REPRODUCING ACTIVATION FUNCTION FOR DEEP LEARNING

SENWEI LIANG

DEPARTMENT OF MATHEMATICS, PURDUE UNIVERSITY, WEST LAFAYETTE, IN 47907, USA (LIANG339@PURDUE.EDU)

LIYAO LYU

DEPARTMENT OF COMPUTATIONAL MATHEMATICS, SCIENCE, AND ENGINEERING, MICHIGAN STATE UNIVERSITY, EAST LANSING, MI, 48824, USA (LYULIYAO@MSU.EDU)

CHUNMEI WANG

DEPARTMENT OF MATHEMATICS & STATISTICS, TEXAS TECH UNIVERSITY, 1108 MEMORIAL CIRCLE, LUBBOCK, TX 79409, USA (CHUNMEI.WANG@TTU.EDU)

HAIZHAO YANG

DEPARTMENT OF MATHEMATICS, PURDUE UNIVERSITY, WEST LAFAYETTE, IN 47907, USA (HAIZHAO@PURDUE.EDU)

Abstract. In this paper, we propose the reproducing activation function to improve deep learning accuracy for various applications ranging from computer vision problems to scientific computing problems. 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 and, therefore, approximate target functions with a smaller number of parameters than traditional neural networks. In terms of training dynamics of deep learning, reproducing activation functions can generate neural tangent kernels with a better condition number than traditional activation functions lessening the spectral bias of deep learning. As demonstrated by extensive numerical tests, the proposed activation function 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.

Key words. Deep Neural Network; Activation Function; Approximation; Neural Tangent Kernel; Partial Differential Equation; Data Reconstruction.

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, not limited to machine learning, 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, 35, 25, 46, 80, 62, 36]. First of all, as a form of function approximation via the compositions of nonlinear functions [22], deep neural networks (DNNs) as a mesh-free parametrization can efficiently approximate various useful functions lessening the curse of dimensionality [2, 53, 19, 18, 55, 61, 78, 47, 54, 30, 67] and achieving exponential approximation rates [76, 55, 47, 43, 17, 59, 67]. Therefore, it might be computationally affordable to apply DNNs with a small number of parameters to approximate the solution of a high-dimensional PDE. 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. Popular choices of the variational formulation include the residual method in the least-squares sense [14, 39] as a special case of variational formulation, the Ritz method [20], and the Nitsche method [44]. It is believed that the implicit regularization of SGD alleviates the difficulty of solving a certain class of nonlinear PDEs and helps to obtain approximate solutions, though current optimization analysis [49] and generalization analysis [26, 4, 68, 49] are limited to the case of linear PDEs. Finally, it is simple to apply neural network-based optimization to solve

PDEs defined in complicated domains, while traditional methods require profound expertise and programming skills to discretize the domain.

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 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 [39, 24, 50]. Applying first-order methods to reformulate high-order PDEs can reduce the difficulty of neural network optimization [7, 50]. Improving the sampling strategy of SGD [56, 9] or the sample weights in the objective function [23] 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 [46]; and other spectral structures [24]. Finally, hybrid algorithms combining neural network-based solvers and traditional iterative solvers can provide highly accurate solutions to low-dimensional nonlinear PDEs efficiently [75, 29].

Though high-dimensional problems are the main application domains of deep learning with a superpower, 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 (e.g. meshes and voxel grids), 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, 34, 21, 51, 60, 45, 64, 70]. 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 have been an active research direction

In this paper, we propose the reproducing activation function 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, 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 [67], the Floor-Exponential-Sign networks [66].

In terms of training dynamics of deep learning, reproducing activation functions can 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 only find the smoothest solution with the fastest decay in the frequency domain due to the implicit regularization of network structures and the stochastic gradient descent (SGD) for solving the minimization problem, no matter how the initial guess is randomly selected. 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. Through the analysis of the optimization energy landscape of SGD, it was shown that SGD with small batches tends to converge to the flattest minimum [57, 41, 12]. Though the above optimization and generalization analysis work only for regression problems, they can be generalized to PDE problems, e.g., the optimization and generalization analysis of PDE solvers in [49] and the spectral bias of PDE solvers in [74]. 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 also including 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 paper is organized as follows. In Section 2, preliminary knowledge of deep learning will be introduced. In Section 3, we introduce the reproducing activation function. Numerical results will be presented in Section 4 to demonstrate the efficiency of the reproducing activation function. 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 [22]. The justification of this kind of approximation is given by the universal approximation theorems of DNNs in [38, 2, 76, 77] with newly developed quantitative and explicit error characterization [65, 47, 67], which shows that 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 $\boldsymbol{h}_\ell(\boldsymbol{x}) = \sigma\left(\boldsymbol{W}_\ell \boldsymbol{x} + \boldsymbol{b}_\ell\right)$ 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 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 [28]. We present its variant defined recursively as follows:

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

where $\mathbf{V} \in \mathbb{R}^{\mathbf{N_0 \times d}}$, $\mathbf{W}_\ell \in \mathbb{R}^{\mathbf{N}_\ell \times \mathbf{N_0}}$, $\tilde{\mathbf{U}}_\ell \in \mathbb{R}^{N_0 \times N_0}$, $U_\ell \in \mathbb{R}^{N_0 \times N_\ell}$, $\mathbf{b}_\ell \in \mathbb{R}^{\mathbf{N}_\ell}$ for $\ell = 1, \cdots, L$, $\mathbf{a} \in \mathbb{R}^{\mathbf{N_0}}$, $\mathbf{h_{-1}} = \mathbf{0}$. Throughout this paper, we consider $N_0 = N_\ell = N$ and U_ℓ is set as the identity matrix in the numerical implementation of ResNets for the purpose of simplicity. Furthermore, as used in [20], we set $\tilde{U}_\ell$ as the identity matrix when ℓ is even and set $\tilde{U}_\ell = 0$ 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

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \hat{\mathcal{J}}(\boldsymbol{\theta}), \tag{2.4}$$

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 [14, 39, 20, 44]. In this paper, we will take the example of BVP and the least squares method (LSM) [14, 39] without loss of generality. The generalization to other problems and methods is similar. Consider the BVP

$$\begin{aligned} &\mathcal{D}u(\boldsymbol{x}) = f(u(\boldsymbol{x}), \boldsymbol{x}), \text{ in } \Omega, \\ &\mathcal{B}u(\boldsymbol{x}) = g(\boldsymbol{x}), \text{ on } \partial\Omega, \end{aligned} \tag{2.5}$$

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 discusssed in [23]. 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

$$\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) := \|\mathcal{D}u(\boldsymbol{x}; \boldsymbol{\theta}) - f(u, \boldsymbol{x})\|_{L^2(\Omega)}^2 + \lambda\|\mathcal{B}u(\boldsymbol{x}; \boldsymbol{\theta}) - g(\boldsymbol{x})\|_{L^2(\partial\Omega)}^2, \tag{2.6}$$

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 (2.6) 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 (2.5). 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 (2.5) is well-posed (e.g. the elliptic PDE with Neumann boundary condition, see Theorem 4.1 in [23]).

In the implementation of LSM, the minimization problem in (2.6) is solved by SGD or its variants (e.g. Adagrad [16], Adam [37] and AMSGrad [63]). In each iteration of the SGD, a stochastic loss function defined below is minimized instead of the original loss function in (2.6):

$$\min_{\boldsymbol{\theta}} \hat{\mathcal{L}}(\boldsymbol{\theta}) := \frac{1}{N}\sum_{i=1}^N \left(\mathcal{D}u(\mathbf{x_i}; \boldsymbol{\theta}) - \mathbf{f}(\mathbf{x_i})\right)^{\mathbf{2}} + \frac{\mathbf{1}}{\mathbf{M}}\lambda\sum_{\mathbf{j=1}}^{\mathbf{M}} \left(\boldsymbol{\mathcal{B}}\mathbf{u}(\mathbf{x_j}; \boldsymbol{\theta}) - \mathbf{g}(\mathbf{x_j})\right)^{\mathbf{2}}, \tag{2.7}$$

where $\{\mathbf{x_i}\}_{\mathbf{i=1}}^{\mathbf{N}}$ are N uniformly sampled random points in Ω and $\{\mathbf{x_j}\}_{\mathbf{j=1}}^{\mathbf{M}}$ 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 (2.6) to

$$\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) := \|\mathcal{D}u(\boldsymbol{x};\boldsymbol{\theta}) - f(u,\boldsymbol{x})\|_{L^2(\Omega)}^2, \tag{2.8}$$

since $\mathcal{B}u(\mathbf{x};\boldsymbol{\theta}) = \mathbf{g}(\mathbf{x})$ is satisfied by construction. Correspondingly, the stochastic loss function is reduced to

$$\min_{\boldsymbol{\theta}} \hat{\mathcal{L}}(\boldsymbol{\theta}) := \frac{1}{N}\sum_{i=1}^{N}\left(\mathcal{D}u(\mathbf{x_i};\boldsymbol{\theta}) - \mathbf{f}(\mathbf{x_i})\right)^{\mathbf{2}}. \tag{2.9}$$

In numerical implementation, the LSM loss function in (2.8) 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 (2.8), as shown by extensive numerical experiments in [39, 24, 50]. 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 [24, 50] 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 $u(x;\boldsymbol{\theta})$:

$$u(x;\boldsymbol{\theta}) = h(x)\hat{u}(x;\boldsymbol{\theta}) + l(x). \tag{2.10}$$

Then $u(x;\boldsymbol{\theta})$ is used to approximate the true solution of the PDE and is trained through (2.8).

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, 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 (2.8) 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 [31, 15, 79, 11, 49]. 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 is further investigated in [75, 48]. 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 [58, 42, 13]. However, the PDE solution might not be associated with such local minimizers. 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 [31] and further investigated in [1, 40, 8, 49, 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 [40] for simplicity. This introduction is sufficient for us to discuss the advantage of reproducing activation functions later in the next section.

