Reproducing Activation Function for Deep Learning

Senwei Liang [*1] **Liyao Lyu** [*2] **Chunmei Wang** [3] **Haizhao Yang** [1]

Abstract

We propose reproducing activation functions (RAFs) to improve deep learning accuracy for various applications ranging from computer vision to scientific computing. The idea is to employ several basic functions and their learnable linear combination to construct neuron-wise data-driven activation functions for each neuron. Armed with RAFs, neural networks (NNs) can reproduce traditional approximation tools and, therefore, approximate target functions with a smaller number of parameters than traditional NNs. In NN training, RAFs can generate neural tangent kernels (NTKs) with a better condition number than traditional activation functions lessening the spectral bias of deep learning. As demonstrated by extensive numerical tests, the proposed RAFs can facilitate the convergence of deep learning optimization for a solution with higher accuracy than existing deep learning solvers for audio/image/video reconstruction, PDEs, and eigenvalue problems. With RAFs, the errors of audio/video reconstruction, PDEs, and eigenvalue problems are decreased by over 14%, 73%, 99%, respectively, compared with baseline, while the performance of image reconstruction increases by 58%.

1. Introduction

Deep neural networks are an important tool for solving a wide range regression problems with surprising performance. For example, as a mesh-free representation of objects, scene geometry, and appearance, the so-called "coordinate-based" networks (Tancik et al., 2020) take low-dimensional coordinates as inputs and output an object value of the shape, density, and/or color at the given input coordinate. Another example is NN-based solvers for high-dimensional and nonlinear partial differential equations (PDEs) in complicated domains (Dissanayake & Phan-Thien, 1994; Han et al., 2018; Raissi et al., 2019; Khoo et al., 2017). However, the optimization problem for above applications is highly non-convex making it challenging to obtain a highly accurate solution. Exploring different neural network architectures and training strategies for highly accurate solutions have been an active research direction (Sitzmann et al., 2020; Tancik et al., 2020; Jagtap et al., 2020b).

We propose RAFs to improve deep learning accuracy. The idea is to employ several basic functions and their learnable linear combination to construct neuron-wise data-driven activation functions. Armed with RAFs, NNs can reproduce traditional approximation tools efficiently, e.g., orthogonal polynomials, Fourier basis functions, wavelets, radial basis functions. Therefore, NNs with the proposed RAF can approximate a wide class of target functions with a smaller number of parameters than traditional NNs. Therefore, the data-driven activation functions are called reproducing activation functions (RAFs).

In NN training, RAFs can empirically generate NTKs with a better condition number than traditional activation functions lessening the spectrum bias of deep learning. NN optimization usually can only find the smoothest solution with the fastest decay in the frequency domain due to the implicit regularization of network structures (Xu et al., 2020; Cao et al., 2019; Neyshabur et al., 2017a; Lei et al., 2018a), which can be generalized to PDE problems, e.g., the optimization and generalization analysis (Luo & Yang, 2020) and the spectral bias (Wang et al., 2020). Therefore, designing an efficient algorithm to identify oscillatory or singular solutions to regression and PDE problems is challenging.

Contribution. We summarize our contribution as follows,

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

2. Empirically, RAFs can generate NTKs with a smaller condition number than traditional activation functions lessening the spectrum bias of NNs.

[*]Equal contribution [1]Department of Mathematics, Purdue University, West Lafayette, USA [2]Department of Computational Mathematics, Science, and Engineering, Michigan State University, East Lansing, USA [3]Department of Mathematics & Statistics, Texas Tech University, Lubbock, USA. Correspondence to: Haizhao Yang <haizhao@purdue.edu>.

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

2. Related Works

NN-based PDE solvers. First of all, NNs as a mesh-free parametrization can efficiently approximate various high-dimensional solutions with dimension-independent approximation rates (Yarotsky & Zhevnerchuk, 2019; Montanelli & Yang, 2020; Hutzenthaler et al., 2019; Shen et al., 2021; 2020) and/or achieving exponential approximation rates (E & Wang, 2018; Opschoor et al., 2019; Shen et al., 2021; 2020). Second, NN-based PDE solvers enjoy simple implementation and work well for nonlinear PDEs on complicated domains. There has been extensive research on improving the accuracy of these PDE solvers, e.g., improving the sampling strategy of SGD (Nakamura-Zimmerer et al., 2019; Chen et al., 2019a) or the sample weights in the objective function (Gu et al., 2020b), building physics-aware NNs (Cai et al., 2019; Liu et al., 2020; Gu et al., 2020a), combining traditional iterative solvers (Xu et al., 2020; Huang et al., 2020).

Take the example of boundary value problems (BVP) and the least squares method (Dissanayake & Phan-Thien, 1994). Consider the BVP

$$\begin{aligned} \mathcal{D}u(\boldsymbol{x}) &= f(u(\boldsymbol{x}), \boldsymbol{x}), \text{ in } \Omega, \\ \mathcal{B}u(\boldsymbol{x}) &= g(\boldsymbol{x}), \text{ on } \partial\Omega, \end{aligned} \tag{1}$$

where $\mathcal{D} : \Omega \to \Omega$ is a differential operator that can be nonlinear, $\Omega \subset \mathbb{R}^d$ is a bounded domain, and $\mathcal{B}u = g$ characterizes the boundary condition. Special network structures $\phi(\boldsymbol{x};\boldsymbol{\theta})$ parameterized by $\boldsymbol{\theta}$ can be proposed such that $\phi(\boldsymbol{x};\boldsymbol{\theta})$ can satisfy boundary conditions, and the loss function over the given points $\{x_i\}_{i=1}^N$ becomes

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

Coordinate-based network. Deep NNs 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. This strategy is compelling in data compression and reconstruction, e.g., see (Tancik et al., 2020; Chen & Zhang, 2019; Jeruzalski et al., 2020; Genova et al., 2020; Michalkiewicz et al., 2019; Park et al., 2019; Liu et al., 2020; Saito et al., 2019; Sitzmann et al., 2019).

Neural tangent kernel. NTK is a tool to study the training behavior of deep learning in regression problems and PDE problems (Jacot et al., 2018; Cao et al., 2019; Luo & Yang, 2020; Wang et al., 2020). Let $\mathcal{X}$ be $\{\boldsymbol{x}_i\}_{i=1}^N$, $\mathcal{Y}$ be the set of function values, $\hat{\mathcal{J}}(\boldsymbol{\theta})$ be the square error loss for regression problem. Using gradient flow to analyze the training dynamics of $\hat{\mathcal{J}}(\boldsymbol{\theta})$, we have the following evolution equations: $\dot{\boldsymbol{\theta}}_t = -\nabla_{\boldsymbol{\theta}}\phi_t(\mathcal{X})^T \nabla_{\phi_t(\mathcal{X})}\hat{\mathcal{J}}$, and $\dot{\phi}_t(\mathcal{X}) = \nabla_{\boldsymbol{\theta}}\phi_t(\mathcal{X})\dot{\boldsymbol{\theta}}_t = -\hat{\Theta}_t(\mathcal{X},\mathcal{X})\nabla_{\phi_t(\mathcal{X})}\hat{\mathcal{J}}$, where $\boldsymbol{\theta}_t$ is the parameter set at iteration time t, $\phi_t(\mathcal{X}) = \text{vec}([\phi_t(\boldsymbol{x};\boldsymbol{\theta}_t)]_{\boldsymbol{x}\in\mathcal{X}})$ is the $N \times 1$ vector of concatenated function values for all samples, and $\nabla_{\phi_t(\mathcal{X})}\hat{\mathcal{J}}$ is the gradient of the loss with respect to the network output vector $\phi_t(\mathcal{X})$, $\hat{\Theta}_t := \hat{\Theta}_t(\mathcal{X},\mathcal{X})$ in $\mathbb{R}^{N\times N}$ is the NTK at iteration 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.

If the following linearized network by Taylor expansion is considered, $\phi_t^{\text{lin}}(\boldsymbol{x}) := \phi(\boldsymbol{x};\boldsymbol{\theta}_0) + \nabla_{\boldsymbol{\theta}}\phi(\boldsymbol{x};\boldsymbol{\theta}_0)\boldsymbol{\omega}_t$, where $\boldsymbol{\omega}_t := \boldsymbol{\theta}_t - \boldsymbol{\theta}_0$ is the change in the parameters from their initial values. The closed form solutions are

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

and

$$\phi_t^{\text{lin}}(\boldsymbol{x}) - \phi_0(\boldsymbol{x}) = \hat{\Theta}_0(\boldsymbol{x},\mathcal{X})\hat{\Theta}_0^{-1}\left(I - e^{-\hat{\Theta}_0 t}\right)(\mathcal{Y} - \phi_0(\mathcal{X})). \tag{3}$$

There mainly two kinds of observations from (27) 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 (27). 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 (27) approaches to $\phi_0(\mathcal{X}) - \mathcal{Y}$ and, hence, $\phi_t^{\text{lin}}(\boldsymbol{x})$ goes to the desired function value for $\boldsymbol{x} \in \mathcal{X}$. However, in practice, when $\hat{\Theta}_0$ is very ill-conditioned, a small approximation error in $\left(I - e^{-\hat{\Theta}_0 t}\right)(\phi_0(\mathcal{X}) - \mathcal{Y}) \approx \phi_0(\mathcal{X}) - \mathcal{Y}$ may be amplified significantly, resulting in a poor accuracy for $\phi_t^{\text{lin}}(\boldsymbol{x})$ to solve the regression problem.

