COMMUN. MATH. SCI.
Vol. 22, No. 2, pp. 285–314

REPRODUCING ACTIVATION FUNCTION FOR DEEP LEARNING*

SENWEI LIANG†, LIYAO LYU‡, CHUNMEI WANG§, AND HAIZHAO YANG¶

Abstract. We propose reproducing activation functions (RAFs) motivated by applied and computational harmonic analysis to improve deep learning accuracy for various applications ranging from computer vision to scientific computing. The idea is to employ several basic functions and their learnable linear combination to construct neuron-wise data-driven activation functions for each neuron. Armed with RAFs, neural networks (NNs) can reproduce traditional approximation tools and, therefore, approximate target functions with a smaller number of parameters than traditional NNs. As demonstrated by extensive numerical tests, the proposed RAFs can facilitate the convergence of deep learning optimization for a solution with higher accuracy than existing deep learning solvers for audio/image/video reconstruction, PDEs, and eigenvalue problems. With RAFs, the errors of audio/video reconstruction, PDEs, and eigenvalue problems are decreased by over 14%, 73%, 99%, respectively, compared with baseline, while the performance of image reconstruction increases by 58%. Numerically, in the NN training, RAFs can generate neural tangent kernels with better condition numbers than traditional activation functions, which provides a prospective for understanding the improved optimization convergence using the theory of neural tangent kernel. The code is available at https://github.com/LeungSamWai/Reproducing-Activation-Function.

Keywords. Deep Neural Network; Activation Function; Neural Tangent Kernel; Polynomials; Fourier Basis; Wavelets; Radial Basis Functions.

AMS subject classifications. 65M75; 65N75; 62M45.

1. Introduction

High-dimensional problems are ubiquitous in science and engineering. Deep learning has been an important tool for solving a wide range of high-dimensional problems with surprising performance. For example, neural network-based optimization has become a powerful tool for solving high-dimensional and nonlinear differential equations in complicated domains [3,24,34,35,44,61,80]. First of all, as a form of function approximation via the compositions of nonlinear functions [21], deep neural networks (DNNs) as a mesh-free parametrization can efficiently approximate various high-dimensional solutions lessening the curse of dimensionality [2, 17, 18, 29, 45, 52–54, 59, 65, 78] and/or achieving exponential approximation rates [16, 41, 45, 54, 57, 65, 76]. Second, DNN parameters are identified via energy minimization from variational formulation, which usually enjoys a summation form that can be accelerated by the stochastic gradient descent (SGD) for a local minimizer. It is believed that the implicit regularization of SGD and DNNs helps to obtain approximate solutions to a certain class of nonlinear PDEs, though current optimization analysis [47] and generalization analysis [4, 25, 47, 66] are limited to the case of linear PDEs.

Though neural network-based optimization for solving PDEs admits attractive properties mentioned above, it is also well known that the optimization problem is highly non-convex and hence challenging to solve for a highly accurate solution. There has

*Received: June 15, 2022; Accepted (in revised form): June 19, 2023. Communicated by Jianfeng Lu.

†Equal contribution. Department of Mathematics, Purdue University, West Lafayette, IN 47907, USA (liang339@purdue.edu).

‡Equal contribution. Department of Computational Mathematics, Science, and Engineering, Michigan State University, East Lansing, MI 48824, USA (lyuliyao@msu.edu).

§Department of Mathematics, University of Florida, 1400 Stadium Rd, Gainesville, FL 32611, USA (chunmei.wang@ufl.edu).

¶Corresponding author. Department of Mathematics, University of Maryland College Park, 4176 Campus Drive, College Park, MD 20742, USA (hzyang@umd.edu).

been extensive research on improving the accuracy of the PDE solution provided by neural network-based optimization. They include but are not limited to the following examples. Building neural networks satisfying the initial/boundary conditions of the PDE can simplify the optimization formulation and increase the accuracy [23, 38, 48]. Applying first-order methods to reformulate high-order PDEs can reduce the difficulty of neural network optimization [7, 48]. Improving the sampling strategy of SGD [9, 55] or the sample weights in the objective function [22] can facilitate the convergence of neural network-based optimization. Building special neural network structures or neural network solutions according to solution ansatz inspired by physical knowledge can significantly alleviate the training difficulty of neural network optimization, e.g., using oscillatory structures [6], multiscale structures [44], and other spectral structures [23]. Finally, hybrid algorithms combining neural network-based solvers and traditional iterative solvers can provide highly accurate solutions to low-dimensional nonlinear PDEs efficiently [28, 75].

Though high-dimensional problems are the main application domains of deep learning, it was also observed that deep learning can outperform traditional computational tools in low-dimensional problems. Recently in computer vision and graphics, deep neural networks were used as a mesh-free representation of objects, scene geometry, and appearance, resulting in notable performance compared to traditional discrete representations. These deep neural networks are called "coordinate-based" networks in [71] because they take low-dimensional coordinates as inputs and output an object value of the shape, density, and/or color at the given input coordinate. This strategy is compelling in data compression and reconstruction, e.g., see [10, 20, 33, 43, 50, 58, 62, 68]. Similarly to the case of high-dimensional applications, obtaining high accuracy in these applications is also a challenging topic. Exploring different neural network architectures and training strategies for highly accurate solutions to these problems has been an active research direction.

In this paper, we propose the reproducing activation function (RAF) to improve deep learning accuracy in the above applications. The idea of reproducing activation functions is to employ several basic functions and their learnable linear combination to construct neuron-wise data-driven activation functions for each neuron. Armed with such activation functions, deep neural networks can reproduce traditional approximation tools efficiently, e.g., orthogonal polynomials, Fourier basis functions, wavelets, and radial basis functions. Therefore, deep neural networks with the proposed reproducing activation function can approximate a wide class of target functions with a smaller number of parameters than traditional neural networks (e.g., networks with ReLU activation functions). Therefore, these basic functions are referred to as basic activation functions, and the data-driven activation functions are called reproducing activation functions. The proposed reproducing activation function is a general concept including many existing network structures with super approximation power, e.g., the Sine-ReLU networks [78], the Floor-ReLU networks [65], the Floor-Exponential-Sign networks [64].

In terms of training dynamics of deep learning, reproducing activation functions can empirically generate neural tangent kernels with a better condition number than traditional activation functions lessening the spectrum bias of deep learning. Neural network-based optimization usually can find a smooth solution with the fast decay in the frequency domain due to the implicit regularization of network structures and the stochastic gradient descent (SGD) for solving the minimization problem. It was shown through the frequency principle of neural networks [75] and the neural tangent kernel [8] that neural networks have an implicit bias towards functions that decay fast in the

Fourier domain and the gradient descent method tends to fit a low-frequency function better than a high-frequency function. Therefore, designing an efficient deep learning algorithm to identify oscillatory or singular solutions to regression and PDE problems is challenging. The proposed reproducing activation function is a general concept that also includes many existing network structures lessening the spectral bias of deep learning (e.g., the multiscale neural network [6], deep neural networks composed with Fourier feature models [71]).

The incorporation of trainable parameters into basic activation functions is widely used in various applications. These trainable parameters are applied as either coefficients for linear combinations of activation functions [49, 60, 70] or scaling parameters for input [31,32], while our RAF concept utilizes both. Previous works use basic activation functions that are proven to be effective in the corresponding applications, such as ReLU, ELU, SELU [70] in computer vision and tanh in scientific computing. Inspired by applied harmonic analysis, we investigate basic functions like polynomial and Gaussian functions and showcase their effectiveness in RAF through both theory and numerical examples. We also summarize a comparison of RAF with other methods in Table 1.1.

Method	Linear combination?	Input scaling?	Application	Basic functions
Qian et al. [60]	✓	✗	Image classification	Leaky-ReLU, Elu
Manessi et al. [49]	✓	✗	Image classification	Identity, ReLU, Tanh
Sutfeld et al. [70]	✓	✗	Image classification	ReLU, ELU, SELU, Identity and Swish
Adaptive activation function [31,32]	✗	✓	PDE, Function approximation	Tanh
Reproducing activation function (ours)	✓	✓	PDE, Function approximation, Coordinate-based regression	Polynomial, Sine, Gaussian

Table 1.1: *Comparison of our reproducing activation function with related methods.*

Contribution. We summarize our contribution as follows.

(1) We propose RAFs and their approximation theory. NNs with RAFs can reproduce traditional approximation tools (e.g., polynomials, Fourier basis functions, wavelets, radial basis functions) and approximate a certain class of functions with exponential and dimension-independent approximation rates.

(2) Empirically, RAFs can generate neural tangent kernels with smaller condition numbers than traditional activation functions, which provides a prospective for understanding the improved optimization convergence using the theory of neural tangent kernel.

(3) Extensive experiments on coordinate-based data representation and PDEs demonstrate the effectiveness of the proposed activation function.

The paper is organized as follows. In Section 2, preliminary knowledge will be introduced. In Section 3, we introduce the proposed activation function. Numerical results will be presented in Section 4 to demonstrate the effectiveness. Finally, we conclude this paper in Section 5.

2. Preliminaries

In this section, we will introduce deep neural networks and their applications in regression problems and solving PDEs.

2.1. Deep neural networks. Mathematically, DNNs are a form of highly non-linear function parametrization via function compositions using simple non-linear functions [21]. The justification of this kind of approximation is given by the universal approximation theorems of DNNs in [2, 37, 76, 77] with newly developed quantitative and explicit error characterization [45,63,65], which shows those function compositions

are more powerful than other traditional approximation tools. There are two popular neural network structures used in deep learning-based PDE solvers.

The first one is the fully connected feed-forward neural network (FNN), which is the composition of L simple nonlinear functions as follows:

$$\phi(\boldsymbol{x};\boldsymbol{\theta}) := \boldsymbol{a}^T \boldsymbol{h}_L \circ \boldsymbol{h}_{L-1} \circ \cdots \circ \boldsymbol{h}_1(\boldsymbol{x}), \tag{2.1}$$

where $\boldsymbol{h}_\ell(\boldsymbol{x}) = \sigma(\boldsymbol{W}_\ell \boldsymbol{x} + \boldsymbol{b}_\ell)$ with $\boldsymbol{W}_\ell \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$, $\boldsymbol{b}_\ell \in \mathbb{R}^{N_\ell}$ for $\ell = 1, \ldots, L$, $\boldsymbol{a} \in \mathbb{R}^{N_L}$, σ is a non-linear activation function, e.g., a rectified linear unit (ReLU) $\sigma(x) = \max\{x, 0\}$ or hyperbolic tangent function $\tanh(x)$. Each $\boldsymbol{h}_\ell$ is referred to as a hidden layer, N_ℓ is the width of the ℓ-th layer, and L is called the depth of the FNN. In the above formulation, $\boldsymbol{\theta} := \{\boldsymbol{a}, \boldsymbol{W}_\ell, \boldsymbol{b}_\ell : 1 \le \ell \le L\}$ denotes the set of all parameters in ϕ, which uniquely determines the underlying neural network.

Another popular network is the residual neural network (ResNet) introduced in [27]. We present its variant defined recursively as follows:

$$\begin{aligned} &\boldsymbol{h}_0 = \boldsymbol{V}\boldsymbol{x}, \boldsymbol{g}_\ell = \sigma(\boldsymbol{W}_\ell \boldsymbol{h}_{\ell-1} + \boldsymbol{b}_\ell), \quad \ell = 1, 2, \ldots, L, \\ &\boldsymbol{h}_\ell = \tilde{\boldsymbol{U}}_\ell \boldsymbol{h}_{\ell-2} + \boldsymbol{U}_\ell \boldsymbol{g}_\ell, \quad \ell = 1, 2, \ldots, L, \quad \phi(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{a}^T \boldsymbol{h}_L, \end{aligned} \tag{2.2}$$

where $\boldsymbol{V} \in \mathbb{R}^{N_0 \times d}$, $\boldsymbol{W}_\ell \in \mathbb{R}^{N_\ell \times N_0}$, $\tilde{\boldsymbol{U}}_\ell \in \mathbb{R}^{N_0 \times N_0}$, $\boldsymbol{U}_\ell \in \mathbb{R}^{N_0 \times N_\ell}$, $\boldsymbol{b}_\ell \in \mathbb{R}^{N_\ell}$ for $\ell = 1, \cdots, L$, $\boldsymbol{a} \in \mathbb{R}^{N_0}$, $\boldsymbol{h}_{-1} = 0$. Throughout this paper, we consider $N_0 = N_\ell = N$ and $\boldsymbol{U}_\ell$ is set as the identity matrix in the numerical implementation of ResNets for the purpose of simplicity. Furthermore, as used in [19], we set $\tilde{\boldsymbol{U}}_\ell$ as the identity matrix when ℓ is even and set $\tilde{\boldsymbol{U}}_\ell = 0$ when ℓ is odd.

2.2. Deep learning for regression problems. Regression problems aim at identifying an unknown target function $f : \mathbf{x} \in \Omega \to y \in \mathbb{R}$ from training samples $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, where $\mathbf{x}_i$'s are usually assumed to be i.i.d samples from an underlying distribution π defined on a domain $\Omega \subseteq \mathbb{R}^n$, and $y_i = f(\mathbf{x}_i)$ (probably with an additive noise). Consider the square loss $\ell(\mathbf{x}, y; \boldsymbol{\theta}) = |\phi(\mathbf{x};\boldsymbol{\theta}) - y|^2$ of a given DNN $\phi(\mathbf{x};\boldsymbol{\theta})$ that is used to approximate $f(\mathbf{x})$, the population risk (error) and empirical risk (error) functions are respectively

$$\mathcal{J}(\boldsymbol{\theta}) = \frac{1}{2}\mathbb{E}_{\mathbf{x}\sim\pi}\left[|\phi(\mathbf{x};\boldsymbol{\theta}) - f(\mathbf{x})|^2\right], \quad \hat{\mathcal{J}}(\boldsymbol{\theta}) = \frac{1}{2N}\sum_{i=1}^{N} |\phi(\mathbf{x}_i;\boldsymbol{\theta}) - y_i|^2, \tag{2.3}$$

which are also functions that depend on the depth L and width N_ℓ of ϕ implicitly. The optimal set $\hat{\boldsymbol{\theta}}$ is identified via $\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \hat{\mathcal{J}}(\boldsymbol{\theta})$, and $\phi(\cdot;\hat{\boldsymbol{\theta}}) : \Omega \to \mathbb{R}$ is the learned DNN that approximates the unknown function f.

2.3. Deep learning for solving PDEs. Deep learning can be applied to solve various PDEs including the initial value problems and boundary value problems (BVP) based on different variational formulations [13,19,38,42]. In this paper, we will take the example of BVP and the least squares method (LSM) [13,38] without loss of generality. The generalization to other problems and methods is similar. Consider the BVP

$$\mathcal{D}u(\boldsymbol{x}) = f(u(\boldsymbol{x}), \boldsymbol{x}), \text{ in } \Omega, \qquad \mathcal{B}u(\boldsymbol{x}) = g(\boldsymbol{x}), \text{ on } \partial\Omega, \tag{2.4}$$

where $\mathcal{D} : \Omega \to \Omega$ is a differential operator that can be nonlinear, $f(u(\boldsymbol{x}), \boldsymbol{x})$ can be a nonlinear function in u, Ω is a bounded domain in $\mathbb{R}^d$, and $\mathcal{B}u = g$ characterizes the boundary condition. Other types of problems like initial value problems can also be

formulated as a BVP as discussed in [22]. Then LSM seeks a solution $u(\boldsymbol{x};\boldsymbol{\theta})$ as a neural network with a parameter set $\boldsymbol{\theta}$ via the following optimization problem

$$\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) := \|\mathcal{D}u(\boldsymbol{x};\boldsymbol{\theta}) - f(u,\boldsymbol{x})\|^2_{L^2(\Omega)} + \lambda \|\mathcal{B}u(\boldsymbol{x};\boldsymbol{\theta}) - g(\boldsymbol{x})\|^2_{L^2(\partial\Omega)}, \tag{2.5}$$

where $\mathcal{L}$ is the loss function consisting of the L^2-norm of the PDE residual $\mathcal{D}u(\boldsymbol{x};\boldsymbol{\theta}) - f(u,\boldsymbol{x})$ and the boundary residual $\mathcal{B}u(\boldsymbol{x};\boldsymbol{\theta}) - g(\boldsymbol{x})$, and $\lambda > 0$ is a regularization parameter.

