenarios, the clean compositional structure required to merge adjacent matrices (e.g., \mathbf{A}_j and \mathbf{W}_{j+1}) is disrupted. Thus, MMNNs remain distinct in both form and training philosophy.



(a) MMNN of size $(4, 2, 2)$, i.e., width 4, rank 2, and depth 2.



(b) FCNN of size $(4, -, 2)$, i.e., width 4 and depth 2.



(c) ResMMNN of size $(4, 2, 3)$, i.e., width 4, rank 2, and depth 3.

FIG. 1. Illustrations of σ -activated MMNN, FCNN, and ResMMNN.

For very deep MMNNs, one can borrow ideas from ResNets [9] to address the gradient vanishing issue, making training more efficient. Incorporating this idea, we propose a new architecture given by a multilayer composition of $\mathbf{I} + \mathbf{h}_i$, i.e., $\mathbf{h} : \mathbb{R}^{d_{in}} \mapsto \mathbb{R}^{d_{out}}$:

$$\mathbf{h} = \mathbf{h}_m \circ (\mathbf{I} + \mathbf{h}_{m-1}) \circ \cdots \circ (\mathbf{I} + \mathbf{h}_3) \circ (\mathbf{I} + \mathbf{h}_2) \circ \mathbf{h}_1,$$

where each $\mathbf{h}_i : \mathbb{R}^{d_{i-1}} \mapsto \mathbb{R}^{d_i}$ is a multicomponent shallow network defined in (2.1) with width n_i ,

$$d_0 = d_{in}, \quad d_1 = \cdots = d_{m-1} = r \ll n_i, \quad d_m = d_{out},$$

and \mathbf{I} is the identity map. We call this architecture ResMMNN. See Figure 1(c) for an illustration of a ResMMNN of size $(4, 2, 3)$.

The above definition of ResMMNNs requires $d_1 = \cdots = d_{m-1} = r$. If this condition does not hold, we can alternatively define ResMMNNs via

$$\mathbf{h} = (\mathbf{I} \oplus \mathbf{h}_m) \circ (\mathbf{I} \oplus \mathbf{h}_{m-1}) \circ \cdots \circ (\mathbf{I} \oplus \mathbf{h}_3) \circ (\mathbf{I} \oplus \mathbf{h}_2) \circ (\mathbf{I} \oplus \mathbf{h}_1),$$

where \oplus is an operation defined as follows. For any functions $\mathbf{f} : \mathbb{R}^d \mapsto \mathbb{R}^{d_f}$ and $\mathbf{g} : \mathbb{R}^d \mapsto \mathbb{R}^{d_g}$, the \oplus operation is given by

$$\mathbf{f} \oplus \mathbf{g} := (\tilde{\mathbf{f}} + \tilde{\mathbf{g}})[1 : d_g],$$

where

$$\tilde{\mathbf{f}} = \begin{bmatrix} \mathbf{f} \\ \mathbf{0} \end{bmatrix} \in \mathbb{R}^{\max\{d_f, d_g\}} \quad \text{and} \quad \tilde{\mathbf{g}} = \begin{bmatrix} \mathbf{g} \\ \mathbf{0} \end{bmatrix} \in \mathbb{R}^{\max\{d_f, d_g\}}.$$

2.2. Learning strategy of MMNNs. Our learning strategy is motivated by the following basic principle: a function can be decomposed in a multicomponent and multilayer structure, each component of which can be approximated and trained effectively using a one-hidden-layer network, which is a linear combination of random basis functions (e.g., of the form $\sigma(\mathbf{W}_i \cdot \mathbf{x} + \mathbf{b}_i)$, see section 4). Therefore, optimizing the linear combination weights of the random basis functions, namely \mathbf{A}_i s and \mathbf{c}_i s, is both computationally efficient and sufficiently expressive. On the other hand, optimizing the weights (orientations of the basis functions) \mathbf{W}_i s and biases \mathbf{b}_i s to make the basis functions more adaptive to fine-tune features of the target function, which would require capturing high-frequency information by a single layer network, leads to not only significantly more parameters to optimize but also difficulties in training, as shown in [45]. Specifically, for each layer of an MMNN, we fix the activation function parameters (\mathbf{W}_i s and \mathbf{b}_i s) as per PyTorch's default setting during the training process. This entails initializing both weights and biases uniformly from the distribution $\mathcal{U}(-\sqrt{k}, \sqrt{k})$, where $k = \frac{1}{\text{in_features}}$.¹ The whole training process optimizes all \mathbf{A}_i s and \mathbf{c}_i s simultaneously using the Adam optimizer [16]. Note that it is important to have a uniform sampling of orientations \mathbf{W}_i and biases \mathbf{b}_i for the random basis functions to be able to approximate an arbitrary smooth function well. Unless stated otherwise, parameter initialization adheres to the default settings provided by PyTorch in our experiments.

To demonstrate the advantages of our training approach (labeled S1), we conduct a comparison with the typical strategy in deep neural networks, denoted as Strategy S2, which uses the default PyTorch initialization and optimizes all parameters during training. In our tests, we select an oscillatory target function $f(x) = \cos(36\pi x^2) - 0.6\cos(12\pi x^2)$ and use fairly compact networks. The tests are performed on a total of 1,000,100 uniform samples in $[-1, 1]$ with a minibatch size of 100 and a learning rate for epoch k set at $0.001 \times 0.9^{\lfloor k/400 \rfloor}$ for $k = 1, 2, \dots, 20000$, where $\lfloor \cdot \rfloor$ denotes the floor operation. The Adam optimizer [16] is applied throughout the training process.

As illustrated in Table 1 and Figure 2, our learning strategy S1 is significantly more effective than strategy S2 with comparable accuracy. There are two main advantages of S1. First, S1 requires training only about half the number of parameters compared to S2, which results in time savings. Second, S1 converges more quickly and performs significantly better when the training is not sufficient. We would like to note that in certain specific cases, S2 may outperform S1, particularly when the network size is relatively small, and S2 is well-trained. This is expected since S2 trains all parameters, whereas S1 only trains a subset. Based on our experience, S1 is more effective in practice, particularly for sufficiently large networks.

TABLE 1
Comparison of test errors averaged over the last 100 epochs.

network	(width, rank, depth)	#parameters (trained / all)	test error (MSE)	test error (MAX)	training time
MMNN1 (S1)	(400, 20, 6)	40501 / 83301	2.01×10^{-5}	4.36×10^{-2}	23.9s / 1000 epochs
MMNN1 (S2)	(400, 20, 6)	83301 / 83301	4.26×10^{-5}	4.71×10^{-2}	30.2s / 1000 epochs
MMNN2 (S1)	(590, 28, 6)	83331 / 170061	1.39×10^{-5}	2.80×10^{-2}	25.2s / 1000 epochs

¹It is noteworthy that this initialization approach is similar to the widely used Xavier initialization [5], which draws weights from the distribution $\mathcal{U}(-\sqrt{k}, \sqrt{k})$ with $k = \frac{\sqrt{6}}{\text{in_features} + \text{out_features}}$ and sets the bias to $\mathbf{0}$.

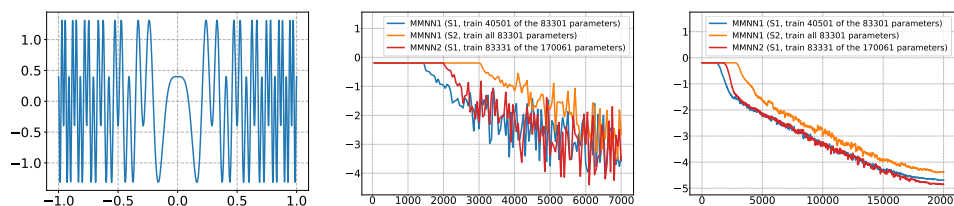


FIG. 2. Left: target function $f(x) = \cos(36\pi x^2) - 0.6 \cos(12\pi x^2)$. Middle: base-10 logarithm of test errors vs. epoch. Right: base-10 logarithm of “test-error-aver” vs. epoch, where “test-error-aver” for epoch k is calculated by averaging the errors in epochs $\max\{1, k - 100\}$ to $\min\{k + 100, \#\text{epochs}\}$.

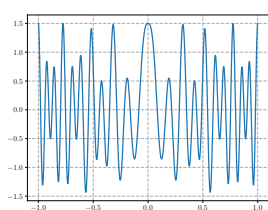


FIG. 3. Plot of f_1 .

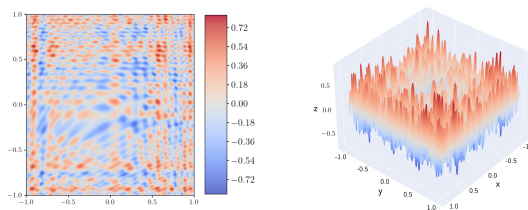


FIG. 4. Plots of f_2 .

3. MMNNs versus FCNNs. In section 2, we outlined the distinctions between MMNNs and FCNNs regarding their representation and learning approaches. Here, we evaluate their numerical performance for approximating oscillatory functions in section 3.1 and solving partial differential equations (PDEs) in section 3.2. To ensure a fair comparison, we use networks with a similar number of parameters and ensure that all networks have sufficient parameters to learn the target function. Typically, when training an FCNN, all parameters are optimized. For a thorough comparison, we will employ two learning strategies for MMNNs as detailed in section 2.2: S1 and S2. S1 involves training approximately half the number of parameters of the MMNN, while S2 involves training all parameters.

3.1. Oscillatory function approximation. We consider a one-dimensional function $f_1(x) = \cos(20\pi|x|^{1.4}) + 0.5 \cos(12\pi|x|^{1.6})$ and a two-dimensional function

$$f_2(x_1, x_2) = \sum_{i=1}^2 \sum_{j=1}^2 a_{ij} \sin(sb_i x_i + sc_{i,j} x_i x_j) \cos(sb_j x_j + sd_{i,j} x_i^2),$$

where $s = 2$ and

$$(a_{i,j}) = \begin{bmatrix} 0.3 & 0.2 \\ 0.2 & 0.3 \end{bmatrix}, \quad (b_i) = \begin{bmatrix} 2\pi \\ 4\pi \end{bmatrix}, \quad (c_{i,j}) = \begin{bmatrix} 2\pi & 4\pi \\ 8\pi & 4\pi \end{bmatrix}, \quad (d_{i,j}) = \begin{bmatrix} 4\pi & 6\pi \\ 8\pi & 6\pi \end{bmatrix}.$$

Refer to Figures 3 and 4 for illustrations of f_1 and f_2 , respectively.

Large network sizes (see Table 2) are selected to ensure that all networks possess sufficient parameters to learn the target functions.² For training the one-dimensional function, we sample a total of 1,000 data points on a uniform grid within $[-1, 1]$ using a minibatch size of 100 and a learning rate of $0.001 \times 0.9^{\lfloor k/400 \rfloor}$ for epochs $k = 1, 2, \dots, 20000$. For training the two-dimensional function, we sample a total of

²FCNNs perform poorly if the network size is small. For a fair comparison, we choose relatively large network sizes for FCNNs and MMNNs, where both perform reasonably well.

TABLE 2
Comparison of test errors averaged over the last 100 epochs.

target function	network	(width, rank, depth)	#parameters (trained / all)	test error (MSE)	test error (MAX)	training time
f_1	MMNN1 (S1)	(388, 18, 6)	35399 / 73035	2.49×10^{-6}	9.93×10^{-3}	23.3s / 1000 epochs
f_1	FCNN1-1	(83, -, 6)	35110 / 35110	2.43×10^{-4}	1.87×10^{-1}	19.5s / 1000 epochs
f_1	MMNN1 (S2)	(388, 18, 6)	73035 / 73035	2.05×10^{-6}	1.88×10^{-2}	27.4s / 1000 epochs
f_1	FCNN1-2	(120, -, 6)	72961 / 72961	1.73×10^{-4}	1.14×10^{-1}	22.3s / 1000 epochs
f_2	MMNN2 (S1)	(789, 36, 12)	313630 / 637120	4.61×10^{-6}	1.55×10^{-2}	30.3s / 10 epochs
f_2	FCNN2-1	(168, -, 12)	312985 / 312985	2.42×10^{-4}	2.75×10^{-1}	26.7s / 10 epochs
f_2	MMNN2 (S2)	(789, 36, 12)	637120 / 637120	6.17×10^{-6}	6.05×10^{-2}	35.8s / 10 epochs
f_2	FCNN2-2	(240, -, 12)	637201 / 637201	3.28×10^{-5}	1.39×10^{-1}	29.3s / 10 epochs

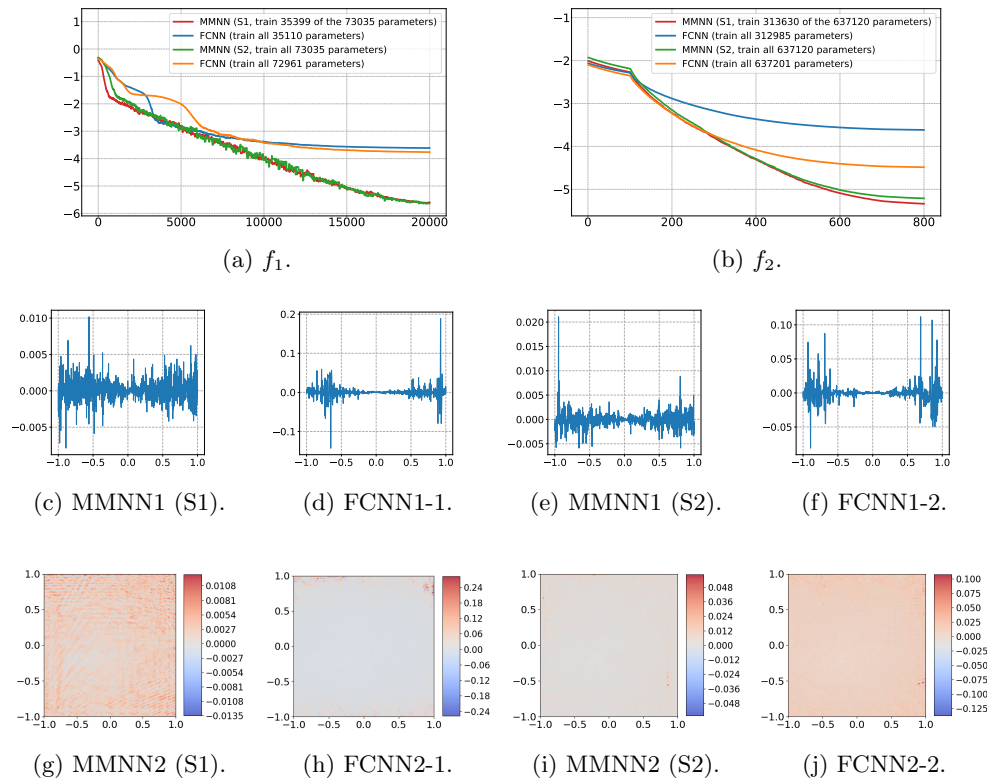


FIG. 5. First row: base-10 logarithm of “test-error-aver” vs. epoch, where “test-error-aver” for epoch k is calculated by averaging the errors in epochs $\max\{1, k - 100\}$ to $\min\{k + 100, \#epochs\}$. All errors shown on the y-axis are in base-10 logarithmic scale. Second row: differences between learned networks and f_1 . Third row: differences between learned networks and f_2 .

