CHEBYSHEV FEATURE NEURAL NETWORK FOR ACCURATE FUNCTION APPROXIMATION

Zhongshu Xu, Yuan Chen,* & Dongbin Xiu

Department of Mathematics, The Ohio State University, Columbus, Ohio 43210, USA

*Address all correspondence to: Yuan Chen, Department of Mathematics, The Ohio State University, Columbus, Ohio 43210, USA, E-mail: chen.11050@osu.edu

We present a new deep neural network (DNN) architecture capable of approximating functions up to machine accuracy. Termed the Chebyshev feature neural network (CFNN), the new structure employs Chebyshev functions with learnable frequencies as the first hidden layer, followed by the standard fully connected hidden layers. The learnable frequencies of the Chebyshev layer are initialized with exponential distributions to cover a wide range of frequencies. Combined with a multistage training strategy, we demonstrate that this CFNN structure can achieve machine accuracy during training. A comprehensive set of numerical examples for dimensions up to 20 are provided to demonstrate the effectiveness and scalability of the method.

KEY WORDS: *deep neural networks, function approximation, Chebyshev function*

1. INTRODUCTION

In recent years, machine learning has become a prominent research area. It has been widely applied in computer vision (Voulodimos et al., 2018), data analysis (James et al., 2013), natural language processing (Chowdhary, 2020), and recommendation systems (Pazzani and Billsus, 2007). In particular, deep neural networks (DNNs) are frequently used in machine learning algorithms. Recently, in the area of scientific computing, DNN-based methods have been widely developed for, for example, solving differential equations (Raissi et al., 2019; Sirignano and Spiliopoulos, 2018; Yu et al., 2018), inverse problems (Jin et al., 2017; Li et al., 2020a, 2019), operator learning (Brunton et al., 2016; Li et al., 2020b; Lu et al., 2021; Qin et al., 2019), etc. In those problems, high-precision function approximations are an intrinsic requirement for the application of DNNs.

The mathematical foundation of DNNs rests on the celebrated universal approximation theorem (Hornik et al., 1989), which states that DNNs are capable of approximating functions with arbitrary precision. However, it is well known that the practical accuracy of DNNs is lacking for many scientific computing problems. For example, the training accuracy of DNNs can often reach a plateau level of $\mathcal{O}(10^{-5})$ to $\mathcal{O}(10^{-2})$. While this may be adequate for imaging analysis and classification problems, it is insufficient for many scientific computing problems, where high precision is critical for long-term predictions. Efforts have been made to partially understand this, see, for example, the so-called spectral bias phenomenon (Fanaskov and Oseledets, 2023; Rahaman et al., 2019) or frequency principle (Xu et al., 2019), although it is unclear when a complete understanding will be available. Practical algorithms have also been designed

Submitted: 9/27/24; Accepted: 12/18/24; Online: 1/7/25
https://dl.begellhouse.com/doi/10.1615/JMachLearnModelComput.2025056536
2689–3967/25/$35.00

to enhance DNNs accuracy, see, for example, Fourier features networks (Tancik et al., 2020), multistage training (Wang and Lai, 2024), and Hat activation functions (Hong et al., 2022).

In this paper, we present the Chebyshev feature neural network (CFNN) architecture and its training procedure to achieve arbitrary training accuracy, down to machine precision, should one desire. Our work is inspired by the original work of Wang and Lai (2024), which employed Fourier feature network and multistage training to reach machine precision during training. The contribution of our work lies in the following aspects: (1) the use of Chebyshev features with learnable frequencies in the first hidden layer. This takes advantage of the superior function approximation properties offered by Chebyshev functions and also contains fewer hyperparameters than the Fourier features network; (2) a random sampling strategy using exponential distribution for the initialization of the Chebyshev frequency in each stage of the multistage training. This alleviates the difficulty of choosing the frequency parameters in Wang and Lai (2024); and (3) a systematical study of the method for a large class of functions. Through extensive numerical study, we demonstrate that CFNN offers excellent approximation capability, for smooth functions or discontinuous functions, in various dimensions.

2. SETUP

We consider the classical function approximation problem. Let $f : \mathbb{R}^d \mapsto \mathbb{R}$, $d \geq 1$, be an unknown function. Let $f(\mathbf{x}_1), \ldots, f(\mathbf{x}_N)$, $N \geq 1$, be its samples. We seek to find an approximation $\widehat{f}(\mathbf{x}) \approx f(\mathbf{x})$.

3. CHEBYSHEV FEATURE NEURAL NETWORK

We consider the classical function approximation problem: Let $f : D \mapsto \mathbb{R}$, $D \subset \mathbb{R}^d$, $d \geq 1$, be an unknown function and let $f(\mathbf{x}_1), \ldots, f(\mathbf{x}_N)$, $N \geq 1$, denote its samples. The objective is to find an approximation function $\widehat{f}$ such that $\widehat{f}(\mathbf{x}) \approx f(\mathbf{x})$, $\forall x \in D$. A common approach is to seek an approximation from a parameterized family of functions $\widehat{f}(\cdot;\theta)$ and determine the parameter θ by minimizing the discrepancy between the samples. The mean squared error (MSE) is often used to quantify this discrepancy:

$$\min_\theta \frac{1}{N} \sum_{i=1}^{N} \Big[f(\mathbf{x}_i) - \widehat{f}(\mathbf{x}_i;\theta) \Big]^2. \tag{1}$$

In this work, we focus on using DNN for function approximation. Let $\mathbf{N} : \mathbb{R}^d \mapsto \mathbb{R}$ represent the DNN-based mapping operator. The DNN approximates f by

$$y = \mathbf{N}(\mathbf{x};\Theta), \tag{2}$$

where Θ represents all the network's parameters (e.g., weights and biases). The values of Θ are determined by minimizing the MSE loss in Eq. (1).

3.1 Chebyshev Features

Chebyshev polynomials, a well-known class of orthogonal polynomials, are frequently employed in function approximation, often achieving near-optimal performance; see, for instance,

Quarteroni et al. (2006), Szego (1939), and Trefethen (2013). For $n \geq 0$, the nth Chebyshev polynomial T_n is defined as

$$T_n(x) = \cos(n \arccos(x)), \qquad x \in [-1, 1].$$

In this paper, we generalize Chebyshev polynomials by extending their degree from integers to real numbers. Specifically, we consider

$$T_\alpha(x) = \cos(\alpha \arccos(x)), \qquad \alpha \in \mathbb{R}_0^+. \tag{3}$$

By introducing this extension, we broaden the "frequency" domain of Chebyshev polynomials to $\mathbb{R}_0^+$. We refer to these generalized functions as Chebyshev features.