The goal of (2.5) is to find an appropriate set of parameters $\boldsymbol{\theta}$ such that the DNN $u(\boldsymbol{x};\boldsymbol{\theta})$ minimizes the loss $\mathcal{L}(\boldsymbol{\theta})$. If the loss $\mathcal{L}(\boldsymbol{\theta})$ is minimized to zero with some $\boldsymbol{\theta}$, then $u(\boldsymbol{x};\boldsymbol{\theta})$ satisfies $\mathcal{D}u(\boldsymbol{x};\boldsymbol{\theta}) - f(\boldsymbol{x}) = 0$ in Ω and $\mathcal{B}u(\boldsymbol{x};\boldsymbol{\theta}) - g(\boldsymbol{x}) = 0$ on $\partial\Omega$, implying that $u(\boldsymbol{x};\boldsymbol{\theta})$ is exactly a solution of (2.4). If $\mathcal{L}$ is minimized to a nonzero but small positive number, $u(\boldsymbol{x};\boldsymbol{\theta})$ is close to the true solution as long as (2.4) is well-posed (e.g. the elliptic PDE with Neumann boundary condition, see Theorem 4.1 in [22]).

In the implementation of LSM, the minimization problem in (2.5) is solved by SGD or its variants, e.g., Adam [36]. In each iteration of the SGD, a stochastic loss function defined below is minimized instead of the original loss function in (2.5):

$$\min_{\boldsymbol{\theta}} \hat{\mathcal{L}}(\boldsymbol{\theta}) := \frac{1}{2N} \sum_{i=1}^{N} \big(\mathcal{D}u(\boldsymbol{x}_i;\boldsymbol{\theta}) - f(\boldsymbol{x}_i)\big)^2 + \frac{\lambda}{2M} \sum_{j=1}^{M} \big(\mathcal{B}u(\tilde{\boldsymbol{x}}_j;\boldsymbol{\theta}) - g(\tilde{\boldsymbol{x}}_j)\big)^2, \tag{2.6}$$

where $\{\boldsymbol{x}_i\}_{i=1}^N$ are N uniformly sampled random points in Ω and $\{\tilde{\boldsymbol{x}}_j\}_{j=1}^M$ are M uniformly sampled random points on $\partial\Omega$. These random samples will be renewed in each iteration. Throughout this paper, we will use Adam, which is a variant of SGD based on momentum, to solve the neural network-based optimization.

To facilitate the optimization convergence to the desired PDE solution, special network structures can be proposed such that the DNN can satisfy common boundary conditions, which can simplify the loss function in (2.5) to $\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) := \|\mathcal{D}u(\boldsymbol{x};\boldsymbol{\theta}) - f(u,\boldsymbol{x})\|^2_{L^2(\Omega)}$, since $\mathcal{B}u(\boldsymbol{x};\boldsymbol{\theta}) = g(\boldsymbol{x})$ is satisfied by construction. The stochastic loss function is reduced to

$$\min_{\boldsymbol{\theta}} \hat{\mathcal{L}}(\boldsymbol{\theta}) := \frac{1}{2N} \sum_{i=1}^{N} \big(\mathcal{D}u(\boldsymbol{x}_i;\boldsymbol{\theta}) - f(u,\boldsymbol{x}_i)\big)^2. \tag{2.7}$$

In the numerical implementation, the LSM loss function in (2.7) is more attractive because (2.6) heavily relies on the selection of a suitable weight parameter λ and a suitable initial guess. If λ is not appropriate, it may be difficult to identify a reasonably good minimizer of (2.5), as shown by extensive numerical experiments in [23, 38, 48]. However, we would like to remark that it is difficult to build neural networks that automatically satisfy complicated boundary conditions especially when the domain Ω is irregular.

The design of these special neural networks depends on the type of boundary conditions. We will discuss the case of Dirichlet boundary conditions by taking one-dimensional problems defined in the domain $\Omega = [a,b]$ as an example. Network structures for more complicated boundary conditions in high-dimensional domains can be constructed similarly. The reader is referred to [23, 48] for other kinds of boundary conditions.

Suppose $\hat{u}(x;\boldsymbol{\theta})$ is a generic DNN with trainable parameters $\boldsymbol{\theta}$. We will augment $\hat{u}(x;\boldsymbol{\theta})$ with several specially designed functions to obtain a final network $u(x;\boldsymbol{\theta})$

that satisfies $\mathcal{B}u(x;\boldsymbol{\theta})=g(x)$ automatically. For simplicity, let us consider the boundary conditions $u(a)=a_0$ and $u(b)=b_0$. In this case, we can introduce two special functions $h(x)$ and $l(x)$ to augment $\hat{u}(x;\boldsymbol{\theta})$ to obtain the final network $u(x;\boldsymbol{\theta})$ by $u(x;\boldsymbol{\theta})=h(x)\hat{u}(x;\boldsymbol{\theta})+l(x)$. Then $u(x;\boldsymbol{\theta})$ is used to approximate the true solution of the PDE and is trained through (2.7).

A straightforward choice for $l(x)$ is $l(x)=(b_0-a_0)(x-a)/(b-a)+a_0$, and $h(x)$ can be set as $h(x)=(x-a)^{p_a}(x-b)^{p_b}$, with $0<p_a,\ p_b\leq 1$. To obtain an accurate approximation, p_a and p_b should be chosen to be consistent with the orders of a and b of the true solution, and hence no singularity will be brought into the network structure.

2.4. The training behavior of deep learning. The least-squares optimization problems in (2.6) and (2.7) are highly non-convex and hence they are challenging to solve. For regression problems or solving linear PDEs, under the assumption of over-parameterized DNNs (i.e., the width of DNNs is sufficiently large) and appropriate random initialization of DNN parameters, it was shown that the least-squares optimization admits global convergence by gradient descent with a linear convergence rate [11,14,30,47,79]. Though the over-parametrization assumption might not be realistic, it is still a positive sign for the justification of DNNs in these least-squares problems. However, the convergence rate depends on the spectrum of the target function. The training of a randomly initialized DNN has a stronger preference for reducing the fitting error of low-frequency components of a target solution. The high-frequency component of the target function would not be well captured until the low-frequency error has been eliminated. This phenomenon is called the F-principle in [75] and the spectral bias of deep learning in [8]. Related works on the learning behavior of DNNs in the frequency domain can be found in [46,75]. In the case of nonlinear PDEs, these theoretical works imply that deep learning-based solvers would also have a bias towards reducing low-frequency errors [74]. Without the assumption of over-parametrization, to the best of our knowledge, there is no theoretical guarantee that neural network-based PDE solvers can identify the global minimizer via a standard SGD. Through the analysis of the optimization energy landscape of SGD without the over-parameterization, it was shown that SGD with small batches tends to converge to the flattest minimum [12,40,56]. However, such local minimizers might not give the desired PDE solutions. Hence, designing new training techniques to make SGD capable of identifying better minimizers has been an active research field.

2.5. Neural tangent kernel. Neural tangent kernel (NTK) originally introduced in [30] and further investigated in [1,8,39,47,74] is one of the popular tools to study the training behavior of deep learning in regression problems and PDE problems. Let us briefly introduce the main idea of NTK following the linearized model for regression problems in [39] for simplicity. This brief introduction will be sufficient for our purposes, as we aim to leverage the NTK theory to gain insights into the training convergence of NNs with different activation functions.

Let us use $\mathcal{X}$ to denote the set of training sample locations $\{\boldsymbol{x}_i\}_{i=1}^N$ in the empirical loss function $\hat{\mathcal{J}}(\boldsymbol{\theta})$ in (2.3). Let $\mathcal{Y}$ be the set of function values at these sample locations. Using gradient flow to analyze the training dynamics of $\hat{\mathcal{J}}(\boldsymbol{\theta})$, we have the following evolution equations:

$$\dot{\boldsymbol{\theta}}_t=-\nabla_{\boldsymbol{\theta}}\phi_t(\mathcal{X})^T\nabla_{\phi_t(\mathcal{X})}\hat{\mathcal{J}}, \tag{2.8}$$

and

$$\dot{\phi}_t(\mathcal{X})=\nabla_{\boldsymbol{\theta}}\phi_t(\mathcal{X})\dot{\boldsymbol{\theta}}_t=-\hat{\Theta}_t(\mathcal{X},\mathcal{X})\nabla_{\phi_t(\mathcal{X})}\hat{\mathcal{J}}, \tag{2.9}$$

where $\boldsymbol{\theta}_t$ is the parameter set at iteration time t, $\phi_t(\mathcal{X}) = \text{vec}([\phi_t(\boldsymbol{x};\boldsymbol{\theta}_t)]_{\boldsymbol{x}\in\mathcal{X}})$ is the $N\times 1$ vector of concatenated function values for all samples, and $\nabla_{\phi_t(\mathcal{X})}\hat{\mathcal{J}}$ is the gradient of the loss with respect to the network output vector $\phi_t(\mathcal{X})$, $\hat{\Theta}_t := \hat{\Theta}_t(\mathcal{X},\mathcal{X})$ in $\mathbb{R}^{N\times N}$ is the NTK at iteration t defined by

$$\hat{\Theta}_t = \nabla_{\boldsymbol{\theta}}\phi_t(\mathcal{X})\nabla_{\boldsymbol{\theta}}\phi_t(\mathcal{X})^T. \tag{2.10}$$

The NTK can also be defined for general arguments, e.g., $\hat{\Theta}_t(\boldsymbol{x},\mathcal{X})$ with $\boldsymbol{x}$ as a test sample location.

After initialization, the training dynamics of deep learning can be characterized by (2.8) and (2.9). The steady-state solutions of these evolution equations give the learned network parameters and the learned neural network in the regression problem. However, these evolution equations are highly nonlinear and it is difficult to obtain the explicit formulations of their solutions. Fortunately, as discussed in the literature [1,8,30,39,47], when the network width goes to infinity, these evolution equations can be approximately characterized by their linearization, the solution of which admits simple explicit formulas.

For simplicity, we consider the linearization in [39] to obtain explicit solutions to discuss the training dynamics. In particular, the following linearized network by Taylor expansion is considered,

$$\phi_t^{\text{lin}}(\boldsymbol{x}) := \phi(\boldsymbol{x};\boldsymbol{\theta}_0) + \nabla_{\boldsymbol{\theta}}\phi(\boldsymbol{x};\boldsymbol{\theta}_0)\boldsymbol{\omega}_t, \tag{2.11}$$

where $\boldsymbol{\omega}_t := \boldsymbol{\theta}_t - \boldsymbol{\theta}_0$ is the change in the parameters from their initial values. The dynamics of gradient flow using this linearized function are governed by

$$\dot{\boldsymbol{\omega}}_t = -\nabla_{\boldsymbol{\theta}}\phi_0(\mathcal{X})^T\nabla_{\phi_t^{\text{lin}}(\mathcal{X})}\hat{\mathcal{J}}, \quad \dot{\phi}_t^{\text{lin}}(\boldsymbol{x}) = -\hat{\Theta}_0(\boldsymbol{x},\mathcal{X})\nabla_{\phi_t^{\text{lin}}(\mathcal{X})}\hat{\mathcal{J}}. \tag{2.12}$$

The above evolution equations have closed-form solutions

$$\boldsymbol{\omega}_t = -\nabla_{\boldsymbol{\theta}}\phi_0(\mathcal{X})^T\hat{\Theta}_0^{-1}\left(I - e^{-\hat{\Theta}_0 t}\right)(\phi_0(\mathcal{X}) - \mathcal{Y}), \quad \phi_t^{\text{lin}}(\mathcal{X}) = \left(I - e^{-\hat{\Theta}_0 t}\right)\mathcal{Y} + e^{-\hat{\Theta}_0 t}\phi_0(\mathcal{X}). \tag{2.13}$$

For an arbitrary point $\boldsymbol{x}$,

$$\phi_t^{\text{lin}}(\boldsymbol{x}) = \phi_0(\boldsymbol{x}) - \hat{\Theta}_0(\boldsymbol{x},\mathcal{X})\hat{\Theta}_0^{-1}\left(I - e^{-\hat{\Theta}_0 t}\right)(\phi_0(\mathcal{X}) - \mathcal{Y}). \tag{2.14}$$

Therefore, once the initialized network $\phi_0(\boldsymbol{x})$ and the NTK at initialization $\hat{\Theta}_0$ are computed, we can obtain the time evolution of the linearized neural network without running gradient descent. The solution in (2.14) serves as an approximate solution to the nonlinear evolution equation in (2.9). Based on (2.14), we see that deep learning can be approximated by a kernel method with the NTK $\hat{\Theta}_0$ that updates the initial prediction $\phi_0(\boldsymbol{x})$ to a correct one.

There are mainly two kinds of observations from (2.14) from the perspective of kernel methods. The first one is through the eigendecomposition of the initial NTK. If the initial NTK is positive definite, ϕ_t^{lin} will eventually converge to a neural network that fits all training examples and its generalization capacity is similar to kernel regression by (2.14). The error of ϕ_t^{lin} along the direction of eigenvectors of $\hat{\Theta}_0$ corresponding to large eigenvalues decays much faster than the error along the direction of eigenvectors of small eigenvalues, which is referred to as the spectral bias of deep learning. The second one is through the condition number of the initial NTK. Since NTK is real symmetric, its condition number is equal to its largest eigenvalue over its smallest eigenvalue. If the initial NTK is positive definite, in the ideal case when t goes to

infinity, $\left(I-e^{-\hat{\Theta}_0 t}\right)(\phi_0(\mathcal{X})-\mathcal{Y})$ in (2.14) approaches to $\phi_0(\mathcal{X})-\mathcal{Y}$ and, hence, $\phi_t^{\text{lin}}(\boldsymbol{x})$ goes to the desired function value for $\boldsymbol{x}\in\mathcal{X}$. However, in practice, when $\hat{\Theta}_0$ is very ill-conditioned, a small approximation error in $\left(I-e^{-\hat{\Theta}_0 t}\right)(\phi_0(\mathcal{X})-\mathcal{Y})\approx\phi_0(\mathcal{X})-\mathcal{Y}$ may be amplified significantly, resulting in a poor accuracy for $\phi_t^{\text{lin}}(\boldsymbol{x})$ to solve the regression problem.

The above discussion is for the NTK in a regression setting. In the case of PDE solvers, we introduce the NTK below

$$\hat{\Theta}_t=(\nabla_{\boldsymbol{\theta}}\mathcal{D}\phi_t(\mathcal{X}))(\nabla_{\boldsymbol{\theta}}\mathcal{D}\phi_t(\mathcal{X}))^T, \tag{2.15}$$

where $\mathcal{D}$ is the differential operator of the PDE. Similar to the discussion for regression problems, the spectral bias and the conditioning issue also exist in deep learning-based PDE solvers by almost the same arguments.