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:

$$\dot{\boldsymbol{\theta}}_t = -\nabla_{\boldsymbol{\theta}}\phi_t(\mathcal{X})^T \nabla_{\phi_t(\mathcal{X})}\hat{\mathcal{J}}, \tag{2.11}$$

and

$$\dot{\phi}_t(\mathcal{X}) = \nabla_{\boldsymbol{\theta}}\phi_t(\mathcal{X})\dot{\boldsymbol{\theta}}_t = -\hat{\Theta}_t(\mathcal{X},\mathcal{X})\nabla_{\phi_t(\mathcal{X})}\hat{\mathcal{J}}, \tag{2.12}$$

where $\boldsymbol{\theta}_t$ is the parameter set at iteration time t, $\phi_t(\mathcal{X}) = \mathrm{vec}([\phi_t(\boldsymbol{x};\boldsymbol{\theta}_t)]_{\boldsymbol{x}\in\mathcal{X}}$ 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 time t defined by

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

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 (2.11) and (2.12). 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 [31, 1, 40, 8, 49], when the network width goes to infinity, these evolution equations can be approximately characterized by their linearization, the solution of which admit simple explicit formulas.

For simplicity, we consider the linearization in [40] to obtain explicit solutions to discuss the training dynamics of deep learning. In particular, the following linearized network by Taylor expansion is considered,

$$\phi_t^{\text{lin}}(\boldsymbol{x}) := \phi(\boldsymbol{x};\boldsymbol{\theta}_0) + \nabla_{\boldsymbol{\theta}}\phi(\boldsymbol{x};\boldsymbol{\theta}_0)\boldsymbol{\omega}_t, \tag{2.13}$$

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

$$\dot{\boldsymbol{\omega}}_t = -\nabla_{\boldsymbol{\theta}}\phi_0(\mathcal{X})^T \nabla_{\phi_t^{\text{lin}}(\mathcal{X})}\hat{\mathcal{J}}, \tag{2.14}$$

and

$$\dot{\phi}_t^{\text{lin}}(\boldsymbol{x}) = -\hat{\Theta}_0(\boldsymbol{x},\mathcal{X})\nabla_{\phi_t^{\text{lin}}(\mathcal{X})}\hat{\mathcal{J}}. \tag{2.15}$$

The above evolution equations have closed form solutions

$$\boldsymbol{\omega}_t = -\nabla_{\boldsymbol{\theta}}\phi_0(\mathcal{X})^T\hat{\Theta}_0^{-1}\left(I - e^{-\hat{\Theta}_0 t}\right)(\phi_0(\mathcal{X}) - \mathcal{Y}),$$

and

$$\phi_t^{\text{lin}}(\mathcal{X}) = \left(I - e^{-\hat{\Theta}_0 t}\right)\mathcal{Y} + e^{-\hat{\Theta}_0 t}\phi_0(\mathcal{X}). \tag{2.16}$$

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

$$\phi_t^{\text{lin}}(\boldsymbol{x}) = \phi_0(\boldsymbol{x}) - \hat{\Theta}_0(\boldsymbol{x},\mathcal{X})\hat{\Theta}_0^{-1}\left(I - e^{-\hat{\Theta}_0 t}\right)(\phi_0(\mathcal{X}) - \mathcal{Y}), \tag{2.17}$$

which is equivalent to

$$\phi_t^{\text{lin}}(\boldsymbol{x}) - \phi_0(\boldsymbol{x}) = \hat{\Theta}_0(\boldsymbol{x},\mathcal{X})\hat{\Theta}_0^{-1}\left(I - e^{-\hat{\Theta}_0 t}\right)(\mathcal{Y} - \phi_0(\mathcal{X})). \tag{2.18}$$

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 (2.17) serves as an approximate solution to the nonlinear evolution equation in (2.12). Based on (2.18), 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 mainly two kinds of observations from (2.17) 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 (2.17). 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 (2.17) 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 $\phi_t^{\mathrm{lin}}(\boldsymbol{x})$ to solve the regression problem. We will discuss the advantage of the proposed reproducing activation functions in terms of these two observations later in the next two sections.

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

$$\hat{\Theta}_t = \left(\nabla_{\boldsymbol{\theta}} \mathcal{D}\phi_t(\mathcal{X})\right) \left(\nabla_{\boldsymbol{\theta}} \mathcal{D}\phi_t(\mathcal{X})\right)^T , \tag{2.19}$$

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 learnign based PDE solvers by almost the same arguments.

3. Reproducing Activation Functions.

In this section, we will introduce the concept of reproducing activation functions, their examples, their 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 every 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 $\{\alpha_{p,i,\ell}, \gamma_{p,i,\ell}\}_{p=1}^P$ 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, 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).

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 set of parameters $\{\hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\beta}}\}$ is identified via

$$\{\hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\beta}}\} = \arg\min_{\boldsymbol{\theta}, \boldsymbol{\alpha}, \boldsymbol{\beta}} \hat{\mathcal{J}}(\boldsymbol{\theta}, \boldsymbol{\alpha}, \boldsymbol{\beta}), \tag{3.3}$$

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 deep learning is applied to solve the PDE in (2.5), the population loss function in (2.6) becomes

$$\min_{\boldsymbol{\theta}, \boldsymbol{\alpha}, \boldsymbol{\beta}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\alpha}, \boldsymbol{\beta}) := \|\mathcal{D}u(\boldsymbol{x}; \boldsymbol{\theta}, \boldsymbol{\alpha}, \boldsymbol{\beta}) - f(u, \boldsymbol{x})\|^2_{L^2(\Omega)} + \lambda\|\mathcal{B}u(\boldsymbol{x}; \boldsymbol{\theta}, \boldsymbol{\alpha}, \boldsymbol{\beta}) - g(\boldsymbol{x})\|^2_{L^2(\partial\Omega)}, \tag{3.4}$$

and in each iteration of the SGD, the empirical loss function in (2.7) becomes

$$\min_{\boldsymbol{\theta}, \boldsymbol{\alpha}, \boldsymbol{\beta}} \hat{\mathcal{L}}(\boldsymbol{\theta}, \boldsymbol{\alpha}, \boldsymbol{\beta}) := \frac{1}{N}\sum_{i=1}^{N}\left(\mathcal{D}u(\mathbf{x_i}; \boldsymbol{\theta}, \boldsymbol{\alpha}, \boldsymbol{\beta}) - \mathbf{f}(\mathbf{x_i})\right)^{\mathbf{2}} + \frac{\mathbf{1}}{\mathbf{M}}\lambda\sum_{\mathbf{j=1}}^{\mathbf{M}}\left(\mathcal{B}\mathbf{u}(\mathbf{x_j}; \boldsymbol{\theta}, \boldsymbol{\alpha}, \boldsymbol{\beta}) - \mathbf{g}(\mathbf{x_j})\right)^{\mathbf{2}}. \tag{3.5}$$

3.2. Examples and Reproducing Properties. Here, a few examples of reproducing activation functions will be discussed, include 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

3.2.1. Example 1: Sine-ReLU Networks. Sine-ReLU networks proposed in [78] applies sine function $\sin(x)$ or a ReLU function $\max\{0,x\}$ in each neuron to construct a deep neural network. Hence, the proposed reproducing activation function concept includes the Sine-ReLU network 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 Sine-ReLU networks proved in [78] provides a theoretical lower bound of the approximation power of reproducing activation function when the set of basic activation functions $\mathcal{A} = \{\sin(x), \max\{0,x\}\}$. In fact, $\sin(x)$ can be replaced with 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 $H^{r,\infty}([0,1]^d)$. We rephrase this theorem using the terminology of reproducing activation functions below.

THEOREM 3.1. *Fix r,d. Let $\sigma:\mathbb{R}\to\mathbb{R}$ be a Lipschitz periodic function with period T. Suppose that $\sigma(x) > 0$ for $x \in (0,T/2)$ and $\sigma(x) < 0$ for $x \in (T/2,T)$, and also that $\max_{x\in\mathbb{R}}\sigma(x) = -\min_{x\in\mathbb{R}}\sigma(x)$. Let $\mathcal{A} = \{\sigma(x), \max\{0,x\}\}$ be set of basic activation functions. For any integer number $W > 0$ and any $f(\boldsymbol{x}) \in F_{r,d}$, there exists a deep neural network $\phi(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\alpha},\boldsymbol{\beta})$ built with reproducing activation functions associated with $\mathcal{A}$ with W parameters in $\{\boldsymbol{\theta},\boldsymbol{\alpha},\boldsymbol{\beta}\}$ such that*

$$\|f(\boldsymbol{x}) - \phi(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\alpha},\boldsymbol{\beta})\|_\infty \le exp\left(-c_{r,d}W^{1/2}\right)$$

with a constant $c_{r,d} > 0$ only depending on r and d.

It is worth pointing out that Theorem 6.1 in [78] also shows that only one parameter of the network there depends on f and other parameters are f-independent. Ideally, this is exciting news since it means that deep learning can be reduced to a one-dimensional optimization problem if optimization and computation with arbitrary precision are affordable. Under current hardware and software constraints, the concept of reproducing activation functions may be more realistic since employing more learnable parameters in our networks may alleviate the requirement of computation precision while keeping the exponential approximation power as shown in Theorem 3.1.