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

$$\hat{\Theta}_t = \left(\nabla_{\boldsymbol{\theta}}\mathcal{D}\phi_t(\mathcal{X})\right)\left(\nabla_{\boldsymbol{\theta}}\mathcal{D}\phi_t(\mathcal{X})\right)^T, \tag{4}$$

where $\mathcal{D}$ is the differential operator of the PDE. Similar to the discussion for regression problems, the spectral bias and

the conditioning issue also exist in deep learning based PDE solvers by almost the same arguments.

3. Reproducing Activation Functions

3.1. Abstract Framework

The concept of RAFs is to apply different activation functions in different neurons. Let $\mathcal{A} = \{\gamma_1(x), \ldots, \gamma_P(x)\}$ be a set of P basic activation functions. In the i-th neuron of the ℓ-th layer, an activation function

$$\sigma_{i,\ell}(x) = \sum_{p=1}^{P} \alpha_{p,i,\ell}\gamma_p(\beta_{p,i,\ell}x) \tag{5}$$

is applied, where $\alpha_{p,i,\ell}$ is a learnable combination coefficient and $\beta_{p,i,\ell}$ is a learnable scaling parameter. Let $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ be the union of all learnable combination coefficients and scaling parameters, respectively, we use $\phi(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\alpha},\boldsymbol{\beta})$ to denote an NN with $\boldsymbol{\theta}$ as the set of all other parameters.

In regression problems, given samples $\{\mathbf{x}_i, y_i\}_{i=1}^N$, the empirical loss with RAFs is

$$\hat{\mathcal{J}}(\boldsymbol{\theta},\boldsymbol{\alpha},\boldsymbol{\beta}) = \frac{1}{2N}\sum_{i=1}^{N} |\phi(\mathbf{x}_i;\boldsymbol{\theta},\boldsymbol{\alpha},\boldsymbol{\beta}) - y_i|^2 . \tag{6}$$

3.2. Examples and Reproducing Properties

3.2.1. Example 1: Sine-ReLU

Sine-ReLU networks proposed in (Yarotsky & Zhevnerchuk, 2019) apply sine $\sin(x)$ or ReLU $\max\{0,x\}$ in each neuron. Instead, the proposed RAF here has a set of trainable parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. In fact, $\sin(x)$ can be replaced by any Lipschitz periodic function. Let $F_{r,d}$ be the unit ball of the d-dimensional Sobolev space $H^{r,\infty}([0,1]^d)$. We have the following theorem according to Thm. 6.1 of (Yarotsky & Zhevnerchuk, 2019).

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

There are other types of network structures utilizing both $\sin(x)$ and ReLU activation functions but for different application purposes and with different strategies, e.g., (Zhong et al., 2020; Mildenhall et al., 2020; Han et al., 2020; Liu et al., 2020; Wang, 2020; Tancik et al., 2020).

3.2.2. Example 2: Floor-Exponential-Sign

Recently, networks with super approximation power (e.g., an exponential approximation rate without the curse of dimensionality for Hölder continuous functions) have been proposed in (Shen et al., 2021; 2020), e.g., the Floor-Exponential-Sign Network that uses one of the following three activation functions in each neuron:

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

Here, $\mathcal{T}(x) := \mathbb{1}_{x\ge 0} = \begin{cases} 1, & x \ge 0, \\ 0, & x < 0. \end{cases}$ The proposed RAF has a set of trainable parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. By Thm. 1.1 in (Shen et al., 2020), we have the following theorem.

Theorem 2 (Dimension-Independent and Exponential Approximation Rate) *Given f in $C([0,1]^d)$ and $W \in \mathbb{N}^+$, there exists an NN $\phi(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\alpha},\boldsymbol{\beta})$ of width W and depth 4 built with RAFs associated with $\mathcal{A} = \{\sigma_1(x), \sigma_2(x), \sigma_3(x)\}$ such that, for any $\boldsymbol{x} \in [0,1)^d$,*

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

with at most $2W^2 + (d+22)W + 1$ parameters.

Here, $\omega_f(\cdot)$ is the modulus of continuity of f defined as

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

for any $r \ge 0$, where $\|\boldsymbol{x}\|_2$ is the length of $\boldsymbol{x} \in \mathbb{R}^d$.

3.2.3. Example 3: Poly-Sine-Gaussian

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

Theorem 3 (Reproducing Polynomials) *Assume $P(\boldsymbol{x}) = \sum_{j=1}^{J} c_j \boldsymbol{x}^{\boldsymbol{\alpha}_j}$ for $\boldsymbol{\alpha}_j \in \mathbb{N}^d$. For any $N, L, a, b \in \mathbb{N}^+$ such that $ab \ge J$ and $(L-2b-b\log_2 N)N \ge b\max_j |\boldsymbol{\alpha}_j|$, there exists a poly-sine-Gaussian network ϕ with width $2Na+d+1$ and depth L such that $\phi(\boldsymbol{x}) = P(\boldsymbol{x})$ for any $\boldsymbol{x} \in \mathbb{R}^d$.*

Thm. 3 characterizes how well poly-sine-Gaussian networks reproduce arbitrary polynomials including orthogonal polynomials. Compared to the results of ReLU NNs for polynomials in (Yarotsky, 2017; Lu et al., 2020), poly-sine-Gaussian networks require less parameters. Orthogonal

polynomials are important tools for classical approximation theory and numerical computation. For example, the Chebyshev series lies at the heart of approximation theory. In particular, for analytic functions, the truncated Chebyshev series defined as $f_n(x) = \sum_{k=0}^{n} c_k T_k(x/M)$ are *exponentially accurate* approximations Thm. 8.2 (Trefethen, 2013), where T_k is the Chebyshev polynomial of degree k defined on $[-1, 1]$. More precisely, for some scalars $M \geq 1$ and $s > 1$, if we define $a_s^M = M\frac{s+s^{-1}}{2}, b_s^M = M\frac{s-s^{-1}}{2}$, and the *Bernstein s-ellipse scaled to* $[-M, M]$,

$$E_s^M = \left\{ x + iy \in \mathbb{C} : \frac{x^2}{(a_s^M)^2} + \frac{y^2}{(b_s^M)^2} = 1 \right\},$$

then we have the following theorem.

Theorem 4 (Exponential Approximation Rate) *For any $M \geq 1$, $s > 1$, $C_f > 0$, $0 < \epsilon < 1$, and any real-valued analytic function f on $[-M, M]$ that is analytically continuable to the open ellipse E_s^M, where it satisfies $|f(x)| \leq C_f$, there is a poly-sine-Gaussian network ϕ with width $2N + 2$ and depth L such that $\|\phi(x) - f(x)\|_{L^\infty([-M,M])} \leq \epsilon$, where N and L are positive integers satisfying $(L - 2n - 2 - (n+1)\log_2 N)N \geq n(n+1)$ and $n = \mathcal{O}\left(\frac{1}{\log_2 s} \log_2 \frac{2C_f}{\epsilon}\right)$.*

By choosing $N = \mathcal{O}(n)$ and $L = \mathcal{O}(n \log_2(n))$ in Thm. 4, the width and depth of ϕ are $\mathcal{O}\left(\log_2 \frac{1}{\epsilon}\right)$ and $\mathcal{O}\left(\left(\log_2 \frac{1}{\epsilon}\right) \log_2 \left(\log_2 \frac{1}{\epsilon}\right)\right)$, respectively, leading to a network size smaller than that of the ReLU NN in Thm. 2.6 in (Montanelli et al., 2019).

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

Definition 1 *Let $d \geq 2$ be an integer, $M \geq 1$ be a scalar, and $B = [0, 1]^d$. Suppose $K : \mathbb{R} \to \mathbb{C}$ is analytic and bounded by a constant $D_K \in (0, 1]$ on $[-dM, dM]$ and K satisfies the assumption of Thm. 4 for $s > 1$ and $C_K > 0$. We define the Hilbert space $\mathcal{H}_{K,M}(B)$ of generalized bandlimited functions via*

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

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

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

Theorem 5 (Dimension-Independent Approximation) *For any real-valued function f in $\mathcal{H}_{K,M}(B)$, $M \geq 1$, $s > 1$, $C_K > 0$, and $d \geq 2$. Let us assume that $\int_{\mathbb{R}^d} |F(\boldsymbol{w})| d\boldsymbol{w} = \int_{[-M,M]^d} |F(\boldsymbol{w})| d\boldsymbol{w} = C_F$. For any measure μ and $\epsilon \in (0, 1)$, there exists a poly-sine-Gaussian network ϕ on $B = [0, 1]^d$, that has width $\mathcal{O}\left(\frac{4C_F\sqrt{\mu(B)}}{\epsilon^2 \log_2 s} \log_2 \frac{4C_F\sqrt{\mu(B)}C_K}{\epsilon}\right)$ and depth $\mathcal{O}\left(\left(\frac{1}{\log_2 s} \log_2 \frac{4C_F\sqrt{\mu(B)}C_K}{\epsilon}\right) \log_2 \log_2 \frac{4C_F\sqrt{\mu(B)}C_K}{\epsilon}\right)$ such that $\|\phi - f\|_{L^2(\mu,B)} = \sqrt{\int_B |\phi(\boldsymbol{x}) - f(\boldsymbol{x})|^2 d\mu(\boldsymbol{x})} \leq \epsilon$.*

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

Lemma 1 *(i) Poly-sine-Gaussian networks can reproduce all basis functions in the discrete cosine transform and the discrete windowed cosine transform with a Gaussian window function in an arbitrary dimension.*

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

Lem. 1 above implies that poly-sine-Gaussian networks may be useful in many computer vision and audio tasks involving Fourier transforms and wavelet transforms. Due to the advantage of wavelets to represent functions with singularity, poly-sine-Gaussian networks may also be useful in representing functions with singularity. We would like to highlight that the Gaussian function may not be the optimal choice in the concept of RAF. Other window functions in wavelet analysis may provide better performance and this would be problem-dependent.