600^2 data points on a uniform grid within $[-1, 1]^2$ using a minibatch size of 1000 and a learning rate of $0.001 \times 0.9^{\lfloor k/16 \rfloor}$ for epochs $k = 1, 2, \dots, 800$. The Adam optimizer is employed for both functions.

As illustrated in Table 2 and Figure 5, MMNNs outperform FCNNs when both have the same depth and a comparable number of parameters, particularly for relatively oscillatory target functions. Moreover, as indicated in Table 2, the training time for MMNN (S1) is similar to that of FCNN, while MMNN (S2) takes a bit more time. We remark that the primary advantage of MMNNs lies in capturing

high-frequency components. As we can see from Figure 5, the differences between network approximations and the corresponding target functions show that FCNNs approximate high-frequency parts of the target functions poorly. In contrast, the approximation errors for MMNNs, especially with the S1 learning strategy, are more evenly distributed across the entire domain, indicating their effectiveness in capturing high-frequency components. The Adam optimizer [16] is applied throughout the training process.

3.2. Solving partial differential equations. Next, we compare the performance of MMNNs and FCNNs for solving PDEs. We consider a classical example: the two-dimensional Poisson equation with zero Dirichlet boundary conditions:

$$-\Delta u(x, y) = f(x, y), \quad (x, y) \in (-1, 1)^2, \quad u|_{\partial\Omega} = 0,$$

where the source term is given by

$$f(x, y) = -113\pi^2 \sin(7\pi x) \sin(8\pi y) - 117\pi^2 \sin(6\pi x) \sin(9\pi y).$$

It is easy to verify that the exact solution is

$$u(x, y) = \sin(7\pi x) \sin(8\pi y) + \sin(6\pi x) \sin(9\pi y).$$

We approximate the solution $u(x, y)$ by a neural network $u_{\theta}(x, y)$, where θ denotes the network parameters, and solve the Poisson equation using the physics-informed neural network (PINN) [27] formulation. The PDE residual is defined as $\mathcal{R}(x, y) = -\Delta u_{\theta}(x, y) - f(x, y)$, where $\Delta u_{\theta} = \frac{\partial^2 u_{\theta}}{\partial x^2} + \frac{\partial^2 u_{\theta}}{\partial y^2}$, and the neural network is trained to minimize the loss function $\mathcal{L}_{\text{total}} = \lambda_{\text{PDE}} \mathcal{L}_{\text{PDE}} + \lambda_{\text{BC}} \mathcal{L}_{\text{BC}}$, combining the PDE loss $\mathcal{L}_{\text{PDE}} = \frac{1}{N_r} \sum_{i=1}^{N_r} \mathcal{R}(x_i, y_i)^2$ for collocation points $(x_i, y_i) \in (-1, 1)^2$ and the boundary loss $\mathcal{L}_{\text{BC}} = \frac{1}{N_b} \sum_{j=1}^{N_b} u_{\theta}(x_j, y_j)^2$ for boundary points $(x_j, y_j) \in \partial\Omega$.

To compare FCNNs and MMNNs in terms of representation accuracy, we avoid soft boundary condition enforcement by including the penalty term $\lambda_{\text{BC}} \mathcal{L}_{\text{BC}}$ in the loss, which requires careful tuning of λ_{BC} . We adopt a hard-constraint formulation that guarantees the boundary condition is satisfied exactly. Specifically, we define the network output as

$$u_{\theta}(x, y) = h_{\theta}(x, y) \cdot \cos\left(\frac{\pi x}{2}\right) \cos\left(\frac{\pi y}{2}\right),$$

where $h_{\theta}(x, y)$ is modeled by either an FCNN or an MMNN. This construction ensures that $u_{\theta}(x, y) = 0$ on the boundary $\partial\Omega$ for all values of θ .

We select an MMNN of size (301, 16, 6), another MMNN of size (503, 20, 6), and an FCNN of size (100, -, 6), denoted as MMNN1, MMNN2, and FCNN, respectively, for simplicity. Since ReLU is not differentiable, we use the sine function as the activation function. We sample 100^2 data points for $f(x, y)$ on a uniform grid in $(-1, 1)^2$ using a minibatch size of 2000 and setting $\lambda_{\text{PDE}} = 0.001$. We adopt the Adam optimizer, with the learning rate set to $\lfloor k/200 \rfloor / 800$ for $k < 16000$ and to $0.001 \times 0.9^{\lfloor (k-16000)/1600 \rfloor}$ for $k \geq 16000$, where $k = 1, 2, \dots, 160000$ denotes the training epoch. We note that for $k < 16000$, we use an increasing learning rate to facilitate warm-up (see, e.g., [1, 15]), which enhances training performance.

Our experiments reveal that initial parameters significantly impact training, particularly for FCNNs. To ensure experimental reliability, we repeated the experiments with 16 different seeds. As demonstrated in Figures 6, 7, and Table 3, both MMNNs surpass the FCNN in solving PDEs with PINNs, even with comparable depth and

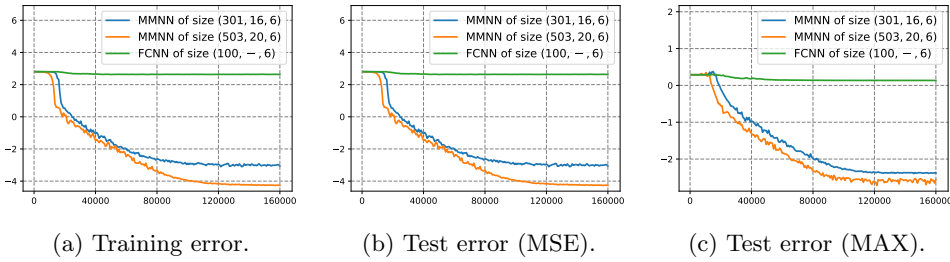


FIG. 6. Average errors across 16 seeds versus epoch. The training error corresponds to the PDE loss, while the test error (MSE or MAX) quantifies the difference between the learned network and the true solution. All errors shown on the y-axis are in base-10 logarithmic scale.

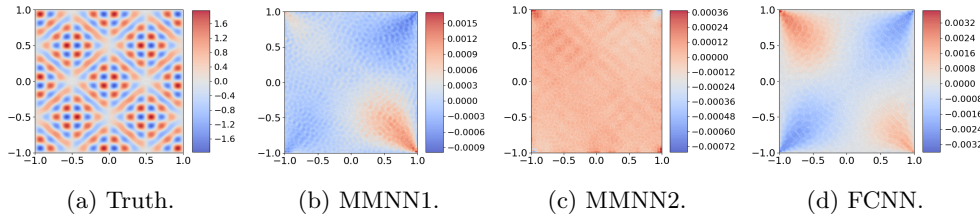


FIG. 7. Comparison of three networks: (a) true solution; (b, c, d) differences between the true solution and predictions from learned networks. For each network, we select the best trained model from 16 seeds.

TABLE 3
Comparison of test errors.

#parameters (trained / all)	24462 / 50950		50904 / 105228		50901 / 50901	
	MMNN1 of size (301, 16, 6)		MMNN2 of size (503, 20, 6)		FCNN of size (100, -, 6)	
seed for randomness	MSE	MAX	MSE	MAX	MSE	MAX
0	2.60×10^{-6}	6.22×10^{-3}	1.77×10^{-8}	3.38×10^{-3}	4.95×10^{-1}	1.94×10^0
1	2.18×10^{-7}	2.08×10^{-3}	9.32×10^{-9}	6.76×10^{-4}	7.95×10^{-7}	3.35×10^{-3}
2	1.18×10^{-6}	4.18×10^{-3}	2.38×10^{-8}	1.55×10^{-3}	5.01×10^{-1}	1.94×10^0
3	3.19×10^{-6}	5.57×10^{-3}	6.68×10^{-9}	1.19×10^{-3}	5.03×10^{-1}	1.94×10^0
4	5.34×10^{-7}	3.12×10^{-3}	2.35×10^{-8}	2.89×10^{-3}	4.99×10^{-1}	1.94×10^0
5	1.76×10^{-6}	5.15×10^{-3}	1.78×10^{-8}	2.51×10^{-3}	4.96×10^{-1}	1.94×10^0
6	4.92×10^{-7}	2.61×10^{-3}	9.68×10^{-9}	4.65×10^{-3}	5.04×10^{-1}	1.94×10^0
7	1.97×10^{-6}	5.34×10^{-3}	3.87×10^{-8}	3.37×10^{-3}	5.01×10^{-1}	1.94×10^0
8	4.98×10^{-7}	2.44×10^{-3}	4.04×10^{-8}	9.83×10^{-4}	8.20×10^{-7}	3.22×10^{-3}
9	2.57×10^{-6}	4.10×10^{-3}	1.69×10^{-9}	6.66×10^{-4}	5.00×10^{-1}	1.94×10^0
10	2.47×10^{-6}	6.15×10^{-3}	2.72×10^{-8}	3.88×10^{-3}	5.02×10^{-1}	1.94×10^0
11	1.03×10^{-6}	3.86×10^{-3}	2.02×10^{-8}	1.31×10^{-3}	5.03×10^{-1}	1.94×10^0
12	1.61×10^{-6}	5.13×10^{-3}	4.38×10^{-9}	1.16×10^{-3}	1.60×10^{-5}	1.12×10^{-2}
13	9.35×10^{-8}	1.52×10^{-3}	3.85×10^{-8}	1.07×10^{-2}	2.86×10^{-2}	4.39×10^{-1}
14	7.42×10^{-7}	3.41×10^{-3}	1.51×10^{-8}	1.77×10^{-3}	5.01×10^{-1}	1.95×10^0
15	6.58×10^{-6}	6.25×10^{-3}	2.74×10^{-8}	1.23×10^{-3}	6.85×10^{-6}	7.97×10^{-3}

total (or training) parameters. We note that using the same seed may yield different outcomes across various code environments. Our tests consistently show that MMNNs always succeed, whereas FCNNs fail with high probability (increasing FCNN width may improve the likelihood of success).

4. Multicomponent and multilayer decomposition. It has been shown in [45] that a one-hidden-layer neural network acts as a low-pass filter and cannot effectively represent or learn high-frequency features. Using mathematical construction, we demonstrate that MMNNs, which are composed of one-hidden-layer neural networks, can overcome this difficulty by decomposing complexity through their components and/or depth. We emphasize that the decomposition is highly nonunique. Our construction is “man-made” which can be different from the one by computer through an optimization (learning) process. Our discussion begins with one-dimensional construction in section 4.1 and later extends to higher dimensions in section 4.2.

4.1. One-dimensional construction. We begin with a two-component decomposition in the one-dimensional as both an illustration and an example in section 4.1.1. Later in section 4.1.2, we introduce the general multicomponent decomposition. Finally, in section 4.1.3, we use concrete examples for demonstration.

4.1.1. Two-component decomposition. We demonstrate a simple “divide and conquer” strategy for an example function $f(x) = \cos(2n\pi x)$, a high frequency Fourier mode when n is large. Define

$$f_2 : (u, v) \in [-1, 1]^2 \mapsto \cos(n\pi(u + 1)) + \cos(n\pi(v - 1)) \in \mathbb{R},$$

and $\mathbf{f}_1 = \begin{bmatrix} f_{1,1} \\ f_{1,2} \end{bmatrix} : [-1, 1] \mapsto [-1, 1]^2$, where the components $f_{1,1}$ and $f_{1,2}$ are given by

$$(4.1) \quad f_{1,1}(x) = \text{ReLU}(2x) - 1 = \begin{cases} -1 & \text{for } x \in [-1, 0), \\ 2x - 1 & \text{for } x \in [0, 1], \end{cases}$$

and

$$(4.2) \quad f_{1,2}(x) = -\text{ReLU}(-2x) + 1 = \begin{cases} 2x + 1 & \text{for } x \in [-1, 0), \\ 1 & \text{for } x \in [0, 1]. \end{cases}$$

Then, for any $x \in [-1, 1]$ we have

$$\begin{aligned} f(x) &= \cos(n\pi \cdot \text{ReLU}(2x)) + \cos(-n\pi \cdot \text{ReLU}(-2x)) \\ &= \cos(n\pi(f_{1,1}(x) + 1)) + \cos(n\pi(f_{1,2}(x) - 1)) = f_2 \circ \mathbf{f}_1(x). \end{aligned}$$

Through this decomposition and piecewise linear transformation, which can be approximated easily by a single layer of ReLU network, one only needs to approximate a function that is smoother than the original f : \mathbf{f}_1 is simplified, while f_2 is reduced to half of the frequency of the original target function f .