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 use ReLU in other layers for the purpose of high-resolution image reconstruction in cryo-electron microscopy. The same idea is applied in [52] for high-resolution scene and shape reconstruction in various applications. The same structure is used in [27] for the purpose of generating networks satisfying periodic boundary conditions. A variant of this structure with several blocks is designed in [46, 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 Networks. Recently, networks with super approximation power (e.g., an exponential convergence rate without the curse of dimensionality for Hölder continuous functions) have been proposed in [67, 66]. Let us consider the Floor-Exponential-Sign Networks in [66]. The key idea is to use one of the following three activation functions in each neuron:

$$\sigma_1(x) := \lfloor x \rfloor, \quad \sigma_2(x) := 2^x, \quad \text{and} \quad \sigma_3 := \mathcal{T}(x - \lfloor x \rfloor - \frac{1}{2}), \tag{3.6}$$

for any $x \in \mathbb{R}$. Here,

$$\mathcal{T}(x) := \mathbb{1}_{x \geq 0} = \begin{cases} 1, & x \geq 0, \\ 0, & x < 0, \end{cases}$$

for any $x \in \mathbb{R}$.. Obviously, the concept of reproducing activation functions also 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 [66] provides a theoretical lower bound of the approximation power of reproducing activation function when the set of basic activation functions $\mathcal{A} = \{\sigma_1(x), \sigma_2(x), \sigma_3(x)\}$. We rephrase Theorem 1.1 in [66] using the terminology of reproducing activation functions below.

Theorem 3.2. *Given f in $C([0,1]^d)$ and $W \in \mathbb{N}^+$, there exists a deep neural network $\phi(\boldsymbol{x}; \boldsymbol{\theta}, \boldsymbol{\alpha}, \boldsymbol{\beta})$ of width W and depth 4 (i.e., three hidden layers) built with reproducing activation functions associated with $\mathcal{A} = \{\sigma_1(x), \sigma_2(x), \sigma_3(x)\}$ such that*

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

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

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

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 [66], 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 problem 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 is no efficient optimization methods for training networks with piecewise constant activation functions. Therefore, it is worth exploring other reproducing activation functions as we shall see in the next section.

3.2.3. Example 3: Poly-Sine-Gaussian Networks. Considering both the approximation power and the computational efficiency, we propose the poly-sine-Gaussian network as a new example of reproducing activation functions in this paper. The main idea is to use $\mathcal{A} = \{x, x^2, \sin(x), e^{-x^2}\}$ such that deep neural networks can reproduce traditional approximation tools efficiently, e.g., orthogonal polynomials, Fourier basis functions, wavelets, radial basis functions, etc. 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, since existing approximation theory with a continuous weight selection of ReLU networks are established by using ReLU networks to approximate x and x^2 as basic building blocks. We will present several lemmas and theorems to illustrate the approximation capacity of poly-sine-Gaussian networks as follows.

Let us start by reproducing polynomial approximations.

Lemma 3.3. *A list of basic lemmas of the poly-sine-Gaussian networks.*

(i) Any 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*

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

(v) 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*

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

Proof. Part (i) to (iii) are trivial. We will only prove Part (iv) and (v).

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 3.1. 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.

Part (v): The main idea of the proof is to apply Part (iv) 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 3.2, 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 $P(\boldsymbol{x})$.

By Part (iv), 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 3.2. 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 3.1).

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. □

Lemma 3.3 plays a key role to characterize the approximation power of poly-sine-Gaussian networks since it characterizes how well poly-sine-Gaussian networks approximate arbitrary polynomials including orthogonal polynomials. Orthogonal polynomials are important tools for classical approximation theory and numerical computation. For example, the Chebyshev series lies at the

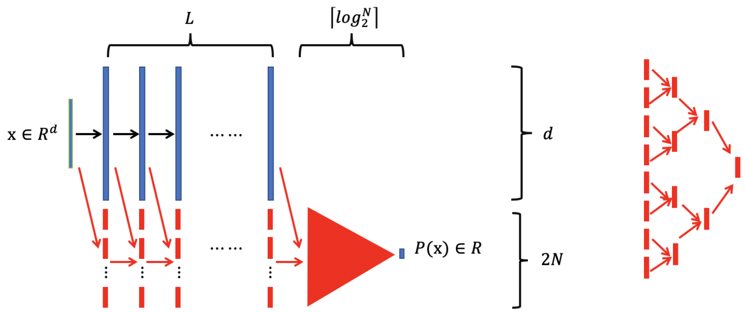

Fig. 3.1: Left: An illustration of the proof of Lemma 3.3 (iv). Green vectors represent the input and output of the poly-sine-Gaussian network carrying out $P(\boldsymbol{x})$. Blue vectors represent the poly-sine-Gaussian network that implements a d-dimensional identity map in Part (i), which was repeatedly 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.

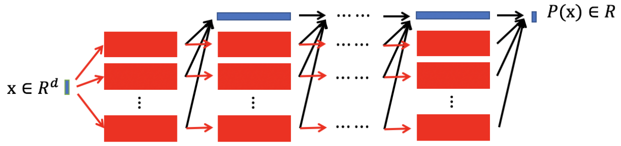

Fig. 3.2: An illustration of the proof of Lemma 3.3 (v). Green vectors represent the input and output of the poly-sine-Gaussian network $\tilde{\phi}(\boldsymbol{x})$ carrying out $P(\boldsymbol{x})$. Each red rectangle represents one poly-sine-Gaussian network $\phi_j(\boldsymbol{x})$ and each blue 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 $P(\boldsymbol{x})$.

heart of approximation theory. In particular, for analytic functions, the truncated Chebyshev series defined as $f_n(x) = \sum_{k=0}^{n} c_k T_k(x/M)$ are *exponentially accurate* approximations [72, Thm. 8.2], where T_k as the Chebyshev polynomial of degree k defined on $[-1, 1]$.

More precisely, for some scalars $M \geq 1$ and $s > 1$, if we define

$$a_s^M = M\frac{s+s^{-1}}{2}, \quad b_s^M = M\frac{s-s^{-1}}{2},$$

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. The proof

THEOREM 3.4 (Poly-sine-Gaussian networks for analytic functions). *For any scalar $M \geq 1$, $s > 1$, $C_f > 0$ and $0 < \epsilon < 1$, and any real-valued analytic function f with input $x \in [-M, M]$ 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 inputs $x \in [-M, M]$, that has 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 $n = \mathcal{O}\left(\frac{1}{\log_2 s} \log_2 \frac{2C_f}{\epsilon}\right)$.

Proof. 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 Lemma 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

$$\max_{0 \leq k \leq n} |c_k| = \mathcal{O}\left(C_f s\right), \text{ via [72, Thm. 8.1].} \tag{3.7}$$

Since, f_n is a polynomial of degree n, by Lemma 3.3 (v) 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

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

□

By choosing $N = \mathcal{O}(n)$ and $L = \mathcal{O}(n \log_2(n))$ in Theorem 3.4, it is easy to see that the width and depth of ϕ approximating an analytic function f with ϵ accuracy can be as small as $\mathcal{O}\left(\log_2 \frac{1}{\epsilon}\right)$ and $\mathcal{O}\left(\left(\log_2 \frac{1}{\epsilon}\right) \log_2\left(\log_2 \frac{1}{\epsilon}\right)\right)$, respectively. Hence, the size of the poly-sine-Gaussian network is smaller than the size of the ReLU network for approximating the same f in Theorem 2.6 in [55].

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

DEFINITION 3.5 (Generalized bandlimited functions [55]). *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 that K satisfies the assumption of Thm. 3.4 for some $s > 1$ and $C_K > 0$. We define the Hilbert space $\mathcal{H}_{K,M}(B)$ of generalized bandlimited functions via*

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

with inner product $\langle f, g\rangle_{\mathcal{H}_{K,M}(B)} := \int_{[-M,M]^d} F_f(\boldsymbol{w})\overline{F}_g(\boldsymbol{w})d\boldsymbol{w}$ *and its induced norm* $\|f\|_{\mathcal{H}_{K,M}(B)}$*, where*

$$F_f = \arg\min_{F\in S_f} \|F\|_{L^2([-M,M]^d)}, \quad S_f = \left\{F \;\middle|\; f(\boldsymbol{x}) = \int_{[-M,M]^d} F(\boldsymbol{w})K(\boldsymbol{w}\cdot\boldsymbol{x})d\boldsymbol{w}\right\}.$$

Note that

$$|f(\boldsymbol{x})| \le D_K \int_{[-M,M]^d} |F_f(\boldsymbol{w})|d\boldsymbol{w} \le (2M)^{d/2}\|F_f\|_{L^2([-M,M]^d)} = (2M)^{d/2}\|f\|_{\mathcal{H}_{K,M}(B)},$$

which shows that if we consider an evaluation functional $L_{\boldsymbol{x}}$ defined on $\mathcal{H}_{K,M}(B)$ by

$$f(\boldsymbol{x}) = L_{\boldsymbol{x}}(f) := \int_{[-M,M]^d} F_f(\boldsymbol{w})K(\boldsymbol{w}\cdot\boldsymbol{x})d\boldsymbol{w},$$

then $L_{\boldsymbol{x}}$ is bounded on $\mathcal{H}_{K,M}(B)$. Hence, $\mathcal{H}_{K,M}(B)$ is a 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.

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

Now, we are ready to show the approximation of poly-sine-Gaussian networks to generalized bandlimited functions below.

Theorem 3.7 (Poly-sine-Gaussian networks for $\mathcal{H}_{K,M}$). *Suppose f is an arbitrary real-valued function in $\mathcal{H}_{K,M}(B)$, for some function K, scalars $M \ge 1$, $s > 1$ and $C_K > 0$, and integer $d \ge 2$. Let us assume that $\int_{\mathbb{R}^d} |F(\boldsymbol{w})|d\boldsymbol{w} = \int_{[-M,M]^d} |F(\boldsymbol{w})|d\boldsymbol{w} = C_F$. Then, for any measure μ and any scalar $0 < \epsilon < 1$, there exists a poly-sine-Gaussian network ϕ with inputs $\boldsymbol{x} \in B = [0,1]^d$, 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

$$\|\phi - f\|_{L^2(\mu,B)} = \sqrt{\int_B |\phi(\boldsymbol{x}) - f(\boldsymbol{x})|^2 d\mu(\boldsymbol{x})} \le \epsilon.$$

Proof. 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

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

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 $2C_F D_K\sqrt{\mu(B)} \leq 2C_F\sqrt{\mu(B)}$, Theorem 3.6 tells us that there exist real coefficients b_j's and β_j's such that[1]

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

for some $0<\epsilon_0<1$ to be determined later, such that

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

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

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

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

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

which yields

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

The total approximation error satisfies

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

We take

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

to complete the proof. □

[1] We use Theorem 3.6 with $b = 2C_F\sqrt{\mu(B)}$, $c = b^2 > b^2 - \|f\|^2$, and $\|\cdot\| = \|\cdot\|_{L^2(\mu,B)}$.