Although the condition number of the NTK matrix provides a perspective for understanding the improved optimization convergence of training error, they may not necessarily indicate better test performance in general. In this paper, we demonstrate the effectiveness of our RAF approach using coordinate-based data representation and PDE with the least squares method. In both applications, the training error closely matches the testing error because the training points are repeatedly and randomly sampled from a distribution at every iteration, and the testing points are sampled from the same distribution. Therefore, the test performance in our numerical examples is reflected in the training error, which in turn can be reflected in the condition numbers of the NTK matrix.

In fact, better optimization convergence would also help to improve testing errors for other applications in certain cases. In theory, the testing error is bounded by the neural network approximation error plus the neural network optimization error plus the neural network statistical error. Therefore, if the optimization error dominates the error bound, then improving the optimization convergence (e.g., either reducing the condition number or improving the convergence rate) will lead to a smaller optimization error and, hence, a smaller testing error.

3. Reproducing activation functions

In this section, we will introduce the concept of reproducing activation functions, for example, reproducing properties, and the corresponding NTK.

3.1. Abstract framework. In Section 2.1, we have introduced deep neural networks built with the same activation function $\sigma(x)$ used in each neuron of the network. The concept of reproducing activation functions is to apply different activation functions in different neurons. Let $\mathcal{A}=\{\gamma_1(x),\dots,\gamma_P(x)\}$ be a set of P different basic activation functions. In the i-th neuron of the ℓ-th layer, an activation function

$$\sigma_{i,\ell}(x)=\sum_{p=1}^{P}\alpha_{p,i,\ell}\gamma_p(\beta_{p,i,\ell}x) \tag{3.1}$$

is applied, where $\{\alpha_{p,i,\ell},\beta_{p,i,\ell}\}_{p=1}^{P}$ is a set of learnable parameters. In this paper, $\alpha_{p,i,\ell}$ is called a learnable combination coefficient and $\beta_{p,i,\ell}$ is called a learnable scaling parameter. Let $\boldsymbol{\alpha}$ be the union of all learnable combination coefficients and $\boldsymbol{\beta}$ be the union of all learnable scaling parameters in all reproducing activation functions, and then we use $\phi(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\alpha},\boldsymbol{\beta})$ to denote a deep neural network, where $\boldsymbol{\theta}$ is the set of all weights and bias introduced in (2.1) and (2.2). The key idea of reproducing activation functions is

to use a small number of basic activation functions that can reproduce a large class of functions. If a large number of basic activation functions are used, it is computationally expensive, and overfitting phenomena may occur. As we shall see later in our theory, a small number of basic activation functions motivated by applied harmonic analysis is sufficiently powerful.

In deep learning for regression problems, the new population and empirical loss functions with reproducing activation functions become

$$\mathcal{J}(\boldsymbol{\theta},\boldsymbol{\alpha},\boldsymbol{\beta}) = \frac{1}{2}\mathbb{E}_{\mathbf{x}\sim\pi}\left[|\phi(\mathbf{x};\boldsymbol{\theta},\boldsymbol{\alpha},\boldsymbol{\beta}) - f(\mathbf{x})|^2\right], \quad \hat{\mathcal{J}}(\boldsymbol{\theta},\boldsymbol{\alpha},\boldsymbol{\beta}) = \frac{1}{2N}\sum_{i=1}^{N}|\phi(\mathbf{x}_i;\boldsymbol{\theta},\boldsymbol{\alpha},\boldsymbol{\beta}) - y_i|^2, \tag{3.2}$$

respectively. The optimal parameter set is identified via $\{\hat{\boldsymbol{\theta}},\hat{\boldsymbol{\alpha}},\hat{\boldsymbol{\beta}}\} = \arg\min_{\boldsymbol{\theta},\boldsymbol{\alpha},\boldsymbol{\beta}}\hat{\mathcal{J}}(\boldsymbol{\theta},\boldsymbol{\alpha},\boldsymbol{\beta})$, and $\phi(\cdot;\hat{\boldsymbol{\theta}},\hat{\boldsymbol{\alpha}},\hat{\boldsymbol{\beta}}):\Omega\to\mathbb{R}$ is the learned DNN that approximates the unknown function f.

Similarly, when solving the PDE in (2.4), the population loss function in (2.5) becomes

$$\min_{\boldsymbol{\theta},\boldsymbol{\alpha},\boldsymbol{\beta}} \mathcal{L}(\boldsymbol{\theta},\boldsymbol{\alpha},\boldsymbol{\beta}) := \|\mathcal{D}u(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\alpha},\boldsymbol{\beta}) - f(u,\boldsymbol{x})\|^2_{L^2(\Omega)} + \lambda\|\mathcal{B}u(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\alpha},\boldsymbol{\beta}) - g(\boldsymbol{x})\|^2_{L^2(\partial\Omega)}, \tag{3.3}$$

and in each iteration of the SGD, the empirical loss function in (2.6) becomes

$$\min_{\boldsymbol{\theta},\boldsymbol{\alpha},\boldsymbol{\beta}} \hat{\mathcal{L}}(\boldsymbol{\theta},\boldsymbol{\alpha},\boldsymbol{\beta}) := \frac{1}{2N}\sum_{i=1}^{N}\big(\mathcal{D}u(\boldsymbol{x}_i;\boldsymbol{\theta},\boldsymbol{\alpha},\boldsymbol{\beta}) - f(\boldsymbol{x}_i)\big)^2 + \frac{\lambda}{2M}\sum_{j=1}^{M}\big(\mathcal{B}u(\tilde{\boldsymbol{x}}_j;\boldsymbol{\theta},\boldsymbol{\alpha},\boldsymbol{\beta}) - g(\tilde{\boldsymbol{x}}_j)\big)^2. \tag{3.4}$$

3.2. Examples and reproducing properties. Here, a few examples of RAFs will be discussed, including existing examples with super approximation power in the literature, examples lessening the spectral bias in the literature, and our new examples. We will only introduce examples with multiple basic activation functions for simplicity.

3.2.1. Example 1: Sine-ReLU. The Sine-ReLU network proposed in [78] applies sine function $\sin(x)$ or ReLU function $\max\{0,x\}$ in each neuron to construct a deep neural network. Instead, the proposed reproducing activation function here has a set of trainable parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. The theory of Sine-ReLU networks proved in [78] provides a theoretical upper bound of the approximation capacity of reproducing activation function for the set of basic activation functions $\mathcal{A} = \{\sin(x),\max\{0,x\}\}$. In fact, $\sin(x)$ can be replaced by any Lipschitz periodic function as shown in Theorem 6.1 of [78]. Let $F_{r,d}$ be the unit ball of the d-dimensional Sobolev space $H^{r,\infty}([0,1]^d)$ where $r>0$. Following the proof of this theorem, we have the following theorem for reproducing activation functions below.

THEOREM 3.1 (Dimension-Independent and Exponential Approximation Rate). *Fix $r>0$ and an integer d. Let σ be a Lipschitz periodic function with period T. Suppose $\sigma(x)>0$ for $x\in(0,T/2]$, $\sigma(x)<0$ for $x\in(T/2,T)$, and $\max_{x\in\mathbb{R}}\sigma(x) = -\min_{x\in\mathbb{R}}\sigma(x)$. For any sufficiently large integer $W>0$ and any $f(\boldsymbol{x})\in F_{r,d}$, there exists an NN $\phi(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\alpha},\boldsymbol{\beta})$ such that: (1) The total number of parameters in $\{\boldsymbol{\theta},\boldsymbol{\alpha},\boldsymbol{\beta}\}$ is less than or equal to W; (2) $\phi(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\alpha},\boldsymbol{\beta})$ is built with RAFs associated with $\mathcal{A} = \{\sigma(x),\max\{0,x\}\}$; (3) $\|f(\boldsymbol{x}) - \phi(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\alpha},\boldsymbol{\beta})\|_\infty \le exp\big(-c_{r,d}W^{1/2}\big)$ with a constant $c_{r,d}>0$ only depending on r and d.*

There are other types of network structures utilizing both $\sin(x)$ and ReLU activation functions together in a single network but for different application purposes and with different strategies. For example, the network structure in [81] uses plane waves with different frequencies as activation functions in the first hidden layer and uses ReLU in other layers for the purpose of high-resolution image reconstruction in cryo-electron microscopy. The same idea is applied in [51] for high-resolution scene and shape reconstruction in various applications. The same structure is used in [26] for the purpose of generating networks satisfying periodic boundary conditions. A variant of this structure with several blocks is designed in [44,73] for solving high-frequency PDEs. As discussed in [71], using plane waves with different frequencies in the first hidden layer may lessen the spectral bias of deep learning using NTK analysis.

3.2.2. Example 2: Floor-Exponential-Sign. Recently, networks with super approximation power (e.g., an exponential approximation rate without the curse of dimensionality for Hölder continuous functions) have been proposed in [64,65], e.g., the Floor-Exponential-Sign network that uses one of the following three activation functions in each neuron:

$$\sigma_1(x) := \lfloor x \rfloor, \quad \sigma_2(x) := 2^x, \quad \sigma_3 := \mathcal{T}(x - \lfloor x \rfloor - \frac{1}{2}). \tag{3.5}$$

Here, $\mathcal{T}(x) := \mathbb{1}_{x \geq 0} = \begin{cases} 1, & x \geq 0, \\ 0, & x < 0. \end{cases}$ Obviously, the concept of reproducing activation functions includes the Floor-Exponential-Sign networks as a special case when the set of combination coefficients $\boldsymbol{\alpha}$ is a fixed binary set and the set of scaling coefficients $\boldsymbol{\beta}$ is a set of constant ones. The theory of Floor-Exponential-Sign networks proved in [64] provides a theoretical upper bound of the approximation power of reproducing activation function for the set of basic activation functions $\mathcal{A} = \{\sigma_1(x), \sigma_2(x), \sigma_3(x)\}$. Following the proof of Theorem 1.1 in [64], we have the theorem for reproducing activation functions below.

THEOREM 3.2 (Dimension-Independent and Exponential Approximation Rate). *Given f in $C([0,1]^d)$ and $W \in \mathbb{N}^+$, there exists an NN $\phi(\boldsymbol{x}; \boldsymbol{\theta}, \boldsymbol{\alpha}, \boldsymbol{\beta})$ of width W and depth 4 built with RAFs associated with $\mathcal{A} = \{\sigma_1(x), \sigma_2(x), \sigma_3(x)\}$ such that, for any $\boldsymbol{x} \in [0,1]^d$,*

$$|\phi(\boldsymbol{x}; \boldsymbol{\theta}, \boldsymbol{\alpha}, \boldsymbol{\beta}) - f(\boldsymbol{x})| \leq 2\omega_f(2\sqrt{d})2^{-W} + \omega_f(2\sqrt{d}2^{-W}) \tag{3.6}$$

for any $\boldsymbol{x} = (x_1, \cdots, x_d) \in [0,1]^d$. The total number of parameters in $\{\boldsymbol{\theta}, \boldsymbol{\alpha}, \boldsymbol{\beta}\}$ is bounded by $2W^2 + (d+22)W + 1$.

In the above theorem, $\omega_f(\cdot)$ is the modulus of continuity of f defined as

$$\omega_f(r) := \sup\{|f(\boldsymbol{x}) - f(\boldsymbol{y})| : \|\boldsymbol{x} - \boldsymbol{y}\|_2 \leq r, \ \boldsymbol{x}, \boldsymbol{y} \in [0,1]^d\}, \tag{3.7}$$

for any $r \geq 0$, where $\|\boldsymbol{x}\|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_d^2}$ for any $\boldsymbol{x} = (x_1, x_2, \cdots, x_d) \in \mathbb{R}^d$. In Theorem 1.1 in [64], it was shown that only W parameters in the Floor-Exponential-Sign network depend on f. However, introducing more parameters in the reproducing activation function concept may alleviate the optimization difficulty of identifying parameters. We would like to point out that, although the approximation power of the network in Theorem 3.2 is very attractive, there are no efficient optimization methods for training networks with piecewise constant activation functions. Hence, it is worth exploring other reproducing activation functions as we shall see in the next part.

3.2.3. Example 3: Poly-Sine-Gaussian. Finally, we propose the poly-sine-Gaussian network using $\mathcal{A}=\{x,x^2,\sin(x),e^{-x^2}\}$ such that NNs can reproduce traditional approximation tools efficiently, e.g., orthogonal polynomials, Fourier basis functions, wavelets, radial basis functions, etc. Therefore, this new NN may approximate a wide class of target functions with a smaller number of parameters than existing NNs, e.g., ReLU NNs, since existing approximation theory with a continuous weight selection of ReLU NNs is established by using ReLU networks to approximate x and x^2 as basic building blocks. We present several theorems proved in the Appendix to illustrate the approximation capacity of this new network.

THEOREM 3.3 (Reproducing Polynomials). *Assume* $P(\boldsymbol{x})=\sum_{j=1}^{J}c_j\boldsymbol{x}^{\boldsymbol{\alpha}_j}$ *for* $\boldsymbol{\alpha}_j\in\mathbb{N}^d$. *For any* $N,L,a,b\in\mathbb{N}^+$ *such that* $ab\geq J$ *and* $(L-2b-b\log_2 N)N\geq b\max_j|\boldsymbol{\alpha}_j|$, *there exists a poly-sine-Gaussian network* ϕ *with width* $2Na+d+1$ *and depth* L *such that* $\phi(\boldsymbol{x})=P(\boldsymbol{x})$ *for any* $\boldsymbol{x}\in\mathbb{R}^d$.

The proof of Theorem 3.3 can be found in Appendix A. Theorem 3.3 characterizes how well poly-sine-Gaussian networks reproduce arbitrary polynomials including orthogonal polynomials. Compared to the results of ReLU NNs for polynomials in [45, 76], poly-sine-Gaussian networks require fewer parameters. Orthogonal polynomials are important tools for classical approximation theory and numerical computation. For example, the Chebyshev series lies at the heart of approximation theory. In particular, for analytic functions, the truncated Chebyshev series defined as $f_n(x)=\sum_{k=0}^{n}c_kT_k(x/M)$ are *exponentially accurate* approximations by Theorem 8.2 [72], where T_k is the Chebyshev polynomial of degree k defined on $[-1,1]$. More precisely, for some scalars $M\geq 1$ and $s>1$, if we define $a_s^M=M\frac{s+s^{-1}}{2}, b_s^M=M\frac{s-s^{-1}}{2}$, and the *Bernstein s-ellipse scaled to* $[-M,M]$, $E_s^M=\left\{x+iy\in\mathbb{C}:\frac{x^2}{(a_s^M)^2}+\frac{y^2}{(b_s^M)^2}=1\right\}$, then we have the following theorem.

THEOREM 3.4 (Exponential Approximation Rate). *For any* $M\geq 1$, $s>1$, $C_f>0$, $0<\epsilon<1$, *and any real-valued analytic function* f *on* $[-M,M]$ *that is analytically continuable to the open ellipse* E_s^M, *where it satisfies* $|f(x)|\leq C_f$, *there is a poly-sine-Gaussian network* ϕ *with width* $2N+2$ *and depth* L *such that* $\|\phi(x)-f(x)\|_{L^\infty([-M,M])}\leq\epsilon$, *where* N *and* L *are positive integers satisfying* $(L-2n-2-(n+1)\log_2 N)N\geq n(n+1)$ *and* $n=\mathcal{O}\left(\frac{1}{\log_2 s}\log_2\frac{2C_f}{\epsilon}\right)$.