We observe that this decomposition approach is universally applicable for any function $f : [-1, 1] \mapsto \mathbb{R}$. Specifically, the decomposition is defined as

$$f_2 : (u, v) \in [-1, 1]^2 \mapsto f\left(\frac{u+1}{2}\right) + f\left(\frac{v-1}{2}\right) - f(0) \in \mathbb{R},$$

and $\mathbf{f}_1 = \begin{bmatrix} f_{1,1} \\ f_{1,2} \end{bmatrix} : [-1, 1] \mapsto [-1, 1]^2$, where $f_{1,1}$ and $f_{1,2}$ are given in (4.1) and (4.2). Hence, for any $x \in [-1, 1]$, we achieve the following reconstruction of $f(x)$:

$$\begin{aligned} f(x) &= f\left(\frac{\text{ReLU}(2x)}{2}\right) + f\left(\frac{-\text{ReLU}(-2x)}{2}\right) - f(0) \\ &= f\left(\frac{f_{1,1}(x) + 1}{2}\right) + f\left(\frac{f_{1,2}(x) - 1}{2}\right) - f(0) = f_2 \circ \mathbf{f}_1(x), \end{aligned}$$

demonstrating a structured decomposition that allows the function to be expressed through the composition of a smoother function with a piecewise (component-wise) transformation and rescaling.

4.1.2. General multicomponent decomposition. Now, we extend to a general multicomponent adaptive decomposition, a “divide and conquer” strategy, that can distribute the complexity of a target function evenly to multiple components.

Given a sequence $x_0 < x_1 < \dots < x_n$, where the target function is defined on the interval $[x_0, x_n]$, we will demonstrate how our new architecture allows us to partition the complexities of the function f into smaller intervals $[x_{i-1}, x_i]$. By rescaling each subinterval, one only needs to deal with a much smoother function in each interval. This approach enables us to effectively approximate the target function over the entire interval $[x_0, x_n]$.

Let $\mathcal{L}_i : [a_i, b_i] \rightarrow [x_{i-1}, x_i]$ be the linear map with

$$(4.3) \quad \mathcal{L}_i(a_i) = x_{i-1} \quad \text{and} \quad \mathcal{L}_i(b_i) = x_i.$$

Define

$$(4.4) \quad f_i = f \circ \mathcal{L}_i : [a_i, b_i] \rightarrow \mathbb{R}.$$

To decompose the target function into smoother pieces, we define a piecewise linear transformation ψ_i using a linear combination of two ReLU functions (or a simple single layer network),

$$(4.5) \quad \psi_i(x) = s_i \cdot \text{ReLU}(x - x_{i-1}) - s_i \cdot \text{ReLU}(x - x_i) + a_i.$$

Here, $s_i = \frac{b_i - a_i}{x_i - x_{i-1}}$ is the “slope” of \mathcal{L}_i^{-1} , which is a local rescaling. For example, f_i becomes a smoother function than f after stretching $[x_{i-1}, x_i]$ to a larger domain $[a_i, b_i]$. See an illustration of $\psi_i(x)$ in Figure 8.

THEOREM 4.1. *Given $x_0 < x_1 < \dots < x_n$, suppose \mathcal{L}_i and ψ_i are given in (4.3) and (4.5), respectively. Then, the target function $f : [x_0, x_n] \rightarrow \mathbb{R}$ has the following (smoother) decomposition (f_i) with a piecewise linear transformation (ψ_i),*

$$f(x) = \sum_{i=1}^n f_i \circ \psi_i(x) - \underbrace{\sum_{i=1}^{n-1} f(x_i)}_{\text{constant}} \quad \text{for any } x \in [x_0, x_n],$$

where f_i is given in (4.4).

Proof. By definition of ψ_i in (4.5), it is easy to check

$$\psi_i(x) = \begin{cases} b_i & \text{if } x > x_i, \\ \mathcal{L}_i^{-1}(x) & \text{if } x \in [x_{i-1}, x_i], \\ a_i & \text{if } x < x_{i-1}, \end{cases} \implies \psi_i(x) = \begin{cases} b_i & \text{if } i \leq j-1, \\ \mathcal{L}_j^{-1}(x) & \text{if } i = j, \\ a_i & \text{if } i \geq j+1, \end{cases}$$

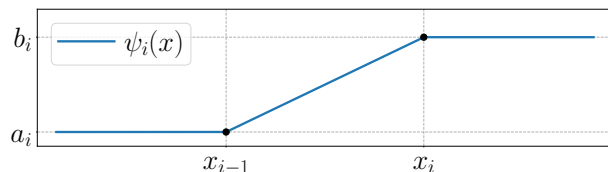


FIG. 8. An illustration of $\psi_i(x)$.

for a fixed $j \in \{1, 2, \dots, n\}$ and any $x \in [x_{j-1}, x_j]$. It follows that

$$\begin{aligned} \sum_{i=1}^n f_i \circ \psi_i(x) &= \sum_{i=1}^n f \circ \mathcal{L}_i \circ \psi_i(x) \\ &= \sum_{i=1}^{j-1} f \circ \mathcal{L}_i \circ \psi_i(x) + f \circ \mathcal{L}_j \circ \psi_j(x) + \sum_{i=j+1}^n f \circ \mathcal{L}_i \circ \psi_i(x) \\ &= \sum_{i=1}^{j-1} f \circ \mathcal{L}_i(b_i) + f \circ \mathcal{L}_j \circ \mathcal{L}_j^{-1}(x) + \sum_{i=j+1}^n f \circ \mathcal{L}_i(a_i) \\ &= \sum_{i=1}^{j-1} f(x_i) + f(x) + \sum_{i=j+1}^n f(x_{i-1}) = f(x) + \underbrace{\sum_{i=1}^{n-1} f(x_i)}_{\text{constant}}. \end{aligned}$$

It follows that

$$f(x) = \sum_{i=1}^n f_i \circ \psi_i(x) - \underbrace{\sum_{i=1}^{n-1} f(x_i)}_{\text{constant}} \quad \text{for any } x \in [x_{j-1}, x_j].$$

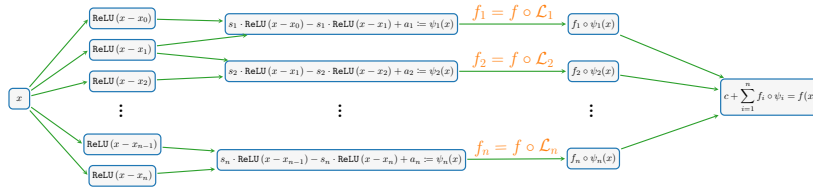
Since j is arbitrary, the above equation holds for all $x = \cup_{j=1}^n [x_{j-1}, x_j] = [x_0, x_n]$. \square

For each smoother f_i , one can use a shallow network component ϕ_i , a linear combination of random basis functions, to approximate f_i well on $[a_i, b_i]$. Then,

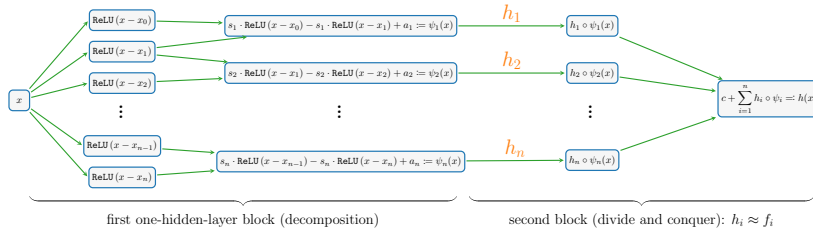
$$f(x) = \sum_{i=1}^n f_i \circ \psi_i(x) - \underbrace{\sum_{i=1}^{n-1} f(x_i)}_{\text{constant}} \approx \sum_{i=1}^n \phi_i \circ \psi_i(x) - \underbrace{\sum_{i=1}^{n-1} f(x_i)}_{\text{constant}} =: h(x),$$

$h(x)$ is a one-hidden-layer neural network approximation of the target function $f(x)$ that can approximate a complex function better than a single layer. See Figure 9 for an illustration. In practice, one can choose repeated decomposition using a multicomponent and multilayer network structure which is the motivation for MMNN. It is well-known that neural networks can approximate smooth functions well. For localized rapid change/oscillation, our construction shows that a small network in terms of the width and the number of components and layers can achieve adaptive decomposition and deal with it rather easily. Hence, MMNN is effective in approximating a function with localized fine features. This is an important advantage in dealing with low-dimensional structures embedded in high dimensions. The most difficult situation is approximating global highly oscillatory functions, especially with diverse frequency modes, for which wider networks with more components and layers are needed to deal with both the complexity and curse of dimensions.

4.1.3. Examples. Here, we use two examples to demonstrate the complexity decomposition strategy presented in the previous section. We start with the Runge function $f(x) = \frac{1}{25x^2+1}$ and modify it to $f(x) = \frac{1}{1000x^2+1}$, which has a localized rapid change near 0. As an example, we use four components $n = 4$, choose points x_0, x_1, x_2, x_3, x_4 at $-1, -0.2, 0, 0.2, 1$, and let $a_i = -1$ and $b_i = 1$ for all i . In practice, each component is approximated by a single-layer network—a linear combination of basis functions and trained by an optimization method, e.g., Adam. Our examples here are just a proof of concept for the decomposition of a target function into smoother components using MMNN structure in the form



(a) Decomposition of target function $f = c + \sum_{i=1}^n f_i \circ \psi_i$: oscillatory f to smooth f_i 's.



(b) Neural network architecture of $h = c + \sum_{i=1}^n h_i \circ \psi_i$ by using $h_i \approx f_i$.

FIG. 9. Visual representations of the decompositions of f and h are provided with $c = \sum_{i=0}^{n-1} f(x_i)$ being a constant and s_i being the slope. Here, the function f is dissected into several simpler functions, labeled as f_i . Each f_i represents a simplified, more manageable segment of f , allowing for the straightforward application of subnetwork h_i to closely approximate f_i even with the use of shallow networks.

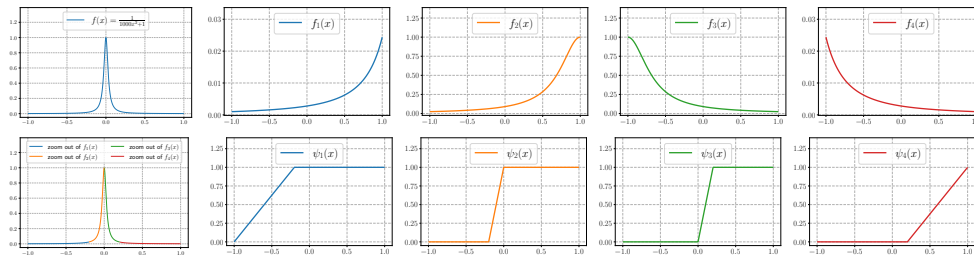


FIG. 10. Illustrations of $f(x) = \frac{1}{1000x^2 + 1}$ and its multicomponent decomposition through f_i and ψ_i $i = 1, 2, 3, 4$, where $f(x) = \sum_{i=1}^4 f_i \circ \psi_i(x) - \sum_{i=1}^3 f(x_i)$.

$$f(x) = \sum_{i=1}^4 f_i \circ \psi_i(x) - \underbrace{\sum_{i=1}^3 f(x_i)}_{\text{constant}}$$

where f_i and ψ_i (piecewise transformation/rescaling) are defined as in (4.4) and (4.5), respectively. These components are illustrated in Figure 10. Each component is relatively smooth, making it easier for approximation and learning through shallow networks. This approach essentially utilizes a divide-and-conquer principle.

The second example is a globally oscillatory function of the form

$$f(x) = \cos^2(6\pi x) + \sin(10\pi x^2).$$

Again, we illustrate using four components $n = 4$, selecting points x_0, x_1, x_2, x_3, x_4 at $-1, -0.7, 0, 0.7, 1$, and setting $a_i = -1$ and $b_i = 1$ for all i . As shown in Figure 11, the

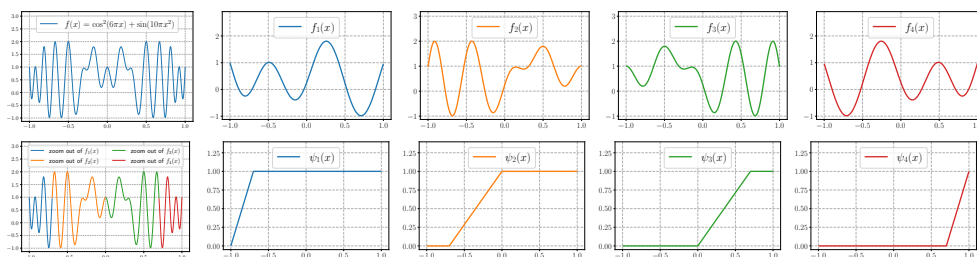


FIG. 11. Illustrations of $f(x) = \cos^2(6\pi x) + \sin(10\pi x^2)$ and its decomposition components f_i and ψ_i such that $f(x) = \sum_{i=1}^4 f_i \circ \psi_i(x) - \sum_{i=1}^3 f(x_i)$.

target function $f(x)$ is decomposed into components that are less oscillatory, again facilitating their approximation and learning through shallow networks.