This concept parallels the extension used in the Fourier features network (Tancik et al., 2020). However, unlike Fourier features, Chebyshev features do not include a "phase" parameter.

3.2 CFNN Construction

The CFNN is designed as a fully connected feedforward network. The dimensionality of the input and output layers corresponds to the dimensions of the target function: the input layer contains d nodes, while the output layer consists of a single node (as in the case of this paper).

Let $L \geq 1$ represent the number of hidden layers between the input and output layers. For simplicity, we assume each hidden layer contains the same number of $K \geq 1$ nodes. In CFNN, the first hidden layer employs the Chebyshev features (3) as the activation function, while the remaining hidden layers are the standard feedforward layers. An illustration of the CFNN structure is provided in Fig. 1. CFNN then defines a mapping operator:

$$\mathbf{N}_{\text{CF}} := \phi_{\text{out}} \circ \phi_L \circ \cdots \circ \phi_2 \circ \phi_{\text{CF}}. \tag{4}$$

Here, ϕ_{CF} is the Chebyshev features operator (3) in the first hidden layer,

$$\phi_{\text{CF}}(\mathbf{x}) = \cos(W_{\text{CF}} \arccos(\mathbf{x})), \tag{5}$$

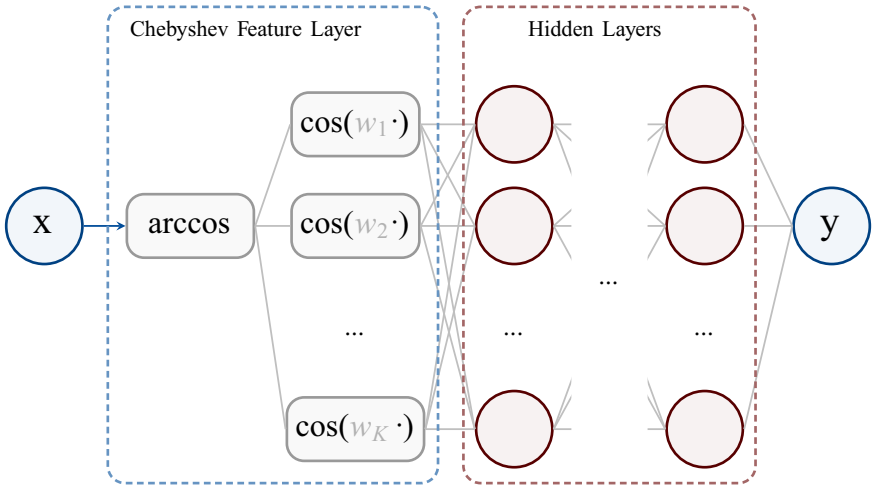

FIG. 1: Architechture of the proposed CFN

where $\mathbf{x} \in \mathbb{R}^d$ is the input, and $W_{\text{CF}} = [w_1^T, ..., w_K^T]$ is the weight matrix, with $K \geq 1$ representing the number of neurons in the next hidden layer. Both $\arccos(\cdot)$ and $\cos(\cdot)$ are applied component-wise. The remaining layers accomplish the standard operations,

$$\phi_i(\mathbf{x}) := \sigma(W_i\mathbf{x} + b_i), \qquad i = 2, \ldots, L,$$

where $\mathbf{x}$ is the outputs from the $(i-1)$th layer; W_i and b_i are the weights and biases, respectively; and $\sigma(\cdot)$ is the activation function. In this paper, we employ $\tanh(\cdot)$ in these layers. Finally, the output layer is a linear layer:

$$\phi_{\text{out}} := W_{\text{out}}\mathbf{x} + b_{\text{out}}.$$

3.3 CFNN Training

In addition to the network architecture, network training plays a crucial role in ensuring that CFNN achieves machine accuracy. Here, we discuss the key components of CFNN training.

3.3.1 Multistage Training

The idea of multistage training is to train NN on the training residual of the NN at the previous stage. It is a general approach that can be adopted for any NN learning. In our setting, let us assume we have trained a CFNN on the given training data set and denote it as $\mathbf{N}_{\text{CF}}^{(0)}(\mathbf{x})$ at stage 0. Let $E_1(\mathbf{x}) = f(\mathbf{x}) - \mathbf{N}_{\text{CF}}^{(0)}(\mathbf{x})$ be the residue. In the next stage, we train a second CFNN, $\mathbf{N}_{\text{CF}}^{(1)}$, to approximate the scaled residual using the data set $(\mathbf{x}_i, E_1(\mathbf{x}_i)/\epsilon_1)$, $i = 1, ..., N$, where the normalizing factor ϵ_1 is defined as

$$\epsilon_1 = \sqrt{\frac{1}{N}\sum_{i=1}^{N}[E_1(\mathbf{x}_i)]^2}. \tag{6}$$

The final approximation is then

$$\widehat{f}^{(1)}(\mathbf{x}) = \mathbf{N}_{\text{CF}}^{(0)}(\mathbf{x}) + \epsilon_1\mathbf{N}_{\text{CF}}^{(1)}(\mathbf{x}). \tag{7}$$

This iterative process can be readily extended to an arbitrary number of stages $S \geq 1$ to achieve an approximation

$$\widehat{f}^{(S)}(\mathbf{x}) = \mathbf{N}_{\text{CF}}^{(0)}(\mathbf{x}) + \sum_{s=1}^{S-1}\epsilon_s\mathbf{N}_{\text{CF}}^{(s)}(\mathbf{x}), \tag{8}$$

where $\mathbf{N}_{\text{CF}}^{(s)}(\mathbf{x})$ is trained on the residue data set $E_s = f - \widehat{f}^{(s-1)}$,

$$\left(\mathbf{x}_i, \frac{1}{\epsilon_s}E_s(\mathbf{x}_i)\right), \qquad i = 1, ..., N,$$

with ϵ_s being the normalizing factor of E_s.

3.3.2 Network Initialization

Network initialization is an important computational aspect. In Wang and Lai (2024), scaling parameters were introduced to multistage Fourier NN in order to achieve machine accuracy. However, the selection of the scaling parameters depends critically on the properties of the target

function. This makes it challenging to apply in practical situations. For CFNN, we propose a network initiation approach that can apply to general target functions.