Finally, we have the next lemma for radial basis functions.

Lemma 2 *Poly-sine-Gaussian networks can reproduce Gaussian radial basis functions and approximate radial basis functions defined on a bounded closed domain with analytic kernels with an exponential approximation rate.*

We will end this section with an informal discussion about the NTK of poly-sine-Gaussian networks. As we shall discuss in Section 2, deep learning can be approximated by kernel methods with a kernel $\hat{\Theta}_0$ in (27). Therefore, from the perspective of kernel regression for regressing $f(\boldsymbol{x})$ with training samples $\{(\boldsymbol{x}_i, f(\boldsymbol{x}_i))\}_{i=1}^N$, $\hat{\Theta}_0(\boldsymbol{x}, \boldsymbol{x}_i)$ quantifies the similarity of the point $\boldsymbol{x}$ and a training point $\boldsymbol{x}_i \in \mathcal{X}$ and, hence, serves as a weight of $f(\boldsymbol{x}_i)$ in a toy regression formulation: $\phi(\boldsymbol{x}; \boldsymbol{\omega}) := \sum_{i=1}^N \omega_i f(\boldsymbol{x}_i) \hat{\Theta}(\boldsymbol{x}, \boldsymbol{x}_i)$, where

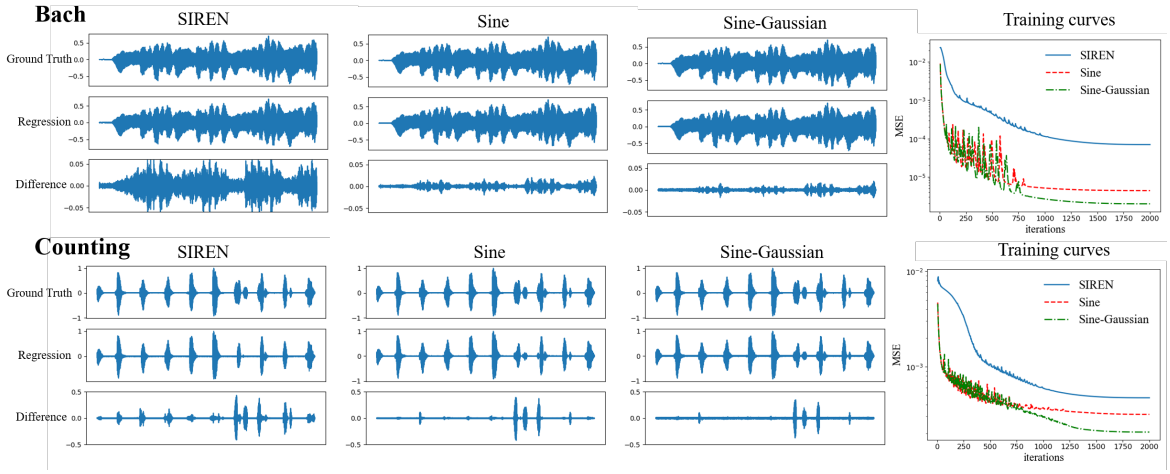


Figure 1. The comparison of fitted signal and training curves on Bach and Counting for SIREN and our RAF (Sine and Sine-Gaussian).

$\boldsymbol{\omega} = [\omega_1, \ldots, \omega_N]$ is a set of learnable parameters and $\phi(\boldsymbol{x}; \boldsymbol{\omega})$ is the approximant of the target function $f(\boldsymbol{x})$. To enable a kernel method to learn both smooth functions and highly oscillatory functions, the kernel function $\hat{\Theta}$ should have a widely spreading Fourier spectrum. By using $\sin(\beta x)$ with a tunable β in the poly-sine-Gaussian, the poly-sine-Gaussian network could learn an appropriate kernel for both kinds of functions. Similarly, by using $\exp(-(\beta x)^2))$ with a tunable β in the poly-sine-Gaussian, the poly-sine-Gaussian network could learn an appropriate kernel for both smooth and singular functions. We will provide numerical examples to demonstrate this empirically in the next section.

4. Numerical Results

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

4.1. Coordinate-based Data Representation

We verify the performance of RAFs on data representations using coordinate-based NNs. Mean square error (MSE) quantifies the difference between the ground truth and the NN output. Standard NNs, e.g., ReLU NNs, were shown to have poor performance to fit high-frequency components of signals (Sitzmann et al., 2020; Tancik et al., 2020). SIREN activation function (Sitzmann et al., 2020), i.e., $\sin(30x)$, improves the ability of NNs to represent complex signals.

As we discussed in Section 3.2, the SIREN function is a special case of the poly-sine-Gaussian activation function in our framework. We will show that poly-sine-Gaussian activation function can provide better performance than SIREN when the combination coefficients $\boldsymbol{\alpha}$ and the scaling parameters $\boldsymbol{\beta}$ are specified or trained appropriately in a problem-dependent manner. We follow the official implementation of SIREN on representations of audio, image, and video signal (refer to (Sitzmann et al., 2020) for details). The main difference between the SIREN code and ours is the activation function. All trainable parameters are trained to minimize the empirical loss function in (6). The NN is optimized by Adam optimizer with an initial learning rate 10^{-4} and cosine learning rate decay.

4.1.1. Audio Signal

We start from modeling audio signals on two audio clips, Bach and Counting as shown in Figure 1. An NN is trained to regress from a one-dimensional time coordinate to the corresponding sound level. Note that audio signals are purely oscillatory signals. Therefore, in the reproducing activation framework, x and x^2 are not necessary. We apply two forms of RAFs, Sine and Sine-Gaussian. The Sine one is set as $\alpha_1 \sin(\beta_1 x)$, while the Sine-Gaussian one is set as $\alpha_1 \sin(\beta_1 x) + \alpha_2 \exp(-x^2/(2\beta_2^2))$, where α_1 is initialized as $\mathcal{N}(2, 0.1)$, α_2 is initialized as $\mathcal{N}(1.0, 0.1)$, β_1 is initialized as $\mathcal{N}(30, 0.001)$, and β_2 is initialized with a uniform distribution $\mathcal{U}(0.01, 0.05)$. We use a 3-hidden-layer neural network with 256 neurons per layer to fit the audio signal following the network structure of SIREN. The NNs are trained for 2000 iterations. Figure 1 displays the fitted signals and training curves. Figure 1 shows our method has the capacity of modeling the audio signals more accurately than SIREN and leads to a smaller error in regression. Besides, our RAFs can converge to a better local minimum at a faster speed compared with SIREN. Moreover, we can see the add-in Gaussian function enhances the fitting ability.

4.1.2. Image Signal

We regress a grayscale image by learning a mapping from two-dimensional pixel coordinates to the corresponding pixel value. Four image of size 256×256 are used, including Camera, Astronaut, Cat and Coin, which are avail-

Table 1. The comparison of PSNR/SSIM of the fitted images using different activation functions. The larger these numbers are, the better the performance is.

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

Table 2. The comparison of PSNR of videos fitted by different activation functions. The mean and average are computed over 250 frames.

Activation	Mean PSNR	Std PSNR
SIREN	32.17	2.16
Sine-Gaussian	32.79	2.10

Table 3. The relative L^2 error of different activation functions for the regression problem, Poisson equation (8), PDE with low regularity (9), PDE with an oscillatory solution (10) and Eigenvalue problem with $d = 5$ or $d = 10$. $\oplus$ means the concatenation of different activation functions in the network.