4.2. High-dimensional cases. Let us now consider the extension to multiple dimensions using the case of two dimensions as an example since the straightforward dimension-by-dimension strategy can be applied to any number of dimensions.

Given $x_0 < x_1 < \dots < x_n$ and $y_0 < y_1 < \dots < y_m$, we divide the domain of the function $f(x, y)$ into small Cartesian rectangles $[x_{i-1}, x_i] \times [y_{j-1}, y_j]$. Let $\mathcal{L}_{1,i} : [a_i, b_i] \rightarrow [x_{i-1}, x_i]$ and $\mathcal{L}_{2,j} : [c_j, d_j] \rightarrow [y_{j-1}, y_j]$ be the linear maps with

$$(4.6) \quad \begin{cases} \mathcal{L}_{1,i}(a_i) = x_{i-1}, \\ \mathcal{L}_{1,i}(b_i) = x_i \end{cases} \quad \text{and} \quad \begin{cases} \mathcal{L}_{2,j}(c_j) = y_{j-1}, \\ \mathcal{L}_{2,j}(d_j) = y_j. \end{cases}$$

For $i = 1, 2, \dots, n$ and $j = 1, 2, \dots, m$, we define

$$(4.7) \quad \begin{cases} f_{i,0}(x, y) := f(\mathcal{L}_{1,i}(x), y), \\ f_{0,j}(x, y) := f(x, \mathcal{L}_{2,j}(y)), \\ f_{i,j}(x, y) := f(\mathcal{L}_{1,i}(x), \mathcal{L}_{2,j}(y)) = f_{0,j}(\mathcal{L}_{1,i}(x), y) = f_{i,0}(x, \mathcal{L}_{2,j}(y)). \end{cases}$$

It is evident that with appropriate transformation and rescaling, $f_{i,0}(x, y)$ is smooth in x when y is held constant, $f_{0,j}(x, y)$ is smooth in y when x is fixed, and $f_{i,j}(x, y)$ is smooth in both x and y . Define

$$(4.8) \quad \psi_i(x) = \begin{cases} b_i & \text{if } x > x_i, \\ \mathcal{L}_{1,i}^{-1}(x) & \text{if } x \in [x_{i-1}, x_i], \\ a_i & \text{if } x < x_{i-1} \end{cases} \quad \text{and} \quad \phi_j(y) = \begin{cases} d_j & \text{if } y > y_j, \\ \mathcal{L}_{2,j}^{-1}(y) & \text{if } y \in [y_{j-1}, y_j], \\ c_j & \text{if } y < y_{j-1}. \end{cases}$$

The theorem below provides a decomposition of f that fits into the MMNN structure.

THEOREM 4.2. *Given $x_0 < x_1 < \dots < x_n$ and $y_0 < y_1 < \dots < y_m$, suppose $\mathcal{L}_{1,i}, \mathcal{L}_{2,j}$, and ψ_i, ϕ_j are given in (4.6) and (4.8), respectively. Then, the function $f : [x_0, x_n] \times [y_0, y_m] \rightarrow \mathbb{R}$ can be expressed as*

$$(4.9) \quad \begin{aligned} f(x, y) = & \sum_{i=1}^n \sum_{j=1}^m f_{i,j}(\psi_i(x), \phi_j(y)) - \sum_{i=1}^n \sum_{j=1}^{m-1} f_{i,0}(\psi_i(x), y_j) \\ & - \sum_{i=1}^{n-1} \sum_{j=1}^m f_{0,j}(x_i, \phi_j(y)) + \sum_{i=1}^{n-1} \sum_{j=1}^{m-1} f(x_i, y_j) \end{aligned}$$

for all $(x, y) \in [x_0, x_n] \times [y_0, y_m]$, where $f_{i,j}$ are given in (4.7).

Proof. Fixing (k, j) , for any $(x, y) \in [x_{k-1}, x_k] \times [y_{\ell-1}, y_\ell]$, we have

$$\psi_i(x) = \begin{cases} b_i & \text{if } i \leq k-1, \\ \mathcal{L}_{1,k}^{-1}(x) & \text{if } i = k, \\ a_i & \text{if } i \geq k+1 \end{cases} \quad \text{and} \quad \phi_j(y) = \begin{cases} d_j & \text{if } j \leq \ell-1, \\ \mathcal{L}_{2,\ell}^{-1}(y) & \text{if } j = \ell, \\ c_j & \text{if } j \geq \ell+1. \end{cases}$$

It follows that

$$\begin{aligned} \sum_{i=1}^n f_{i,0}(\psi_i(x), y) &= \sum_{i=1}^n f(\mathcal{L}_{1,i} \circ \psi_i(x), y) \\ &= \sum_{i=1}^{k-1} f(\mathcal{L}_{1,i} \circ \psi_i(x), y) + f(\mathcal{L}_{1,k} \circ \psi_k(x), y) + \sum_{i=k+1}^n f(\mathcal{L}_{1,i} \circ \psi_i(x), y) \\ &= \sum_{i=1}^{k-1} f(\mathcal{L}_{1,i}(b_i), y) + f(\mathcal{L}_{1,k} \circ \mathcal{L}_{1,k}^{-1}(x), y) + \sum_{i=k+1}^n f(\mathcal{L}_{1,i}(a_i), y) \\ &= \sum_{i=1}^{k-1} f(x_i, y) + f(x, y) + \sum_{i=k+1}^n f(x_{i-1}, y) = f(x, y) + \sum_{i=1}^{n-1} f(x_i, y), \end{aligned}$$

implying

$$f(x, y) = \sum_{i=1}^n f_{i,0}(\psi_i(x), y) - \sum_{i=1}^{n-1} f(x_i, y).$$

For each i , using the one-dimensional decomposition technique described in section 4.1, we find the decompositions for $f_{i,0}(\psi_i(x), y)$ and $f(x_i, y)$. We have

$$\begin{aligned} \sum_{j=1}^m f_{i,j}(\psi_i(x), \phi_j(y)) &= \sum_{j=1}^m f_{i,0}(\psi_i(x), \mathcal{L}_{2,j} \circ \phi_j(y)) \\ &= \sum_{j=1}^{\ell-1} f_{i,0}(\psi_i(x), \mathcal{L}_{2,j} \circ \phi_j(y)) + f_{i,0}(\psi_i(x), \mathcal{L}_{2,\ell} \circ \phi_\ell(y)) \\ &\quad + \sum_{j=\ell+1}^m f_{i,0}(\psi_i(x), \mathcal{L}_{2,j} \circ \phi_j(y)) \\ &= \sum_{j=1}^{\ell-1} f_{i,0}(\psi_i(x), \mathcal{L}_{2,j}(d_j)) + f_{i,0}(\psi_i(x), \mathcal{L}_{2,\ell} \circ \mathcal{L}_{2,\ell}^{-1}(y)) \\ &\quad + \sum_{j=\ell+1}^m f_{i,0}(\psi_i(x), \mathcal{L}_{2,j}(c_j)) \\ &= \sum_{j=1}^{\ell-1} f_{i,0}(\psi_i(x), y_j) + f_{i,0}(\psi_i(x), y) + \sum_{j=\ell+1}^m f_{i,0}(\psi_i(x), y_{j-1}) \\ &= f_{i,0}(\psi_i(x), y) + \sum_{j=1}^{m-1} f_{i,0}(\psi_i(x), y_{j-1}), \end{aligned}$$

implying

$$(4.10) \quad f_{i,0}(\psi_i(x), y) = \sum_{j=1}^m f_{i,j}(\psi_i(x), \phi_j(y)) - \sum_{j=1}^{m-1} f_{i,0}(\psi_i(x), y_j).$$

Moreover,

$$\begin{aligned} \sum_{j=1}^m f_{0,j}(x_i, \phi_j(y)) &= \sum_{j=1}^m f(x_i, \mathcal{L}_{2,j} \circ \phi_j(y)) \\ &= \sum_{j=1}^{\ell-1} f(x_i, \mathcal{L}_{2,j} \circ \phi_j(y)) + f(x_i, \mathcal{L}_{2,\ell} \circ \phi_\ell(y)) + \sum_{j=\ell+1}^m f(x_i, \mathcal{L}_{2,j} \circ \phi_j(y)) \\ &= \sum_{j=1}^{\ell-1} f(x_i, \mathcal{L}_{2,j}(d_j)) + f(x_i, \mathcal{L}_{2,\ell} \circ \mathcal{L}_{2,\ell}^{-1}(y)) + \sum_{j=\ell+1}^m f(x_i, \mathcal{L}_{2,j}(c_j)) \\ &= \sum_{j=1}^{\ell-1} f(x_i, y_j) + f(x_i, y) + \sum_{j=\ell+1}^m f(x_i, y_{j-1}) = f(x_i, y) + \sum_{j=1}^{m-1} f(x_i, y_j), \end{aligned}$$

implying

$$(4.11) \quad f(x_i, y) = \sum_{j=1}^m f_{0,j}(x_i, \phi_j(y)) - \sum_{j=1}^{m-1} f(x_i, y_j).$$

Therefore, for any $(x, y) \in [x_{k-1}, x_k] \times [y_{\ell-1}, y_\ell]$, by (4.10) and (4.11), we have

$$\begin{aligned} f(x, y) &= \sum_{i=1}^n f_{i,0}(\psi_i(x), y) - \sum_{i=1}^{n-1} f(x_i, y) \\ &= \sum_{i=1}^n \sum_{j=1}^m f_{i,j}(\psi_i(x), \phi_j(y)) - \sum_{i=1}^n \sum_{j=1}^{m-1} f_{i,0}(\psi_i(x), y_j) \\ &\quad - \sum_{i=1}^{n-1} \sum_{j=1}^m f_{0,j}(x_i, \phi_j(y)) + \sum_{i=1}^{n-1} \sum_{j=1}^{m-1} f(x_i, y_j). \end{aligned}$$

Since k and ℓ are arbitrary, the above equation holds for all $(x, y) \in \cup_{k=1}^n [x_{k-1}, x_k] \times \cup_{\ell=1}^m [y_{\ell-1}, y_\ell] = [x_0, x_n] \times [y_0, y_m]$. \square

4.3. Related work. Several lines of research are closely related to this work, including approximation theory, low-rank methods, random feature models, and architectures inspired by the Kolmogorov–Arnold representation.

Approximation. Extensive research has examined the approximation capabilities of neural networks, focusing on various architectures to approximate diverse target functions. Early studies concentrated on the universal approximation power of single-hidden-layer networks [4, 11, 12], which demonstrated that sufficiently large neural networks could approximate specific functions with arbitrary precision mathematically without quantifying the error relative to network size. Subsequent research, such as [2, 3, 7, 8, 21, 23, 33, 34, 35, 36, 37, 38, 40, 41, 42, 43, 44, 46], analyzed the approximation error for different networks in terms of size characterized by width, depth, or the number of parameters. Those studies have primarily concentrated on the mathematical theory that supports the existence theory for such neural networks. However, there has been limited focus on determining the parameters within these networks computationally and the numerical errors, particularly those arising from finite precision in computer simulations. This gap motivated our current investigation, which considers practical training processes and numerical errors. Specifically, the balanced structure of MMNN, the choice of training parameters, and the associated learning strategy discussed here are intended to facilitate a smooth decomposition of the function, thereby promoting an efficient training process.

Low-rank methods. Low-rank structures in the weight matrix \mathbf{W} of a fully connected neural network have been investigated by various groups. For example, the methods proposed in [13, 29, 31] focus on accelerating training and reducing memory requirements while maintaining final performance. The concept of low-rank structures is further extended to tensor train decomposition in [24]. The MMNN proposed here differs in two key aspects. First, each layer contains two matrices: \mathbf{A} outside and \mathbf{W} inside the activation functions. Each row of \mathbf{A} represents the weights for a linear combination of a set of random basis functions, forming a component in each layer. The number of rows in \mathbf{A} , which equals the number of components, is selected based on the complexity of the function and is typically much smaller than the number of columns corresponding to the number of basis functions. Each row of (\mathbf{W}, \mathbf{b}) represents a random parameterization of a basis function, with the number of rows in \mathbf{W} corresponding to the number of basis functions, usually much larger than the number of columns in \mathbf{W} , which is the input dimension. Second, in our MMNN, only \mathbf{A} is trained while \mathbf{W} remains fixed with randomly initialized values. Theoretical studies and numerical experiments demonstrated that the architecture of MMNN, combined with the learning strategy, is effective in approximating complex functions.

Random features. Fixing (\mathbf{W}, \mathbf{b}) of each layer and use of random basis functions in the MMNNs is inspired by a previous approach known as random features [18, 25, 26, 30, 39]. In typical random feature methods, only the linear combination parameters at the output layer are trained, which also leads to the issue of ill-conditioning of the representation. In MMNNs, matrix \mathbf{A} and vector \mathbf{c} of each layer are trained. Our MMNN employs a composition architecture and learning mechanism that enhances the approximation capabilities compared to random feature methods while achieving a more effective training process than a standard fully connected network of equivalent size. Extensive experiments demonstrate that our approach can strike a satisfactory balance between approximation accuracy and training cost.

Komogolov-Arnold (KA) representation. The KA representation theorem [17] states that any multivariate continuous function on a hypercube can be expressed as a finite composition of continuous univariate functions and the binary operation of addition. However, this elegant mathematical representation may result in compositions of nonsmooth or even fractal univariate functions in general, a computational challenge one has to address in practice. KA representation has been explored in several studies [14, 19, 22, 35]. A recently proposed network known as the KA network (KAN) utilizes spline functions to approximate the univariate functions in the KA representation. The proposed MMNN is motivated by a multicomponent and multilayer smooth decomposition, or a “divide and conquer” approach, employing distinct network architectures, activation functions, and training strategies.