Our initialization method employs random sampling of the network parameters with different distributions. For the first hidden layer, i.e., the Chebyshev feature layer (5), the weights W_{CF} are sampled via exponential distribution, with different parameters at different training stages during the multistage training (8). Specifically, at the sth stage, we set

$$W_{\text{CF}}^{(s)} \sim \text{Exp}(\lambda^{(s)}) + c^{(s)}, \qquad s \geq 0.$$

In stage 0, $\lambda^{(0)} = 5$ and $c^{(0)} = 0$; in the later stages $\lambda^{(s)} = 5^{1-s}$ and $c^{(s)} = 2 \times 5^{s-1}$. This initialization encourages the network to first capture the lower-frequency components of the target function, and then in the later stages to capture the high-frequency residuals. The exponential distribution increases the likelihood of some of the parameters in the Chebyshev feature layers aligning with the predominant frequencies of the target function and residues.

For the hidden layers after the Chebyshev feature layer, the weight matrices W_i and biases b_i are initialized with the widely used Xavier method for all the training stages:

$$W_i \sim \mathcal{N}(0, \sigma^2), \qquad \sigma = \sqrt{2/(N_{i-1} + N_i)},$$

where N_{i-1} and N_i are the numbers of neurons in the previous and next layers. All biases b_i are initialized to zero.

4. NUMERICAL EXAMPLE

In this section, we present a comprehensive set of numerical examples to examine the accuracy of the proposed CFNN approach. In one dimension ($d = 1$), we consider the following six functions:

$$\begin{aligned}
&f_1(x) = x,\\
&f_2(x) = \sin(2x+1) + 0.2e^{1.3x},\\
&f_3(x) = |\sin(\pi x)|^2,\\
&f_4(x) = \left(1 - \frac{x^2}{2}\right)\cos\left[m\left(x + 0.5x^3\right)\right], \qquad m = 30,\\
&f_5(x) = |x|,\\
&f_6(x) = \text{sign}(x).
\end{aligned} \tag{9}$$

In multidimension ($d > 1$), we test the following three functions that are widely used for multi-dimensional function approximation:

$$\begin{aligned}
f_7(\mathbf{x}) &= \sum_{i=1}^{d} x_i,\\
f_8(\mathbf{x}) &= \exp\left(-\sum_{i=1}^{d} \sigma_i^2 \left(\frac{x_i+1}{2} - \omega_i\right)^2\right),\\
f_9(\mathbf{x}) &= \sum_{i=1}^{N} \alpha_i \exp\left(-\sum_{j=1}^{d} \sigma_{ij}^2 \left(\frac{x_j+1}{2} - \omega_{ij}\right)^2\right).
\end{aligned} \tag{10}$$

The domain of these functions will be specified in the following subsections. The parameters α, σ, and ω determine the behavior of the functions and are specified in each example. For the

Gaussian peak function f_8, we set $\sigma_i = 1$ and $\omega_i = 1, \forall i$. For the multimodal function f_9, we set $N = 10$, $\sigma_{ij} \equiv 1$, and randomly sample ω_{ij} uniformly in [–1, 1] and α_i uniformly in [–10, 10].

In most examples, we employ CFNN with three hidden layers, each containing 40 neurons (including the first Chebyshev feature layer). For the network training at each stage, we employ the Adam optimizer for 5000 epochs, followed by the L-BFGS optimizer for an additional 20,000 epochs. The Adam optimization utilizes an exponential decay learning rate, which is initialized at 0.01 and decays by a factor of 0.97 for every 100 epochs. We conducted tests on both single and double precision, and similar results were observed. In this paper, we only report double-precision results.

4.1 One-Dimensional Examples

For one-dimensional examples, we set the domain as $D = [-1, 1]$. The training set consists of 3000 equidistant points in D. To evaluate the generalization capability, we test the trained network on a separate set of 10,000 equidistant points in D.

4.1.1 Smooth Functions

In this section, we consider the four smooth functions, f_1, f_2, f_3, and f_4, in Eq. (9).

For the linear function $f_1(x)$, we conduct its approximation with a four-stage CFNN. The combined results are presented in Fig. 2. In the left panel of the figure, the training loss is displayed at the top, where we observe that the training RMSE is reduced to $\mathcal{O}(10^{-30})$ through four stages of training. The target function and the CFNN approximation are plotted in the middle, where no visible difference can be observed. The pointwise difference, termed test error, is shown in the bottom plot of the left panel. We observe the pointwise errors are of $\mathcal{O}(10^{-13})$.

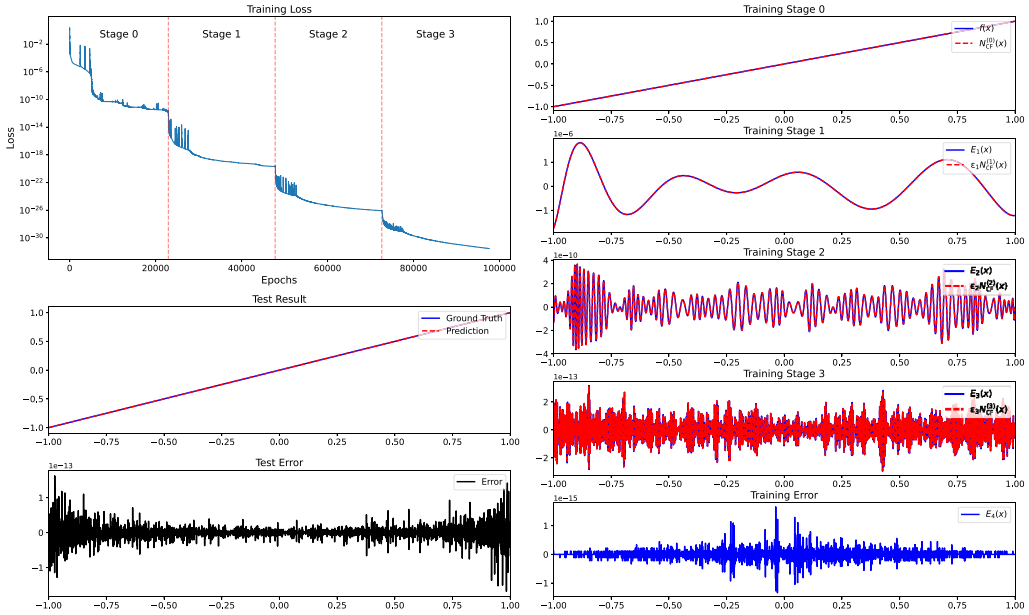

FIG. 2: Results of $f_1(x)$. Left top: training loss history across all stages; Left bottom: test data ground truth vs. prediction and prediction error; Right: training results at four stages showing the network output (red dashed) against the ground truth/residual (blue solid) at each stage.