The proof of Theorem 3.4 can be found in Appendix B. By choosing $N=\mathcal{O}(n)$ and $L=\mathcal{O}(n\log_2(n))$ in Theorem 3.4, the width and depth of ϕ are $\mathcal{O}\left(\log_2\frac{1}{\epsilon}\right)$ and $\mathcal{O}\left(\left(\log_2\frac{1}{\epsilon}\right)\log_2\left(\log_2\frac{1}{\epsilon}\right)\right)$, respectively, leading to a network size smaller than that of the ReLU NN in Theorem 2.6 in [54].

Next, we prove the approximation of poly-sine-Gaussian networks to generalized bandlimited functions below.

DEFINITION 3.1. *Let* $d\geq 2$ *be an integer,* $M\geq 1$ *be a scalar, and* $B=[0,1]^d$. *Suppose* $K:\mathbb{R}\to\mathbb{C}$ *is analytic and bounded by a constant* $D_K\in(0,1]$ *on* $[-dM,dM]$ *and* K *satisfies the assumption of Theorem 3.4 for* $s>1$ *and* $C_K>0$. *We define the Hilbert space* $\mathcal{H}_{K,M}(B)$ *of generalized bandlimited functions via*

$$\mathcal{H}_{K,M}(B)=\left\{f(\boldsymbol{x})=\int_{[-M,M]^d}F(\boldsymbol{w})K(\boldsymbol{w}\cdot\boldsymbol{x})d\boldsymbol{w}\;\middle|\;F:[-M,M]^d\to\mathbb{C}\text{ is in }L^2([-M,M]^d)\right\},\quad(3.8)$$

with $\langle f,g\rangle_{\mathcal{H}_{K,M}(B)}:=\int_{[-M,M]^d}F_f(\boldsymbol{w})\overline{F}_g(\boldsymbol{w})d\boldsymbol{w}$ *and its induced norm* $\|f\|_{\mathcal{H}_{K,M}(B)}$, *where* $F_f=\arg\min_{F\in S_f}\|F\|_{L^2([-M,M]^d)}$ *and* $S_f=\left\{F\;\middle|\;f(\boldsymbol{x})=\int_{[-M,M]^d}F(\boldsymbol{w})K(\boldsymbol{w}\cdot\boldsymbol{x})d\boldsymbol{w}\right\}$.

Note that $\mathcal{H}_{K,M}(B)$ is a reproducing kernel Hilbert space (RKHS); a classical example of interest is $K(t)=e^{it}$. For simplicity, we will use F instead of F_f for $f\in\mathcal{H}_{K,M}(B)$, when the dependency on f is clear.

THEOREM 3.5 (Dimension-Independent Approximation). *For any real-valued function f in $\mathcal{H}_{K,M}(B)$, $M\geq 1$, $s>1$, $C_K>0$, and $d\geq 2$. Assume $\int_{\mathbb{R}^d}|F(\boldsymbol{w})|d\boldsymbol{w}=\int_{[-M,M]^d}|F(\boldsymbol{w})|d\boldsymbol{w}=C_F$. For any measure μ and $\epsilon\in(0,1)$, there exists a poly-sine-Gauss. network ϕ on $B=[0,1]^d$, that has width $\mathcal{O}\left(\frac{4C_F\sqrt{\mu(B)}}{\epsilon^2\log_2 s}\log_2\frac{4C_F\sqrt{\mu(B)}C_K}{\epsilon}\right)$ and depth $\mathcal{O}\left(\left(\frac{1}{\log_2 s}\log_2\frac{4C_F\sqrt{\mu(B)}C_K}{\epsilon}\right)\log_2\log_2\frac{4C_F\sqrt{\mu(B)}C_K}{\epsilon}\right)$ such that $\|\phi-f\|_{L^2(\mu,B)}=\sqrt{\int_B|\phi(\boldsymbol{x})-f(\boldsymbol{x})|^2d\mu(\boldsymbol{x})}\leq\epsilon$.*

The proof of Theorem 3.5 can be found in Appendix C. We would like to revisit the discussion in [2, 54] about C_F and $\mu(B)$. If F is a mollifier then $C_F=1$, whereas if F is a normal distribution truncated to $[-M,M]^d$ then $C_F<1$. In general, however, C_F might grow algebraically or exponentially with the dimension d. If μ is a probability measure, then $\mu(B)\leq 1$ for any compact domain B. If μ is a Lebesgue measure, then $\mu(B)=1$ for $B=[0,1]^d$, but grows exponentially with the dimension d if $B=[0,\ell]^d$, $\ell>1$. Hence, the curse of dimensionality of approximation may exist due to large C_F and $\mu(B)$. However, the approximation rate of poly-sine-Gaussian networks is dimension-independent. Compared to ReLU networks in Theorem 3.2 in [54] approximating the same bandlimited function, the poly-sine-Gaussian network in Theorem 3.5 requires fewer parameters.

Poly-sine-Gaussian networks can also reproduce typical applied harmonic analysis tools as in the following lemma, the proof of which can be found in Appendix D.

LEMMA 3.1.

(i) *Poly-sine-Gaussian networks can reproduce all basis functions in the discrete cosine transform and the discrete windowed cosine transform with a Gaussian window function in an arbitrary dimension.*

(ii) *Poly-sine-Gaussian networks with complex parameters can reproduce all basis functions in the discrete Fourier transform and the discrete Gabor wavelet transform in an arbitrary dimension.*

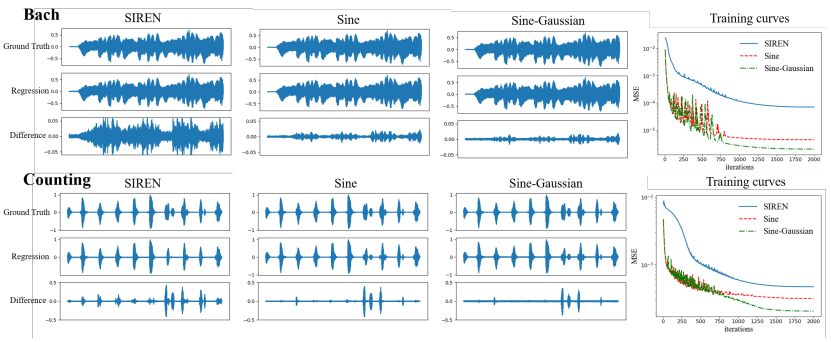


Fig. 3.1: *The comparison of fitted signal and training curves on two audio clips, Bach and Counting, for SIREN and our RAFs (Sine and Sine-Gaussian).*

(a) Camera (b) Astronaut (c) Cat (d) Coin

Fig. 3.2: *Training dataset for fitting images in coordinate-based data representation.*

Activation	Camera	Astronaut	Cat	Coin
SIREN	45.80/0.9913	44.84/0.9962	49.58/0.9970	43.05/0.9868
Sine	60.60/0.9995	59.37/0.9997	65.94/0.9999	62.66/0.9998
Poly-Sine	61.21/0.9996	59.99/0.9997	66.41/0.9999	63.57/0.9998
Poly-Sine-Gauss.	**73.80/1.0000**	**70.98/1.0000**	**82.55/1.0000**	**74.92/1.0000**

Table 3.1: *The comparison of PSNR/SSIM of the fitted images using SIREN and our RAFs (Sine, Poly-Sine, Poly-Sine-Gauss.). The larger these numbers, the better the performance.*

Lemma 3.1 above implies that poly-sine-Gaussian networks may be useful in many computer vision and audio tasks involving Fourier transforms and wavelet transforms. Due to the advantage of wavelets to represent functions with singularity, poly-sine-Gaussian networks may also be useful in representing functions with singularity. We would like to highlight that the Gaussian function may not be the optimal choice in the concept of RAF. Other window functions in wavelet analysis may provide better performance and this would be problem-dependent.

Finally, we have the next lemma for radial basis functions. The proof of Lemma 3.2 can be found in Appendix E.

LEMMA 3.2. *Poly-sine-Gaussian networks can reproduce Gaussian radial basis functions and approximate radial basis functions defined on a bounded closed domain with analytic kernels with an exponential approximation rate.*

We will conclude this section with an informal discussion about the NTK of poly-sine-Gaussian networks. As we discussed in Section 2, deep learning can be approximated by kernel methods with a kernel $\hat{\Theta}_0$ in (2.14). Therefore, from the perspective of kernel regression for regressing $f(\boldsymbol{x})$ with training samples $\{(\boldsymbol{x}_i, f(\boldsymbol{x}_i))\}_{i=1}^N$, $\hat{\Theta}_0(\boldsymbol{x},\boldsymbol{x}_i)$ quantifies the similarity of the point $\boldsymbol{x}$ and a training point $\boldsymbol{x}_i \in \mathcal{X}$ and, hence, serves as a weight of $f(\boldsymbol{x}_i)$ in a toy regression formulation: $\phi(\boldsymbol{x};\boldsymbol{\omega}) := \sum_{i=1}^N \omega_i f(\boldsymbol{x}_i)\hat{\Theta}(\boldsymbol{x},\boldsymbol{x}_i)$, where $\boldsymbol{\omega} = [\omega_1,\ldots,\omega_N]$ is a set of learnable parameters and $\phi(\boldsymbol{x};\boldsymbol{\omega})$ is the approximant of the target function $f(\boldsymbol{x})$. To enable a kernel method to learn both smooth functions and highly oscillatory functions, the kernel function $\hat{\Theta}$ should have a widely spreading Fourier spectrum. By using $\sin(\beta x)$ with a tunable β in the poly-sine-Gaussian, the poly-sine-Gaussian network could learn an appropriate kernel for both kinds of functions. Similarly, by using $\exp(-(\beta x)^2))$ with a tunable β in the poly-sine-Gaussian, the poly-sine-Gaussian network could learn an appropriate kernel for both smooth and singular functions. We will provide numerical examples to demonstrate this empirically in the next section.

4. Numerical results

In this section, we will illustrate the advantages of RAFs in two kinds of applications, data representation and scientific computing. The optimal choice of basic activation functions would be problem-dependent.

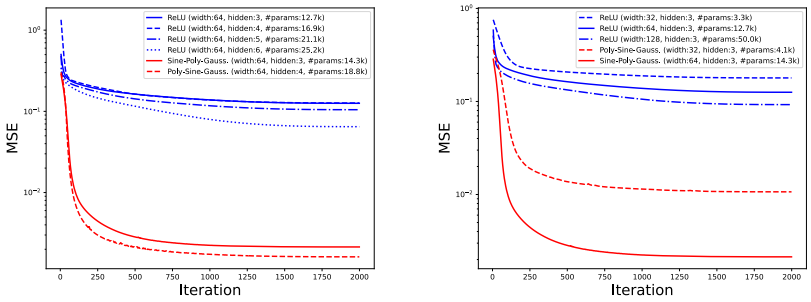
4.1. Coordinate-based data representation. We verify the performance of RAFs on data representations using coordinate-based NNs. Mean square error (MSE) quantifies the difference between the ground truth and the NN output. Standard NNs, e.g., ReLU NNs, were shown to have poor performance to fit high-frequency components of signals [67,71]. SIREN activation function [67], i.e., $\sin(30x)$, improves the ability of NNs to represent complex signals.

As we discussed in Section 3.2, the SIREN function is a special case of the poly-sine-Gaussian activation function in our framework. We will show that the poly-sine-Gaussian activation function can provide better performance than SIREN when the combination coefficients $\boldsymbol{\alpha}$ and the scaling parameters $\boldsymbol{\beta}$ are specified or trained appropriately in a problem-dependent manner. We follow the official implementation of SIREN on representations of audio, image, and video signals (refer to [67] for details). The main difference between the SIREN code and ours is the activation function. In coordinate-based data representation, fully connected networks are memory-efficient and continuous representations for objects such as shapes or scenes. One important application is signal compression [15,69], where the signal is stored in the parameters of a neural network. The training error characterizes how well the network fits the data. Additionally, we compare our method with SIREN [67] and report the training loss as done in SIREN. All trainable parameters are trained to minimize the empirical loss function in (3.2). The NN is optimized by Adam optimizer with an initial learning rate 10^{-4} and cosine learning rate decay.

4.1.1. Audio signal. We start by modeling audio signals on two audio clips, Bach and Counting as shown in Figure 3.1. An NN is trained to regress from a one-dimensional time coordinate to the corresponding sound level. Note that audio signals are purely oscillatory signals. Therefore, in the reproducing activation framework, x and x^2 are not necessary. We apply two forms of RAFs, Sine, and Sine-Gaussian. The Sine one is set as $\alpha_1 \sin(\beta_1 x)$, while the Sine-Gaussian one is set as $\alpha_1 \sin(\beta_1 x) + \alpha_2 \exp(-x^2/(2\beta_2^2))$, where α_1 is initialized as $\mathcal{N}(2,0.1)$, α_2 is initialized as $\mathcal{N}(1.0,0.1)$, β_1 is initialized as $\mathcal{N}(30,0.001)$, and β_2 is initialized with a uniform distribution $\mathcal{U}(0.01,0.05)$. We use a 3-hidden-layer neural network with 256 neurons per layer to fit the audio signal following the network structure of SIREN. The NNs are trained for 2000 iterations. Figure 3.1 displays the fitted signals and training curves. Figure 3.1 shows our method has the capacity of modeling the audio signals more accurately than SIREN and leads to a smaller error in regression. Besides, our RAFs can converge to a better local minimum at a faster speed compared with SIREN. Moreover, we can see the add-in Gaussian function enhances the fitting ability.

4.1.2. Image signal. We regress a grayscale image by learning a mapping from two-dimensional pixel coordinates to the corresponding pixel value. Four images of size 256×256 are used, including Camera, Astronaut, Cat, and Coin (as shown in Figure 3.2), which are available in Python Pillow. Note that images usually contain a cartoon part and a texture part. We apply three types of RAFs for image fitting: Sine, $\alpha_1 \sin(\beta_1 x)$; Poly-Sine, $\alpha_1 \sin(\beta_1 x) + \alpha_3 x + \alpha_4 x^2$; and Poly-Sine-Gaussian,

$$\alpha_1 \sin(\beta_1 x) + \alpha_2 \exp(-x^2/(2\beta_2^2)) + \alpha_3 x + \alpha_4 x^2. \tag{4.1}$$



(a) Layer (b) Width

Fig. 4.1: *Comparison of training loss for fitting the image "Astronaut" with different activation functions in neural networks of different numbers of neurons (width) and hidden layers.*

Fig. 4.2: *Comparison of the video frames fitted by different activation functions and the corresponding training curves.*

Here, α_1, α_2, α_3, α_4, β_1, and β_2 are initialized as $\mathcal{N}(2,0.1)$, $\mathcal{N}(1,0.1)$, $\mathcal{N}(0.0,0.1)$, $\mathcal{N}(1.0,0.1)$, $\mathcal{N}(30,0.001)$, and $\mathcal{U}(0.01,0.05)$, respectively. An NN with 3 hidden layers and 256 neurons per layer is trained for 2,000 iterations. Here we use peak signal-to-noise ratio peak (PSNR) and structural similarity (SSIM) to characterize the quality of the fitted image.

Table 3.1 summarizes the PSNR and SSIM of the fitted images showing that RAFs outperform SIREN by a significant margin.

Since RAF is applied to each hidden neuron, the cost caused by RAF will grow linearly with the number of hidden nodes, while the cost increases quadratically with the number of hidden nodes for the ReLU network. Second, the improvement benefited by RAF is more significant than that benefited by enlarging the network size. In Figure 4.1, we compare the training loss for different network sizes by increasing the width of the hidden layer or the number of hidden layers. We can see the training loss decreases with the growth of the network size for specific activation functions, but it is marginal simply by increasing the width of the hidden layer or the number of hidden layers. However, the Poly-Sine-Gaussian network with 14.3k parameters achieves significantly lower error than the ReLU network with 25.2k parameters or 50.0k parameters.