5. Numerical experiments. We perform extensive experiments to validate our analysis and demonstrate the effectiveness of MMNNs through the multicomponent and multilayer decomposition studied in section 4. In particular, our tests show its ability to (1) adaptively capture localized high-frequency features in section 5.1, (2) approximate highly oscillatory functions in section 5.2, (3) approximate discontinuous functions with porous structures in section 5.3, (4) exhibit some interesting learning dynamics in section 5.4, and (5) solve problems in three and higher dimensions in section 5.5. All our experiments involve target functions that include high-frequency components in various ways and are difficult to handle by shallow networks (no matter how wide), as shown in our previous work [45]. Moreover, our experience with these tests shows that using a fully connected deep neural network would require many

more parameters and is much harder (if possible) to train to get a comparable result. This is mainly due to a balanced and structured network design of MMNN in terms of the (1) network width w , which is the number of hidden neurons or random basis functions in each component, (2) rank r , which is the number of components in each layer, and (3) network depth l , which is the number of layers in the network. The use of a controllable number of collective components (through \mathbf{A}) in each layer instead of a large number of individual neurons and the use of fixed and randomly chosen weights (\mathbf{W}, \mathbf{b}) make the training process more effective.

In all tests, (1) data are sampled enough to resolve fine features in the target function, (2) the Adam optimizer is used in training, (3) the mean squared error (MSE) is the loss function, (4) the default activation function used is ReLU, (5) all computation and training use single precision in PyTorch, (6) all parameters are initialized according to the PyTorch default initialization (section 2.2) unless otherwise specified, (7) \mathbf{W} s and \mathbf{b} s (the parameters inside the activation functions; see section 2.1) are fixed and only \mathbf{A} s and \mathbf{c} s (the parameters outside the activation functions) are trained, and (8) computations are conducted on a NVIDIA RTX 3500 Ada Generation Laptop GPU (power cap 130W), with most experiments concluding within a range from a few dozen to several thousand seconds. All our MMNN setups are specified by three parameters (w, r, l) which depend on the function complexity. Another tuning parameter is the learning rate, which is guided by the following criteria: (1) not too large initially due to stability, and (2) a decreasing rate with iterations such that the learning rate becomes small near the equilibrium to achieve good accuracy while not decreasing too fast (especially during a long training process for more difficult target functions) so that the training is stalled.

5.1. Localized rapid changes. We begin with two one-dimensional examples. The first is $f(x) = \arctan(100x + 20)$, which is smooth but features a rapid transition at zero. As demonstrated in our previous work [45], a shallow network struggles to capture such a simple local fast transition which contains high frequencies, and we show that this function can be approximated easily by a composition of a smooth function on top of a (repeated) spatial decomposition and local rescaling using the MMNN structure in section 2.1. Our test indeed verifies that our new architecture can effectively capture a localized fast transition rather easily using a very small network of size $(16, 4, 3)$, as shown in Figure 12. For this test, a total of 1,000 data points are uniformly sampled in the range $[-1, 1]$, with a minibatch size of 100, a learning rate of 10^{-3} , and the number of epochs set to 2,000. Figure 13 gives the error plot.

Next, we consider a more complicated target function, $f(x) = \mathbb{1}_{\{|x+0.2| < 0.02\}} \cdot \sin(50\pi x)$, which represents a localized fast oscillation. For this example, we will conduct two tests. The first one is to show the flexibility of MMNN to automatically adapt to local features. The network has a small size, as above $(16, 4, 3)$. Each layer has a network width of 16. In other words, each component is a linear combination

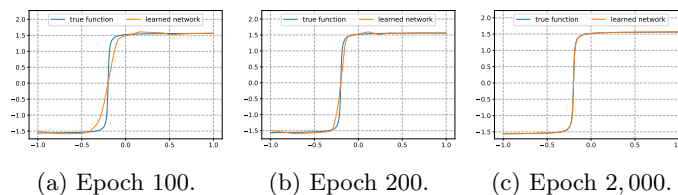


FIG. 12. Illustrations of training process.



FIG. 13. Training and test errors (MSE).

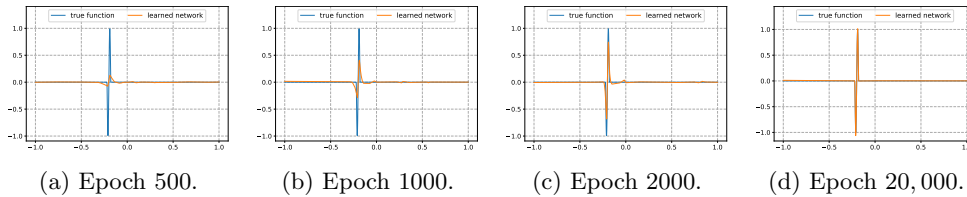


FIG. 14. Illustrations of the training process.

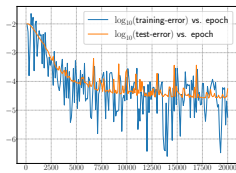


FIG. 15. Training and test errors (in MSE) vs. epoch.

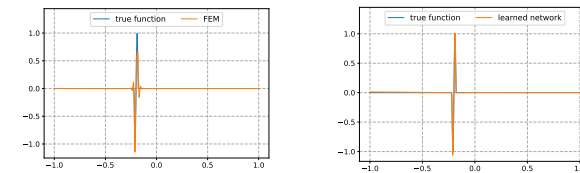


FIG. 16. Left: Least square using equally spaced 153 FEM bases. Right: MMNN with 153 free parameters.

of 16 ReLU functions, which has no way to approximate such a target function well. However, with a multilayer and multicomponent decomposition with parameters appropriately trained by Adam, MMNN can adapt to the behavior of the target function, as shown in Figure 14. Figure 15 gives the error plot. Also, the test shows that this example is more difficult to train. For this test, there are a total of 1,000 uniformly sampled points in $[-1, 1]$ with a minibatch size of 100 and a learning rate of $0.002 \times 0.95^{\lfloor k/1000 \rfloor}$, where $\lfloor \cdot \rfloor$ denotes floor operation and $k = 1, 2, \dots, 20000$ is the epoch number. It should be noted that in this test, we initialize the biases \mathbf{b} s to $\mathbf{0}$ and use the PyTorch default initialization method for the weights \mathbf{W} . This approach, inspired by Xavier initialization, is chosen because the target function is locally oscillatory, and the MMNN size is quite small, necessitating a setup adaptive to the target function to facilitate the training. For other experiments, both the biases and weights use the PyTorch default initialization. We then compare with least square approximation using uniform finite element method (FEM) basis with the same degrees of freedom. As shown in Figure 16, MMNN renders a better approximation due to automatic adaptation through the training process. We remark that when training an MMNN with an extreme compact size with respect to the target function complexity, due to the lack of flexibility/redundancy, the training may become more subtle and need more careful calibration, such as initialization, learning rate, min-batch size, and others. However, introducing slight redundancy into an MMNN, such as by increasing its size marginally, enhances its flexibility and makes training more tractable. On the other hand, when the network becomes too large, then training a large number of parameters and over-redundancy will lead to potential difficulties for optimization. This also shows that there is a trade-off between representation and optimization that needs to be balanced in practice.

Finally, we show an example in the two-dimensional case shown in Figure 17 and defined in polar coordinates by

$$f(r, \theta) = \begin{cases} 0 & \text{if } 0.5 + 25\rho - 25r \leq 0, \\ 1 & \text{if } 0.5 + 25\rho - 25r \geq 1, \\ 0.5 + 25\rho - 25r & \text{otherwise,} \end{cases} \quad \text{where } \rho = 0.1 + 0.02 \cos(8\pi\theta).$$

Again, a rather compact MMNN of size (100, 10, 6) can produce a good approximation. Figure 18 shows the log plot of training and testing errors in MSE. For this test, there are a total of 400^2 uniformly sampled points in $[-1, 1]^2$ with minibatch size of 1000 and a learning rate of $10^{-3} \times 0.9^{\lfloor k/25 \rfloor}$, where $k = 1, 2, \dots, 1000$ is the epoch number. In Figure 19, we compare the result with piecewise linear interpolation and least square approximation using FEM basis on a uniform grid with the same number of degrees of freedom. As observed before, MMNN renders the best result due to its adaptivity through training. When adaptive finite element is applicable, it is hard to beat. However, adaptive FEMs are more humanly involved, while MMNNs based on training are more automatic. The key message here is to demonstrate that MMNN has adaptive features through training/optimization, which will be useful when an adaptive FEM becomes difficult or impossible in applications.

5.2. Highly oscillatory functions. Globally oscillatory functions with significant high-frequency components can not be approximated well by a shallow network when a global bounded activation function of the form $\sigma(\mathbf{W} \cdot \mathbf{x} - \mathbf{b})$, such as ReLU, is used. Due to almost orthogonality or high decorrelation (in terms of the inner product) between $\sigma(\mathbf{W} \cdot \mathbf{x} - \mathbf{b})$ and oscillatory functions with high likelihood (in terms of a random choice of (\mathbf{W}, \mathbf{b})), the set of parameters that can render a good approximation, namely the *Rashomon set* [32], becomes smaller and smaller (in terms of relative measure) and hence harder and harder to find as the target function becomes more and more oscillatory (see [45]). Although this difficulty can be alleviated by complexity decomposition using MMNN as shown in section 2, it still requires a larger network in terms of width, rank, layers, and more training. Here, we limit our tests to oscillatory functions in one-dimensional and two-dimensional cases due to the dramatic increase in complexity with dimensions, or the curse of dimensions, in general.

We again start with a one-dimensional example, $f(x) = \sin(50\pi x), x \in [-1, 1]$. An MMNN of size (800, 40, 15) produces a good approximation of this highly oscillatory function, as illustrated by the error plot in Figure 21, with a smaller learning rate and a longer training process compared to previous examples with localized fine features. Due to the significant depth, we consider using ResMMNN, as discussed in section 2.1. For this test, a total of 1000 uniformly sampled points in $[-1, 1]$ are used with a minibatch size of 100 and a learning rate of $10^{-4} \times 0.9^{\lfloor k/800 \rfloor}$, where $k = 1, 2, \dots, 40000$ is the epoch number. Also, an interesting learning dynamic for Adam is observed

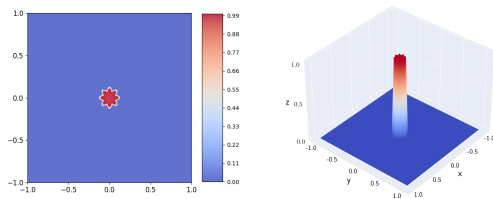


FIG. 17. Target function.

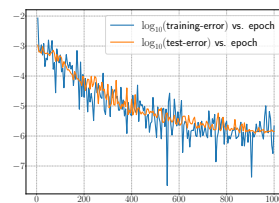


FIG. 18. Errors (in MSE).

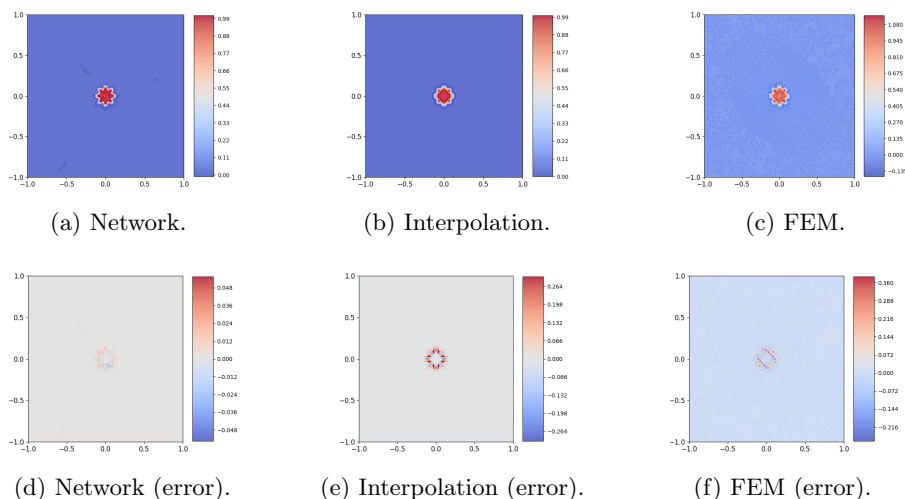


FIG. 19. Comparison among different approximations using MMNN, interpolation, and least square FEM. The interpolation and FEM are all based on a $72 \times 72 = 5184$ uniform grid. MMNN has $(100 + 1) \times 10 \times (6 - 1) + (100 + 1) = 5151$ free parameters. The maximum error is approximately 0.05 for MMNN, 0.31 for interpolation, and 0.38 for FEM. The corresponding MSE errors are 0.85×10^{-6} , 1.95×10^{-4} , and 1.45×10^{-4} , respectively.

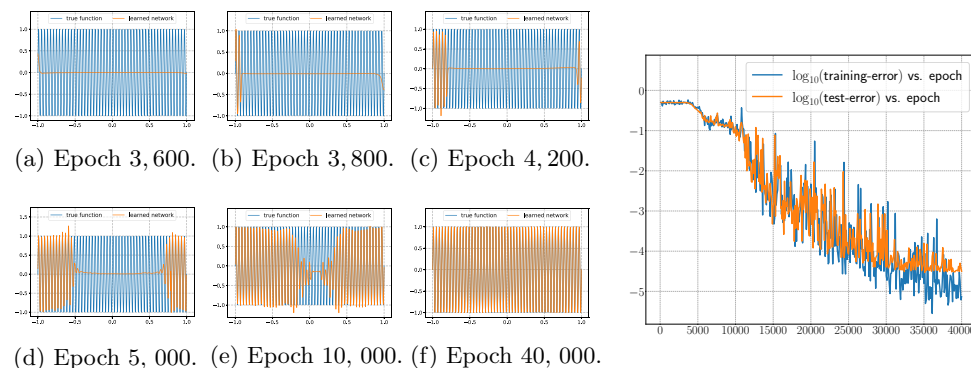


FIG. 20. Illustrations of the training process.