In the right panel of Fig. 2, we present the approximation results on the *training data set* for all four stages of CFNN, from top to bottom. Note that at Training Stage 0, the target function is the original function, whereas at later stages the target functions are the residues of the CFNN approximations of the previous stages. The pointwise error is shown at the very bottom. We observe that for such an extremely oscillatory target function of Training Stage 3, the CFNN can achieve an approximation error of $\mathcal{O}(10^{-15})$, the machine epsilon level for double-precision computation.

We now consider the smooth nonlinear function f_2, whose results are shown in Fig. 3. The left panel shows (from top to bottom) the training loss decay over the four training stages; the CFNN function approximation at the testing set (along with the target function f_2); and pointwise error of the CFNN approximation. We observe that the training loss decays to $\mathcal{O}(10^{-28})$ and trained CFNN approximation error is at $\mathcal{O}(10^{-12})$. The right panel shows the CFNN approximation at the training data set at each training stage (from top down), along with the pointwise error of the last stage CFNN approximation (at the bottow) which is at $\mathcal{O}(10^{-14})$.

The numerical results for f_3 are shown in Fig. 4, and f_4 in Fig. 5. In each plot, we show in the left panel the training loss decay, the trained CFNN approximation, along with its pointwise errors; in the right panel, we show the CFNN approximation after each stage, along with the pointwise error of the last stage.

While the results of f_3 are similar to those of f_1 and f_2, it is worth noting the results for f_4, which is a fairly oscillatory function (with $m = 30$). We observe that the one-stage approximation (at Training Stage 0) is not very accurate, as shown in the top of the right panel of Fig. 5. The DNN approximation missed a few wavefronts completely. We remark that this is quite common for the standard DNN function approximation. With the multistage learning, the

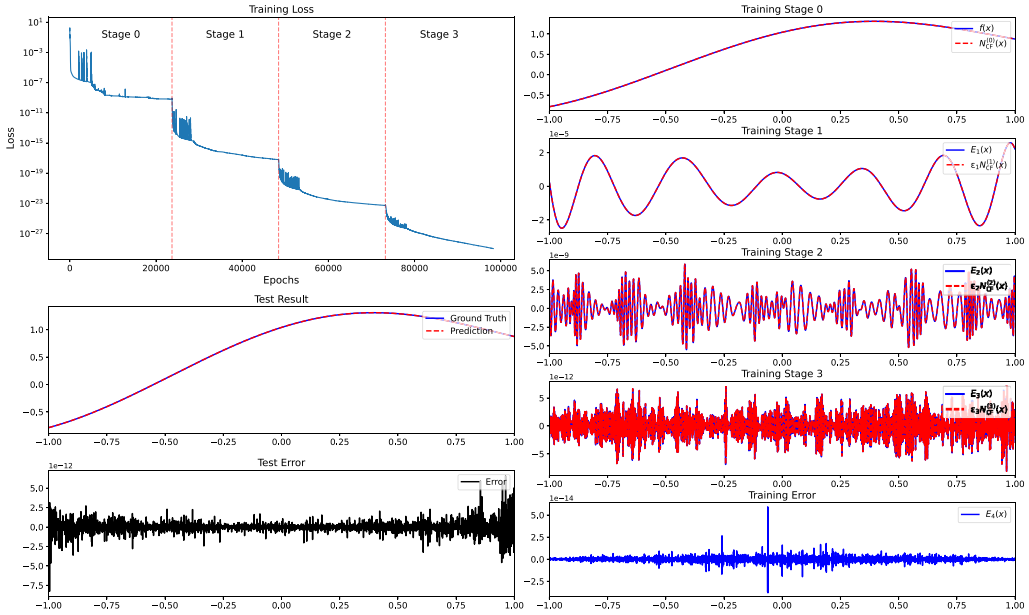

FIG. 3: Results of $f_2(x)$. Left top: training loss history across all stages; Left bottom: test data ground truth vs. prediction and prediction error; Right: training results at four stages showing the network output (red dashed) against the ground truth/residual (blue solid) at each stage.

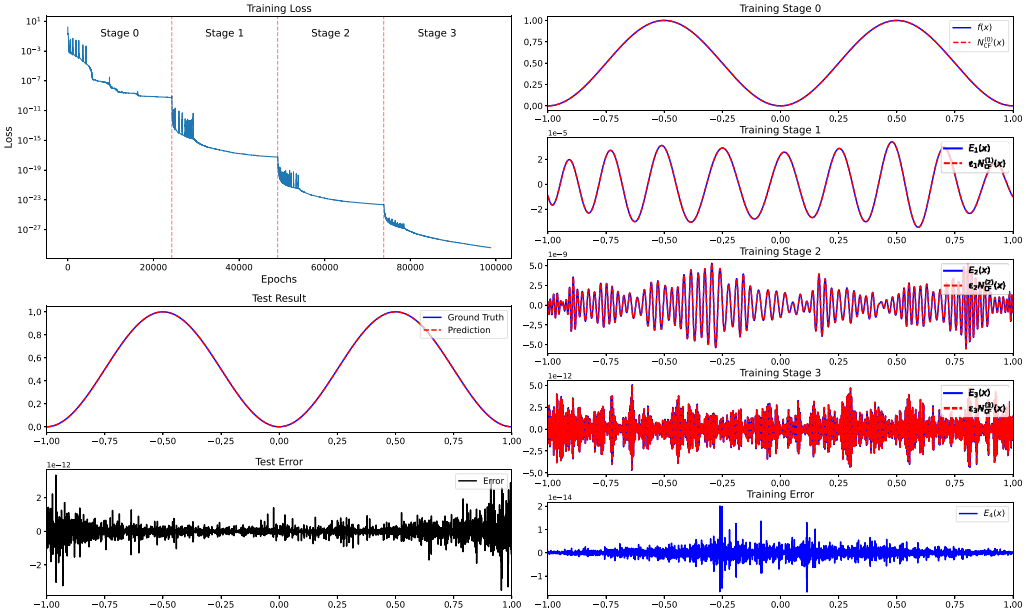

FIG. 4: Results of $f_3(x)$. Left top: training loss history across all stages; Left bottom: test data ground truth vs. prediction and prediction error; Right: training results at four stages showing the network output (red dashed) against the ground truth/residual (blue solid) at each stage.