4.1.3. Video signal. We fit a color video named BIKE with 250 frames available in Python Skvideo Package (example frames can be seen in Figure 4.2). The regression is from three-dimensional coordinates to RGB pixel values. We apply Sine-Guassian as defined in Section 4.1.1, but α_1, α_2, β_1 and β_2 are initialized by $\mathcal{N}(1,0.1)$, $\mathcal{N}(1,0.1)$,

$\mathcal{N}(30,0.001)$ and $\mathcal{U}(0.002,0.01)$, respectively. An NN with 3 hidden layers and 400 neurons per layer is trained for 100,000 iterations. Figure 4.2 displays the video frames fitted by different activation functions and the training curves of video fitting for different activation functions, and our Sine-Gaussian can lead to a better minimizer. Computed with 250 frames, our Sine-Gaussian achieves a larger mean PSNR (32.79) than SIREN (32.17), and has a smaller standard derivation of PSNR (2.10) than SIREN (2.16).

4.2. Scientific computing applications. We will compare the proposed poly-sine-Gaussian RAF with existing activation functions, e.g., the ReLU function, the ReLU^3 function, and the rational activation function in [5]. We will also provide an ablation study to justify the combination of $\mathcal{A}=\{x,x^2,\sin(x),\exp(-x^2)\}$ in the poly-sin-Gaussian network.

The Adam optimizer is employed to minimize the loss functions discussed in Section 2 for regression and solving PDEs. Besides, we will follow the approach of [26] for the loss function of eigenvalue problems.

The overall setting for all examples is summarized as follows.

Optimization. Adam [36] is used with the default hyperparameters. The learning rate decreases step by step in all examples following the formula $\tau_n=\tau_0*q^{\lfloor \frac{n}{s} \rfloor}$, where τ_n is the learning rate in the $n-$th iteration, q is a factor set to be 0.95, and s means that we update learning rate after s steps. The number of training and testing samples for regression and PDE problems are 10,000. The number of training and testing samples for eigenvalue problems are 2,048 following the approach in [26].

Network setting. In all PDE examples, we construct a special network that satisfies the given boundary condition as discussed in Section 2.3. In all examples, we apply ResNet with two residual blocks and each block contains two hidden layers. The width is set as 50 unless specified. Unless specified particularly, all weights and biases in the ℓ-th layer are initialized by $\mathcal{U}(-1/\sqrt{N_{\ell-1}},1/\sqrt{N_{\ell-1}})$, where $N_{\ell-1}$ is the width of the $(\ell-1)$-th layer. Note that the network with RAFs can be expressed by a network with a single activation function in each neuron but different neurons can use different activation functions. For example, in the case of poly-sine-Gaussian networks, we will use 1/4 neurons within each layer with x activation function, 1/4 with x^2, 1/4 with $\sin(x)$, and 1/4 with $\exp(-x^2)$ for coding simplicity. In the case of poly-sine networks, 1/3 neurons for x, x^2, and $\sin(x)$ activation functions. In this new setting, it is not necessary to train extra combination coefficients in the RAF. Though training the scaling parameters in the RAF might be beneficial in general applications, we focus on justifying the poly-sine-Gaussian function without emphasizing the scaling parameters. Hence, in almost all tests, the scaling parameters are set to be one for x, x^2, and $\sin(x)$, and the scaling parameter is set to be 0.1 for $\exp(-x^2)$. In the case of oscillatory target functions, we specify the scaling parameter of $\sin(x)$ to introduce oscillation in the NTK as we shall discuss and improved performance is observed. The idea of scaling parameters has been tested and verified in [31,32].

Performance evaluation. We define the relative L^2 test error by $\left(\frac{\sum_{i=1}^N (u(x_i)-\hat{u}(x_i))^2}{\sum_{i=1}^N u^2(x_i)}\right)^{\frac{1}{2}}$, where $\{x_i\}_{i=1}^N$ are random samples uniformly distributed in the function domain, u is the ground true solution, and $\hat{u}$ is the estimation by deep learning. We will use two criteria to evaluate the performance of different activation functions based on relative L^2 error on test samples. Since the ground truth solution is not available in real-world applications, we cannot determine when to end training. To address this, we will track two test error measures:

- Min: The minimum relative L^2 test error of the past given iteration.
- Moving-average: The average L^2 test error of the past 100 iterations at a given iteration.

The second criterion is the condition number of the NTK matrices. A smaller condition number usually leads to a smaller iteration number to achieve the same accuracy.

4.2.1. Discontinuous function regression. We first show the advantage of the poly-sine-Gaussian activation function by regressing a discontinuous function, $f(x)=-2x+1$, when $x\geq 0$, $f(x)=-2x-1$, when $x<0$, on the domain $\Omega=[-1,1]$. The relative L^2 error is presented in Table 4.1 and the training process is visualized in Figure 4.3(a). The regression result shows that the poly-sine-Gaussian activation has the best performance. We would like to remark that rational activation [5] works well for regression problems but fails in our PDE problems without meaningful solutions. Hence, we only compare RAFs with rational activation functions in this example. The numerical results justify the combination of four kinds of activation functions. Remark that the computational time of rational activation functions is twice the time of RAFs, though their accuracy is almost the same.

Activation Function	Relative L^2 Error (Moving -Average)	Relative L^2 Error (Min)	Forward Evaluation Time	Backward Propagation Time
ReLU	6.61 e-02	4.16 e-02	**1.98 e-03**	**3.05 e-03**
ReLU^3	1.13 e-01	9.71 e-02	2.07 e-03	3.43 e-03
$x\oplus x^2$	3.71 e-01	3.46 e-01	4.79 e-03	5.44 e-03
$x\oplus x^2\oplus\text{ReLU}$	9.98 e-02	7.61 e-02	3.33 e-03	4.12 e-03
$x\oplus x^2\oplus\text{ReLU}^3$	9.12 e-02	6.85 e-02	3.62 s-03	4.34 e-03
$x\oplus x^2\oplus\sin(x)$	9.07 e-02	7.27 e-02	4.83 e-03	6.92 e-03
$x\oplus x^2\oplus\sin(x)\oplus$ Gaussian	**3.46 e-02**	1.08 e-02	5.92 e-03	8.91 e-03
Rational function	3.94 e-02	**5.84 e-03**	5.31 e-03	2.03 e-02

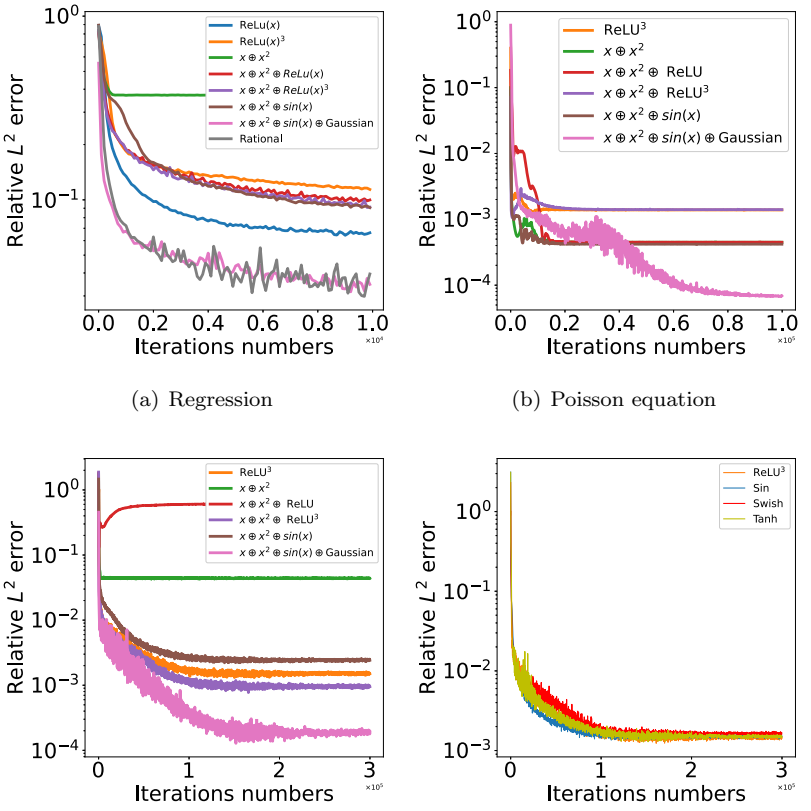
Table 4.1: *The performance comparison for the regression problem in terms of the accuracy after* 50,000 *iterations and the average computational time for one forward or backward evaluation.* $\oplus$ *means that the activation function is applied together with other activation functions.*

Activation Function	Relative L^2 Error (Moving -Average)	Relative L^2 Error (Min)	Optimization Time (Per Iteration)
ReLU^3	1.38 e-03	2.99 e-04	**5.86 e-02**
$x\oplus x^2$	4.48 e-04	4.26 e-04	6.87 e-02
$x\oplus x^2\oplus\text{ReLU}$	4.48 e-04	4.12 e-04	8.20 e-02
$x\oplus x^2\oplus\text{ReLU}^3$	1.40 e-03	9.88 e-04	9.33 e-02
$x\oplus x^2\oplus\sin(x)$	4.18 e-04	3.82 e-04	8.64 e-02
$x\oplus x^2\oplus\sin(x)\oplus\exp(-x^2)$	**6.87 e-05**	**6.36 e-05**	1.11 e-01

Table 4.2: *The accuracy and computational cost for the Poisson equation defined in* (4.2) *with a smooth solution.*

4.2.2. Poisson equation with a smooth solution. Now we solve a two-dimensional Poisson equation

$$-\Delta u=f \text{ for } \boldsymbol{x}\in\Omega \text{ and } u=0 \text{ for } \boldsymbol{x}\in\partial\Omega \tag{4.2}$$



(a) Regression

(b) Poisson equation

(c) Non-smooth solution

(d) Non-smooth solution with other activation function

Fig. 4.3: *The training process of (a) the regression problem, (b) the Poisson equation in (4.2) with a smooth solution, and (c) the Equation (4.3) with a solution which is not smooth at the origin.*

with a smooth solution $u(\boldsymbol{x}) = x_1^2(1-x_1)x_2^2(1-x_2)$ defined on $\Omega = [0,1]^2$. The numerical solution can be constructed as $\hat{u}(\boldsymbol{x};\boldsymbol{\theta}) = \left(\Pi_{i=1}^2 x_i(1-x_i)\right)\phi(\boldsymbol{x};\boldsymbol{\theta})$, where $\phi(\boldsymbol{x};\boldsymbol{\theta})$ is an NN. We apply the loss function (2.7) to identify an estimated solution. The relative L^2 errors for different activation functions are shown in Table 4.2 and the corresponding training process is visualized in Figure 4.3(b). The RAF with $\mathcal{A} = \{x, x^2, \sin(x), \exp(-x^2)\}$ achieves the best performance. The networks with other activation functions reach local minima and cannot escape from these minima after $20k$ iterations, while the poly-sine-Gaussian network continuously reduces the error even after $50k$ iterations. The numerical results also justify the combination of four kinds of activation functions. The proposed activation functions can improve the accuracy by one to two digits while the computational cost only increases by a factor of 2.

4.2.3. PDE with low regularity. Next, we consider a two-dimensional PDE

$$-\nabla\cdot(|\boldsymbol{x}|\nabla u) = f \text{ for } \boldsymbol{x}\in\Omega \text{ and } u = 0 \text{ for } \boldsymbol{x}\in\partial\Omega \tag{4.3}$$

with a solution $u(\boldsymbol{x})=\sin(2\pi(1-|\boldsymbol{x}|))$ defined on $\Omega=\{\boldsymbol{x}:|\boldsymbol{x}|\leq 1\}$. The exact solution has low regularity at the origin. Let $\hat{u}(\boldsymbol{x};\boldsymbol{\theta})=(1-|\boldsymbol{x}|)\phi(\boldsymbol{x};\boldsymbol{\theta})$, where $\phi(\boldsymbol{x};\boldsymbol{\theta})$ is an NN and $\hat{u}(\boldsymbol{x};\boldsymbol{\theta})$ satisfies the boundary condition automatically. The loss function (2.7) is used to identify an estimated solution to the Equation (4.3). The relative L^2 errors for different activation functions are shown in Table 4.3 and the training curves are visualized in Figure 4.3(c). Since the true solution has low regularity, it is more challenging than the Example (4.2) to obtain good accuracy. The RAF with $\mathcal{A}=\{x,x^2,\sin(x),\exp(-x^2)\}$ achieves the lowest test error, which justifies the combination of four activation functions. The proposed activation functions can improve the accuracy by one digit while the computational cost only increases by a factor of 2.

Activation Function	Relative L^2 Error (Moving -Average)	Relative L^2 Error (Min)	Optimization Time (Per Iteration)
ReLU^3	1.49 e-03	2.99 e-04	**5.83 e-02**
$x\oplus x^2$	4.39 e-02	2.97 e-02	6.98 e-02
$x\oplus x^2\oplus\text{ReLU}$	6.06 e-01	1.20 e-01	8.27 e-02
$x\oplus x^2\oplus\text{ReLU}^3$	9.48 e-04	2.35 e-04	9.17 e-02
$x\oplus x^2\oplus\sin(x)$	2.42 e-03	7.82 e-04	8.71 e-02
$x\oplus x^2\oplus\sin(x)\oplus$ Gaussian	**1.91 e-04**	**3.96 e-05**	1.09 e-01

Table 4.3: *The accuracy and computation cost for (4.3) with a solution that is not smooth at the origin.*

4.2.4. PDE with an oscillatory solution. Next, to verify the performance of $\sin(x)$ in the RAF, we consider a two-dimensional nonlinear PDE as follows,

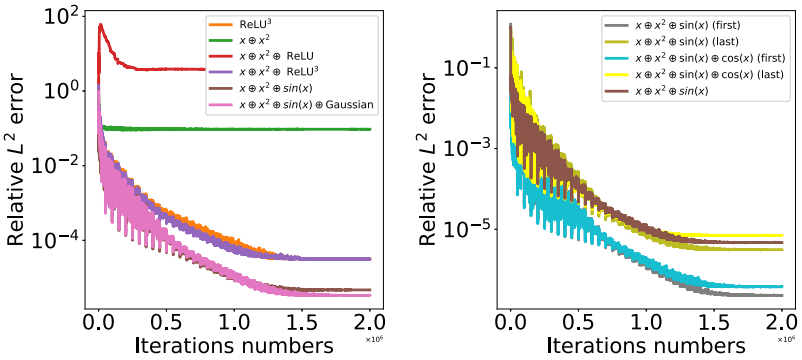
$$-\Delta u+(u+2)^2=f \text{ for } \boldsymbol{x}\in\Omega \tag{4.4}$$

with a Dirichlet boundary condition and an oscillatory solution $u(\boldsymbol{x})=\sin(6\pi x_1)\sin(6\pi x_2)$ defined on $\Omega=[0,1]^2$. The NN is constructed as in Section 4.2.2 with width 100. The least squares method is used to identify the NN solution. The test error is shown in Table 4.4 and Figure 4.4(a). RAFs with $\{x,x^2,\sin(x),\exp(-x^2)\}$ achieve the best performance.