Examples	Regression	Poisson Equation	Low Regularity	Oscillation	Eigen. ($d = 5$)	Eigen. ($d = 10$)
ReLU	6.61 e-02	-	-	-	6.38e-03	4.59 e-03
ReLU^3	1.13 e-01	9.40 e-04	6.12 e-03	3.16 e-05	0.307	0.223
$x \oplus x^2$	3.71 e-01	4.50 e-04	4.41 e-02	9.46 e-02	-	-
$x \oplus x^2 \oplus \text{ReLU}$	9.98 e-02	4.49 e-04	6.18 e-01	3.81 e+00	-	-
$x \oplus x^2 \oplus \text{ReLU}^3$	9.12 e-02	1.39 e-03	2.46 e-03	3.15 e-05	-	-
$x \oplus x^2 \oplus \sin(x)$	9.07 e-02	4.45 e-04	6.59 e-03	4.69 e-06	-	-
$x \oplus x^2 \oplus \sin(x) \oplus$ Gaussian	3.46 e-02	6.87 e-05	2.20 e-04	3.35 e-06	2.09 e-03	1.10 e-03
Rational (Boullé et al., 2020)	3.94 e-02	-	-	-	-	-

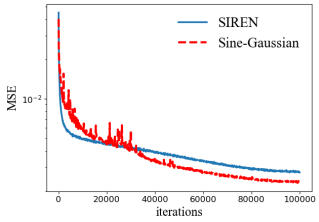


Figure 2. Comparison of training curves on video fitting for different activation functions.

able in Python Pillow. Note that images usually contain a cartoon part and a texture part. We apply three types of RAFs for image fitting: Sine, $\alpha_1 \sin(\beta_1 x)$; Poly-Sine, $\alpha_1 \sin(\beta_1 x) + \alpha_3 x + \alpha_4 x^2$; and Poly-Sine-Gaussian,

$$\alpha_1 \sin(\beta_1 x) + \alpha_2 \exp(-x^2/(2\beta_2^2)) + \alpha_3 x + \alpha_4 x^2. \quad (7)$$

Here, α_1, α_2, α_3, α_4, β_1, and β_2 are initialized as $\mathcal{N}(2, 0.1)$, $\mathcal{N}(1, 0.1)$, $\mathcal{N}(0.0, 0.1)$, $\mathcal{N}(1.0, 0.1)$, $\mathcal{N}(30, 0.001)$, and $\mathcal{U}(0.01, 0.05)$, respectively. An NN with 3 hidden layers and 256 neurons per layer is trained for $2,000$ iterations. Table 1 summarizes the Peak signal-to-noise ratio (PSNR) and Structural similarity (SSIM) of the fitted images showing that RAFs outperform SIREN with a significant margin.

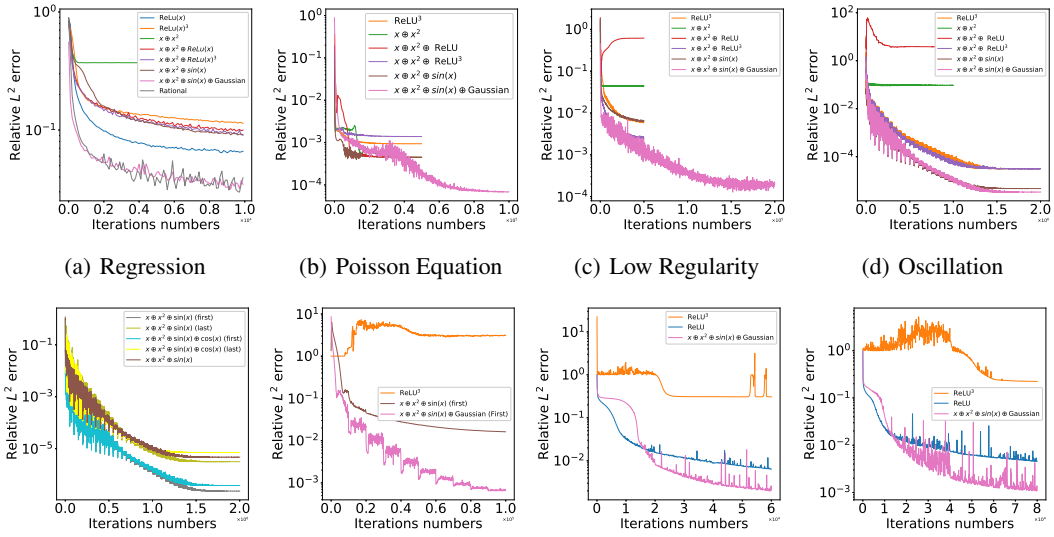
4.1.3. Video Signal

We fit a color video named Bike with 250 frames available in Python Skvideo Package. The regression is from three-dimensional coordinates to RGB pixel values. We apply Sine-Guassian as defined in Section 4.1.1, but α_1, α_2, β_1 and β_2 are initialized by $\mathcal{N}(1, 0.1)$, $\mathcal{N}(1, 0.1)$, $\mathcal{N}(30, 0.001)$ and $\mathcal{U}(0.002, 0.01)$, respectively. An NN with 3 hidden layers and 400 neurons per layer is trained for $100,000$ iterations. Figure 2 displays the training curves of video fitting for different activation function. Table 2 shows the mean and standard derivation of PSNR for video over 250 frames. From Figure 2, RAFs can lead to a better minimizer with a larger PSNR than SIREN.

4.2. Scientific Computing Applications

We compare RAFs with popular activation functions in scientific computing and provide ablation study to justify the combination of $\mathcal{A} = \{x, x^2, \sin(x), \exp(-x^2)\}$. The relative L^2 error is defined by $\left(\frac{\sum_{i=1}^N (u(x_i) - \hat{u}(x_i))^2}{\sum_{i=1}^N u^2(x_i)}\right)^{\frac{1}{2}}$, where $\{x_i\}_{i=1}^N$ are random points uniformly sampled in the domain, u is the true solution, and $\hat{u}$ is the estimated solution. We will adopt two metrics to quantify the performance of activation functions. The first one is the relative L^2 error on test samples. The second metric is the condition number of the NTK matrices for PDE solvers. A smaller condition number usually leads to a smaller iteration number to achieve the same accuracy.

Network setting. In all examples, we employ ResNet with two residual blocks and each block contains two hidden layers. Unless specified particularly, the width is set as 50 and all weights and biases in the ℓ-th layer are initialized by $\mathcal{U}(-\sqrt{1/N_{\ell-1}}, \sqrt{1/N_{\ell-1}})$, where $N_{\ell-1}$ is the width of the $\ell - 1$-th layer. Note that the network with RAFs can be expressed by a network with a single activation function in each neuron but different neurons can use different activation functions. For example, in the case of poly-sine-Gaussian networks, we will use $1/4$ neurons within each layer with x activation function, $1/4$ with x^2, $1/4$ with $\sin(x)$, and $1/4$ with $\exp(-x^2)$. In this new setting, it is not



(a) Regression (b) Poisson Equation (c) Low Regularity (d) Oscillation

(e) Oscillation (f) Oscillation (g) Eigenvalue (d=5) (h) Eigenvalue (d=10)

Figure 3. The relative L^2 error vs. iteration of different activation functions for (a) the regression, (b) Poisson equation (8), (c) PDE with low regularity (9), (d) PDE with an oscillatory solution (10), PDE with (e) an oscillatory solution or (f) a super oscillatory solution solved by using different scaling parameters in trigonometric activation function, and (g-h) Eigenvalue problem with $d = 5$ or $d = 10$.

necessary to train extra combination coefficients in the RAF. Though training the scaling parameters in the RAF might be beneficial in general applications, we focus on justifying the poly-sine-Gaussian activation function without emphasizing the scaling parameters. Hence, in almost all tests, the scaling parameters are set to be 1 for x, x^2, and $\sin(x)$, and the scaling parameter is set to be 0.1 for $\exp(-x^2)$. In the case of oscillatory solution (10), we specify the scaling parameter of $\sin(x)$ to introduce oscillation in the NTK. The idea of scaling parameters was also tested and verified in (Jagtap et al., 2020b;a). The other implementation detail can be found in Appendix.

4.2.1. Discontinuous Function Regression

We first show the advantage of the poly-sine-Gaussian activation function by regression a discontinuous function, $f(x) = -2x + 1$, when $x \geq 0$, $f(x) = -2x - 1$, when $x < 0$, on the domain $\Omega = [-1, 1]$. The relative L^2 error is presented in Table 3 and the training process is visualized in Figure 3(a). The regression result shows that the poly-sine-Gaussian activation has the best performance. We would like to remark that rational activation (Boullé et al., 2020) works well for regression problems but fails in our PDE problems without meaningful solutions. Hence, we only compare RAFs with rational activation functions in this example. The numerical results justify the combination of four kinds of activation functions. Remark that the computational time of rational activation functions is twice of the time of RAFs, though their accuracy is almost the same.

4.2.2. Poisson Equation with a Smooth Solution

Now we solve a two-dimensional Poisson equation

$$-\Delta u = f \text{ for } \boldsymbol{x} \in \Omega \text{ and } u = 0 \text{ for } \boldsymbol{x} \in \partial\Omega \tag{8}$$

with a smooth solution $u(\boldsymbol{x}) = x_1^2(1 - x_1)x_2^2(1 - x_2)$ defined on $\Omega = [0, 1]^2$. The numerical solution can be constructed as $\hat{u}(\boldsymbol{x}; \boldsymbol{\theta}) = \left(\Pi_{i=1}^2 x_i(1 - x_i)\right)\phi(\boldsymbol{x}; \boldsymbol{\theta})$, where $\phi(\boldsymbol{x}; \boldsymbol{\theta})$ is an NN. We apply the loss function (19) to identify an estimated solution. The relative L^2 errors for different activation functions are shown in Table 3 and the corresponding training process is visualized in Figure 3(b). The RAF with $\mathcal{A} = \{x, x^2, \sin(x), \exp(-x^2)\}$ achieves the best performance. The networks with other activation functions reach local minimal and cannot escape from these minimal after $20k$ iterations, while the poly-sine-Gaussian network continuously reduces the error even after $50k$ iterations. The numerical results also justify the combination of four kinds of activation functions.