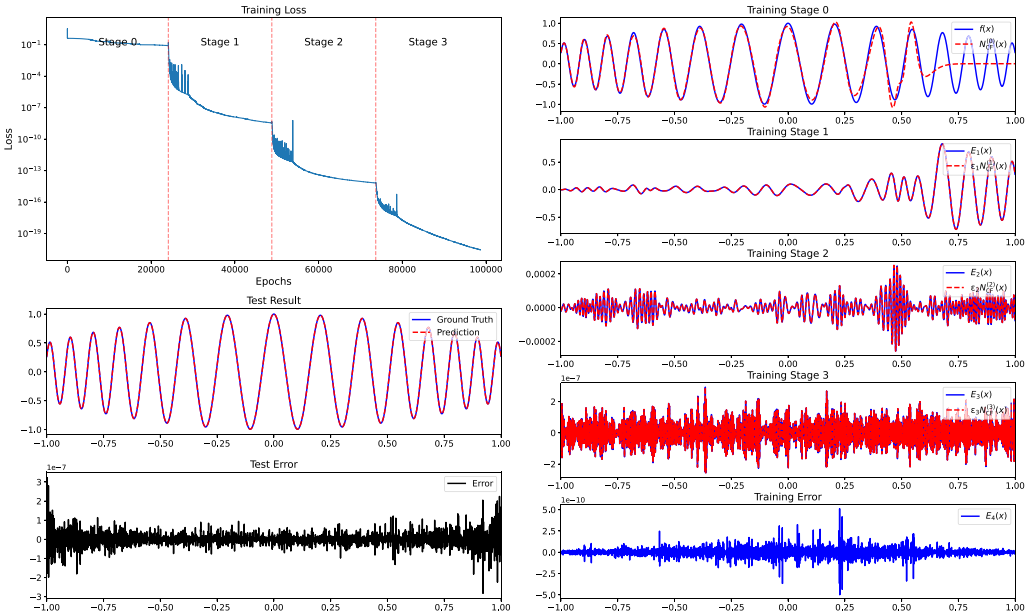

FIG. 5: Results of $f_4(x)$. Left top: training loss history across all stages; Left bottom: test data ground truth vs. prediction and prediction error; Right: training results at four stages showing the network output (red dashed) against the ground truth/residual (blue solid) at each stage.

DNN approximation becomes increasingly accurate. After four stages of training, the pointwise approximation error is at $\mathcal{O}(10^{-10})$ over the training data set. (See the bottom of the right panel of Fig. 5.) The pointwise approximation error over the testing data set is of $\mathcal{O}(10^{-7})$. (See the bottom of the left panel of Fig. 5.) This error is higher than the errors in the previous examples. However, it is not unexpected because of the oscillatory nature of the function.

4.1.2 Ablation Study

To further examine the proposed CFNN structure, we perform an ablation study to the relatively more challenging function f_4. The objective is to understand the roles of the components in the CFNN structure. In particular, we consider the following cases:

Case 1. In the first layer of CFNN, change its activation function from the Chebyshev function (5) to the standard $\tanh(\cdot)$ activation function, which is initialized by the standard Xavier initialization. By doing so, the network becomes a standard feedforward DNN with multistage training. This case shall examine the effect of the Chebyshev function used in the first layer of CFNN.

Case 2. Since CFNN at each stage does not share parameters, one may consider the multistage CFNN a much larger DNN. To examine this, we employ a CFNN with 160 nodes per layer, so that this CFNN has roughly the same number of hyperparameters as the one used in Fig. 5. We also conduct a standard one-stage training for up to 10,000 epochs. [The Chebyshev feature layer is initialized with $W_{\mathtt{CF}} \sim \mathrm{Exp}(5^{-3})$.] This case shall examine the impact of multistage training in CFNN.

The results of the two ablation cases are shown in Fig. 6, where Case 1 is on the left and Case 2 is on the right. Compared with the original four-stage CFNN (Fig. 5), these two cases

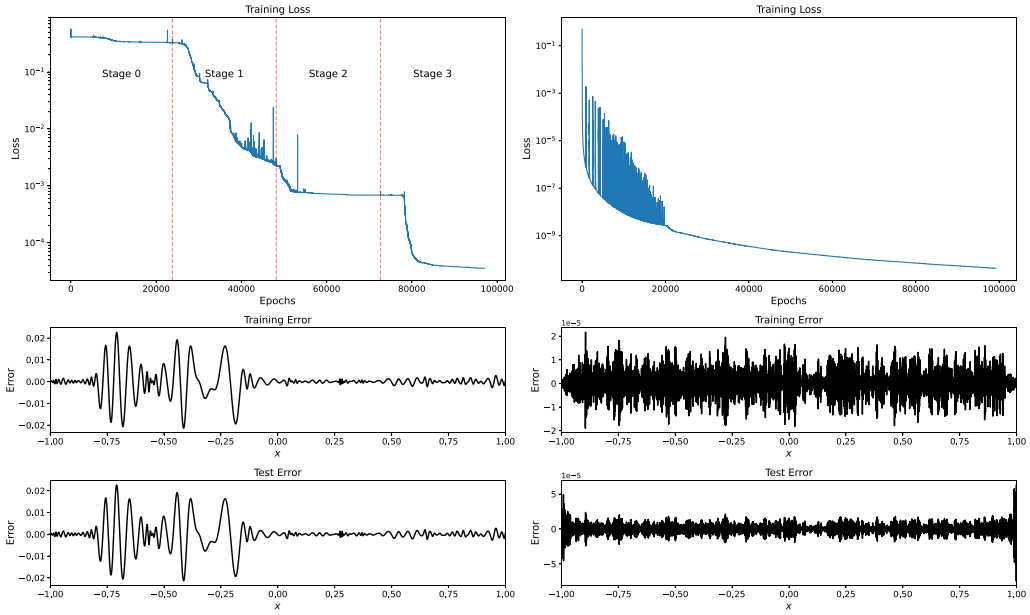

FIG. 6: Results of ablation tests. Left: Case 1; Right: Case 2. From top to bottom, training error decay; pointwise approximation error at the training data; pointwise error at the testing data.

demonstrate notable performance degradation. In particular, Case 1 can only achieve a pointwise approximation error of $\mathcal{O}(10^{-2})$, whereas for Case 2 it is of $\mathcal{O}(10^{-5})$. Compared to the $\mathcal{O}(10^{-10})$ error achieved by the proposed multistage CFNN, we conclude that both the Chebyshev function layer and the multistage training are essential in achieving high accuracy, with the Chebyshev function layer playing perhaps a relatively larger role.