As discussed in Section 3, introducing oscillation in NNs is crucial to lessen the spectral bias of NNs. Fixing different scaling parameters in $\sin(x)$ can help to lessen the spectral bias better and obtain a high-resolution image reconstruction [71]. Therefore, in the case of oscillatory target functions, we also specify scaling parameters in $\sin(x)$ to verify the performance. If n sine functions are used in a layer, we will use $\{\sin(2\pi x),\sin(4\pi x),\ldots,\sin(2n\pi x)\}$. Besides, it is also of interest to see the performance of $\cos(x)$. Since specifying a wide range of scaling parameters in every hidden layer creates too much oscillation, we only specify scaling parameters either in the first or the last hidden layer. Therefore, four tests were conducted and the results are shown in Table 4.5 and Figure 4.4(b). The results show that $\cos(x)$ does not have an effective gain, but specifying different scaling parameters improves the performance, especially in the first hidden layer. Furthermore, we consider a more high oscillatory solution $u(\boldsymbol{x})=\sin(40\pi x_1)\sin(40\pi x_2)$. ReLU^3, $x\oplus x^2\oplus\sin(x)$ and $x\oplus x^2\oplus\sin(x)$ (first) $\oplus$ Gaussian is compared as activation function. The numerical result can be seen in Table 4.6 and Figure 4.5. The proposed activation functions can improve the accuracy by one digit while the computational cost only increases by a factor of 2 in the case of non-oscillatory solutions. In the case of oscillatory solutions, the proposed activation function can significantly improve the accuracy and computational cost.

Activation Function	Relative L^2 Error (Moving -Average)	Relative L^2 Error (Min)	Optimization Time (Per Iteration)
ReLU^3	3.16 e-05	3.01 e-05	**7.10 e-02**
$x \oplus x^2$	9.45 e-02	8.03 e-02	8.22 e-02
$x \oplus x^2 \oplus \mathrm{ReLU}$	3.81 e+01	9.99 e-01	1.02 e-01
$x \oplus x^2 \oplus \mathrm{ReLU}^3$	3.15 e-05	2.93 e-05	1.10 e-01
$x \oplus x^2 \oplus \sin(x)$	4.69 e-06	4.48 e-06	1.09 e-01
$x \oplus x^2 \oplus \sin(x) \oplus$ Gaussian	**3.35 e-06**	**3.21 e-06**	1.29 e-01

Table 4.4: *The accuracy and computation cost for the equation in (4.4) with an oscillatory solution.*



(a) Oscillatory solution (b) Different scaling params

Fig. 4.4: *The training process of the Equation (4.4) with an oscillatory solution. Here "first" and "last" refer to the locations of pre-fixed scaling parameters.*

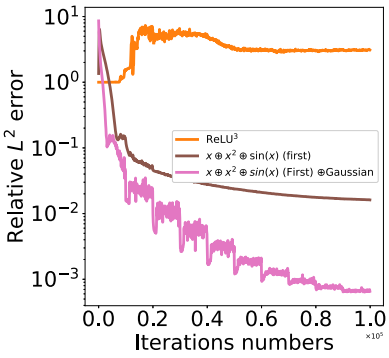


Fig. 4.5: *The training process of the Equation (4.4) with solution* $u(\boldsymbol{x}) = \sin(40\pi x_1)\sin(40\pi x_2)$.

4.2.5. Nonlinear Schrödinger equation. We consider a nonlinear Schrödinger operator with a cubic term,

$$\mathcal{L}\varphi = -\Delta\varphi + \varphi^3 + V\varphi, \qquad \text{in } \Omega = [0, 2\pi]^d, \tag{4.5}$$

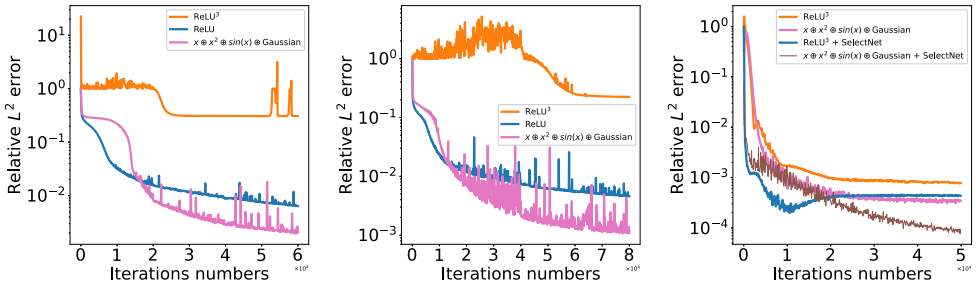
Activation Function	Relative L^2 Error (Moving -Average)	Relative L^2 Error (Min)	Optimization Time (Per Iteration)
$x \oplus x^2 \oplus \sin(x)$ (first)	**2.31 e-07**	**2.22 e-07**	**1.04 e-01**
$x \oplus x^2 \oplus \sin(x)$ (last)	2.14 e-06	2.99 e-06	1.05 e-01
$x \oplus x^2 \oplus \sin(x) \oplus \cos(x)$ (first)	3.79 e-07	3.62 e-07	1.11 e-01
$x \oplus x^2 \oplus \sin(x) \oplus \cos(x)$ (last)	1.90 e-03	6.73 e-06	1.09 e-01

Table 4.5: *The accuracy and computation cost for the Equation (4.4) when the scaling parameters of sine activation functions either in the first hidden layer or the last hidden layer are pre-fixed.*

Activation Function	Relative L^2 Error (Moving -Average)	Relative L^2 Error (Min)	Optimization Time (Per Iteration)
ReLU^3	3.09	0.99	6.73 e-01
$x \oplus x^2 \oplus \sin(x)$	1.61 e-02	1.57 e-02	1.08 e-01
$x \oplus x^2 \oplus \sin(x) \oplus$ Gaussian	**6.58 e-04**	**5.54 e-04**	1.28 e-01

Table 4.6: *The accuracy for the equation in (4.4) with solution* $u(\boldsymbol{x}) = \sin(40\pi x_1)\sin(40\pi x_2)$.

where $V(x) = -\frac{1}{c^2}\exp(\frac{2}{d}\sum_{i=1}^{d}\cos x_i) + \sum_{i=1}^{d}(\frac{\sin^2 x_i}{d^2} - \frac{\cos x_i}{d}) - 3$. Here, $\lambda = -3$ and $\varphi(x) = \exp(\frac{1}{d}\sum_{j=1}^{d}\cos(x_j))/c$ is the leading eigenpair of the operator $\mathcal{L}$. c is a positive constant such that $\int_\Omega \varphi^2(x)dx = |\Omega|$. We follow the approach in [26] to solve the leading eigenpair. The NN used in [26] consists of two parts: (1) the first hidden layer uses $\sin(x)$ and $\cos(x)$ with different frequencies so that the whole network satisfies periodic boundary conditions; (2) the other hidden layers use ReLU activation functions. We compare three activation functions, ReLU, ReLU^3, and poly-sine-Gaussian, after the first hidden layer. For $d = 5$, the relative L^2 errors are 0.307, $6.38e-03$, and $2.09e-03$, respectively. For $d = 10$, the relative L^2 errors are 0.223, 4.59 and $1.10e-03$, respectively. Figure 4.6(a) and 4.6(b) display the training curves for $d = 5$ and $d = 10$, respectively. One can see that poly-sine-Gaussian reaches a smaller minimum than ReLU, ReLU^3.



(a) Eigenpair $d = 5$ (b) Eigenpair $d = 10$ (c) SelectNet

Fig. 4.6: *Training process with different activation functions for the Equation (4.5) when (a)* $d = 5$ *and (b)* $d = 10$*, and (c) the Equation (4.6) using SelectNet.*

4.2.6. SelectNet. We consider solving high dimensional PDE using SelectNet [22]:

$$\begin{aligned} -\nabla(a(x)\nabla u)+\mu|u| &= f, \quad x\in\Omega=\{x:|x|<1\}, \\ u(x) &= g(x), \quad x\in\partial\Omega, \end{aligned} \tag{4.6}$$

with $a(x)=1+\frac{1}{2}|x|^2$. In this case, we specify the exact solution by

$$u(x)=\sin(\frac{\pi}{2}(1-|x|))^{2.5}. \tag{4.7}$$

Let $\mu=1$ and $d=10$. The problem is non-linear and defined in high dimensions. A DNN $\phi(\boldsymbol{x};\theta)$ is used to approximate the solution $u(\boldsymbol{x})$, and we also introduce a DNN $\phi_s(\boldsymbol{x},\theta_s)\in[m_1,m_2]$ for weighting. The loss function is defined as

$$\begin{aligned} \min_{\theta}\max_{\theta_s}\mathbf{L}(\theta,\theta_s) =& \mathbf{E}_{\boldsymbol{x}\in\Omega}[\phi_s(\boldsymbol{x},\theta_s)|-\nabla(a(x)\nabla\theta)+\mu|\theta|-f|^2] \\ &+\lambda\mathbf{E}_{\boldsymbol{x}\in\partial\Omega}[\phi_s(\boldsymbol{x},\theta_s)|\phi(\boldsymbol{x},\theta)-g(\boldsymbol{x})|^2] \\ &-\epsilon^{-1}(\frac{1}{|\Omega|}\int_{\Omega}\phi_s(\boldsymbol{x},\theta_s)-1)^2-\epsilon^{-1}(\frac{1}{|\partial\Omega|}\int_{\partial\Omega}\phi_s(\boldsymbol{x},\theta_s)-1)^2. \end{aligned} \tag{4.8}$$

The numerical result is shown in Table 4.7 and Figure 4.6(c). We compared the ReLU^3 and $x\oplus x^2\oplus\sin(x)\oplus$ Gaussian as activation functions under the condition of whether or not we use SelectNet. One can find $x\oplus x^2\oplus\sin(x)\oplus$ Gaussian cooperating with SelectNet gets the best result. The proposed activation functions can improve the accuracy by one digit while the computational cost only increases by a factor of 2.

Activation Function	Relative L^2 Error (Moving -Average)	Relative L^2 Error (Min)	Optimization Time (Per Iteration)
ReLU^3	7.74 e-4	6.88 e-04	**4.9 e-01**
$x\oplus x^2\oplus\sin(x)\oplus$ Gaussian	3.46 e-04	2.06 e-04	5.4 e-01
ReLU^3 + SelectNet	4.33 e-04	1.47 e-04	6.0 e-01
$x\oplus x^2\oplus\sin(x)\oplus$ Gaussian + SelectNet	**7.72 e-05**	**4.74 e-05**	7.9 e-01

Table 4.7: *The accuracy and computational cost for the Equation* (4.6).

4.2.7. Neural tangent kernel of PDE solvers. In Section 2.5 analysis, the condition number of NTK can provide a perspective for understanding the training behavior of deep learning. We summarize the condition numbers of the NTK matrices for different PDE problems at the NN initialization with various activation functions in Table 4.8. We evaluate the NTK matrix using 100 random samples, resulting in a 100×100 size matrix. As shown in the table, the NTK matrices associated with the poly-sine-Gaussian activation function exhibit the smallest condition numbers. This finding supports the improved training convergence of combining basic activation functions in the poly-sine-Gaussian activation function from the perspective of the NTK theory.

5. Conclusion

We propose RAF and its approximation theory. NNs with this activation function can reproduce traditional approximation tools (e.g., polynomials, Fourier basis functions, wavelets, radial basis functions) and approximate a certain class of functions

Activation	Eqn. (4.2)	Eqn. (4.3)	Eqn. (4.4)
ReLU^3	1.32 e+11	2.60 e+10	3.23 e+11
$x \oplus x^2$	4.71 e+11	1.74 e+11	4.28 e+11
$x \oplus x^2 \oplus \mathrm{ReLU}$	1.01 e+11	1.09 e+10	3.11 e+10
$x \oplus x^2 \oplus \mathrm{ReLU}^3$	2.03 e+12	1.65 e+11	3.45 e+11
$x \oplus x^2 \oplus \sin(x)$	1.92 e+12	5.18 e+10	1.10 e+11
$x \oplus x^2 \oplus \sin(x) \oplus$ Gaussian	**3.91 e+08**	**4.11 e+09**	**1.36 e+10**

Table 4.8: *The condition number of the NTK matrix in (2.15) of PDE solvers at the NN initialization.*

with exponential and dimension-independent approximation rates. We have numerically demonstrated that RAFs can generate neural tangent kernels with a better condition number than traditional activation functions, which provides a prospective for understanding the improved optimization convergence using the theory of neural tangent kernel. Extensive experiments on coordinate-based data representation and PDEs demonstrate the effectiveness of the proposed activation function. We have not explored the optimal choice of basic activation functions in this paper, which would be problem-dependent and is left for future work.

Acknowledgments. C.W. is partially supported by the US National Science Foundation CAREER Award DMS-1849483. H.Y. was partially supported by the US National Science Foundation CAREER Award DMS-1945029, DMS-2206333, and ONR Young Investigator Award. S. L. acknowledges the support of the Ross-Lynn fellowship of Purdue University.

Appendix A. Proof of Theorem 3.3. The proof of Theorem 3.3 relies on the following lemma.

LEMMA A.1.

(i) *An identity map in $\mathbb{R}^d$ can be realized exactly by a poly-sine-Gaussian network with one hidden layer and d neurons.*

(ii) *$f(x)=x^2$ can be realized exactly by a poly-sine-Gaussian network with one hidden layer and one neuron.*

(iii) *$f(x,y)=xy=\frac{(x+y)^2-(x-y)^2}{4}$ can be realized exactly by a poly-sine-Gaussian network with one hidden layer and two neurons.*

(iv) *Assume $P(\boldsymbol{x})=\boldsymbol{x}^{\boldsymbol{\alpha}}=x_1^{\alpha_1}x_2^{\alpha_2}\cdots x_d^{\alpha_d}$ for $\boldsymbol{\alpha}\in\mathbb{N}^d$. For any $N,L\in\mathbb{N}^+$ such that $NL+2^{\lfloor \log_2 N\rfloor}\geq|\boldsymbol{\alpha}|$, there exists a poly-sine-Gaussian network ϕ with width $2N+d$ and depth $L+\lceil \log_2 N\rceil$ such that*

$$\phi(\boldsymbol{x})=P(\boldsymbol{x}) \quad \text{for any } \boldsymbol{x}\in\mathbb{R}^d.$$

Proof. Part (i) to (iii) are trivial. We will only prove Part (iv). In the case of $|\boldsymbol{\alpha}|=k\leq 1$, the proof is simple and left for the reader. When $|\boldsymbol{\alpha}|=k\geq 2$, the main idea of the proof of (v) can be summarized in Figure A.1. By Part (i), we can apply a poly-sine-Gaussian network to implement a d-dimensional identity map. This identity map maintains necessary entries of $\boldsymbol{x}$ to be multiplied together. We apply poly-sine-Gaussian networks to implement the multiplication function in Part (iii) and carry out the multiplication N times per layer. After L layers, there are $k-NL\leq N$ multiplications to be implemented. Finally, these at most N multiplications can be carried out with a small poly-sine-Gaussian network in a dyadic tree structure. ∎

Now we are ready to prove Theorem 3.3.