4.2.3. PDE with Low Regularity

Next, we consider a two-dimensional PDE

$$-\nabla \cdot (|\boldsymbol{x}|\nabla u) = f \text{ for } \boldsymbol{x} \in \Omega \text{ and } u = 0 \text{ for } \boldsymbol{x} \in \partial\Omega \tag{9}$$

with a solution $u(\boldsymbol{x}) = \sin(2\pi(1 - |\boldsymbol{x}|))$ defined on $\Omega = \{\boldsymbol{x} : |\boldsymbol{x}| \leq 1\}$. The exact solution has low regularity at the origin. Let $\hat{u}(\boldsymbol{x}; \boldsymbol{\theta}) = (1 - |\boldsymbol{x}|)\phi(\boldsymbol{x}; \boldsymbol{\theta})$, where $\phi(\boldsymbol{x}; \boldsymbol{\theta})$ is an NN and $\hat{u}(\boldsymbol{x}; \boldsymbol{\theta})$ satisfies the boundary condition automatically. The loss function (19) is used to identify an estimated solution to the equation (9). The relative L^2

Table 4. The best historical accuracy for the equation in (10) 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.

Activation	Position	L2 error
$x \oplus x^2 \oplus \sin(x)$	first	2.31 e-07
$x \oplus x^2 \oplus \sin(x)$	last	3,14 e-06
$x \oplus x^2 \oplus \sin(x) \oplus \cos(x)$	first	3.79 e-07
$x \oplus x^2 \oplus \sin(x) \oplus \cos(x)$	last	1.90 e-03

Table 5. The condition number of the NTK in (29) of PDE solvers with different activation functions at initialization. The NTK matrix is evaluated with 100 samples, i.e., the matrix size is 100×100.

Activation	Eqn. (8)	Eqn. (9)	Eqn. (10)
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
$x \oplus x^2 \oplus$ ReLU	1.01 e+11	1.09 e+10	3.11 e+10
$x \oplus x^2 \oplus$ 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

errors for different activation functions are shown in Table 3 and the training curves is visualized in Figure 3(c). Since the true solution has low regularity, it is more challenging than the example (8) to obtain good accuracy. The RAF with $\mathcal{A} = \{x, x^2, \sin(x), \exp(-x^2)\}$ achieves lowest test error, which justifies the combination of four activation functions.

4.2.4. PDE WITH AN OSCILLATORY SOLUTION

Next, to verify the performance of $\sin(x)$ in the RAF, we consider a two-dimensional Poisson equation as follows,

$$-\Delta u + (u+2)^2 = f \text{ for } \boldsymbol{x} \in \Omega \tag{10}$$

with a Dirichlet boundary condition and an oscillatory solution $u(\boldsymbol{x}) = \sin(6\pi x_1)\sin(6\pi x_2)$ defined on $\Omega = [0,1]^2$. The NN is constructed as in Section 4.2.2 with width 100. The loss function (19) is used to identify the NN solution. The test error is shown in Table 3 and Figure 3(d). RAFs with $\{x, x^2, \sin(x), \exp(-x^2)\}$ achieve the best performance.

As discussed in Section 3, introducing oscillation in NNs is crucial to lessen the spectral bias of NNs. Fixing different scaling parameters in $\sin(x)$ can help to lessen the spectral bias better and obtain high-resolution image reconstruction (Tancik et al., 2020). Therefore, in the case of oscillatory target functions, we also specify scaling parameters in $\sin(x)$ to verify the performance. If n $\sin(x)$ functions are used in a layer, we will use $\{\sin(2\pi x), \sin(4\pi x), \ldots, \sin(2n\pi x)\}$. Besides, it is also of interest to see the performance of $\cos(x)$. Since specifying a wide range of scaling parameters in every hidden layer will create too much oscillation, we only specify scaling parameters either in the first or the last hidden layer. Therefore, four tests were conducted and the results are shown in Table 4 and Figure 3(e). The results show that $\cos(x)$ does not have an effective gain, but specifying different scaling parameters improves the performance especially in the first hidden layer. Further, we test a super oscillatory solution $u(x) = \sin(40x_1)\sin(40x_2)$ to the equation (10) and the result in Figure 3(f) shows our methods outperform ReLU^3.

4.2.5. NONLINEAR SCHRÖDINGER EQUATION

Last, we consider a d-dimensional nonlinear Schrödinger operator defined as $\mathcal{L}\varphi = -\Delta\varphi + \varphi^3 + V\varphi$ on Ω, where $V(x) = -\frac{1}{c^2}\exp(\frac{2}{d}\sum_{i=1}^d \cos x_i) + \sum_{i=1}^d (\frac{\sin^2 x_i}{d^2} - \frac{\cos x_i}{d}) - 3$ and $\Omega = [0, 2\pi]^d$. $\lambda = -3$ and $\varphi(x) = \exp(\frac{1}{d}\sum_{j=1}^d \cos(xj))/c$ is the leading eigenpair of the operator $\mathcal{L}$. Here c is a positive constant such that $\int_\Omega \varphi^2(x)dx = |\Omega|$. We follow the approach in (Han et al., 2020) to solve for the leading eigenpair. The NN in (Han et al., 2020) consists of two parts: 1) the first hidden layer uses $\sin(x)$ and $\cos(x)$ with different frequencies so that the whole network satisfies periodic boundary conditions; 2) the other hidden layers uses ReLU activation functions. We compare three activation functions, ReLU, ReLU^3, poly-sine-Gaussian, after the first hidden layer. Table 3 shows the error for different activation function and Figure 3(g) and 3(h) display the training curves for $d = 5$ and $d = 10$, respectively. One can see poly-sine-Gaussian reaches a smaller minimal than ReLU, ReLU^3.

4.2.6. NEURAL TANGENT KERNEL OF PDE SOLVERS

As discussed in Section 2, the condition number of NTK is also a crucial factor that determines the performance of deep learning. The condition numbers of NTK for the different PDE problems at initialization when different activation functions are used are summarized in Table 5. The condition number of the poly-sine-Gaussian activation function is smallest. Hence, from the perspective of NTK, we have also justified the combination of basic activation functions in the poly-sine-Gaussian activation function.

5. Conclusion

We propose RAF and its approximation theory. NNs with this activation function can reproduce traditional approximation tools (e.g., polynomials, Fourier basis functions, wavelets, radial basis functions) and approximate a certain class of functions with exponential and dimension-independent approximation rates. We have numerically demonstrated that RAFs can generate neural tangent kernels

with a better condition number than traditional activation functions, lessening the spectral bias of deep learning. Extensive experiments on coordinate-based data representation and PDEs demonstrate the effectiveness of the proposed activation function. We have not explored the optimal choice of basic activation functions in this paper, which would be problem-dependent and is left for future work.

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

Arora, S., Du, S. S., Hu, W., Li, Z., and Wang, R. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. *arxiv:1901.08584*, 2019.

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

Boullé, N., Nakatsukasa, Y., and Townsend, A. Rational neural networks. *arXiv:2004.01902*, 2020.

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

Cao, Y., Fang, Z., Wu, Y., Zhou, D.-X., and Gu, Q. Towards understanding the spectral bias of deep learning. *arXiv preprint arXiv:1912.01198*, 2019.

Chen, J., Du, R., Li, P., and Lyu, L. Quasi-monte carlo sampling for machine-learning partial differential equations. *ArXiv*, abs/1911.01612, 2019a.

Chen, Z. and Zhang, H. Learning implicit fields for generative shape modeling. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5932–5941, 2019. doi: 10.1109/CVPR.2019.00609.

Chen, Z., Cao, Y., Zou, D., and Gu, Q. How much over-parameterization is sufficient to learn deep relu networks? *CoRR*, arXiv:1911.12360, 2019b. URL https://arxiv.org/abs/1911.12360.

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

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

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

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

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

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

Genova, K., Cole, F., Sud, A., Sarna, A., and Funkhouser, T. Local deep implicit functions for 3d shape. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4856–4865, 2020. doi: 10.1109/CVPR42600.2020.00491.

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

Gu, Y., Wang, C., and Yang, H. Structure probing neural network deflation. *arXiv preprint arXiv:2007.03609*, 2020a.

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

Han, J., Jentzen, A., and Weinan, E. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018.

Han, J., Lu, J., and Zhou, M. Solving high-dimensional eigenvalue problems using deep neural networks: A diffusion monte carlo like approach. *Journal of Computational Physics*, 423:109792, 2020. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2020.109792. URL http://www.sciencedirect.com/science/article/pii/S0021999120305660.

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

Huang, J., Wang, H., and Yang, H. Int-deep: A deep learning initialized iterative method for nonlinear problems. *Journal of Computational Physics*, 419:109675, 2020. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2020.109675. URL http://www.sciencedirect.com/science/article/pii/S0021999120304496.

Hutzenthaler, M., Jentzen, A., Kruse, T., and Nguyen, T. A. 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. URL https://www.sam.math.ethz.ch/sam_reports/reports_final/reports2019/2019-10.pdf.

Jacot, A., Gabriel, F., and Hongler, C. Neural tangent kernel: Convergence and generalization in neural networks. *CoRR*, abs/1806.07572, 2018. URL http://arxiv.org/abs/1806.07572.

Jagtap, A. D., Kawaguchi, K., and Em Karniadakis, G. 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, 2020a. doi: 10.1098/rspa.2020.0334. URL https://royalsocietypublishing.org/doi/abs/10.1098/rspa.2020.0334.

Jagtap, A. D., Kawaguchi, K., and Karniadakis, G. E. Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *Journal of Computational Physics*, 404:109136, 2020b. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2019.109136. URL http://www.sciencedirect.com/science/article/pii/S0021999119308411.