We would like to revisit the discussion in [2, 55] 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 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.

Poly-sine-Gaussian networks can also reproduce typical applied harmonic analysis tools as in the following lemma.

Lemma 3.8. *A list of basic lemmas of the poly-sine-Gaussian networks for reproducing basis functions in applied harmonic analysis.*

(i) Poly-sine-Gaussian networks can reproduce all basis functions in the discrete cosine transform and discrete sine transform in an arbitrary dimension.

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

(iii) Poly-sine-Gaussian networks with complex parameters in the last affine linear transform can reproduce all basis functions in the discrete Fourier transform in an arbitrary dimension.

(iv) Poly-sine-Gaussian networks with complex parameters in the last affine linear transform can reproduce all basis functions in the discrete Gabor wavelet transform in an arbitrary dimension.

Proof. 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 3.3 shows that the x^2 activation function can reproduce multiplication. □

In theory, Lemma 3.8 above implies that poly-sine-Gaussian networks can be very useful in many computer vision tasks involving Fourier transforms and wavelet transforms. As we shall see in the numerical section, a few examples of audio/image reconstruction will be presented to demonstrate this advantage numerically. Due to the advantage of wavelets to represent functions with singularity, poly-sine-Gaussian networks may also be useful in representing functions with singularity, the performance of which is as good as or even slightly better than rational neural networks as we shall see in the numerical section. We would like to highlight that the Gaussian function may not be the optimal choice in the concept of reproducing activation function. Other window functions in wavelet analysis may provide better performance and this would be problem-dependent. We will leave this for future exploration.

Finally, in terms of approximation capacity, we have the following lemma for radial basis functions.

Lemma 3.9. *Poly-sine-Gaussian networks for radial basis functions.*

(i) Poly-sine-Gaussian networks can reproduce radial basis functions with arbitrary shape parameters and a Gaussian kernel.

(ii) Poly-sine-Gaussian networks can approximate radial basis functions defined on a bounded closed domain with arbitrary shape parameters and analytic kernels with an exponential convergence rate.

Proof. The proof of this lemma is trivial by Lemma 3.3 and the proof of Theorem 3.4. □

We have finished the discussion of the poly-sine-Gaussian networks in terms of approximation theory. We will end this section with a short and informal discussion about the NTK of poly-sine-Gaussian networks. As we have seen in Section 2.5, deep learning can be approxiated by kernel methods with a kernel $\hat{\Theta}_0$ in (2.18). 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 the following toy regression formulation:

$$\phi(\boldsymbol{x};\boldsymbol{\omega}) := \sum_{i=1}^{N} \omega_i f(\boldsymbol{x}_i)\hat{\Theta}(\boldsymbol{x}, \boldsymbol{x}_i), \tag{3.8}$$

where $\boldsymbol{\omega} = [\omega_1, \dots, \omega_N] \in \mathbb{R}^N$ is a set of learnable parameters and $\phi(\boldsymbol{x};\boldsymbol{\omega})$ is the approximant of the target function $f(\boldsymbol{x})$. According to (3.8), 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 reproducing activation functions in two kinds of applications. In the first part, several examples in scientific computing will be provided, e.g., regression problems, solving high dimensional and nonlinear PDEs, and eigenvalue problems. In the second part, several examples in signal processing and computer vision will be provided.

We would like to emphasize that the optimal choice of basic activation functions would be problem-dependent. According to the discussion in the previous section, we know that $\mathcal{A} = \{x, x^2, \sin(x), \exp(-x^2)\}$ has nice approximation properties for various functions. However, deep learning optimization usually cannot identify the best parameter set to verify these properties. Different training strategies are required to specify or train the combination coefficients $\boldsymbol{\alpha}$, the scaling parameters $\boldsymbol{\beta}$, and other network parameters $\boldsymbol{\theta}$ in a problem-dependent manner. Generally speaking, $\mathcal{A} = \{x, x^2, \sin(x), \exp(-x^2)\}$ achieves the best performance in various scientific computing problems in Part I. In Part II, we will have different strategies for $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ according to the characteristic of tasks.

4.1. Part I: Scientific Computing Applications.

In Part I, we will compare the proposed poly-sine-Gaussian reproducing activation function with existing activation functions, e.g., the ReLU function, the ReLU^3 function, the rational activation function in [5]. We will also provide 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 method in PyTorch is employed to minimize the loss functions discussed in Section 2 for regression and solving PDEs. We will follow the approach of [27] for the loss function of eigenvalue problems. We define the L^2 relative error by

$$\left(\frac{\sum_{i=1}^N (u(x_i) - \hat{u}(x_i))^2}{\sum_{i=1}^N u^2(x_i)}\right)^{\frac{1}{2}}, \tag{4.1}$$

where $\{x_i\}_{i=1}^N$ are random samples uniformly distributed in the function domain, u is the ground truth solution, and $\hat{u}$ is the estiation by deep learning.

The overall setting for all examples is summarized as follows.

- **Environment.** The experiments are performed in Python 3.7 environment. We utilize PyTorch library for neural network implementation and CUDA 10.0 toolkit for GPU-based parallel computing.

- **Optimizer.** In all examples, the optimization problems are solved by *Adam* subroutine from PyTorch library with default hyperparameters. This subroutine implements the Adam algorithm in [37].
- **Learning rate.** The learning rate will be decreased step by step in all examples following the formula

  $$\tau_n = \tau_0 * q^{\lfloor \frac{n}{s} \rfloor}, \tag{4.2}$$

  where τ_n is the learning rate in the $n-$th iteration, q is a factor set to be 0.95, and s means taht we update learning rate after s steps.
- **Numbers of samples.** The numbers of training and testing samples for regression and PDE problems are $10,000$. The numbers of training and testing samples for eigenvalue problems are 2048 following the approach in [27].
- **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 are initialized by $\boldsymbol{W}_\ell, \boldsymbol{b}_\ell \sim U(-\sqrt{N_{\ell-1}}, \sqrt{N_{\ell-1}})$, where $N_{\ell-1}$ is the width of the ℓ-th layer. Note that the network with reproducing activation functions can be expressed by a network with a single activation function in each neuron but different activation functions in different neurons. 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 for each x, x^2, and $\sin(x)$ activation functions. In this new setting, it is not necessary to train extra combination coefficients in the reproducing activation function. Though training the scaling parameters in the reproducing activation function might be beneficial in general applications, we focus on justifying the poly-sine-Gaussian activation function without emphasizing the scaling parameters. Hence, in almost all tests in Part I, the scaling parameters are set to be one for x, x^2, and $\sin(x)$, and the scaling parameters are set to be 0.1 for $\exp(-x^2)$. In the case of oscillatory target functions, we specify the scaling parameters 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 [33, 32].
- **Performance Evaluation.** We will adopt two criteria to quantify the performance of different activation functions. The first one is the L^2 error on test samples. Note that the ground truth solution is not available in real applications and, hence, it is not known when to stop the training. Therefore, we will keep the best historical L^2 test error and the best historical moving-average L^2 test error. In the moving-average error calculation, the error at a given iteration is the average L^2 test error of 100 previous iterations. 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.

4.1.1. Discontinuous Function Regression. We first demonstrate the advantage of the poly-sine-Gaussian activation function in a regression problem when the target function is discontinuous. For example, consider the following target function

$$f(x) = \begin{cases} 1-x, & x > 0, \\ x-1, & x < 0, \end{cases} \tag{4.3}$$

on the domain $\Omega = [-1, 1]$. The empirical loss function in (2.3) is applied with different random samples in each SGD iteration. The relative L^2 error is presented in Table 4.1 and the training process can be seen in Figure 4.1. Numerical results show that the poly-sine-Gaussian activation has the best performance in terms of the moving-average error and its best historical error is only slightly worse than the rational activation function recently proposed in [5]. We would like to remark that rational activation functions in [5] works well for regression problems but fails in our PDE problems without any meaningful solutions. Hence, we only compare reproducing activation functions with rational activation functions in this example. The numerical results also justify the combination of four kinds of activation functions.

Activation Function	Relative L^2 Error (Moving-Average)	Relative L^2 Error (Min)
ReLU^3	1.13 e-01	1.01 e-01
$x \oplus x^2$	3.71 e-01	3.58 e-01
$x \oplus x^2 \oplus$ ReLU	9.98 e-02	8.36 e-02
$x \oplus x^2 \oplus$ ReLU^3	9.12 e-02	6.96 e-02
$x \oplus x^2 \oplus \sin(x)$	9.07 e-02	7.90 e-02
$x \oplus x^2 \oplus \sin(x) \oplus$ Gaussian	3.46 e-02	1.39 e-02
Rational function [5]	3.94 e-02	1.03 e-02

Table 4.1: The best historical accuracy for the regression problem with different activation functions after 50,000 iterations. The $\oplus$ notation here means that the activation is applied together with other activation functions in the network.

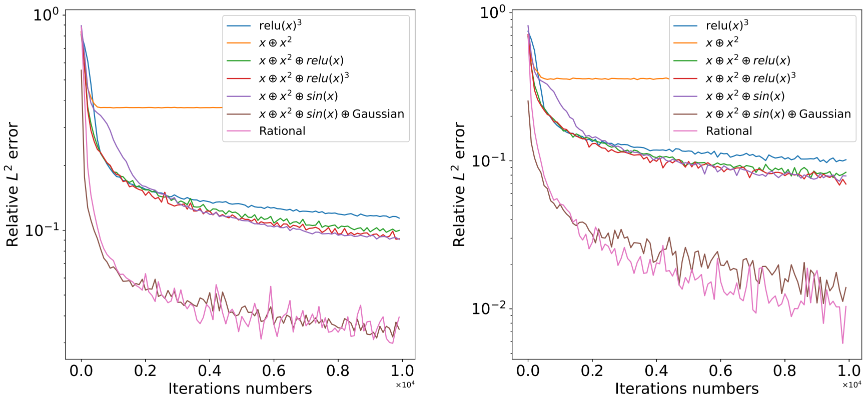

Fig. 4.1: The training process of the regression problem with different activation functions. Left: the moving-average minimal historical error. Right: the minimal historical error.