FIG. 21. Training and test errors (in MSE) vs. epoch.

from Figure 20. In the beginning, nothing seems to happen until about epoch 3600, when learning starts from the boundary. Then, more and more features are captured gradually from the boundary to the inside. Eventually, all features are captured and then fine-tuned together to improve the overall approximation.

Next, we consider a two-dimensional target function of the following form:

$$f_s(x_1, x_2) = \sum_{i=1}^2 \sum_{j=1}^2 a_{i,j} \sin(sb_i x_i + sc_{i,j} x_i x_j) \cos(sb_j x_j + sd_{i,j} x_i^2),$$

where

$$(a_{i,j}) = \begin{bmatrix} 0.3 & 0.2 \\ 0.2 & 0.3 \end{bmatrix}, \quad (b_i) = \begin{bmatrix} 2\pi \\ 4\pi \end{bmatrix}, \quad (c_{i,j}) = \begin{bmatrix} 2\pi & 4\pi \\ 8\pi & 4\pi \end{bmatrix}, \quad (d_{i,j}) = \begin{bmatrix} 4\pi & 6\pi \\ 8\pi & 6\pi \end{bmatrix}.$$

In our test, we choose $s = 3$ to ensure the function exhibits significant oscillations and contains diverse Fourier modes as illustrated by Figure 22. Given the complexity

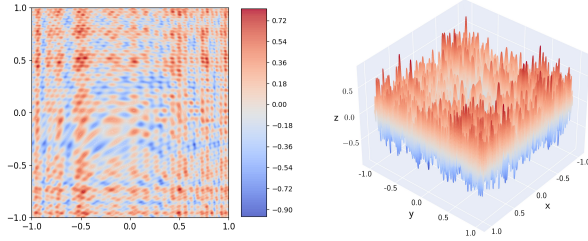


FIG. 22. Illustrations of the target function.

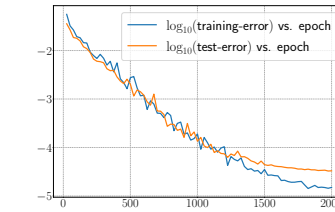


FIG. 23. Training and test errors (in MSE) vs. epoch.

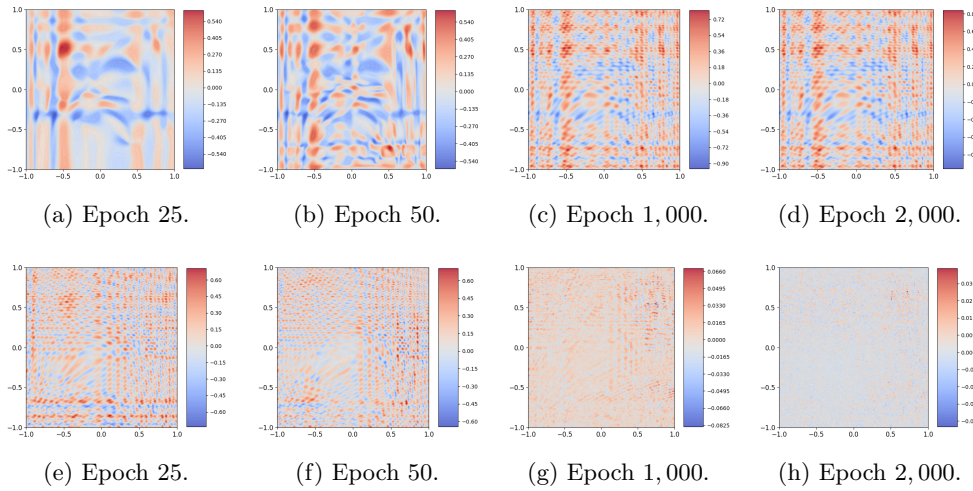


FIG. 24. The top row: the learned neural network; the bottom row: error.

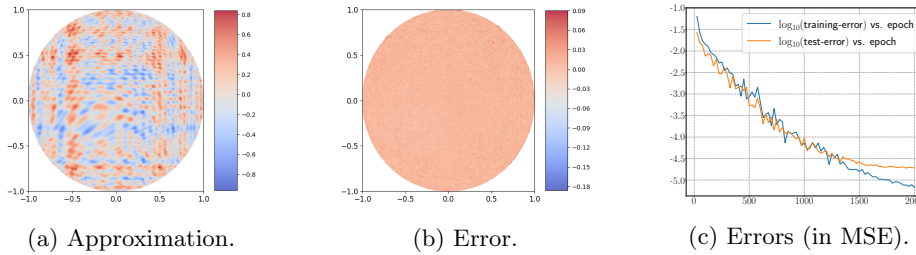


FIG. 25. MMNN approximation in a unit disk.

of the function, we employ an MMNN with size $(600, 30, 15)$. Again, ResMMNN is used due to the depth. For this test, a total of 400^2 data are sampled on a uniform grid in $[-1, 1]^2$ with a minibatch size of 1000 and a learning rate of $10^{-3} \times 0.9^{\lfloor k/40 \rfloor}$, where $k = 1, 2, \dots, 2000$ is the epoch number. The training process is illustrated by Figure 24. Figure 23 shows the log-error plot.

We trained the same function using identical network settings, except we limited the domain of interest to a unit disc. We sampled 452^2 data points uniformly distributed over the $[-1, 1]^2$ area, then filtered to retain only those points that fall within the unit disc, totaling approximately $159692 (\approx 400^2)$ samples. As illustrated in Figure 25, our network successfully learned the target function in the disc with

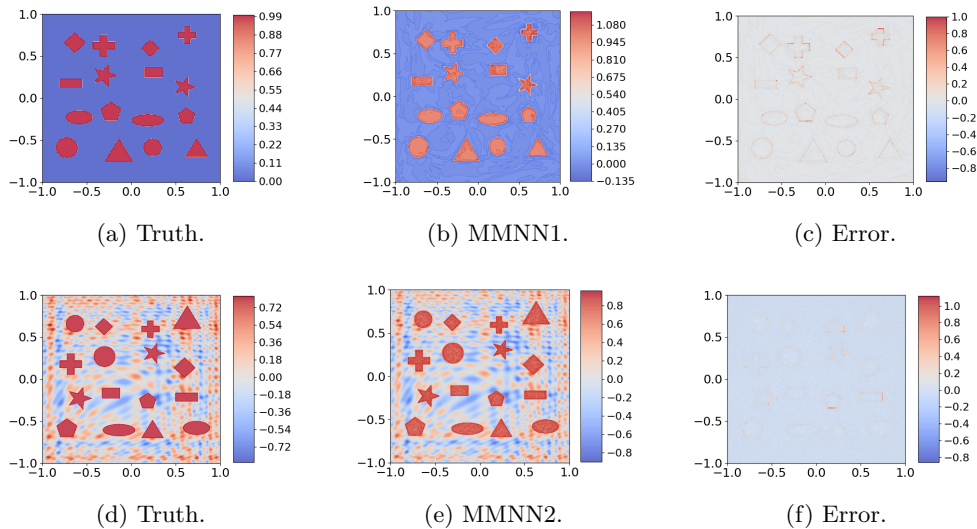


FIG. 26. True functions, corresponding learned networks, and their differences (errors). The top and bottom rows correspond to two different target functions.

TABLE 4
Test errors for two approximation results in Figure 26.

target function	network	(width, rank, depth)	#parameters (trained / all)	test error (MSE)	test error (MAX)
Figure 26(a)	MMNN1	(256, 12, 6)	15677 / 33085	1.35×10^{-3}	9.94×10^{-1}
Figure 26(d)	MMNN2	(1024, 32, 6)	165025 / 337057	1.14×10^{-3}	1.12×10^0

no adjustments or modifications. This test highlights the network's flexibility for domain geometry, an advantage over traditional mesh or grid-based methods, especially in higher dimensions.

5.3. Discontinuous functions with porous structures. Earlier, we tested highly oscillatory functions that are still continuous. To be more inclusive in our experiments, we now consider piecewise smooth functions with porous structures and complicated interfaces. Two target functions are shown in Figure 26(a,d). The first is a constant function with holes of various shapes removed (piecewise constant), while the second is based on the function in Figure 4 with holes introduced (piecewise smooth). We choose an MMNN of size (256, 12, 6), denoted MMNN1, and another of size (1024, 32, 6), denoted MMNN2, to learn these two functions, respectively. For training, we sample 600^2 data points on a uniform grid in $[-1, 1]^2$ using a minibatch size of 1,000 and a learning rate of $0.001 \times 0.9^{\lfloor k/20 \rfloor}$ for epochs $k = 1, 2, \dots, 1600$.

As shown in Figure 26, MMNNs demonstrate an impressive ability to simultaneously localize and capture discontinuities, geometric features, and oscillatory behaviors. This indicates that MMNNs are adaptive in both spatial and frequency domains. While the errors presented in Table 4 are somewhat larger, they are primarily concentrated near the discontinuous parts, as illustrated in Figure 26, which is reasonable.

5.4. Learning dynamics. Here, we show some interesting learning dynamics observed during the training process. As the first example in section 5.2 and the following examples show, the training process not just learns from low frequency first but can also learn feature by feature, i.e., can be localized in both frequency

domain and spatial domain. We believe this is due to the combination of MMNN’s “divide and conquer” ability and the Adam optimizer which utilizes momentum. More understanding is needed and will be studied in our future research.

We again start with a one-dimensional example, $f(x) = \sin(36\pi|x|^{1.5}), x \in [-1, 1]$. An MMNN of size (600, 30, 8) produces a good approximation of this highly oscillatory function, as illustrated by the error plot in Figure 28. For this test, a total of 1000 uniformly sampled points in $[-1, 1]$ are used with a minibatch size of 100 and a learning rate of $10^{-3} \times 0.9^{\lfloor k/200 \rfloor}$, where $k = 1, 2, \dots, 10000$ is the epoch number. As illustrated in Figure 27, the function is less oscillatory near 0. Therefore, we might anticipate that the network will initially learn the part near 0 and then feature by feature from the middle to the boundary. The experimental results presented in Figure 29 agree with our expectations.

Now, we show an example of two-dimensional function $f(r, \theta)$ (see Figure 30) defined in polar coordinates (r, θ) as

$$f(r, \theta) = \begin{cases} 0 & \text{if } 0.5 + 5\rho - 5r \leq 0, \\ 1 & \text{if } 0.5 + 5\rho - 5r \geq 1, \\ 0.5 + 5\rho - 5r & \text{otherwise,} \end{cases} \quad \text{where } \rho = 0.5 + 0.1 \cos(\pi^2\theta^2).$$

Our MMNN is of a compact size (500, 20, 8). For this test, a total of 600^2 data are sampled on a uniform grid in $[-1, 1]^2$ with a minibatch size of 1000 and a learning rate of $0.001 \times 0.9^{\lfloor k/6 \rfloor}$ for epochs $k = 1, 2, \dots, 300$. Figure 31 gives the error plot. The training process shown in Figure 38 illustrates that an overall coarse scale or low-frequency component of the shape is learned first, and then localized features are learned one by one from coarse to fine.

5.5. Tests in three dimensions and higher. Here, we test a few examples in three and four dimensions. Even sampling an interesting function becomes challenging as the dimension becomes higher. Although our examples are limited by our computation power using a laptop, our tests show that MMNN performs well and is more effective than a fully connected network.

The first example is a three-dimensional function a level set of which is shown in Figure 32. Using polar coordinates (r, θ, ϕ) , $\theta \in [0, \pi]$, $\phi \in [0, 2\pi)$, the target function $f(x, y, z)$ is defined as

$$f(r, \theta, \phi) = \begin{cases} 0 & \text{if } 0.5 + 5\rho - 5r \leq 0, \\ 1 & \text{if } 0.5 + 5\rho - 5r \geq 1, \\ 0.5 + 5\rho - 5r & \text{otherwise,} \end{cases}$$

where

$$\rho = \rho(\theta, \phi) = 0.5 + 0.2 \sin(6\theta) \cos(6\phi) \sin^2(\theta).$$

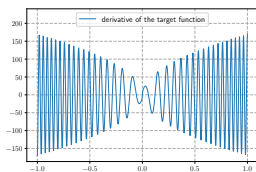


FIG. 27. Derivative of f .

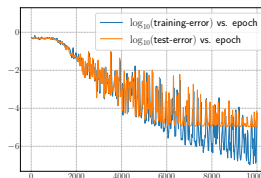


FIG. 28. Errors (in MSE) vs. epoch.

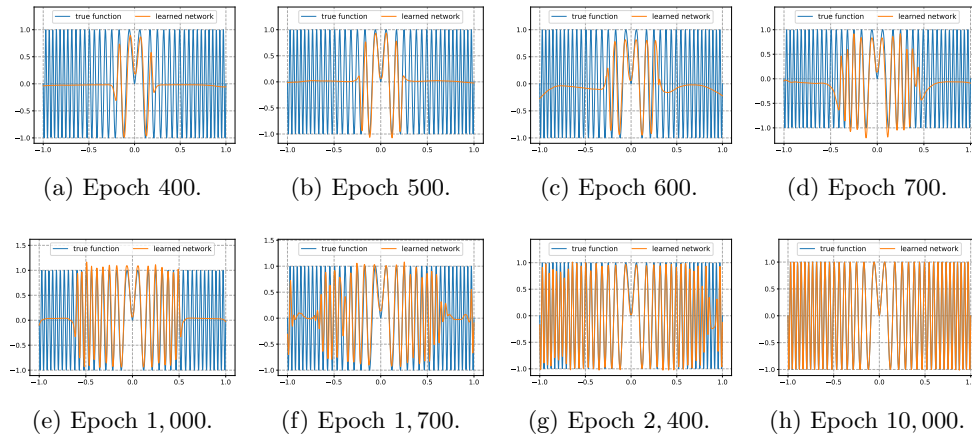


FIG. 29. Illustration of training process.