Jeruzalski, T., Deng, B., Norouzi, M., Lewis, J. P., Hinton, G., and Tagliasacchi, A. Nasa: Neural articulated shape approximation. *ArXiv*, abs/1912.03207, 2020.

Khoo, Y., Lu, J., and Ying, L. Solving parametric pde problems with artificial neural networks. *arXiv: Numerical Analysis*, 2017.

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

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

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

Lee, J., Xiao, L., Schoenholz, S. S., Bahri, Y., Novak, R., Sohl-Dickstein, J., and Pennington, J. 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. doi: 10.1088/1742-5468/abc62b. URL https://doi.org/10.1088/1742-5468/abc62b.

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

Lei, D., Sun, Z., Xiao, Y., and Wang, W. Y. Implicit regularization of stochastic gradient descent in natural language processing: Observations and implications. *arXiv preprint arXiv:1811.00659*, 2018b.

Liao, Y. and Ming, P. Deep nitsche method: Deep ritz method with essential boundary conditions. *arXiv preprint arXiv:1912.01309*, 2019.

Liu, S., Zhang, Y., Peng, S., Shi, B., Pollefeys, M., and Cui, Z. Dist: Rendering deep implicit signed distance function with differentiable sphere tracing. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2016–2025, 2020. doi: 10.1109/CVPR42600.2020.00209.

Liu, Z., Cai, W., and Xu, Z.-Q. J. Multi-scale deep neural network (mscalednn) for solving poisson-boltzmann equation in complex domains. *Communications in Computational Physics*, 28(5):1970–2001, Jun 2020. ISSN 1991-7120. doi: 10.4208/cicp.oa-2020-0179. URL http://dx.doi.org/10.4208/cicp.OA-2020-0179.

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

Luo, T. and Yang, H. Two-Layer Neural Networks for Partial Differential Equations: Optimization and Generalization Theory. *arXiv e-prints*, arXiv:2006.15733, 2020.

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

Lyu, L., Wu, K., Du, R., and Chen, J. Enforcing exact boundary and initial conditions in the deep mixed residual method. *ArXiv*, abs/2008.01491, 2020.

Michalkiewicz, M., Pontes, J. K., Jack, D., Baktashmotlagh, M., and Eriksson, A. Implicit surface representations as layers in neural networks. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4742–4751, 2019. doi: 10.1109/ICCV.2019.00484.

Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., and Ng, R. Nerf: Representing scenes as neural radiance fields for view synthesis. *arxiv:2003.08934*, 2020.

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

Montanelli, H., Yang, H., and Du, Q. Deep relu networks overcome the curse of dimensionality for bandlimited functions. *arXiv preprint arXiv:1903.00735*, 2019.

Nakamura-Zimmerer, T., Gong, Q., and Kang, W. Adaptive deep learning for high dimensional hamilton-jacobi-bellman equations. *ArXiv*, abs/1907.05317, 2019.

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

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

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

Park, J. J., Florence, P., Straub, J., Newcombe, R., and Lovegrove, S. Deepsdf: Learning continuous signed distance functions for shape representation. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 165–174, 2019. doi: 10.1109/CVPR.2019.00025.

Raissi, M., Perdikaris, P., and Karniadakis, G. 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. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.10.045. URL http://www.sciencedirect.com/science/article/pii/S0021999118307125.

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

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

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

Shen, Z., Yang, H., and Zhang, S. Neural network approximation: Three hidden layers are enough. *arXiv:2010.14075*, 2020.

Shen, Z., Yang, H., and Zhang, S. Deep network with approximation error being reciprocal of width to power of square root of depth. *Neural Computation*, 2021.

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

Sitzmann, V., Martel, J., Bergman, A., Lindell, D., and Wetzstein, G. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33, 2020.

Tancik, M., Srinivasan, P. P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron, J. T., and Ng, R. Fourier features let networks learn high frequency functions in low dimensional domains. *arxiv:2006.10739*, 2020.

Trefethen, L. *Approximation Theory and Approximation Practice*. Other Titles in Applied Mathematics. SIAM, 2013. ISBN 9781611972405. URL https://books.google.com/books?id=h80N5JHm-u4C.

Wang, B. Multi-scale deep neural network (mscalednn) methods for oscillatory stokes flows in complex domains. *Communications in Computational Physics*, 28(5):2139–2157, Jun 2020. ISSN 1991-7120. doi: 10.4208/cicp.oa-2020-0192. URL http://dx.doi.org/10.4208/cicp.OA-2020-0192.

Wang, S., Yu, X., and Perdikaris, P. When and why pinns fail to train: A neural tangent kernel perspective. *arXiv:2007.14527*, 2020.

Xu, Z.-Q. J., Zhang, Y., Luo, T., Xiao, Y., and Ma, Z. Frequency principle: Fourier analysis sheds light on deep neural networks. *Communications in Computational Physics*, 28(5):1746–1767, 2020. ISSN 1991-7120. doi: https://doi.org/10.4208/cicp.OA-2020-0085. URL http://global-sci.org/intro/article_detail/cicp/18395.html.

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

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

Yarotsky, D. and Zhevnerchuk, A. The phase diagram of approximation rates for deep neural networks. *arXiv e-prints*, art. arXiv:1906.09477, June 2019.

Z. A.-Zhu, Y. Li, Z. S. A convergence theory for deep learning via over-parameterization. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 242–252, Long Beach, California, USA, 2019. PMLR.

Zhong, E. D., Bepler, T., Davis, J. H., and Berger, B. Reconstructing continuous distributions of 3d protein structure from cryo-em images. *arXiv:1909.05215*, 2020.

A. Preliminaries

A.1. Deep Neural Networks

Mathematically, NNs are a form of highly non-linear function parametrization via function compositions using simple non-linear functions (Goodfellow et al., 2016). The justification of this kind of approximation is given by the universal approximation theorems of NNs in (Kůrková, 1992; Barron, 1993; Yarotsky, 2017; 2018) with newly developed quantitative and explicit error characterization (Shen et al., 2019; Lu et al., 2020; Shen et al., 2021), which shows that function compositions are more powerful than other traditional approximation tools. There are two popular neural network structures used in NN-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:

$$\phi(\boldsymbol{x};\boldsymbol{\theta}) := \boldsymbol{a}^T \boldsymbol{h}_L \circ \boldsymbol{h}_{L-1} \circ \cdots \circ \boldsymbol{h}_1(\boldsymbol{x}), \tag{11}$$

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 (He et al., 2016). We present its variant defined recursively as follows:

$$\begin{aligned} \boldsymbol{h}_0 &= \boldsymbol{V}\boldsymbol{x}, \\ \boldsymbol{g}_\ell &= \sigma(\boldsymbol{W}_\ell \boldsymbol{h}_{\ell-1} + \boldsymbol{b}_\ell), \qquad \ell = 1, 2, \ldots, L, \\ \boldsymbol{h}_\ell &= \tilde{\boldsymbol{U}}_\ell \boldsymbol{h}_{\ell-2} + \boldsymbol{U}_\ell \boldsymbol{g}_\ell, \quad \ell = 1, 2, \ldots, L, \\ \phi(\boldsymbol{x};\boldsymbol{\theta}) &= \boldsymbol{a}^T \boldsymbol{h}_L, \end{aligned} \tag{12}$$

where $\boldsymbol{V} \in \mathbb{R}^{N_0 \times d}$, $\boldsymbol{W}_\ell \in \mathbb{R}^{N_\ell \times N_0}$, $\tilde{\boldsymbol{U}}_\ell \in \mathbb{R}^{N_0 \times N_0}$, $\boldsymbol{U}_\ell \in \mathbb{R}^{N_0 \times N_\ell}$, $\boldsymbol{b}_\ell \in \mathbb{R}^{N_\ell}$ for $\ell = 1, \cdots, L$, $\boldsymbol{a} \in \mathbb{R}^{N_0}$, $\boldsymbol{h}_{-1} = 0$. Throughout this paper, we consider $N_0 = N_\ell = N$ and $\boldsymbol{U}_\ell$ is set as the identity matrix in the numerical implementation of ResNets for the purpose of simplicity. Furthermore, as used in (E & Yu, 2018), we set $\tilde{\boldsymbol{U}}_\ell$ as the identity matrix when ℓ is even and set $\tilde{\boldsymbol{U}}_\ell = 0$ when ℓ is odd.

A.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 NN $\phi(\mathbf{x};\boldsymbol{\theta})$ that is used to approximate $f(\mathbf{x})$, the population risk (error) and empirical risk (error) functions are respectively

$$\mathcal{J}(\boldsymbol{\theta}) = \frac{1}{2}\mathbb{E}_{\mathbf{x}\sim\pi}\left[|\phi(\mathbf{x};\boldsymbol{\theta}) - f(\mathbf{x})|^2\right], \quad \hat{\mathcal{J}}(\boldsymbol{\theta}) = \frac{1}{2N}\sum_{i=1}^{N} |\phi(\mathbf{x}_i;\boldsymbol{\theta}) - y_i|^2, \tag{13}$$

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

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \hat{\mathcal{J}}(\boldsymbol{\theta}), \tag{14}$$

and $\phi(\cdot;\hat{\boldsymbol{\theta}}) : \Omega \to \mathbb{R}$ is the learned NN that approximates the unknown function f.