4.1.2. Poisson Equation with a Smooth Solution. Now we solve a two-dimensional Poisson equation

$$\begin{cases} -\Delta u = f, & \boldsymbol{x} \in \Omega, \\ u = 0, & \boldsymbol{x} \in \partial\Omega, \end{cases} \tag{4.4}$$

with a smooth solution $u(\boldsymbol{x}) = x_1^2(1-x_1)x_2^2(1-x_2)$ defined on $[0,1]^2$. The numerical solution can be constructed as

$$\hat{u}(\boldsymbol{x};\boldsymbol{\theta}) = \left(\Pi_{i=1}^2 x_i(1-x_i)\right)\phi(\boldsymbol{x};\boldsymbol{\theta}) \tag{4.5}$$

where $\phi(\boldsymbol{x};\boldsymbol{\theta})$ is a deep neural network. Since the constructed network $\hat{u}(\boldsymbol{x};\boldsymbol{\theta})$ satisfies the boundary condition automatically, we apply the loss function without boundary penalty introduced in Section 2.3 to identify an approximate solution to the Poisson equation.

Different activation functions are applied to construct the deep neural network in the above solver. The resulting relative L^2 errors are shown in Table 4.2. The corresponding training process is visualized in Figure 4.2. It is obvious that the reproducing activation function with $\mathcal{A} = \{x, x^2, \sin(x), \exp(-x^2)\}$ achieves the best performance. After around $20,000$ iterations, networks with other activation functions have achieved local minimizers and cannot escape from these minimizers. The poly-sine-Gaussian activation enables networks to escape from bad local minimizers (e.g., around $3,000$ and $20,000$ iterations) and gradient descent can still reduce the solution error after $50,000$ iterations since the error curve is far away from being a flat line. The numerical results also justify the combination of four kinds of activation functions.

Activation Function	Relative L^2 Error (Moving-Average)	Relative L^2 Error (Min)
ReLU^3	9.40 e-04	8.98 e-04
$x \oplus x^2$	4.50 e-04	4.38 e-04
$x \oplus x^2 \oplus$ ReLU	4.49 e-04	4.36 e-04
$x \oplus x^2 \oplus$ ReLU^3	1.39 e-03	1.35 e-03
$x \oplus x^2 \oplus \sin(x)$	4.45 e-04	4.33 e-04
$x \oplus x^2 \oplus \sin(x) \oplus$ Gaussian	6.87 e-05	6.85 e-05

Table 4.2: The best historical accuracy for the Poisson equation defined in (4.4) with a smooth solution. Different activation functions are applied to solve the Poisson equation.

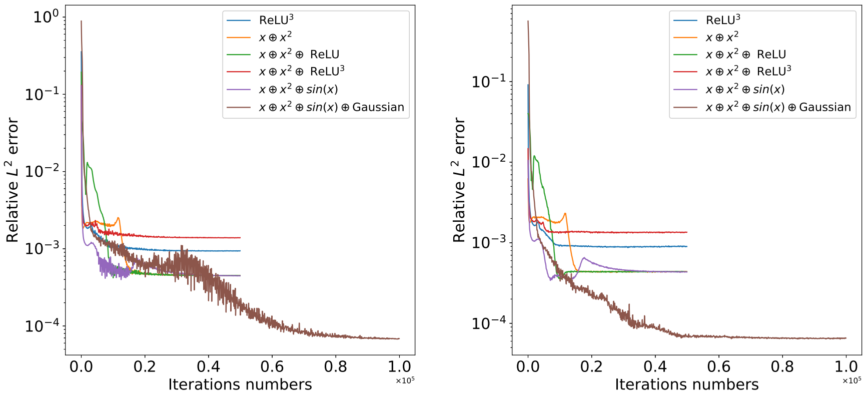

Fig. 4.2: The training process for the Poisson equation in (4.4) with a smooth solution. Left: the moving-average minimal historical error. Right: the minimal historical error.

4.1.3. Poisson Equation with Low Regularity. Next, we consider a two-dimensional Poisson equation

$$\begin{cases} -\nabla \cdot (|\boldsymbol{x}|\nabla u) = f, & \boldsymbol{x} \in \Omega, \\ u = 0, & \boldsymbol{x} \in \partial\Omega, \end{cases} \tag{4.6}$$

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 $\phi(\boldsymbol{x};\boldsymbol{\theta})$ be an arbitrary network, then

$$\hat{u}(\boldsymbol{x};\boldsymbol{\theta}) = (1 - |\boldsymbol{x}|)\phi(\boldsymbol{x};\boldsymbol{\theta}) \tag{4.7}$$

satisfies the boundary condition automatically. We apply the loss function without boundary penalty introduced in Section 2.3 to identify an approximate solution $\hat{u}(\boldsymbol{x};\boldsymbol{\theta})$ to the Poisson equation in (4.6).

Different activation functions are applied to construct the deep neural network in the above solver. The resulting relative L^2 errors are shown in Table 4.3. The corresponding training process is visualized in Figure 4.3. Since the exact solution has low regularity, it is more challenging than the previous example and the numerical accuracy is not as good as those in the previous example. It is obvious that the reproducing activation function with $\mathcal{A} = \{x, x^2, \sin(x), \exp(-x^2)\}$ achieves the best performance. Note that gradient descent can still reduce the solution error after $50,000$ iterations in the case of poly-sine-Gaussian activation function since the error curve is still not flat. However, the error curves for other activation functions are close to flat lines. The numerical results also justify the combination of four kinds of activation functions.

Activation Function	Relative L^2 Error (Moving-Average)	Relative L^2 Error (Min)
ReLU^3	6.12 e-03	4.49 e-03
$x \oplus x^2$	4.41 e-02	3.86 e-02
$x \oplus x^2 \oplus$ ReLU	6.18 e-01	5.97 e-01
$x \oplus x^2 \oplus$ ReLU^3	2.46 e-03	1.56 e-03
$x \oplus x^2 \oplus \sin(x)$	6.59 e-03	3.99 e-03
$x \oplus x^2 \oplus \sin(x) \oplus$ Gaussian	1.60 e-03	8.23 e-04

Table 4.3: The best historical accuracy for the Poisson equation (4.6) with a solution not smooth at the origin.

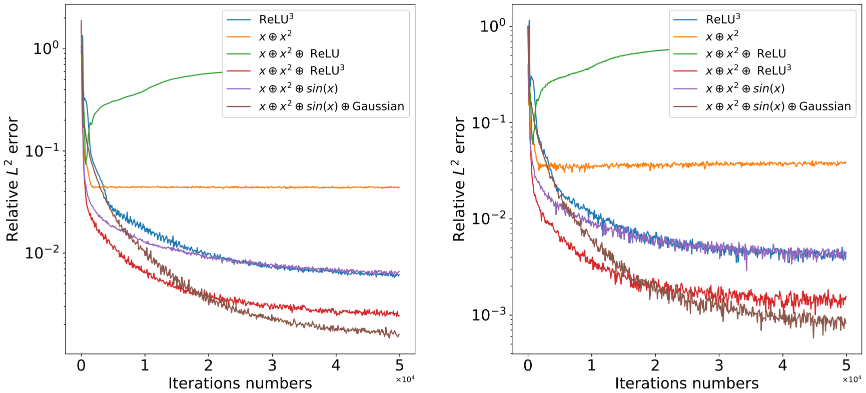

Fig. 4.3: The training process for the Poisson equation (4.6) with a solution not smooth at the origin. Left: the moving-average minimal historical error. Right: the minimal historical error.

4.1.4. Poisson Equation with an Oscillatory Solution. Next, to test the benefit of using $\sin(x)$ in the reproducing activation function, we consider a two-dimensional Poisson equation

$$\begin{cases} -\Delta u - (u-1)^3 + (u+2)^2 = f, & \boldsymbol{x} \in \Omega, \\ u = 0 & \boldsymbol{x}, \in \partial\Omega, \end{cases} \tag{4.8}$$

with an oscillatory solution $u(\boldsymbol{x}) = \sin(6\pi x_1)\sin(6\pi x_2)$ defined on $\Omega = [0,1]^2$. The deep neural network is constructed following (4.5) so that it satisfies the boundary condition automatically. We apply the loss function without boundary penalty introduced in Section 2.3 to identify an approximate solution $\hat{u}(\boldsymbol{x};\boldsymbol{\theta})$ to the Poisson equation in (4.8).

Different activation functions are applied to construct the deep neural network in the above solver. Their performance is shown in Table 4.4 and Figure 4.4 (left). It is obvious that reproducing activation functions with $\mathcal{A} = \{x, x^2, \sin(x)\}$ and $\mathcal{A} = \{x, x^2, \sin(x), \exp(-x^2)\}$ achieve the best performance after $50,000$ iteration. Note that gradient descent in the case of $\mathcal{A} = \{x, x^2, \sin(x), \exp(-x^2)\}$ has a steeper error curve than other activation functions. When the number of iteration is sufficiently large, the performance of $\mathcal{A} = \{x, x^2, \sin(x), \exp(-x^2)\}$ may become the best.