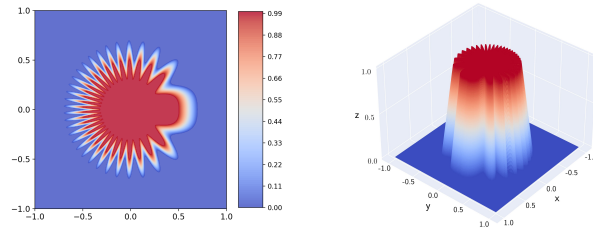


FIG. 30. Illustration of target function.

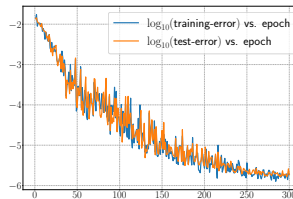


FIG. 31. Training and test errors (in MSE) vs. epoch.

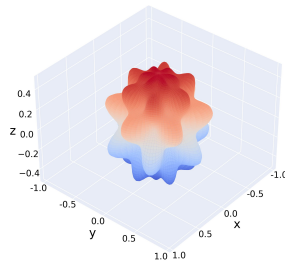


FIG. 32. Surface of levelset $f(r, \theta, \phi) = 0.5$.

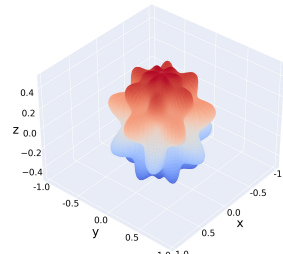


FIG. 33. Surface of levelset $h(r, \theta, \phi) = 0.5$.

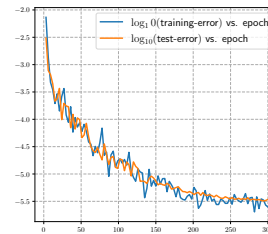


FIG. 34. Training and test errors (MSE) vs. epoch.

Our MMNN is of a compact size $(600, 20, 8)$. For this test, a total of 111^3 data are sampled on a uniform grid in $[-1, 1]^3$ with a minibatch size of 999 and a learning rate of $0.0005 \times 0.9^{\lfloor k/6 \rfloor}$ for epochs $k = 1, 2, \dots, 300$. Figure 34 gives the error plot. As shown in Figures 32 and 33, the levelsets corresponding to the target function f and the learned MMNN approximation h are nearly identical. To visually demonstrate the quality of the approximation and complex structure of the three-dimensional function, we present several slices of the target function and the MMNN approximation by fixing x , y , or z in Figure 35.

Next, we consider the probability density function (PDF) of a Gaussian (normal) distribution in four dimensions as

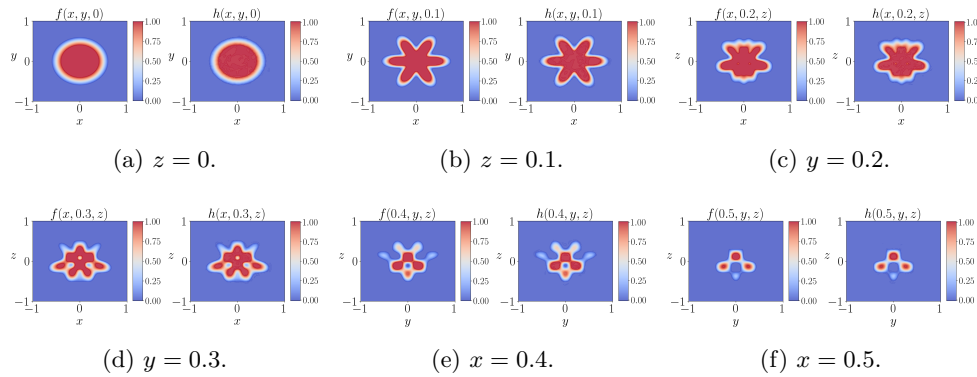


FIG. 35. Slices of true function $f(x, y, z)$ vs. those of MMNN approximation $h(x, y, z)$.

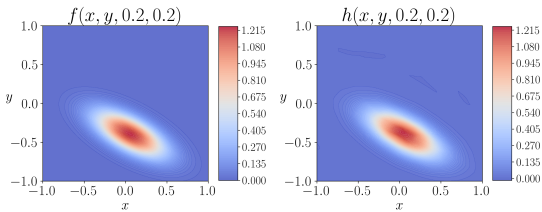


FIG. 36. True function $f(x, y, z, u)$ versus the learned network $h(x, y, z, u)$ with $z = u = 0.2$.

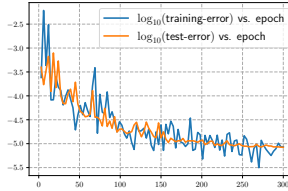


FIG. 37. Training and test errors (MSE) vs. epoch.

$$f(\mathbf{x}) = f(x_1, \dots, x_4) = \frac{\exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right)}{\sqrt{(2\pi)^k \det(\boldsymbol{\Sigma})}},$$

where $\boldsymbol{\Sigma}$ is the covariance matrix. We set $\boldsymbol{\mu} = \mathbf{0}$ and

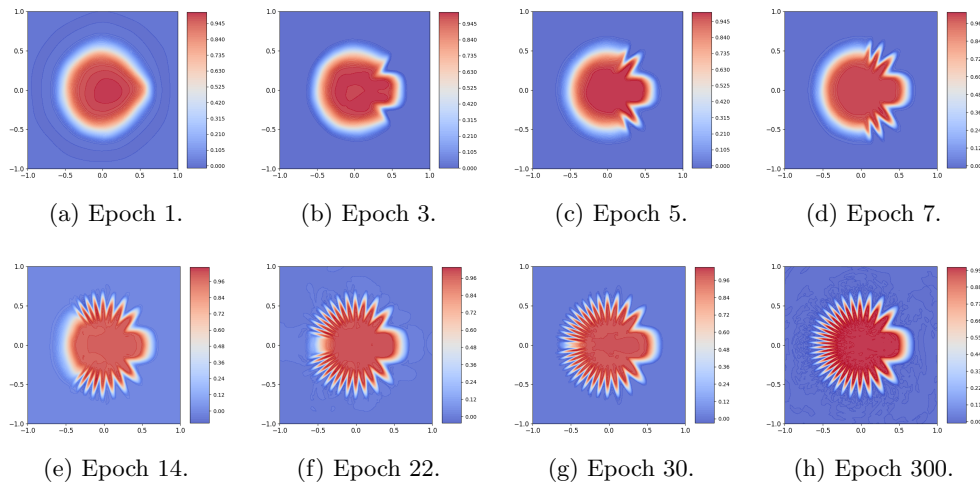
$$\boldsymbol{\Sigma}^{-1} = 20 \begin{bmatrix} 1.0 & 0.9 & 0.8 & 0.7 \\ 0.9 & 2.0 & 1.9 & 1.8 \\ 0.8 & 1.9 & 3.0 & 2.9 \\ 0.7 & 1.8 & 2.9 & 4.0 \end{bmatrix}.$$

We remark that the eigenvalues of $\boldsymbol{\Sigma}^{-1}$ are 6.82, 9.93, 25.28, 158.05 which means that the distribution is quite anisotropic and concentrated near the center.

A compact MMNN with size of (500, 12, 6) produces a good approximation, as shown in the error plot in Figure 37. Figure 36 compares the true function $f(x, y, z, u)$ and the MMNN approximation $h(x, y, z, u)$ with $z = u = 0.2$. For this test, a total of 35^4 data are sampled on a uniform grid in $[-1, 1]^4$ with a minibatch size of 35^2 and a learning rate at $10^{-3} \times 0.9^{\lfloor k/6 \rfloor}$ for epochs $k = 1, 2, \dots, 300$.

6. Further discussion. In this section, we provide a few more comments about MMNNs. First, in section 6.1, we discuss the advantages of MMNNs over fully connected networks (FCNNs) or multilayer perceptrons (MLPs). Next, in section 6.2, we offer practical guidelines for determining the appropriate MMNN size based on our theoretical understanding and extensive numerical experiments. Finally in section 6.3, we discuss the use of alternative activation functions beyond ReLU in MMNNs.

6.1. Advantages compared to FCNNs or MLPs. The two key differences between a standard FCNN or MLP and an MMNN are the (1) introduction of the

FIG. 38. *Illustration of learning dynamics.*

weights \mathbf{A}, \mathbf{c} for different linear combinations of hidden neurons (or perceptrons) as the multicomponents in each layer, and (2) training strategy that fixes those randomly initialized \mathbf{W}, \mathbf{b} (random features) in the hidden neurons. Hence, it is extremely easy to modify a FCNN or MLP to an MMNN.

MMNNs are much more effective than FCNNs in terms of representation, training, and accuracy, especially for complex functions. In comparison, as shown in those experiments in section 3, MMNNs (1) have much fewer training parameters, (2) converge much faster in training, and (3) achieve much better accuracy. Moreover, experiments show that the training process of MMNNs converges not only faster but also with a steady rate while FCNNs saturate pretty early to a quite low accuracy, as commonly observed in practices. These nice behaviors of MMNNs are due to their balanced structure for smooth decomposition, as well as the training strategy. In practice, the introduction of \mathbf{A}, \mathbf{c} in MMNNs provides an important balance between the network width, which is the number of hidden neurons (basis functions) and can be very large, and the dimension of the input space, which is the number of components from the previous layer and can be much smaller than the network width. In other words, using a few linear combinations of the basis functions can capture smooth structures in the input space well. On the other hand, for FCNNs, the two are the same, and no balance is exerted.

6.2. Practical guidelines for MMNNs. There are three hyperparameters for the configuration of MMNN sizes: the network width, the number of components (rank), and the number of layers (depth). Here are the general guidelines based on our mathematical construction and extensive experiments.

1. The network width should provide enough resolution to capture fine details of the target function. This means that the width should be at least comparable to the size of an adaptive mesh that can approximate the target function well.
2. The number of components (rank) is related to the overall complexity of the target function, which depends on its spatial domain and Fourier domain representation, as well as the input dimension. As indicated by our mathematical multicomponent construction, it is related to the “divide and conquer” strategy.

3. The number of layers (depth) is also related to the overall complexity of the target function as for the number of components. Rank and depth are complementary but work together effectively for a smooth decomposition of the target function. The rule of thumb for depth is similar to that for the rank.

Here, we use more concrete examples to illustrate the guidelines. For simplicity, we fix the input dimension and domain of interest. As the domain size and dimension increases, the network size needs to increase correspondingly. For a smooth target function, a compact MMNN in terms of width, rank, and depth is enough, and an easy training process can render accurate results. Larger MMNNs are needed for target functions with localized rapid changes. Even with a relative compact size, the training process can allocate resources adaptive to the target function and render good approximation. The most difficult situation is to approximate globally highly oscillatory functions with diverse Fourier modes for which large MMNNs are needed. For instance, if the oscillation frequency doubles, the network width should increase by 2^d , where d is the dimension. In general the network width needs to deal with the curse of dimensionality just like a mesh based method. However, the growth in the number of components and layers with the increase in complexity seems to be relative mild (maybe polylogarithmic as suggested by our mathematical construction).

Overall, for a given target function, MMNNs can work well with quite a large range of configurations with a trade-off between the network size and training process. For example, the training process for a network more of a compact size with respect to the complexity of a given target function may become more subtle and challenging, e.g., choosing the appropriate learning rate and batch size, due to the lack of flexibility (or redundancy) of the representation. On the other hand, a network of too large a size (or redundancy) with respect to the complexity of a given target function requires unnecessarily expensive training costs. There is a trade-off between representation and optimization that one needs to balance in practice. An important question for future research is how to develop a self-adaptive strategy to adjust the network size.

The most advantageous situation for using MMNNs is when approximating a function in a relative high dimension which is mostly smooth, except for localized fine features, e.g., a distribution in high dimensions concentrated on a low dimensional manifold. Through training, MMNNs can provide an automatic adaptive approximation of the underlying structure which can be challenging for a mesh-based method.

We would like to remark that a learning rate scheduler can be a subtle and important issue for all training processes in practice. For all our training process, the Step Learning Rate suffices. However, one could consider using other learning rate schedulers, such as the Cosine Scheduler [20] or the gradual warm-up strategy [6]. Exploring and designing a more efficient learning rate scheduler with some automatic restart mechanism is a potential interesting topic for future work.

6.3. Beyond ReLU to other activation functions. We also tried using different activation functions for MMNNs, e.g., GELU [10], Swish [28], Sigmoid, and Tanh. In general, ReLU provides the overall best results for various target functions. However, in situations where a smooth (e.g., C^s or real analytic) approximation is needed, one might consider using smooth alternatives to ReLU, such as GELU or Swish, which generally yield results comparable to ReLU.

Additionally, other popular S-shaped activation functions like Sigmoid and Tanh have demonstrated poor performance in our tests, possibly due to the vanishing gradient problem. For highly oscillatory target functions, when using Sigmoid or Tanh, training errors did not even decrease during the training process.

7. Conclusion. In this work, we introduced the multicomponent and multilayer neural network (MMNN) and demonstrated its effectiveness in approximating complex functions. By incorporating the principles of structured and balanced decomposition, the MMNN architecture addresses the limitations of shallow networks, particularly in capturing high-frequency components and localized fine features. Our proposed network structure as confirmed by extensive numerical experiments can approximate highly oscillatory functions and functions with rapid transitions efficiently and accurately. Additionally, we highlight the advantages of our training strategy, which optimizes only the linear combination weights of basis functions for each component while keeping the parameters within the activation (basis) functions fixed, leading to a more efficient and stable training process.