4.1.3 Nonsmooth Functions

We now consider the two nonsmooth functions in Eq. (9), the C^0 function f_5 and the discontinuous function f_6. Both are approximated by four-stage CFNN. The results for f_5 are shown in Fig. 7, and the results for f_6 in Fig. 8. For f_5, the training loss decays to $\mathcal{O}(10^{-17})$ and the training error is at $\mathcal{O}(10^{-8})$. For the discontinuous function f_6, the training loss decays to $\mathcal{O}(10^{-18})$ and the training error is at $\mathcal{O}(10^{-9})$. These errors are larger than the machine accuracy errors in the previous examples for smooth functions. This is expected as both functions are nonsmooth at $x = 0$. On the other hand, achieving approximation error at $\mathcal{O}(10^{-8})$ for discontinuous functions is quite remarkable, as most function approximation methods cannot achieve this level of accuracy without special treatment of the singularity.

4.2 Multidimensional Examples

For high-dimensional examples, we consider the functions in Eq. (10) in $[-1, 1]^d$ with various values of $d > 1$. We use the proposed CFNN with four hidden layers and 40 neurons per layer

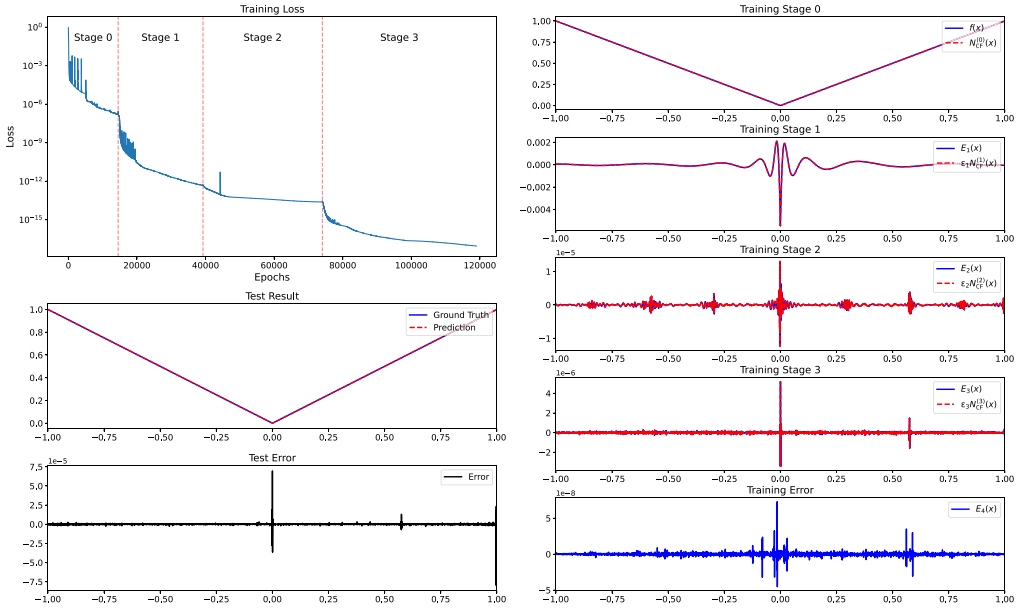

FIG. 7: Results of $f_5(x)$. Left top: training loss history across all stages; Left bottom: test data ground truth vs. prediction and prediction error; Right: training results at four stages showing the network output (red dashed) against the ground truth/residual (blue solid) at each stage.

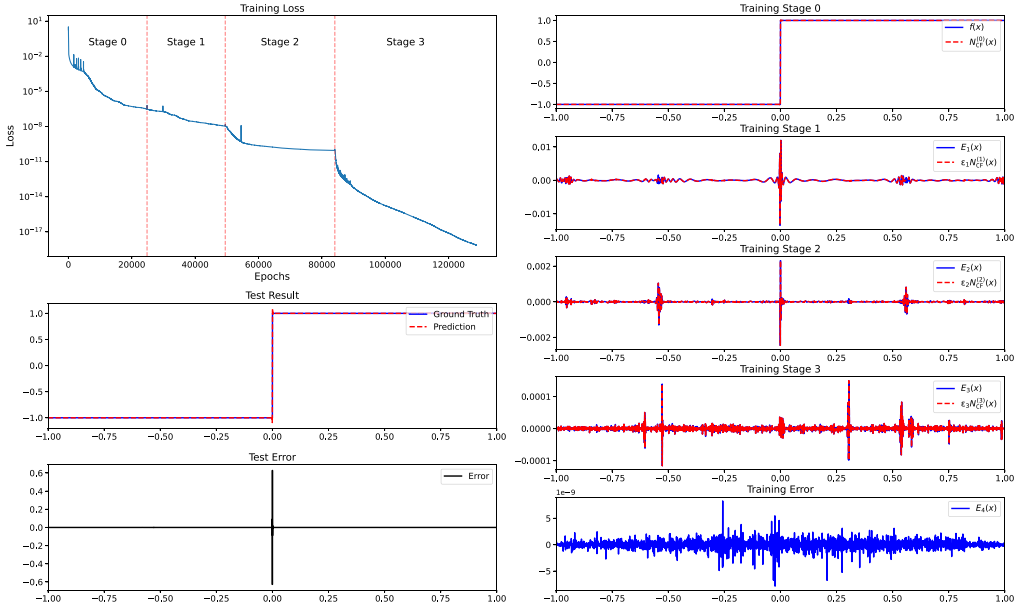

FIG. 8: Results of $f_6(x)$. Left top: training loss history across all stages; Left bottom: test data ground truth vs. prediction and prediction error; Right: training results at four stages showing the network output (red dashed) against the ground truth/residual (blue solid) at each stage.