According to the discussion about NTK in Section 2.5, introducing oscillation in neural networks is a crucial step to lessen the spectral bias of deep learning. Although using $\sin(x)$ in reproducing activation functions has helped to relieve the spectral bias as we have seen in the above results. As discussed in [71], pre-specifying a wide range of different scaling parameters in $\sin(x)$ can help to lessen the spectral bias better and obtain high-resolution image (coordinate-wise low-dimensional function) reconstruction. Therefore, in the case of oscillatory target functions, we also specify the scaling parameters as different frequencies in $\sin(x)$ to check the performance. Particularly, if n $\sin(x)$ activation functions is used in a layer, we will replace them with $\{\sin(2\pi x), \sin(4\pi x), \ldots, \sin(2n\pi x)\}$. Besides, it is also of interest to check whether adding $\cos(x)$ in the set of basic activation functions can improve the performance. Note that specifying a wide range of scaling parameters in every hidden layer would create too much oscillation and, hence, we would only specify scaling parameters either in the first or in the last hidden layer. Therefore, a set of four tests were conducted and their results are shown in Table 4.5 and Figure 4.4 (right). Numerical results show that there is no much difference whether $\cos(x)$ is used or not, but specifying different scaling parameters does improve the performance. Specifying different scaling parameters in the first hidden layer is better than in the last hidden layer.

Activation Function	Relative L^2 Error (Average)	Relative L^2 Error (Min)
ReLU^3	3.60 e-03	2.01 e-03
$x \oplus x^2$	6.33 e-01	6.19 e-01
$x \oplus x^2 \oplus$ ReLU	9.47 e-00	9.24 e-00
$x \oplus x^2 \oplus$ ReLU^3	5.58 e-03	2.75 e-03
$x \oplus x^2 \oplus \sin(x)$	6.48 e-04	3.46 e-04
$x \oplus x^2 \oplus \sin(x) \oplus$ Gaussian	3.05 e-03	1.28 e-03

Table 4.4: The best historical accuracy for the Poisson equation in (4.8) with an oscillatory solution.

4.1.5. Nonlinear Schrödinger Equation. The last example of Part I is a d-dimensional nonlinear Schrödinger operator defined below.

$$\mathcal{L}\varphi = -\Delta\varphi + \varphi^3 + V\varphi, \tag{4.9}$$

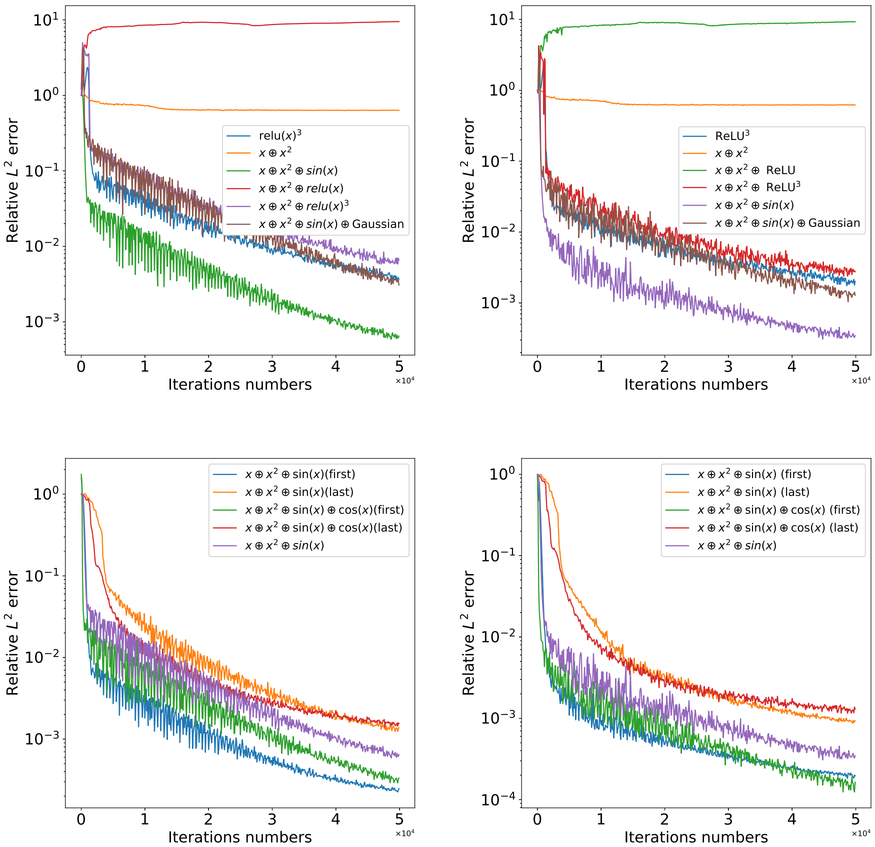

Fig. 4.4: The training process of the Poisson equation in (4.8) with an oscillatory solution.

Activation Function	Relative L^2 Error (Average)	Relative L^2 Error (Min)
$x \oplus x^2 \oplus \sin(x)$ (first)	2.48 e-04	1.96 e-04
$x \oplus x^2 \oplus \sin(x)$ (last)	1.38 e-03	9.36 e-04
$x \oplus x^2 \oplus \sin(x) \oplus \cos(x)$ (first)	3.32 e-04	1.63 e-04
$x \oplus x^2 \oplus \sin(x) \oplus \cos(x)$ (last)	1.55 e-03	1.30 e-03

Table 4.5: The best historical accuracy for the Poisson equation in (4.8) with an oscillatory solution when the scaling parameters of $\sin(x)$ activation functions either in the first hidden layer or the last hidden layer are pre-fixed.

where $V(x) = -\frac{1}{c^2}\exp(\frac{2}{d}\sum_{i=1}^{d}\cos x_i) + \sum_{i=1}^{d}(\frac{\sin^2 x_i}{d^2} - \frac{\cos x_i}{d}) - 3$ such that $\lambda = -3$ and $\varphi(x) = \exp(\frac{1}{d}\sum_{j=1}^{d}\cos(xj))/c$ is the leadingan eigenpair of the operator. This example is considered in [27] and we follow the approach in [27] to solve for the leading eigenpair. The network structure in [27] consists of two parts: 1) the first hidden layer uses $\sin(x)$ and $\cos(x)$ with different scaling parameters so that the whole network satisfies periodic boundary conditions; 2) the other hidden layers uses ReLU activation functions. In our test, we compare different activation functions (e.g., ReLU, ReLU^3, poly-sine-Gaussian) after the first hidden layer. In the case when $d = 5$, the relative L^2 error of the estimated leading eigenfunction after $60,000$ iterations is 0.307, $6.38e - 03$, and

Activation Function	Equation (4.4)	Equation (4.6)	Equation (4.8)
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
$x \oplus x^2 \oplus \text{ReLU}$	1.01 e+11	1.09 e+10	3.11 e+10
$x \oplus x^2 \oplus \text{ReLU}^3$	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 4.6: The condition number of the NTK in (2.19) of different PDE solvers with different activation functions at initialization. The NTK matrix is evaluated with 100 samples, i.e., the matrix size is 100 × 100.

$2.09e-03$ for ReLU^3, ReLU, and poly-sine-Gaussian activation functions, respectively. In the case when $d = 10$, the relative L^2 error of the estimated leading eigenfunction after $80,000$ iterations is 0.223, $4.59e-03$, and $1.10e-03$ for ReLU^3, ReLU, and poly-sine-Gaussian activation functions, respectively. The training process of these solvers have been visualized in Figure 4.5.

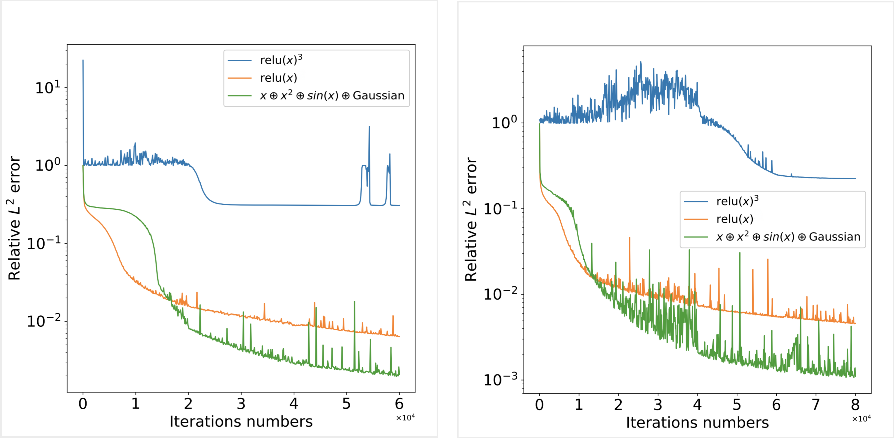

Fig. 4.5: The training process of the nonlinear Schrödinger Equation in (4.9) with different activation functions.

4.1.6. Neural Tangent Kernel of PDE Solvers. As discussed in Section 2.5, the condition number of NTK is also a crucial factor that determines the performance of deep learning. We compute the NTK of the PDE problems we considered in previous examples at initialization when different activation functions are used. These condition numbers are summarized in Table 4.6. The condition number of the poly-sine-Gaussian activation function is the smallest one. Hence, from the perspective of NTK, we have also justified the combination of basic activation functions in the poly-sine-Gaussian activation function.

4.2. Coordinate-based Data Representation. In this part, we verify the performance of reproducing activation functions on the task of continuous representations using coordinate-based neural networks. In this task, neural networks are trained to regress a given signal, which takes low-dimensional coordinates as input and outputs the signal value at the corresponding coordinate point. Mean square error (MSE) is used to quantify the difference between the ground truth and the neural network output. Standard neural networks, e.g., ReLU networks, were shown to have poor performance of fitting some signal patterns, for example, the high-frequency component of signals [69, 71]. SIREN activation function [69], i.e., $\sin(30x)$, improves the ability of neural networks to represent complex signals.

As we discussed in the theoretical part of reproducing activation function, the SIREN activation function is a special case of the poly-sine-Gaussian activation function in our framework. We will show that 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. In our implementation, we follow the official implementation of SIREN on representations of audio, image, and video signal (refer to [69] for implementation details). The main difference between the SIREN code and our implementation is the activation function. In all of the examples in Part II, each neuron will have a data-driven reproducing activation function as in (3.1). All trainable parameters are trained to minimize the empirical loss function in (3.2).