A.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 (Dissanayake & Phan-Thien, 1994; Lagaris et al., 1998; E & Yu, 2018; Liao & Ming, 2019). In this paper, we will take the example of BVP and the least squares method (LSM) (Dissanayake &

Phan-Thien, 1994; Lagaris et al., 1998) without loss of generality. The generalization to other problems and methods is similar. Consider the BVP

$$\begin{aligned} &\mathcal{D}u(\boldsymbol{x}) = f(u(\boldsymbol{x}), \boldsymbol{x}), \text{ in } \Omega, \\ &\mathcal{B}u(\boldsymbol{x}) = g(\boldsymbol{x}), \text{ on } \partial\Omega, \end{aligned} \tag{15}$$

where $\mathcal{D} : \Omega \to \Omega$ is a differential operator that can be nonlinear, $f(u(\boldsymbol{x}), \boldsymbol{x})$ can be a nonlinear function in u, Ω is a bounded domain in $\mathbb{R}^d$, and $\mathcal{B}u = g$ characterizes the boundary condition. Other types of problems like initial value problems can also be formulated as a BVP as discussed in (Gu et al., 2020b). Then LSM seeks a solution $u(\boldsymbol{x}; \boldsymbol{\theta})$ as a neural network with a parameter set $\boldsymbol{\theta}$ via the following optimization problem

$$\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) := \|\mathcal{D}u(\boldsymbol{x}; \boldsymbol{\theta}) - f(u, \boldsymbol{x})\|^2_{L^2(\Omega)} + \lambda \|\mathcal{B}u(\boldsymbol{x}; \boldsymbol{\theta}) - g(\boldsymbol{x})\|^2_{L^2(\partial\Omega)}, \tag{16}$$

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 (16) is to find an appropriate set of parameters $\boldsymbol{\theta}$ such that the NN $u(\boldsymbol{x}; \boldsymbol{\theta})$ 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 (15). 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 (15) is well-posed (e.g. the elliptic PDE with Neumann boundary condition, see Thm. 4.1 in (Gu et al., 2020b)).

In the implementation of LSM, the minimization problem in (16) is solved by SGD or its variants (e.g. Adagrad (Duchi et al., 2011), Adam (Kingma & Ba, 2014) and AMSGrad (Reddi et al., 2019)). In each iteration of the SGD, a stochastic loss function defined below is minimized instead of the original loss function in (16):

$$\min_{\boldsymbol{\theta}} \hat{\mathcal{L}}(\boldsymbol{\theta}) := \frac{1}{N} \sum_{i=1}^{N} \big(\mathcal{D}u(\boldsymbol{x}_i; \boldsymbol{\theta}) - f(\boldsymbol{x}_i)\big)^2 + \frac{1}{M} \lambda \sum_{j=1}^{M} \big(\mathcal{B}u(\boldsymbol{x}_j; \boldsymbol{\theta}) - g(\boldsymbol{x}_j)\big)^2, \tag{17}$$

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

To facilitate the optimization convergence to the desired PDE solution, special network structures can be proposed such that the NN can satisfy common boundary conditions, which can simplify the loss function in (16) to

$$\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) := \|\mathcal{D}u(\boldsymbol{x}; \boldsymbol{\theta}) - f(u, \boldsymbol{x})\|^2_{L^2(\Omega)}, \tag{18}$$

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

$$\min_{\boldsymbol{\theta}} \hat{\mathcal{L}}(\boldsymbol{\theta}) := \frac{1}{N} \sum_{i=1}^{N} \big(\mathcal{D}u(\boldsymbol{x}_i; \boldsymbol{\theta}) - f(\boldsymbol{x}_i)\big)^2. \tag{19}$$

In numerical implementation, the LSM loss function in (18) is more attractive because (16) 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 (16), as shown by extensive numerical experiments in (Lagaris et al., 1998; Gu et al., 2020a; Lyu et al., 2020). However, we would like to remark that it is difficult to build NNs that automatically satisfy complicated boundary conditions especially when the domain Ω is irregular.

The design of these special NNs 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 (Gu et al., 2020a; Lyu et al., 2020) for other kinds of boundary conditions.

Suppose $\hat{u}(x; \boldsymbol{\theta})$ is a generic NN 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 $u(x;\boldsymbol{\theta})$:

$$u(x;\boldsymbol{\theta}) = h(x)\hat{u}(x;\boldsymbol{\theta}) + l(x). \tag{20}$$

Then $u(x;\boldsymbol{\theta})$ is used to approximate the true solution of the PDE and is trained through (18).

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.

A.4. The Training Behavior of Deep Learning

The least-squares optimization problems in (16) and (18) are highly non-convex and hence they are challenging to solve. For regression problems or solving linear PDEs, under the assumption of over-parameterized NNs (i.e., the width of NNs is sufficiently large) and appropriate random initialization of NN parameters, it was shown that the least-squares optimization admits global convergence by gradient descent with a linear convergence rate (Jacot et al., 2018; Du et al., 2018; Z. A.-Zhu, 2019; Chen et al., 2019b; Luo & Yang, 2020). Though the over-parametrization assumption might not be realistic, it is still a positive sign for the justification of NNs in these least-squares problems. However, the convergence rate depends on the spectrum of the target function. The training of a randomly initialized NN 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 (Xu et al., 2020) and the spectral bias of deep learning in (Cao et al., 2019). Related works on the learning behavior of NNs in the frequency domain is further investigated in (Xu et al., 2020; Luo et al., 2019). In the case of nonlinear PDEs, these theoretical works imply that NN-based solvers would also have a bias towards reducing low-frequency errors (Wang et al., 2020). Without the assumption of over-parametrization, to the best of our knowledge, there is no theoretical guarantee that NN-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 (Neyshabur et al., 2017b; Lei et al., 2018b; Dai & Zhu, 2018). However, such local minimizers might not give the desired PDE solutions. Hence, designing new training techniques to make SGD capable of identifying better minimizers has been an active research field.

A.5. Neural Tangent Kernel

Neural tangent kernel (NTK) originally introduced in (Jacot et al., 2018) and further investigated in (Arora et al., 2019; Lee et al., 2020; Cao et al., 2019; Luo & Yang, 2020; Wang et al., 2020) 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 (Lee et al., 2020) for simplicity. This introduction is sufficient for us to discuss the advantage of RAFs 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 (13). 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:

$$\dot{\boldsymbol{\theta}}_t = -\nabla_{\boldsymbol{\theta}}\phi_t(\mathcal{X})^T \nabla_{\phi_t(\mathcal{X})}\hat{\mathcal{J}}, \tag{21}$$

and

$$\dot{\phi}_t(\mathcal{X}) = \nabla_{\boldsymbol{\theta}}\phi_t(\mathcal{X})\dot{\boldsymbol{\theta}}_t = -\hat{\Theta}_t(\mathcal{X},\mathcal{X})\nabla_{\phi_t(\mathcal{X})}\hat{\mathcal{J}}, \tag{22}$$

where $\boldsymbol{\theta}_t$ is the parameter set at iteration time t, $\phi_t(\mathcal{X}) = \text{vec}([\phi_t(\boldsymbol{x};\boldsymbol{\theta}_t)]_{\boldsymbol{x}\in\mathcal{X}})$ is the $N \times 1$ vector of concatenated function values for all samples, and $\nabla_{\phi_t(\mathcal{X})}\hat{\mathcal{J}}$ is the gradient of the loss with respect to the network output vector $\phi_t(\mathcal{X})$, $\hat{\Theta}_t := \hat{\Theta}_t(\mathcal{X},\mathcal{X})$ in $\mathbb{R}^{N\times N}$ is the NTK at iteration 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 (21) and (22). 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 (Jacot et al., 2018; Arora et al., 2019; Lee et al., 2020; Cao et al., 2019; Luo & Yang, 2020), 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 (Lee et al., 2020) to obtain explicit solutions to discuss the training dynamics of deep learning. In particular, the following linearized network by Taylor expansion is considered,

$$\phi_t^{\text{lin}}(\boldsymbol{x}) := \phi(\boldsymbol{x}; \boldsymbol{\theta}_0) + \nabla_{\boldsymbol{\theta}} \phi(\boldsymbol{x}; \boldsymbol{\theta}_0) \boldsymbol{\omega}_t, \tag{23}$$

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

$$\dot{\boldsymbol{\omega}}_t = -\nabla_{\boldsymbol{\theta}} \phi_0(\mathcal{X})^T \nabla_{\phi_t^{\text{lin}}(\mathcal{X})} \hat{\mathcal{J}}, \tag{24}$$

and

$$\dot{\phi}_t^{\text{lin}}(\boldsymbol{x}) = -\hat{\Theta}_0(\boldsymbol{x}, \mathcal{X}) \nabla_{\phi_t^{\text{lin}}(\mathcal{X})} \hat{\mathcal{J}}. \tag{25}$$

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

$$\phi_t^{\text{lin}}(\mathcal{X}) = \left(I - e^{-\hat{\Theta}_0 t} \right) \mathcal{Y} + e^{-\hat{\Theta}_0 t} \phi_0(\mathcal{X}). \tag{26}$$

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

$$\phi_t^{\text{lin}}(\boldsymbol{x}) = \phi_0(\boldsymbol{x}) - \hat{\Theta}_0(\boldsymbol{x}, \mathcal{X}) \hat{\Theta}_0^{-1} \left(I - e^{-\hat{\Theta}_0 t} \right) (\phi_0(\mathcal{X}) - \mathcal{Y}), \tag{27}$$

which is equivalent to

$$\phi_t^{\text{lin}}(\boldsymbol{x}) - \phi_0(\boldsymbol{x}) = \hat{\Theta}_0(\boldsymbol{x}, \mathcal{X}) \hat{\Theta}_0^{-1} \left(I - e^{-\hat{\Theta}_0 t} \right) (\mathcal{Y} - \phi_0(\mathcal{X})). \tag{28}$$