The theoretical underpinnings and practical implementations presented in this paper suggest that MMNNs offer a promising direction for constructing neural networks capable of handling complex tasks with fewer parameters and reduced computational overhead. Future research can explore further generalizations and applications of MMNNs, as well as investigate the interplay between representation and optimization in more depth.

REFERENCES

- [1] J. ASH AND R. P. ADAMS, *On warm-starting neural network training*, in Advances in Neural Information Processing Systems, Vol. 33, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, eds., Curran Associates, 2020, pp. 3884–3894, https://proceedings.neurips.cc/paper_files/paper/2020/file/288cd2567953f06e460a33951f55daaf-Paper.pdf.
- [2] H. BÖLCSKEI, P. GROHS, G. KUTYNIOK, AND P. PETERSEN, *Optimal approximation with sparsely connected deep neural networks*, SIAM J. Math. Data Sci., 1 (2019), pp. 8–45, <https://doi.org/10.1137/18M118709X>.
- [3] C. K. CHUI, S.-B. LIN, AND D.-X. ZHOU, *Construction of neural networks for realization of localized deep learning*, Front. Appl. Math. Statist., 4 (2018), 14, <https://doi.org/10.3389/fams.2018.00014>.
- [4] G. CYBENKO, *Approximation by superpositions of a sigmoidal function*, Math. Control Signals Systems, 2 (1989), pp. 303–314, <https://doi.org/10.1007/BF02551274>.
- [5] X. GLOROT AND Y. BENGIO, *Understanding the difficulty of training deep feedforward neural networks*, in Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, Proceedings of Machine Learning Research 9, Y. W. Teh and M. Titterton, eds., PMLR, Chia Laguna Resort, Sardinia, Italy, 2010, pp. 249–256, <https://proceedings.mlr.press/v9/glorot10a.html>.
- [6] P. GOYAL, P. DOLLÁR, R. GIRSHICK, P. NOORDHUIS, L. WESOLOWSKI, A. KYROLA, A. TULLOCH, Y. JIA, AND K. HE, *Accurate, Large Minibatch SGD: Training Imagenet in 1 Hour*, e-prints, 2017, <https://doi.org/10.48550/arXiv.1706.02677>.
- [7] R. GRIBONVAL, G. KUTYNIOK, M. NIELSEN, AND F. VOIGTLAENDER, *Approximation spaces of deep neural networks*, Constr. Approx., 55 (2022), pp. 259–367, <https://doi.org/10.1007/s00365-021-09543-4>.
- [8] I. GÜHRING, G. KUTYNIOK, AND P. PETERSEN, *Error bounds for approximations with deep ReLU neural networks in $W^{s,p}$ norms*, Anal. Appl., 18 (2020), pp. 803–859, <https://doi.org/10.1142/S0219530519410021>.
- [9] K. HE, X. ZHANG, S. REN, AND J. SUN, *Deep residual learning for image recognition*, in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–778, <https://doi.org/10.1109/CVPR.2016.90>.
- [10] D. HENDRYCKS AND K. GIMPEL, *Gaussian Error Linear Units (GELUs)*, e-prints, 2016, arXiv:1606.08415, <https://doi.org/10.48550/arXiv.1606.08415>.
- [11] K. HORNIK, *Approximation capabilities of multilayer feedforward networks*, Neural Netw., 4 (1991), pp. 251–257, [https://doi.org/10.1016/0893-6080\(91\)90009-T](https://doi.org/10.1016/0893-6080(91)90009-T).
- [12] K. HORNIK, M. STINCHCOMBE, AND H. WHITE, *Multilayer feedforward networks are universal approximators*, Neural Netw., 2 (1989), pp. 359–366, [https://doi.org/10.1016/0893-6080\(89\)90020-8](https://doi.org/10.1016/0893-6080(89)90020-8).

- [13] Y. IDELBAYEV AND M.Á. CARREIRA-PERPIÑÁN, *Low-rank compression of neural nets: Learning the rank of each layer*, in 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020, pp. 8046–8056, <https://doi.org/10.1109/CVPR42600.2020.00807>.
- [14] A. ISMAYILOVA AND V. E. ISMAILOV, *On the Kolmogorov neural networks*, *Neural Netw.*, 176 (2024), 106333, <https://doi.org/10.1016/j.neunet.2024.106333>.
- [15] D. S. KALRA AND M. BARKESHLI, *Why warmup the learning rate? Underlying mechanisms and improvements*, in *Advances in Neural Information Processing Systems*, Vol. 37, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, eds., Curran Associates, 2024, pp. 111760–111801, https://proceedings.neurips.cc/paper_files/paper/2024/file/ca98452d4e9ecbc18c40da2aa0da8b98-Paper-Conference.pdf.
- [16] D. P. KINGMA AND J. BA, *Adam: A method for stochastic optimization*, in 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, Conference Track Proceedings, Y. Bengio and Y. LeCun, eds., 2015, <http://arxiv.org/abs/1412.6980>.
- [17] A. N. KOLMOGOROV, *On the representation of continuous functions of several variables by superposition of continuous functions of one variable and addition*, *Dokl. Akad. Nauk SSSR*, (1957), pp. 953–956.
- [18] F. LIU, X. HUANG, Y. CHEN, AND J. A. K. SUYKENS, *Random features for kernel approximation: A survey on algorithms, theory, and beyond*, *IEEE Trans. Pattern Anal. Mach. Intell.*, 44 (2022), pp. 7128–7148, <https://doi.org/10.1109/TPAMI.2021.3097011>.
- [19] Z. LIU, Y. WANG, S. VAIDYA, F. RUEHLE, J. HALVERSON, M. SOLJAČIĆ, T. Y. HOU, AND M. TEGMARK, *KAN: Kolmogorov-Arnold networks*, e-prints, 2024, arXiv:2404.19756, <https://doi.org/10.48550/arXiv.2404.19756>.
- [20] I. LOSHCHELOV AND F. HUTTER, *SGDR: Stochastic gradient descent with warm restarts*, in 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, Conference Track Proceedings, 2017, <https://openreview.net/forum?id=Skq89Scxx>.
- [21] J. LU, Z. SHEN, H. YANG, AND S. ZHANG, *Deep network approximation for smooth functions*, *SIAM J. Math. Anal.*, 53 (2021), pp. 5465–5506, <https://doi.org/10.1137/20M134695X>.
- [22] V. MAIOROV AND A. PINKUS, *Lower bounds for approximation by MLP neural networks*, *Neurocomputing*, 25 (1999), pp. 81–91, [https://doi.org/10.1016/S0925-2312\(98\)00111-8](https://doi.org/10.1016/S0925-2312(98)00111-8).
- [23] H. MONTANELLI AND H. YANG, *Error bounds for deep ReLU networks using the Kolmogorov-Arnold superposition theorem*, *Neural Netw.*, 129 (2020), pp. 1–6, <https://doi.org/10.1016/j.neunet.2019.12.013>.
- [24] A. NOVIKOV, D. PODOPRIKHIN, A. OSOKIN, AND D. P. VETROV, *Tensorizing neural networks*, in *Adv. Neural Inf. Process. Syst.* 28, C. Cortes and N. Lawrence and D. Lee and M. Sugiyama and R. Garnett, eds., Curran Associates, Inc., 2015.
- [25] H. PENG, N. PAPPAS, D. YOGATAMA, R. SCHWARTZ, N. SMITH, AND L. KONG, *Random feature attention*, in *International Conference on Learning Representations*, 2021, <https://arxiv.org/abs/2103.02143>.
- [26] A. RAHIMI AND B. RECHT, *Random features for large-scale kernel machines*, in *Advances in Neural Information Processing Systems*, Vol. 20, J. Platt, D. Koller, Y. Singer, and S. Roweis, eds., Curran Associates, 2007, https://proceedings.neurips.cc/paper_files/paper/2007/file/013a006f03dbc5392effeb8f18fda755-Paper.pdf.
- [27] M. RAISSI, P. PERDIKARIS, AND G. E. KARNIADAKIS, *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*, *J. Comput. Phys.*, 378 (2019), pp. 686–707, <https://doi.org/10.1016/j.jcp.2018.10.045>.
- [28] P. RAMACHANDRAN, B. ZOPH, AND Q. V. LE, *Searching for Activation Functions*, arXiv:1710.05941, 2017, <https://doi.org/10.48550/arXiv.1710.05941>.
- [29] S. RAO KAMALAKARA, A. LOCATELLI, B. VENKITESH, J. BA, Y. GAL, AND A. N. GOMEZ, *Exploring Low Rank Training of Deep Neural Networks*, e-prints, 2022, arXiv:2209.13569, <https://doi.org/10.48550/arXiv.2209.13569>.
- [30] A. RUDI AND L. ROSASCO, *Generalization properties of learning with random features*, in *Advances in Neural Information Processing Systems*, Vol. 30, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds., Curran Associates, 2017, https://proceedings.neurips.cc/paper_files/paper/2017/file/61b1fb3f59e28c67f3925f3c79be81a1-Paper.pdf.
- [31] T. N. SAINATH, B. KINGSBURY, V. SINDHWANI, E. ARISOY, AND B. RAMABHADRAN, *Low-rank matrix factorization for deep neural network training with high-dimensional output targets*, in 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, 2013, pp. 6655–6659, <https://doi.org/10.1109/ICASSP.2013.6638949>.

- [32] L. SEMENOVA, C. RUDIN, AND R. PARR, *On the existence of simpler machine learning models*, in Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency, 2022, pp. 1827–1858.
- [33] Z. SHEN, H. YANG, AND S. ZHANG, *Nonlinear approximation via compositions*, Neural Netw., 119 (2019), pp. 74–84, <https://doi.org/10.1016/j.neunet.2019.07.011>.
- [34] Z. SHEN, H. YANG, AND S. ZHANG, *Deep network approximation characterized by number of neurons*, Commun. Comput. Phys., 28 (2020), pp. 1768–1811, <https://doi.org/10.4208/cicp.OA-2020-0149>.
- [35] Z. SHEN, H. YANG, AND S. ZHANG, *Deep network approximation: Achieving arbitrary accuracy with fixed number of neurons*, J. Mach. Learn. Res., 23 (2022), pp. 1–60, <http://jmlr.org/papers/v23/21-1404.html>.
- [36] Z. SHEN, H. YANG, AND S. ZHANG, *Deep network approximation in terms of intrinsic parameters*, in Proceedings of the 39th International Conference on Machine Learning, Proceedings of Machine Learning Research 162, PMLR, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, eds., Proceedings of Machine Learning Research, PMLR, 2022, pp. 19909–19934, <https://proceedings.mlr.press/v162/shen22g.html>.
- [37] Z. SHEN, H. YANG, AND S. ZHANG, *Neural network architecture beyond width and depth*, in Advances in Neural Information Processing Systems, Vol. 35, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, eds., Curran Associates, 2022, pp. 5669–5681, https://proceedings.neurips.cc/paper_files/paper/2022/hash/257be12f31dfa7cc158dda99822c6fd1-Abstract-Conference.html.
- [38] Z. SHEN, H. YANG, AND S. ZHANG, *Optimal approximation rate of ReLU networks in terms of width and depth*, J. Math. Pures Appl., 157 (2022), pp. 101–135, <https://doi.org/10.1016/j.matpur.2021.07.009>.
- [39] A. SINHA AND J. C. DUCHI, *Learning kernels with random features*, in Advances in Neural Information Processing Systems, Vol. 29, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, eds., Curran Associates, 2016, https://proceedings.neurips.cc/paper_files/paper/2016/file/e70611883d2760c8bbafb4acb29e3446-Paper.pdf.
- [40] D. YAROTSKY, *Error bounds for approximations with deep ReLU networks*, Neural Netw., 94 (2017), pp. 103–114, <https://doi.org/10.1016/j.neunet.2017.07.002>.
- [41] D. YAROTSKY, *Optimal approximation of continuous functions by very deep ReLU networks*, in Proceedings of the 31st Conference On Learning Theory, Proceedings of Machine Learning Research 75, PMLR, S. Bubeck, V. Perchet, and P. Rigollet, eds., Proceedings of Machine Learning Research, PMLR, 2018, pp. 639–649, <http://proceedings.mlr.press/v75/yarotsky18a.html>.
- [42] S. ZHANG, *Deep Neural Network Approximation via Function Compositions*, Ph.D. thesis, National University of Singapore, 2020, <https://scholarbank.nus.edu.sg/handle/10635/186064>.
- [43] S. ZHANG, J. LU, AND H. ZHAO, *On enhancing expressive power via compositions of single fixed-size ReLU network*, Sabato, in Proceedings of the 40th International Conference on Machine Learning, Proceedings of Machine Learning Research 202, PMLR, A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, eds., Proceedings of Machine Learning Research, PMLR, 2023, pp. 41452–41487, <https://proceedings.mlr.press/v202/zhang23ad.html>.
- [44] S. ZHANG, J. LU, AND H. ZHAO, *Deep network approximation: Beyond ReLU to diverse activation functions*, J. Mach. Learn. Res., 25 (2024), pp. 1–39, <http://jmlr.org/papers/v25/23-0912.html>.
- [45] S. ZHANG, H. ZHAO, Y. ZHONG, AND H. ZHOU, *Why Shallow Networks Struggle to Approximate and Learn High Frequencies*, e-prints, 2023, arXiv:2306.17301, <https://doi.org/10.48550/arXiv.2306.17301>.
- [46] D.-X. ZHOU, *Universality of deep convolutional neural networks*, Appl. Comput. Harmon. Anal., 48 (2020), pp. 787–794, <https://doi.org/10.1016/j.acha.2019.06.004>.