4.2.1. Audio Signal. We start from modeling audio signals on two audio clips, Bach and Counting as shown in Figure 4.6. A neural network 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. The reproducing activation function is set as

$$\alpha_1 \sin(\beta_1 x) + \alpha_2 \mathrm{exp}(-x^2/(2\beta_2^2)),$$

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 $\mathcal{N}(30, 10)$, and β_2 is initialized as $\mathcal{N}(0.04, 0.001)$. 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 neural networks are trained for 2000 iterations and optimized by Adam optimizer with an initial learning rate 10^{-4} and cosine learning rate decay.

Figure 4.6 displays the fitting performance and training curves on the audio Bach and Counting. From Figure 4.6, we can see that our method has the capacity of modeling the audio signals more accurately than SIREN and leads to a smaller error in regression. Besides, our reproducing activation function can converge to a better local minimum at a faster speed compared with SIREN.

4.2.2. Image Signal. In this part, we regress a grayscale image by learning a map from two-dimensional pixel coordinates to the corresponding pixel value. We evaluate the performance using two image examples: Cameraman and Astronaut, which are available in Python Pillow Package. Note that images usually contain a cartoon part and a texture part. Therefore, the following reproducing activation function is used

$$\alpha_1 \sin(\beta_1 x) + \alpha_2 x + \alpha_3 x^2, \tag{4.10}$$

where α_1 is initialized as $\mathcal{N}(2, 0.1)$, α_2 is initialized as $\mathcal{N}(0, 0.1)$, α_3 is initialized as $\mathcal{N}(1.0, 0.1)$, and β_1 is initialized as $\mathcal{N}(30, 0.01)$. A neural network with 3 hidden layers and 256 neurons per layer is trained for $2,000$ iterations. The Adam optimizer is used with an initial learning rate 10^{-4} and cosine learning rate decay. Table 4.7 presents the Peak signal-to-noise ratio (PSNR) and Structural similarity (SSIM) of the fitted images by SIREN and our reproducing activation function. From Table 4.7, we see that our method outperforms SIREN with a significant margin.

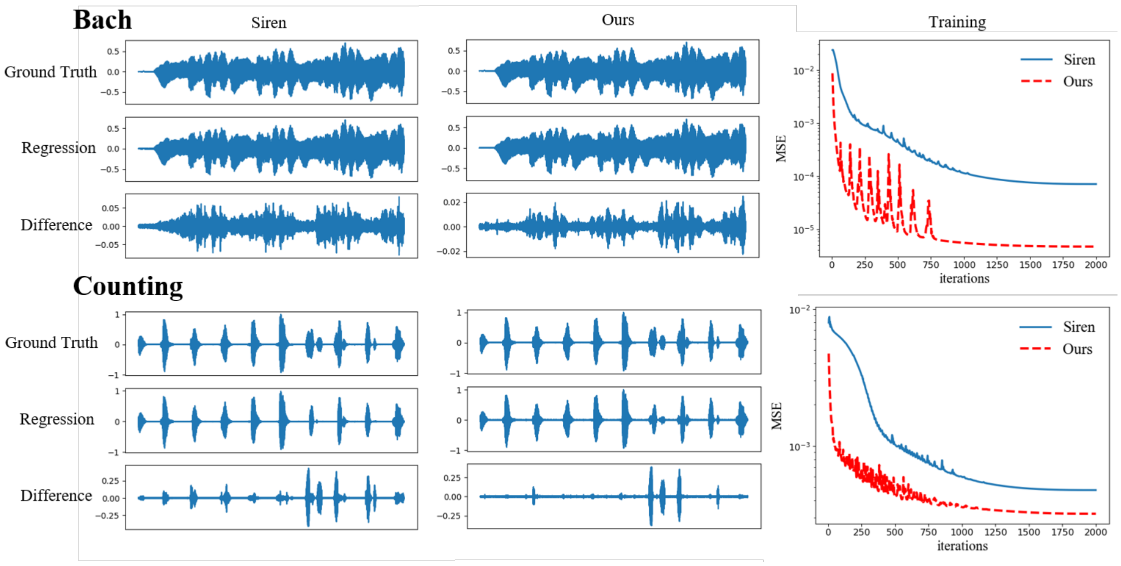

Fig. 4.6: The comparison of fitting performance and training on the audio Bach and Counting for SIREN and our reproducing activation function.

Image Name	Cameraman	Astronaut
SIREN	45.80/0.9913	44.84/0.9962
Ours	61.22/0.9996	58.73/0.9996

Table 4.7: The comparison of PSNR/SSIM of the fitted images using different activation functions. The larger these numbers are, the better the performance is.

Video	Mean PSNR	Std PSNR
SIREN	32.17	2.16
Ours	32.40	2.32

Table 4.8: The comparison of PSNR of videos fitted by different activation functions. The mean and average are computed over 250 frames of the video.

4.2.3. Video Signal. In this part, we fit a color video named Bike with 250 frames, which is available in Python Skvideo Package. The regression is from three-dimensional coordinates to RGB pixel values. We apply the same reproducing activation function in 4.10, but α_1 is initialized as $\mathcal{N}(-0.5, 0.1)$, α_2 is initialized as $\mathcal{N}(1, 0.1)$, α_3 is initialized as $\mathcal{N}(0, 0.1)$, and β_1 is initialized as $\mathcal{N}(30, 0.01)$. A neural networks with 3 hidden layers and 400 neurons per layer is trained for $10,000$ iterations. The Adam optimizer is used with an initial learning rate 10^{-4} and cosine learning rate decay. Figure 4.7 displays the video frames fitted by different activation functions and the corresponding training curves. Table 4.8 shows the mean and standard derivation of PSNR for video over 250 frames. From Figure 4.7, we see that the visual difference is not much but our activation function can converge faster with a larger PSNR.

5. Conclusion. In this paper, we proposed the reproducing activation function to improve deep learning accuracy for various applications ranging from computer vision problems to scientific computing problems. The idea of reproducing activation functions is to employ several basic functions with learnable linear combination and rescaling to construct neuron-wise data-driven activation functions for each neuron. In theory, we have proved that this new concept can lead to powerful deep neural networks approximating various kinds of functions better than ReLU neural networks. In terms of training dynamics of deep learning, we have numerically demonstrated that

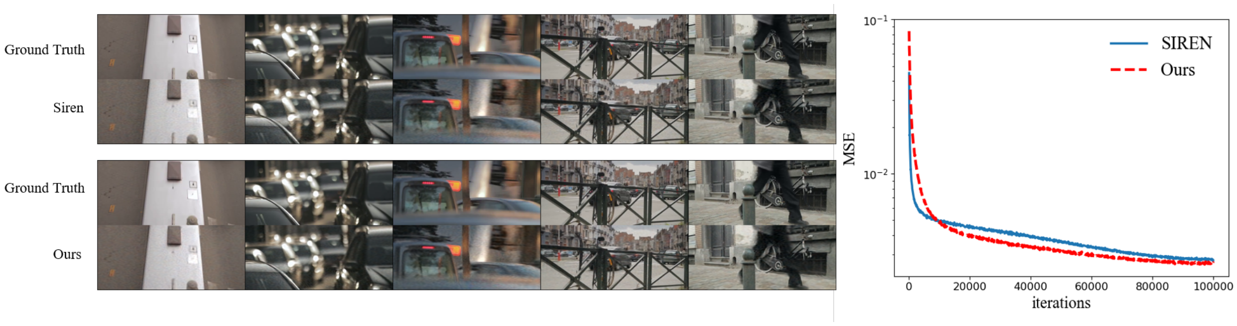

Fig. 4.7: Comparison of the video frames fitted by different activation functions and the corresponding training curves.

reproducing activation functions can generate neural tangent kernels with a better condition number than traditional activation functions, lessening the spectral bias of deep learning. As demonstrated by extensive numerical tests, the proposed activation function 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. 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.

Acknowledgements. C. W. was partially supported by National Science Foundation Award DMS-1849483. H. Y. was partially supported by the US National Science Foundation under award DMS-1945029. The authors thank Mo Zhou for sharing his code for Schrödinger equations.

REFERENCES

[1] Sanjeev Arora, Simon S. Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. *arxiv:1901.08584*, 2019.

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

[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.

[4] Julius Berner, Philipp Grohs, and Arnulf 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. *CoRR*, abs/1809.03062, 2018.

[5] Nicolas Boullé, Yuji Nakatsukasa, and Alex Townsend. Rational neural networks. *arXiv:2004.01902*, 2020.

[6] Wei Cai, X. Li, and L. Liu. A phase shift deep neural network for high frequency approximation and wave problems. *arXiv: Learning*, 2019.

[7] Zhiqiang Cai, Jingshuang Chen, Min Liu, and Xinyu Liu. Deep least-squares methods: An unsupervised learning-based numerical method for solving elliptic pdes. *Journal of Computational Physics*, page 109707, 2020.

[8] Yuan Cao, Zhiying Fang, Yue Wu, Ding-Xuan Zhou, and Quanquan Gu. Towards understanding the spectral bias of deep learning. *arXiv preprint arXiv:1912.01198*, 2019.

[9] J. Chen, Rui Du, Panchi Li, and Liyao Lyu. Quasi-monte carlo sampling for machine-learning partial differential equations. *ArXiv*, abs/1911.01612, 2019.

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

[11] Zixiang Chen, Yuan Cao, Difan Zou, and Quanquan Gu. How much over-parameterization is sufficient to learn deep relu networks? *CoRR*, arXiv:1911.12360, 2019.

[12] X. Dai and Y. Zhu. Towards theoretical understanding of large batch training in stochastic gradient descent. *CoRR*, abs/1812.00542, 2018.

[13] Xiaowu Dai and Yuhua Zhu. Towards theoretical understanding of large batch training in stochastic gradient descent. *arXiv preprint arXiv:1812.00542*, 2018.