Proof. **(Proof of Theorem 3.3.)** The main idea of the proof is to apply Part (iv) of Lemma A.1 J times to construct J poly-sine-Gaussian networks, $\{\phi_j(\boldsymbol{x})\}_{j=1}^J$, to represent $\boldsymbol{x}^{\boldsymbol{\alpha}_j}$ and arrange these poly-sine-Gaussian networks as subnetwork blocks to form a larger poly-sine-Gaussian network $\tilde{\phi}(\boldsymbol{x})$ with ab blocks as shown in Figure A.2, where each red rectangle represents one poly-sine-Gaussian network $\phi_j(\boldsymbol{x})$ and each blue rectangle represents one poly-sine-Gaussian network of width 1 as an identity map of $\mathbb{R}$. There are ab red blocks with a rows and b columns. When $ab \geq J$, these subnetwork blocks can carry out all monomials $\boldsymbol{x}^{\boldsymbol{\alpha}_j}$. In each column, the results of the multiplications of $\boldsymbol{x}^{\boldsymbol{\alpha}_j}$ are added up to the input of the narrow poly-sine-Gaussian network, which can carry the sum over to the next column. After the calculation of b columns, J additions of the monomials $\boldsymbol{x}^{\boldsymbol{\alpha}_j}$ have been implemented, resulting in the output $P(\boldsymbol{x})$.

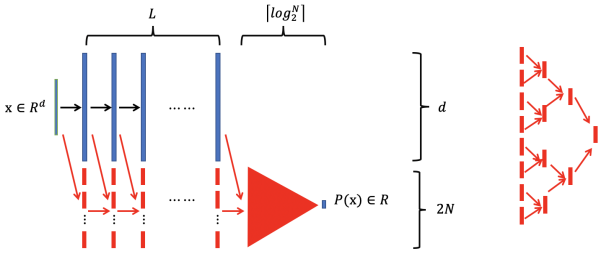


Fig. A.1: *Left: An illustration of the proof of Lemma A.1 (iv). Green vectors represent the input and output of the poly-sine-Gaussian network carrying out $P(\boldsymbol{x})$. Blue vectors represent the poly-sine-Gaussian network that implements a d-dimensional identity map in Part (i), which was repeatedly applied for L times. Black arrows represent the data flow for carrying out the identity maps. Red vectors represent the poly-sine-Gaussian networks implementing the multiplication function in Part (iii) and there are NL such red vectors. Red arrows represent the data flow for carrying out the multiplications. Finally, a red triangle represents a poly-sine-Gaussian network of width at most $2N$ and depth at most $\lceil \log_2^N \rceil$ carrying out the rest of the multiplications. Right: An example of the red triangle is given on the right when it consists of 15 red vectors carrying out 15 multiplications.*

By Part (iv) of Lemma A.1, for any $N \in \mathbb{N}^+$, there exists a poly-sine-Gaussian network $\phi_j(\boldsymbol{x})$ of width $d+2N$ and depth $L_j = \lceil \frac{|\boldsymbol{\alpha}_j|}{N} \rceil + \lceil \log_2 N \rceil$ to implement $\boldsymbol{x}^{\boldsymbol{\alpha}_j}$. Since

$$b \max_j L_j \leq b\left(\frac{\max_j |\boldsymbol{\alpha}_j|}{N} + 2 + \log_2 N\right),$$

there exists a poly-sine-Gaussian network $\tilde{\phi}(\boldsymbol{x})$ of depth $b\left(\frac{\max_j |\boldsymbol{\alpha}_j|}{N} + 2 + \log_2 N\right)$ and width $da + 2Na + 1$ to implement $P(\boldsymbol{x})$ as in Figure A.2. Note that the total width of each column of blocks is $ad + 2Na + 1$ but in fact, this width can be reduced to $d + 2Na + 1$, since the red blocks in each column can share the same identity map of $\mathbb{R}^d$ (the blue part of Figure A.1). Note that $b\left(\frac{\max_j |\boldsymbol{\alpha}_j|}{N} + 2 + \log_2 N\right) \leq L$ is equivalent to $(L - 2b - b\log_2 N)N \geq b \max_j |\boldsymbol{\alpha}_j|$. Hence, for any $N, L, a, b \in \mathbb{N}^+$ such that $ab \geq J$ and $(L - 2b - b\log_2 N)N \geq b \max_j |\boldsymbol{\alpha}_j|$, there exists a poly-sine-Gaussian network $\phi(\boldsymbol{x})$ with width $2Na + d + 1$ and depth L such that $\tilde{\phi}(\boldsymbol{x})$ is a subnetwork of $\phi(\boldsymbol{x})$ in the sense of $\phi(\boldsymbol{x}) = \mathrm{Id} \circ \tilde{\phi}(\boldsymbol{x})$ with Id as an identify map of $\mathbb{R}$, which means that $\phi(\boldsymbol{x}) = \tilde{\phi}(\boldsymbol{x}) = P(\boldsymbol{x})$. The proof of Part (v) is completed.

□

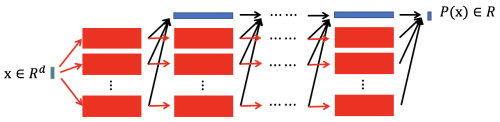


Fig. A.2: *An illustration of the proof of Theorem 3.3. Green vectors represent the input and output of the poly-sine-Gaussian network $\phi(\boldsymbol{x})$ carrying out $P(\boldsymbol{x})$. Each red rectangle represents one poly-sine-Gaussian network $\phi_j(\boldsymbol{x})$ and each blue rectangle represents one poly-sine-Gaussian network of width 1 as an identity map of $\mathbb{R}$. There are $ab \geq J$ red blocks with a rows and b columns. When $ab \geq J$, these subnetwork blocks can carry out all monomials $\boldsymbol{x}^{\boldsymbol{\alpha}_j}$. In each column, the results of the multiplications of $\boldsymbol{x}^{\boldsymbol{\alpha}_j}$ are added up to (indicated by black arrows) the input of the narrow poly-sine-Gaussian network, which can carry the sum over to the next column. Each red arrow passes $\boldsymbol{x}$ to the next red block. After the calculation of b columns, J additions of the monomials $\boldsymbol{x}^{\boldsymbol{\alpha}_j}$ have been implemented, resulting in the output $P(\boldsymbol{x})$.*

Appendix B. Proof of Theorem 3.4.

Proof. **(Proof of Theorem 3.4.)** Let $M \geq 1$, $s > 1$, $C_f > 0$ and $0 < \epsilon < 1$ be four scalars, and f be an analytic function defined on $[-M, M]$ that is analytically continuable to the open Bernstein s-ellipse E_s^M, where it satisfies $|f(x)| \leq C_f$. We first approximate f by a truncated Chebyshev series f_n, and then approximate f_n by a poly-sine-Gaussian network ϕ using Theorem 3.3.

Since f is analytic in the open Bernstein s-ellipse E_s^M then, for any integer $n \geq 2$,

$$\|f_n(x) - f(x)\|_{L^\infty([-M,M])} \leq \frac{2C_f s^{-n}}{s-1} = \mathcal{O}\left(C_f s^{-n}\right).$$

Therefore, if we take $n = \mathcal{O}\left(\frac{1}{\log_2 s} \log_2 \frac{2C_f}{\epsilon}\right)$, then the above term is bounded by ϵ.

Let us now approximate f_n by a poly-sine-Gaussian network ϕ. We first write $f_n(x) = \sum_{k=0}^n c_k T_k\left(\frac{x}{M}\right)$, with

$$\max_{0 \leq k \leq n} |c_k| = \mathcal{O}\left(C_f s\right), \text{via Theorem 8.1 in [72].} \tag{B.1}$$

Since, f_n is a polynomial of degree n, by Theorem 3.3 with $d = 1$, $a = 1$, and $b = n+1$, there exists a poly-sine-Gaussian network ϕ with width $2N+2$ and depth L such that

$$\phi(x) = f_n(x)$$

for $x \in \mathbb{R}$, as long as N and L satisfy $(L - 2n - 2 - (n+1)\log_2 N)N \geq n(n+1)$. This yields

$$|\phi(x) - f(x)| = |f_n(x) - f(x)| \leq \epsilon.$$

∎

Appendix C. Proof of Theorem 3.5.

To show the approximation of poly-sine-Gaussian networks to generalized bandlimited functions, we will need Maurey's unpublished theorem below. It was used to study shallow network approximation by Barron in [2].

THEOREM C.1 (Maurey's theorem). *Let H be a Hilbert space with norm $\|\cdot\|$. Suppose there exists $G \subset H$ such that for every $g \in G$, $\|g\| \leq b$ for some $b > 0$. Then, for every f in the convex hull of G and every integer $n \geq 1$, there is a f_n in the convex hull of n points in G and a constant $c > b^2 - \|f\|^2$ such that $\|f - f_n\|^2 \leq \frac{c}{n}$.*

Proof. **(Proof of Theorem 3.5.)** Let f be an arbitrary function in $\mathcal{H}_{K,M}$, and μ be an arbitrary measure. Let $F(\boldsymbol{w}) = |F(\boldsymbol{w})| e^{i\theta(\boldsymbol{w})}$. Since f is real-valued, we may write

$$\begin{aligned} f(\boldsymbol{x}) &= \mathrm{Re}\left(\int_{\mathbb{R}^d} C_F e^{i\theta(\boldsymbol{w})} K(\boldsymbol{w}\cdot\boldsymbol{x}) \frac{|F(\boldsymbol{w})|}{C_F} d\boldsymbol{w} \right) \\ &= \int_{[-M,M]^d} C_F \left[\cos(\theta(\boldsymbol{w})) K_R(\boldsymbol{w}\cdot\boldsymbol{x}) - \sin(\theta(\boldsymbol{w})) K_I(\boldsymbol{w}\cdot\boldsymbol{x}) \right] \frac{|F(\boldsymbol{w})|}{C_F} d\boldsymbol{w}, \end{aligned}$$

where $K_R(\boldsymbol{w}\cdot\boldsymbol{x}) = \mathrm{Re}(K(\boldsymbol{w}\cdot\boldsymbol{x}))$ and $K_I(\boldsymbol{w}\cdot\boldsymbol{x}) = \mathrm{Im}(K(\boldsymbol{w}\cdot\boldsymbol{x}))$. The integral above represents f as an infinite convex combination of functions in the set

$$G_{K,M} = \left\{ \gamma \left[\cos(\beta)\mathrm{Re}(K(\boldsymbol{w}\cdot\boldsymbol{x})) - \sin(\beta)\mathrm{Im}(K(\boldsymbol{w}\cdot\boldsymbol{x})) \right], |\gamma| \le C_F, \beta \in \mathbb{R}, \boldsymbol{w} \in [-M,M]^d \right\}.$$

Therefore, f is in the closure of the convex hull of $G_{K,M}$. Since functions in $G_{K,M}$ are bounded in the $L^2(\mu,B)$-norm by $2C_F D_K \sqrt{\mu(B)} \le 2C_F \sqrt{\mu(B)}$, Theorem C.1 tells us that there exist real coefficients b_j's and β_j's such that[1]

$$f_{\epsilon_0}(\boldsymbol{x}) = \sum_{j=1}^{\lceil 1/\epsilon_0^2 \rceil} b_j \left[\cos(\beta_j) K_R(\boldsymbol{w}\cdot\boldsymbol{x}) - \sin(\beta_j) K_I(\boldsymbol{w}\cdot\boldsymbol{x}) \right], \quad \sum_{j=1}^{\lceil 1/\epsilon_0^2 \rceil} |b_j| \le C_F,$$

for some $0 < \epsilon_0 < 1$ to be determined later, such that $\|f_{\epsilon_0}(\boldsymbol{x}) - f(\boldsymbol{x})\|_{L^2(\mu,B)} \le 2C_F \sqrt{\mu(B)} \epsilon_0$.

We now approximate $f_{\epsilon_0}(\boldsymbol{x})$ by a poly-sine-Gaussian network $\phi(\boldsymbol{x})$. Note that K_R and K_I are both analytic and satisfy the same assumptions as K. Using Theorem 3.4, they can be approximated to accuracy ϵ_0 using networks $\widetilde{K}_R$ and $\widetilde{K}_I$ of width and depth

$$\mathcal{O}\left(\frac{1}{\log_2 s} \log_2 \frac{C_K}{\epsilon_0} \right) \quad \text{and} \quad \mathcal{O}\left(\left(\frac{1}{\log_2 s} \log_2 \frac{C_K}{\epsilon_0} \right) \log_2 \log_2 \frac{C_K}{\epsilon_0} \right),$$

respectively. We define the poly-sine-Gaussian network $\phi(\boldsymbol{x})$ by

$$\phi(\boldsymbol{x}) = \sum_{j=1}^{\lceil 1/\epsilon_0^2 \rceil} b_j \left[\cos(\beta_j) \widetilde{K}_R(\boldsymbol{w}\cdot\boldsymbol{x}) - \sin(\beta_j) \widetilde{K}_I(\boldsymbol{w}\cdot\boldsymbol{x}) \right].$$

This network has width $\mathcal{O}\left(\frac{1}{\epsilon_0^2 \log_2 s} \log_2 \frac{C_K}{\epsilon_0} \right)$ and depth $\mathcal{O}\left(\left(\frac{1}{\log_2 s} \log_2 \frac{C_K}{\epsilon_0} \right) \log_2 \log_2 \frac{C_K}{\epsilon_0} \right)$, and

$$|\phi(\boldsymbol{x}) - f_{\epsilon_0}(\boldsymbol{x})| \le \sum_{j=1}^{\lceil \frac{1}{\epsilon_0^2} \rceil} |b_j| |\widetilde{K}_R(\boldsymbol{w}_j\cdot\boldsymbol{x}) - K_R(\boldsymbol{w}_j\cdot\boldsymbol{x})| + \sum_{j=1}^{\lceil \frac{1}{\epsilon_0^2} \rceil} |b_j| |\widetilde{K}_I(\boldsymbol{w}_j\cdot\boldsymbol{x}) - K_I(\boldsymbol{w}_j\cdot\boldsymbol{x})| \le 2C_F \epsilon_0,$$

which yields

$$\|\phi(\boldsymbol{x}) - f_{\epsilon_0}(\boldsymbol{x})\|_{L^2(\mu,B)} \le 2C_F \sqrt{\mu(B)} \epsilon_0.$$

The total approximation error satisfies

$$\|\phi(\boldsymbol{x}) - f(\boldsymbol{x})\|_{L^2(\mu,B)} \le 4C_F \sqrt{\mu(B)} \epsilon_0.$$

[1] We use Theorem C.1 with $b = 2C_F \sqrt{\mu(B)}$, $c = b^2 > b^2 - \|f\|^2$, and $\|\cdot\| = \|\cdot\|_{L^2(\mu,B)}$.

We take

$$\epsilon_0 = \frac{\epsilon}{4C_F\sqrt{\mu(B)}}$$

to complete the proof. ∎

Appendix D. Proof of Lemma 3.1.

Proof. **(Proof of Lemma 3.1.)** The proof of this lemma is simple by three facts: (1) the affine linear transforms before activation functions can play the role of translation and dilation in the spatial and Fourier domains; (2) the Gaussian activation function plays the role of localization in the transforms in this lemma; (3) Lemma A.1 shows that the x^2 activation function can reproduce multiplication. ∎

Appendix E. Proof of Lemma 3.2.