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 (27) serves as an approximate solution to the nonlinear evolution equation in (22). Based on (28), 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 (27) 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 (27). 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 (27) approaches to $\phi_0(\mathcal{X}) - \mathcal{Y}$ and, hence, $\phi_t^{\text{lin}}(\boldsymbol{x})$ goes to the desired function value for $\boldsymbol{x} \in \mathcal{X}$. However, in practice, when $\hat{\Theta}_0$ is very ill-conditioned, a small approximation error in $\left(I - e^{-\hat{\Theta}_0 t} \right) (\phi_0(\mathcal{X}) - \mathcal{Y}) \approx \phi_0(\mathcal{X}) - \mathcal{Y}$ may be amplified significantly, resulting in a poor accuracy for $\phi_t^{\text{lin}}(\boldsymbol{x})$ to solve the regression problem. We will discuss the advantage of the proposed RAFs 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

$$\hat{\Theta}_t = \left(\nabla_{\boldsymbol{\theta}} \mathcal{D} \phi_t(\mathcal{X}) \right) \left(\nabla_{\boldsymbol{\theta}} \mathcal{D} \phi_t(\mathcal{X}) \right)^T, \tag{29}$$

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

B. Proof of Theories

B.1. Proof of Theorem 3

The proof of Theorem 3 relies on the following lemma.

Lemma 3 *(i) An identity map in $\mathbb{R}^d$ can be realized exactly by a poly-sine-Gaussian network with one hidden layer and d neurons.*

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

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

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

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

Proof 1 *Part (i) to (iii) are trivial. We will only prove Part (iv). In the case of $|\boldsymbol{\alpha}| = k \leq 1$, the proof is simple and left for the reader. When $|\boldsymbol{\alpha}| = k \geq 2$, the main idea of the proof of (v) can be summarized in Figure 4. By Part (i), we can apply a poly-sine-Gaussian network to implement a d-dimensional identity map. This identity map maintains necessary entries of $\boldsymbol{x}$ to be multiplied together. We apply poly-sine-Gaussian networks to implement the multiplication function in Part (iii) and carry out the multiplication N times per layer. After L layers, there are $k - NL \leq N$ multiplications to be implemented. Finally, these at most N multiplications can be carried out with a small poly-sine-Gaussian network in a dyadic tree structure.*

Now we are ready to prove Thm. 3.

Proof 2 *The main idea of the proof is to apply Part (iv) of Lem. 3 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 5, 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) of Lem. 3, 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 5. 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 4).

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 $\phi(\boldsymbol{x}) = Id \circ \tilde{\phi}(\boldsymbol{x})$ 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.

B.2. Proof of Thm. 4

Proof 3 *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 Thm. 3.*

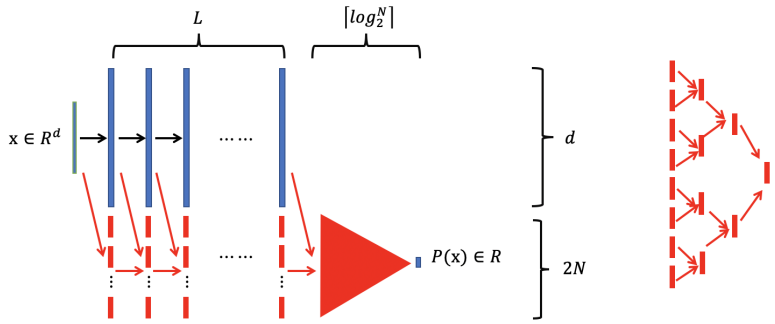


Figure 4. Left: An illustration of the proof of Lem. 3 (iv). Green vectors represent the input and output of the poly-sine-Gaussian network carrying out $P(\boldsymbol{x})$. Blue vectors represent the poly-sine-Gaussian network that implements a d-dimensional identity map in Part (i), which was repeatedly 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.

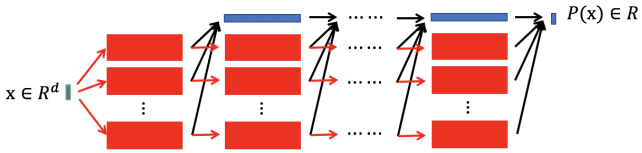


Figure 5. An illustration of the proof of Thm. 3. Green vectors represent the input and output of the poly-sine-Gaussian network $\tilde{\phi}(\boldsymbol{x})$ carrying out $P(\boldsymbol{x})$. Each red rectangle represents one poly-sine-Gaussian network $\phi_j(\boldsymbol{x})$ and each blue rectangle 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 $P(\boldsymbol{x})$.

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

$$\max_{0 \leq k \leq n} |c_k| = \mathcal{O}\left(C_f s\right), \text{ via Thm. 8.1 in (Trefethen, 2013).} \tag{30}$$

Since, f_n is a polynomial of degree n, by Thm. 3 with $d = 1$, $a = 1$, and $b = n + 1$, there exists a poly-sine-Gaussian network ϕ with width $2N + 2$ and depth L such that

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

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

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

B.3. Proof of Thm. 5

To show the approximation of poly-sine-Gaussian networks to generalized bandlimited functions, we will need Maurey's unpublished theorem below. It was used to study shallow network approximation by Barron in (Barron, 1993).

Theorem 6 (Maurey's theorem) *Let H be a Hilbert space with norm $\|\cdot\|$. Suppose there exists $G \subset H$ such that for every $g \in G$, $\|g\| \leq b$ for some $b > 0$. Then, for every f in the convex hull of G and every integer $n \geq 1$, there is a f_n in the convex hull of n points in G and a constant $c > b^2 - \|f\|^2$ such that $\|f - f_n\|^2 \leq \frac{c}{n}$.*

Proof 4 *Let f be an arbitrary function in $\mathcal{H}_{K,M}$, and μ be an arbitrary measure. Let $F(\boldsymbol{w}) = |F(\boldsymbol{w})|e^{i\theta(\boldsymbol{w})}$. Since f is real-valued, we may write*

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

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

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

Therefore, f is in the closure of the convex hull of $G_{K,M}$. Since functions in $G_{K,M}$ are bounded in the $L^2(\mu, B)$-norm by $2C_F D_K \sqrt{\mu(B)} \leq 2C_F\sqrt{\mu(B)}$, Thm. 6 tells us that there exist real coefficients b_j's and β_j's such that[1]

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

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 4, they can be approximated to accuracy ϵ_0 using networks $\widetilde{K}_R$ and $\widetilde{K}_I$ of width and depth

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

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

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

[1] We use Thm. 6 with $b = 2C_F\sqrt{\mu(B)}$, $c = b^2 > b^2 - \|f\|^2$, and $\|\cdot\| = \|\cdot\|_{L^2(\mu,B)}$.

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.

B.4. Proof of Lemma 1

Proof 5 *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) Lem. 3 shows that the x^2 activation function can reproduce multiplication.*

B.5. Proof of Lemma 2

Proof 6 *The proof of this lemma is trivial by Lem. 3, Thm. 3, and the proof of Thm. 4.*

C. Implementation details

C.1. Scientific computing

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 hyper-parameters. This subroutine implements the Adam algorithm in (Kingma & Ba, 2014).
- **Learning rate.** The learning rate will be decreased step by step in all examples following the formula

$$\tau_n = \tau_0 * q^{\lfloor \frac{n}{s} \rfloor}, \tag{31}$$

  where τ_n is the learning rate in the $n-$th iteration, q is a factor set to be 0.95, and s means that we update learning rate after s steps.
- **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 (Han et al., 2020).
- **Network setting.** In all PDE examples, we construct a special network that satisfies the given boundary condition as discussed in Section A.3. In all examples, we apply ResNet with two residual blocks and each block contains two hidden layers. The width is set as 50 unless specified. Unless specified particularly, all weights and biases in the ℓ-th layer are initialized by $U(-\sqrt{N_{\ell-1}}, \sqrt{N_{\ell-1}})$, where $N_{\ell-1}$ is the width of the $\ell-1$-th layer. Note that the network with RAFs can be expressed by a network with a single activation function in each neuron but different neurons can use different activation functions. For example, in the case of poly-sine-Gaussian networks, we will use $1/4$ neurons within each layer with x activation function, $1/4$ with x^2, $1/4$ with $\sin(x)$, and $1/4$ with $\exp(-x^2)$ for coding simplicity. In the case of poly-sine networks, $1/3$ neurons for each x, x^2, and $\sin(x)$ activation functions. In this new setting, it is not necessary to train extra combination coefficients in the RAF. Though training the scaling parameters in the RAF might be beneficial in general applications, we focus on justifying the poly-sine-Gaussian 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 parameter is set to be 0.1 for $\exp(-x^2)$. In the case of oscillatory target functions, we specify the scaling parameter of $\sin(x)$ to introduce oscillation in the NTK as we shall discuss and improved performance is observed. The idea of scaling parameters has been tested and verified in (Jagtap et al., 2020b;a).

- **Performance Evaluation.** We will adopt two criteria to quantify the performance of different activation functions. The first one is the relative 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.