[14] M. W. M. G. Dissanayake and N. Phan-Thien. Neural-network-based Approximations for Solving Partial Differential Equations. *Comm. Numer. Methods Engrg.*, 10:195–201, 1994.

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

[16] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res*, 12:2121–2159, 2011.

[17] W. E and Q. Wang. Exponential convergence of the deep neural network approximation for analytic functions. *CoRR*, abs/1807.00297, 2018.

[18] Weinan E, Chao Ma, and Qingcan Wang. A priori estimates of the population risk for residual networks. *ArXiv*, abs/1903.02154, 2019.

[19] Weinan E, Chao Ma, and Lei Wu. A priori estimates of the population risk for two-layer neural networks. *Communications in Mathematical Sciences*, 17(5):1407 – 1425, 2019.

[20] Weinan E and Bing Yu. The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. *Commun. Math. Stat.*, 6:1–12, 2018.

[21] K. Genova, F. Cole, A. Sud, A. Sarna, and T. Funkhouser. Local deep implicit functions for 3d shape. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4856–4865, 2020.

[22] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning.* MIT Press, Cambridge, 2016.

[23] Y. Gu, H. Yang, and C. Zhou. SelectNet: Self-paced Learning for High-dimensional Partial Differential Equations. *arXiv e-prints*, arXiv:2001.04860, 2020.

[24] Yiqi Gu, Chunmei Wang, and Haizhao Yang. Structure probing neural network deflation. *arXiv preprint arXiv:2007.03609*, 2020.

[25] 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.

[26] Jiequn Han and Jihao Long. Convergence of the deep bsde method for coupled fbsdes. *ArXiv*, abs/1811.01165, 2018.

[27] Jiequn Han, Jianfeng Lu, and Mo Zhou. Solving high-dimensional eigenvalue problems using deep neural networks: A diffusion monte carlo like approach. *Journal of Computational Physics*, 423:109792, 2020.

[28] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[29] Jianguo Huang, Haoqin Wang, and Haizhao Yang. Int-deep: A deep learning initialized iterative method for nonlinear problems. *Journal of Computational Physics*, 419:109675, 2020.

[30] 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.

[31] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *CoRR*, abs/1806.07572, 2018.

[32] Ameya D. Jagtap, Kenji Kawaguchi, and George Em Karniadakis. Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 476(2239):20200334, 2020.

[33] Ameya D. Jagtap, Kenji Kawaguchi, and George Em Karniadakis. Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *Journal of Computational Physics*, 404:109136, 2020.

[34] Timothy Jeruzalski, Boyang Deng, Mohammad Norouzi, J. P. Lewis, G. Hinton, and A. Tagliasacchi. Nasa: Neural articulated shape approximation. *ArXiv*, abs/1912.03207, 2020.

[35] S. Justin and S. Konstantinos. Dgm: A deep learning algorithm for solving partial differential equations. *J. Comput. Phys.*, 375:1339–1364, 2018.

[36] Yuehaw Khoo, Jianfeng Lu, and Lexing Ying. Solving parametric pde problems with artificial neural networks. *arXiv: Numerical Analysis*, 2017.

[37] D. P. Kingma and J. Ba. Adam: a method for stochastic optimization. *arXiv e-prints*, arXiv:1412.6980, 2014.

[38] V. Kůrková. Kolmogorov's theorem and multilayer neural networks. *Neural Networks*, 5:501–506, 1992.

[39] I.E. Lagaris, A. Likas, and D. I. Fotiadis. Artificial Neural Networks for Solving Ordinary and Partial Differential Equations. *IEEE Trans. Neural Networks*, 9:987–1000, 1998.

[40] Jaehoon Lee, Lechao Xiao, Samuel S Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. *Journal of Statistical Mechanics: Theory and Experiment*, 2020(12):124002, dec 2020.

[41] D. Lei, Z. Sun, Y. Xiao, and W. Y. Wang. Implicit regularization of stochastic gradient descent in natural language processing: observations and implications. *arXiv e-prints*, arXiv:1811.00659, 2018.

[42] Deren Lei, Zichen Sun, Yijun Xiao, and William Yang Wang. Implicit regularization of stochastic gradient descent in natural language processing: Observations and implications. *arXiv preprint arXiv:1811.00659*, 2018.

[43] S. Liang and R. Srikant. Why deep neural networks? *CoRR*, abs/1610.04161, 2016.

[44] Yulei Liao and Pingbing Ming. Deep nitsche method: Deep ritz method with essential boundary conditions. *arXiv preprint arXiv:1912.01309*, 2019.

[45] S. Liu, Y. Zhang, S. Peng, B. Shi, M. Pollefeys, and Z. Cui. Dist: Rendering deep implicit signed distance function with differentiable sphere tracing. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2016–2025, 2020.

[46] Ziqi Liu, Wei Cai, and Zhi-Qin John Xu. Multi-scale deep neural network (mscalednn) for solving poisson-boltzmann equation in complex domains. *Communications in Computational Physics*, 28(5):1970–2001, Jun 2020.

[47] J. Lu, Z. Shen, H. Yang, and S. Zhang. Deep Network Approximation for Smooth Functions. *arXiv e-prints*, arXiv:2001.03040, 2020.

[48] T. Luo, Z. Ma, Z.J. Xu, and Y. Zhang. Theory of the frequency principle for general deep neural networks. *CoRR*, abs/1906.09235, 2019.

[49] Tao Luo and Haizhao Yang. Two-Layer Neural Networks for Partial Differential Equations: Optimization and Generalization Theory. *arXiv e-prints*, arXiv:2006.15733, 2020.

[50] Liyao Lyu, Keke Wu, Rui Du, and J. Chen. Enforcing exact boundary and initial conditions in the deep mixed residual method. *ArXiv*, abs/2008.01491, 2020.

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

[52] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *arxiv:2003.08934*, 2020.

[53] H. Montanelli and Q. Du. New error bounds for deep networks using sparse grids. *arXiv e-prints*, arXiv:1712.08688, 2017.

[54] H. Montanelli and H. Yang. Error bounds for deep relu networks using the kolmogorov–arnold superposition theorem. *Neural Networks*, 129:1–6, 2020.

[55] Hadrien Montanelli, Haizhao Yang, and Qiang Du. Deep relu networks overcome the curse of dimensionality for bandlimited functions. *arXiv preprint arXiv:1903.00735*, 2019.

[56] Tenavi Nakamura-Zimmerer, Qi Gong, and Wei Kang. Adaptive deep learning for high dimensional hamilton-jacobi-bellman equations. *ArXiv*, abs/1907.05317, 2019.

[57] B. Neyshabur, R. Tomioka, R. Salakhutdinov, and N. Srebro. Geometry of optimization and implicit regularization in deep learning. *arXiv e-prints*, arXiv:1705.03071, 2017.

[58] Behnam Neyshabur, Ryota Tomioka, Ruslan Salakhutdinov, and Nathan Srebro. Geometry of optimization and implicit regularization in deep learning. *arXiv preprint arXiv:1705.03071*, 2017.

[59] J.A.A. Opschoor, C. Schwab, and J. Zech. Exponential relu dnn expression of holomorphic maps in high dimension. Technical report, Zurich, 2019.

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

[61] 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. *International Journal of Automation and Computing*, 14:503–519, 2017.

[62] 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. *Journal of Computational Physics*, 378:686 – 707, 2019.

[63] S. J. Reddi, S. Kale, and S. Kumar. On the convergence of Adam and beyond. *arXiv e-prints*, arXiv:1904.09237, 2019.

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

[65] Z. Shen, H. Yang, and S. Zhang. Deep network approximation characterized by number of neurons. *arXiv e-prints*, arXiv:1906.05497, 2019.

[66] Zuowei Shen, Haizhao Yang, and Shijun Zhang. Neural network approximation: Three hidden layers are enough. *arXiv:2010.14075*, 2020.

[67] Zuowei Shen, Haizhao Yang, and Shijun Zhang. Deep network with approximation error being reciprocal of width to power of square root of depth. *Neural Computation*, 2021.

[68] Yeonjong Shin, J. Darbon, and G. Karniadakis. On the convergence and generalization of physics informed neural networks. *ArXiv*, abs/2004.01806, 2020.

[69] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33, 2020.

[70] Vincent Sitzmann, M. Zollhöfer, and G. Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. *ArXiv*, abs/1906.01618, 2019.

[71] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *arxiv:2006.10739*, 2020.

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

[73] Bo Wang. Multi-scale deep neural network (mscalednn) methods for oscillatory stokes flows in complex domains. *Communications in Computational Physics*, 28(5):2139–2157, Jun 2020.

[74] Sifan Wang, Xinling Yu, and Paris Perdikaris. When and why pinns fail to train: A neural tangent kernel perspective. *arXiv:2007.14527*, 2020.

[75] Zhi-Qin John Xu, Yaoyu Zhang, Tao Luo, Yanyang Xiao, and Zheng Ma. Frequency principle: Fourier analysis sheds light on deep neural networks. *Communications in Computational Physics*, 28(5):1746–1767, 2020.

[76] D. Yarotsky. Error bounds for approximations with deep relu networks. *Neural Networks*, 94:103–114, 2017.

[77] D. Yarotsky. Optimal approximation of continuous functions by very deep relu networks. In *31st Annual Conference on Learning Theory*, volume 75, pages 1–11. 2018.

[78] Dmitry Yarotsky and Anton Zhevnerchuk. The phase diagram of approximation rates for deep neural networks. *arXiv e-prints*, page arXiv:1906.09477, June 2019.

[79] Z. Song Z. A.-Zhu, Y. Li. A convergence theory for deep learning via over-parameterization. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 242–252, Long Beach, California, USA, 2019. PMLR.

[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.

[81] Ellen D. Zhong, Tristan Bepler, Joseph H. Davis, and Bonnie Berger. Reconstructing continuous distributions of 3d protein structure from cryo-em images. *arXiv:1909.05215*, 2020.