Proof. **(Proof of Lemma 3.2.)** The proof of this lemma is trivial by Lemma A.1, Theorem 3.3, and the proof of Theorem 3.4. ∎

REFERENCES

[1] S. Arora, S. Du, W. Hu, Z. Li, and R. Wang, *Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks*, Proc. Int. Conf. Mach. Learn., 97:322–332, 2019. 6, 7

[2] A.R. Barron, *Universal approximation bounds for superpositions of a sigmoidal function*, IEEE Trans. Inf. Theory, 39(3):930–945, 1993. 1, 3, 12, 25

[3] J. Berg and K. Nyström, *A unified deep artificial neural network approach to partial differential equations in complex geometries*, Neurocomputing, 317:28–41, 2018. 1

[4] J. Berner, P. Grohs, and A. Jentzen, *Analysis of the generalization error: Empirical risk minimization over deep artificial neural networks overcomes the curse of dimensionality in the numerical approximation of Black-Scholes partial differential equations*, SIAM J. Math. Data Sci., 2(3):631–657, 2020. 1

[5] N. Boullé, Y. Nakatsukasa, and A. Townsend, *Rational neural networks*, Adv. Neur. Inf. Process. Syst., 33:14243–14253, 2020. 16, 17

[6] W. Cai, X. Li, and L. Liu, *A phase shift deep neural network for high frequency approximation and wave problems*, SIAM J. Sci. Comput., 42(5):A3285–A3312, 2020. 2, 3

[7] Z. Cai, J. Chen, M. Liu, and X. Liu, *Deep least-squares methods: An unsupervised learning-based numerical method for solving elliptic PDEs*, J. Comput. Phys., 420:109707, 2020. 2

[8] Y. Cao, Z. Fang, Y. Wu, D.-X. Zhou, and Q. Gu, *Towards understanding the spectral bias of deep learning*, Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, 2205– 2211, 2021. 2, 6, 7

[9] J. Chen, R. Du, P. Li, and L. Lyu, *Quasi-Monte Carlo sampling for machine-learning partial differential equations*, Numer. Math. Theor. Meth. Appl., 14:377–404, 2021. 2

[10] Z. Chen and H. Zhang, *Learning implicit fields for generative shape modeling*, 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 5932–5941, 2019. 2

[11] Z. Chen, Y. Cao, D. Zou, and Q. Gu, *How much over-parameterization is sufficient to learn deep ReLU networks?*, arXiv preprint, arXiv:1911.12360, 2019. 6

[12] X. Dai and Y. Zhu, *Towards theoretical understanding of large batch training in stochastic gradient descent*, arXiv preprint, arXiv:1812.00542, 2018. 6

[13] M.W.M.G. Dissanayake and N. Phan-Thien, *Neural-network-based approximations for solving partial differential equations*, Commun. Numer. Meth. Eng., 10:195–201, 1994. 4

[14] S.S. Du, X. Zhai, B. Poczos, and A. Singh, *Gradient descent provably optimizes over-parameterized neural networks*, arXiv preprint, arXiv:1810.02054, 2018. 6

[15] E. Dupont, A. Golinski, M. Alizadeh, Y.W. Teh, and A. Doucet, *COIN: COmpression with implicit neural representations*, Neural Compression: From Information Theory to Applications–Workshop @ ICLR 2021, 2021. 14

[16] W. E and Q. Wang, *Exponential convergence of the deep neural network approximation for analytic functions*, Sci. China Math., 61:1733–1740, 2018. 1

[17] W. E, C. Ma, and Q. Wang, *A priori estimates of the population risk for residual networks*, arXiv preprint, arXiv.1903.02154, 2019. 1

[18] W. E, C. Ma, and L. Wu, *A priori estimates of the population risk for two-layer neural networks*, Commun. Math. Sci., 17(5):1407–1425, 2019. 1
[19] W. E and B. Yu, *The deep Ritz method: A deep learning-based numerical algorithm for solving variational problems*, Commun. Math. Stat., 6:1–12, 2018. 4
[20] K. Genova, F. Cole, A. Sud, A. Sarna, and T. Funkhouser, *Local deep implicit functions for 3D shape*, 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 4856–4865, 2020. 2
[21] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, Cambridge, 2016. 1, 3
[22] Y. Gu, H. Yang, and C. Zhou, *SelectNet: Self-paced learning for high-dimensional partial differential equations*, J. Comput. Phys., 441(15):110444, 2021. 2, 5, 22
[23] Y. Gu, C. Wang, and H. Yang, *Structure probing neural network deflation*, J. Comput. Phys., 434:110231, 2021. 2, 5
[24] J. Han, A. Jentzen, and W. E, *Solving high-dimensional partial differential equations using deep learning*, Proc. Natl. Acad. Sci., 115(34):8505–8510, 2018. 1
[25] J. Han and J. Long, *Convergence of the deep BSDE method for coupled FBSDEs*, Probab. Uncertain. Quant. Risk, 5:5, 2020. 1
[26] J. Han, J. Lu, and M. Zhou, *Solving high-dimensional eigenvalue problems using deep neural networks: A diffusion Monte Carlo like approach*, J. Comput. Phys., 423:109792, 2020. 10, 16, 21
[27] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770–778, 2016. 4
[28] J. Huang, H. Wang, and H. Yang, *Int-deep: A deep learning initialized iterative method for nonlinear problems*, J. Comput. Phys., 419:109675, 2020. 2
[29] M. Hutzenthaler, A. Jentzen, Th. Kruse, and T.A. Nguyen, *A proof that rectified deep neural networks overcome the curse of dimensionality in the numerical approximation of semilinear heat equations*, Technical Report 2019-10, Seminar for Applied Mathematics, ETH Zürich, Switzerland, 2019. 1
[30] A. Jacot, F. Gabriel, and C. Hongler, *Neural tangent kernel: Convergence and generalization in neural networks*, Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing, 2021. 6, 7
[31] A.D. Jagtap, K. Kawaguchi, and G. Em Karniadakis, *Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks*, Proc. Roy. Soc. A Math. Phys. Eng. Sci., 476(2239):2020.0334, 2020. 3, 16
[32] A.D. Jagtap, K. Kawaguchi, and G. Em Karniadakis, *Adaptive activation functions accelerate convergence in deep and physics-informed neural networks*, J. Comput. Phys., 404:109136, 2020. 3, 16
[33] T. Jeruzalski, B. Deng, M. Norouzi, J.P. Lewis, G. Hinton, and A. Tagliasacchi, *NASA: Neural articulated shape approximation*, European Conference on Computer Vision, 612–628, 2020. 2
[34] S. Justin and S. Konstantinos, *DGM: A deep learning algorithm for solving partial differential equations*, J. Comput. Phys., 375:1339–1364, 2018. 1
[35] Y. Khoo, J. Lu, and L. Ying, *Solving Parametric PDE Problems with Artificial Neural Networks*, Cambridge University Press, 2021. 1
[36] D.P. Kingma and J. Ba, *Adam: a method for stochastic optimization*, arXiv preprint, arXiv:1412.6980, 2014. 5, 16
[37] V. Kůrková, *Kolmogorov's theorem and multilayer neural networks*, Neural Netw., 5:501–506, 1992. 3
[38] I.E. Lagaris, A. Likas, and D.I. Fotiadis, *Artificial neural networks for solving ordinary and partial differential equations*, IEEE Trans. Neural Netw., 9:987–1000, 1998. 2, 4, 5
[39] J. Lee, L. Xiao, S.S. Schoenholz, Y. Bahri, R. Novak, J. Sohl-Dickstein, and J. Pennington, *Wide neural networks of any depth evolve as linear models under gradient descent*, J. Stat. Mech. Theory Exp., 2020(12):124002, 2020. 6, 7
[40] D. Lei, Z. Sun, Y. Xiao, and W.Y. Wang, *Implicit regularization of stochastic gradient descent in natural language processing: Observations and implications*, arXiv preprint, arXiv:1811.00659, 2018. 6
[41] S. Liang and R. Srikant, *Why deep neural networks for function approximation?*, arXiv preprint, arXiv:1610.04161, 2016. 1
[42] Y. Liao and P. Ming, *Deep Nitsche method: Deep Ritz method with essential boundary conditions*, Commun. Comput. Phys., 29:1365–1384, 2021. 4
[43] S. Liu, Y. Zhang, S. Peng, B. Shi, M. Pollefeys, and Z. Cui, *DIST: Rendering deep implicit signed distance function with differentiable sphere tracing*, 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2016–2025, 2020. 2
[44] Z. Liu, W. Cai, and Z.-Q.J. Xu, *Multi-scale deep neural network (MscaleDNN) for solving Poisson-*

Boltzmann equation in complex domains, Commun. Comput. Phys., 28(5):1970–2001, 2020. 1, 2, 10

[45] J. Lu, Z. Shen, H. Yang, and S. Zhang, *Deep network approximation for smooth functions*, SIAM J. Math. Anal., 53(5):5465–5506, 2021. 1, 3, 11

[46] T. Luo, Z. Ma, Z.J. Xu, and Y. Zhang, *Theory of the frequency principle for general deep neural networks*, CSIAM Trans. Appl. Math., 2(3):484–507, 2021. 6

[47] T. Luo and H. Yang, *Two-layer neural networks for partial differential equations: optimization and generalization theory*, arXiv preprint, arXiv:2006.15733, 2020. 1, 6, 7

[48] L. Lyu, K. Wu, R. Du, and J. Chen, *Enforcing exact boundary and initial conditions in the deep mixed residual method*, CSIAM Trans. Appl. Math., 2:748–775, 2021. 2, 5

[49] F. Manessi and A. Rozza, *Learning combinations of activation functions*, 2018 IEEE International Conference on Pattern Recognition (ICPR), 61–66, 2018. 3

[50] M. Michalkiewicz, J.K. Pontes, D. Jack, M. Baktashmotlagh, and A. Eriksson, *Implicit surface representations as layers in neural networks*, 2019 IEEE/CVF International Conference on Computer Vision (ICCV), 4742–4751, 2019. 2

[51] B. Mildenhall, P.P. Srinivasan, M. Tancik, J.T. Barron, R. Ramamoorthi, and R. Ng, *NeRF: Representing scenes as neural radiance fields for view synthesis*, in A. Vedaldi, H. Bischof, T. Brox, and J.M. Frahm (eds.), Computer Vision – ECCV 2020, Lecture Notes in Computer Science, vol 12346. Springer, Cham. 405–421, 2020. 10

[52] H. Montanelli and Q. Du, *New error bounds for deep networks using sparse grids*, SIAM J. Math. Data Sci., 1(1):78–92, 2019. 1

[53] H. Montanelli and H. Yang, *Error bounds for deep ReLU networks using the Kolmogorov-Carnold superposition theorem*, Neural Netw., 129:1–6, 2020. 1

[54] H. Montanelli, H. Yang, and Q. Du, *Deep ReLU networks overcome the curse of dimensionality for bandlimited functions*, J. Comp. Math., 39:801–815, 2021. 1, 11, 12

[55] T. Nakamura-Zimmerer, Q. Gong, and W. Kang, *Adaptive deep learning for high dimensional Hamilton-Jacobi-Bellman equations*, SIAM J. Sci. Comput., 43(2):A1221–A1247, 2021. 2

[56] B. Neyshabur, R. Tomioka, R. Salakhutdinov, and N. Srebro, *Geometry of optimization and implicit regularization in deep learning*, arXiv preprint, arXiv:1705.03071, 2017. 6

[57] J.A.A. Opschoor, C. Schwab, and J. Zech, *Exponential ReLU DNN expression of holomorphic maps in high dimension*, Constr. Approx., 55:537–582, 2022. 1

[58] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, *DeepSDF: Learning continuous signed distance functions for shape representation*, 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 165–174, 2019. 2

[59] T. Poggio, H.N. Mhaskar, L. Rosasco, B. Miranda, and Q. Liao, *Why and when can deep–but not shallow–networks avoid the curse of dimensionality: A review*, Inter. J. Auto. Comput., 14:503–519, 2017. 1

[60] S. Qian, H. Liu, C. Liu, S. Wu, and H.S. Wong, *Adaptive activation functions in convolutional neural networks*, Neurocomputing, 272:204–212, 2018. 3

[61] 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., 378:686–707, 2019. 1

[62] S. Saito, Z. Huang, R. Natsume, S. Morishima, A. Kanazawa, and H. Li, *PiFU: Pixel-aligned implicit function for high-resolution clothed human digitization*, 2019 IEEE/CVF International Conference on Computer Vision (ICCV), 2304–2314, 2019. 2

[63] Z. Shen, H. Yang, and S. Zhang, *Deep network approximation characterized by number of neurons*, Commun. Comput. Phys., 28:1768–1811, 2020. 3

[64] Z. Shen, H. Yang, and S. Zhang, *Neural network approximation: Three hidden layers are enough*, 141:160–173, 2021. 2, 10

[65] Z. Shen, H. Yang, and S. Zhang, *Deep network with approximation error being reciprocal of width to power of square root of depth*, Neural Comput., 33(4):1005–1036, 2021. 1, 2, 3, 10

[66] Y. Shin, J. Darbon, and G. Karniadakis, *On the convergence and generalization of physics informed neural networks*, Commun. Comput. Phys., 28:2042–2074, 2020. 1

[67] V. Sitzmann, J. Martel, A. Bergman, D. Lindell, and G. Wetzstein, *Implicit neural representations with periodic activation functions*, Adv. Neural Inf. Process. Syst., 626:7462–7473, 2020. 14

[68] V. Sitzmann, M. Zollhöfer, and G. Wetzstein, *Scene representation networks: Continuous 3D-structure-aware neural scene representations*, Neur. Inf. Process. Syst., 2019. 2

[69] Y. Strümpler, J. Postels, R. Yang, L. Van Gool, and F. Tombari, *Implicit neural representations for image compression*, ECCV 2022: 17th European Conference, Springer, 74–91, 2022. 14

[70] L. René Sütfeld, F. Brieger, H. Finger, S. Füllhase, and G. Pipa, *Adaptive blending units: Trainable activation functions for deep neural networks*, Science and Information Conference, Springer, 37–50, 2020. 3

[71] M. Tancik, P.P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J.T. Barron, and R. Ng, *Fourier features let networks learn high frequency functions in low dimensional domains*, Neur. Inf. Process. Syst., 33:7537–7547, 2020. 2, 3, 10, 14, 19

[72] L.N. Trefethen, *Approximation Theory and Approximation Practice*, Other Titles in Applied Mathematics, SIAM, 2013. 11, 25

[73] B. Wang, *Multi-scale deep neural network (MscaleDNN) methods for oscillatory Stokes flows in complex domains*, Commun. Comput. Phys., 28(5):2139–2157, 2020. 10

[74] S. Wang, X. Yu, and P. Perdikaris, *When and why PINNs fail to train: A neural tangent kernel perspective*, J. Comput. Phys., 449:110768, 2022. 6

[75] Z.-Q.J. Xu, Y. Zhang, T. Luo, Y. Xiao, and Z. Ma, *Frequency principle: Fourier analysis sheds light on deep neural networks*, Commun. Comput. Phys., 28(5):1746–1767, 2020. 2, 6

[76] D. Yarotsky, *Error bounds for approximations with deep ReLU networks*, Neural Netw., 94:103–114, 2017. 1, 3, 11

[77] D. Yarotsky, *Optimal approximation of continuous functions by very deep ReLU networks*, Proceedings of the 31st Conference On Learning Theory, 75:639–649, 2018. 3

[78] D. Yarotsky and A. Zhevnerchuk, *The phase diagram of approximation rates for deep neural networks*, Neur. Inf. Process. Syst., 33:13005–13015, 2020. 1, 2, 9

[79] Z. Song, Z.A.-Zhu, and Y. Li, *A convergence theory for deep learning via over-parameterization*, Proceedings of the 36th International Conference on Machine Learning, 97:242–252, 2019. 6

[80] Y. Zang, G. Bao, X. Ye, and H. Zhou, *Weak adversarial networks for high-dimensional partial differential equations*, J. Comput. Phys., 411:109409, 2020. 1

[81] E.D. Zhong, T. Bepler, J.H. Davis, and B. Berger, *Reconstructing continuous distributions of 3D protein structure from cryo-EM images*, arXiv preprint, arXiv:1909.05215, 2020. 9