(including the first Chebyshev feature layer). The CFNN training is conducted over a set of 20,000 uniformly distributed random points. For validation, the approximation errors are computed over an independent set of 10,000 uniformly distributed random points. Since it is difficult to visualize high-dimensional functions, hereafter we will only report RMSE (root mean squared error) in each case, where "training error" refers to RMSE over the training data points and "testing error" refers to RMSE over the testing data points.

Numerical tests are performed in dimensions $d = 2, 5, 10, 20$, and with learning stages as high as $s = 20$. In Fig. 9, we show both the relative training errors and relative validation errors at each stage for f_7 in Eq. (10), at dimension $d = 2$ and $d = 20$. We observe exponential decay of the training errors with respect to the increase of training stages. The validation errors quickly saturate. This is because the validation errors are computed on a set of fixed, albeit randomly generated, points, whose geometrical property determines the achievable approximation accuracy.

Very similar approximation behaviors are observed for all three examples in Eq. (10). Therefore, we opt not to display all the plots. Instead, we tabulate the relative training errors and relative validation errors for f_7 in Table 1, f_8 in Table 2, and f_9 in Table 3.

5. CONCLUSION

In this paper, we presented the CFNN, as a novel DNN structure for accurate function approximation. CFNN utilizes Chebyshev functions with learnable frequency in the first hidden layer. Combined with multistage learning and a proper initialization procedure, CFNN is capable of

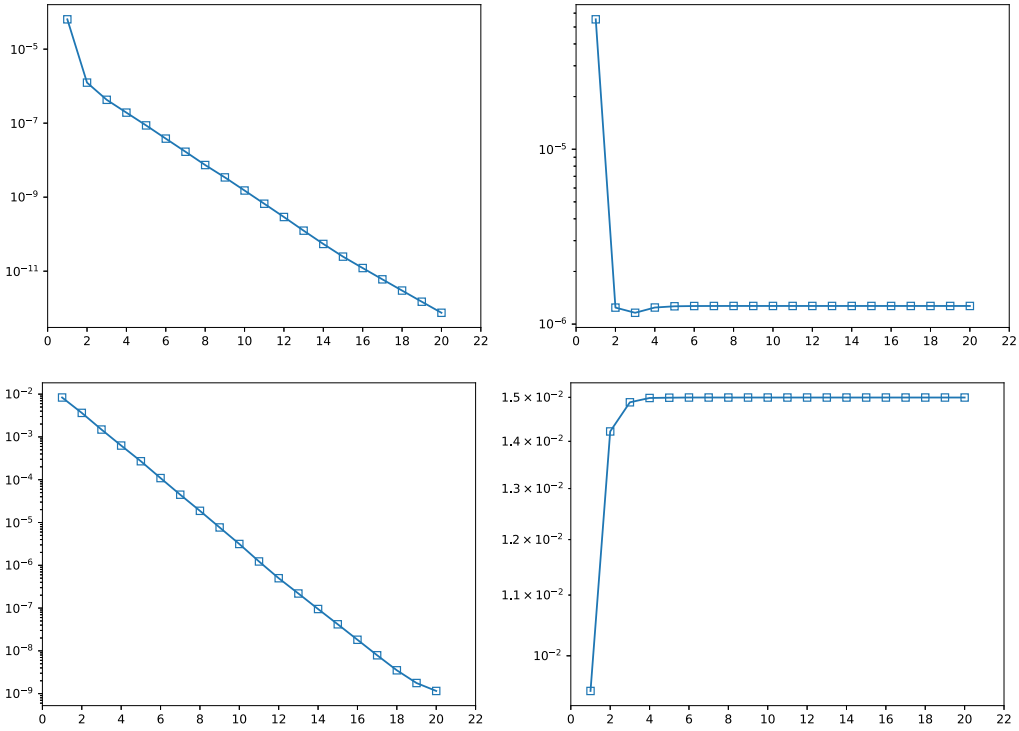

FIG. 9: The relative training and testing errors vs. stages, for f_7 in Eq. (10). Upper left: training error for $d = 2$; Upper right: testing error for $d = 2$; Lower left: training error for $d = 20$; Lower right: testing error for $d = 20$.

TABLE 1: Summary of Errors for f_7 in Eq. (10)

d	Error	Stage 1	Stage 4	Stage 8	Stage 12	Stage 16	Stage 20
2	Training	6.3941E-05	1.9261E-07	7.3827E-09	2.9078E-10	1.2144E-11	7.5117E-13
	Testing	5.5311E-05	1.2439E-06	1.2699E-06	1.2698E-06	1.2698E-06	1.2698E-06
5	Training	3.9762E-04	2.0109E-05	7.8913E-07	3.0380E-08	1.4001E-09	7.6904E-11
	Testing	3.3952E-04	2.6595E-04	2.6726E-04	2.6724E-04	2.6724E-04	2.6724E-04
10	Training	2.5685E-03	2.3697E-04	8.3021E-06	2.9867E-07	1.3257E-08	7.6304E-10
	Testing	2.4612E-03	3.9598E-03	3.9698E-03	3.9698E-03	3.9698E-03	3.9698E-03
20	Training	8.3437E-03	6.2842E-04	1.8745E-05	4.9859E-07	1.8172E-08	1.1561E-09
	Testing	9.4638E-03	1.4985E-02	1.4996E-02	1.4996E-02	1.4996E-02	1.4996E-02

achieving machine accuracy for function approximation, which is a critical component for scientific machine learning.

ACKNOWLEDGMENT

This work was partially supported by AFOSR FA9550-24-1-0237.

TABLE 2: Summary of Errors for f_8 in Eq. (10)

d	Error	Stage 1	Stage 4	Stage 8	Stage 12	Stage 16	Stage 20
2	Training	1.2234E-04	6.1596E-07	2.3968E-08	8.5897E-10	4.0117E-11	2.4655E-12
	Testing	1.0396E-04	4.4164E-06	4.4850E-06	4.4853E-06	4.4853E-06	4.4853E-06
5	Training	9.9211E-04	8.3535E-05	3.2130E-06	1.1872E-07	5.7236E-09	3.2697E-10
	Testing	8.7179E-04	1.1540E-03	1.1574E-03	1.1574E-03	1.1574E-03	1.1574E-03
10	Training	3.6173E-03	3.1649E-04	1.1602E-05	4.3113E-07	1.9065E-08	1.0722E-09
	Testing	7.4386E-03	8.8001E-03	8.8122E-03	8.8124E-03	8.8124E-03	8.8124E-03
20	Training	2.3357E-02	1.9097E-03	6.0061E-05	1.7198E-06	5.9873E-08	3.8842E-09
	Testing	1.3465E-01	1.3750E-01	1.3754E-01	1.3754E-01	1.3754E-01	1.3754E-01

TABLE 3: Summary of Errors for f_9 in Eq. (10)

d	Error	Stage 1	Stage 4	Stage 8	Stage 12	Stage 16	Stage 20
2	Training	1.3982E-04	8.4385E-07	3.2461E-08	1.2081E-09	5.6564E-11	3.5100E-12
	Testing	1.1950E-04	5.8842E-06	5.9729E-06	5.9728E-06	5.9728E-06	5.9728E-06
5	Training	3.1952E-03	3.5137E-04	1.2766E-05	4.5370E-07	2.1068E-08	1.1673E-09
	Testing	3.1045E-03	4.4888E-03	4.5006E-03	4.5005E-03	4.5006E-03	4.5006E-03
10	Training	2.6111E-02	2.3253E-03	8.4740E-05	2.9449E-06	1.2474E-07	7.0201E-09
	Testing	4.0942E-02	5.1587E-02	5.1669E-02	5.1670E-02	5.1670E-02	5.1670E-02
20	Training	7.2815E-02	5.6877E-03	1.6364E-04	4.5827E-06	1.6399E-07	1.0466E-08
	Testing	6.0428E-01	6.1028E-01	6.1019E-01	6.1019E-01	6.1019E-01	6.1019E-01

REFERENCES

Brunton, S.L., Proctor, J.L., and Kutz, J.N., Discovering Governing Equations from Data by Sparse Identification of Nonlinear Dynamical Systems, *Proc. Natl. Acad. Sci.*, vol. **113**, no. 15, pp. 3932–3937, 2016.

Chowdhary, K.R., Natural Language Processing, in *Fundamentals of Artificial Intelligence*, Berlin: Springer, pp. 603–649, 2020.

Fanaskov, V.S. and Oseledets, I.V., Spectral Neural Operators, *Dokl. Math.*, vol. **108**, pp. S226–S232, 2023.

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

Hornik, K., Stinchcombe, M., and White, H., Multilayer Feedforward Networks Are Universal Approximators, *Neural Networks*, vol. **2**, no. 5, pp. 359–366, 1989.

James, G., Witten, D., Hastie, T., and Tibshirani, R., *An Introduction to Statistical Learning*, Vol. 112, Berlin: Springer, 2013.

Jin, K.H., McCann, M.T., Froustey, E., and Unser, M., Deep Convolutional Neural Network for Inverse Problems in Imaging, *IEEE Trans. Image Process.*, vol. **26**, no. 9, pp. 4509–4522, 2017.

Li, H., Schwab, J., Antholzer, S., and Haltmeier, M., NETT: Solving Inverse Problems with Deep Neural Networks, *Inv. Prob.*, vol. **36**, no. 6, p. 065005, 2020a.

Li, L., Wang, L.G., and Teixeira, F.L., Performance Analysis and Dynamic Evolution of Deep Convolutional Neural Network for Electromagnetic Inverse Scattering, *IEEE Antennas and Wireless Propag. Lett.*, vol. **18**, no. 11, pp. 2259–2263, 2019.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A., Fourier Neural Operator for Parametric Partial Differential Equations, arXiv Preprint arXiv:2010.08895, 2020b.

Lu, L., Jin, P., Pang, G., Zhang, Z., and Karniadakis, G.E., Learning Nonlinear Operators via DeepONet Based on the Universal Approximation Theorem of Operators, *Nat. Mach. Intell.*, vol. **3**, no. 3, pp. 218–229, 2021.

Pazzani, M.J. and Billsus, D., Content-Based Recommendation Systems, *The Adaptive Web: Methods and Strategies of Web Personalization*, Berlin: Springer, pp. 325–341, 2007.

Qin, T., Wu, K., and Xiu, D., Data Driven Governing Equations Approximation Using Deep Neural Networks, *J. Comput. Phys.*, vol. **395**, pp. 620–635, 2019.

Quarteroni, A., Sacco, R., and Saleri, F., *Numerical Mathematics*, Vol. 37, New York: Springer Science & Business Media, 2006.

Rahaman, N., Baratin, A., Arpit, D., Draxler, F., Lin, M., Hamprecht, F., Bengio, Y., and Courville, A., On the Spectral Bias of Neural Networks, in *Int. Conf. on Machine Learning*, Long Beach, CA, pp. 5301–5310, 2019.

Raissi, M., Perdikaris, P., and Karniadakis, G.E., Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations, *J. Comput. Phys.*, vol. **378**, pp. 686–707, 2019.

Sirignano, J. and Spiliopoulos, K., DGM: A Deep Learning Algorithm for Solving Partial Differential Equations, *J. Comput. Phys.*, vol. **375**, pp. 1339–1364, 2018.

Szego, G., *Orthogonal Polynomials*, Providence, RI: American Mathematical Society, 1939.

Tancik, M., Srinivasan, P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron, J., and Ng, R., Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains, *Adv. Neural Inf. Process. Syst.*, vol. **33**, pp. 7537–7547, 2020.

Trefethen, L.N., *Approximation Theory and Approximation Practice*, Philadelphia: SIAM, 2013.

Voulodimos, A., Doulamis, N., Doulamis, A., and Protopapadakis, E., Deep Learning for Computer Vision: A Brief Review, *Comput. Intell. Neurosci.*, vol. **2018**, 2018. DOI: 10.1155/2018/7068349

Wang, Y. and Lai, C.Y., Multi-Stage Neural Networks: Function Approximator of Machine Precision, *J. Comput. Phys.*, vol. **504**, p. 112865, 2024.

Xu, Z.Q.J., Zhang, Y., Luo, T., Xiao, Y., and Ma, Z., Frequency Principle: Fourier Analysis Sheds Light on Deep Neural Networks, arXiv Preprint arXiv:1901.06523, 2019.

Yu, B., The Deep Ritz Method: A Deep Learning-Based Numerical Algorithm for Solving Variational Problems, *Commun. Math. Stat.*, vol. **6**, no. 1, pp. 1–12, 2018.