On the eigenvector bias of Fourier feature networks: From regression to solving multi-scale PDEs with physics-informed neural networks

Sifan Wang[a], Hanwen Wang[a], Paris Perdikaris[b],*

[a] *Graduate Group in Applied Mathematics and Computational Science, University of Pennsylvania, Philadelphia, PA 19104, United States of America*
[b] *Department of Mechanical Engineering and Applied Mechanics, University of Pennsylvania, Philadelphia, PA 19104, United States of America*

Abstract

Physics-informed neural networks (PINNs) are demonstrating remarkable promise in integrating physical models with gappy and noisy observational data, but they still struggle in cases where the target functions to be approximated exhibit high-frequency or multi-scale features. In this work we investigate this limitation through the lens of Neural Tangent Kernel (NTK) theory and elucidate how PINNs are biased towards learning functions along the dominant eigen-directions of their limiting NTK. Using this observation, we construct novel architectures that employ spatio-temporal and multi-scale random Fourier features, and justify how such coordinate embedding layers can lead to robust and accurate PINN models. Numerical examples are presented for several challenging cases where conventional PINN models fail, including wave propagation and reaction–diffusion dynamics, illustrating how the proposed methods can be used to effectively tackle both forward and inverse problems involving partial differential equations with multi-scale behavior. All code an data accompanying this manuscript will be made publicly available at https://github.com/PredictiveIntelligenceLab/MultiscalePINNs.

1. Introduction

Leveraging advances in automatic differentiation [1], deep learning tools are introducing a new trend in tackling forward and inverse problems in computational mechanics. Under this emerging paradigm, unknown quantities of interest are typically parametrized by deep neural networks, and a multi-task learning problem is posed with the dual goal of fitting observational data and approximately satisfying a given physical law, mathematically expressed via systems of partial differential equations (PDEs). Since the early studies of Psichogios et al. [2] and Lagaris et al. [3], and their modern re-incarnation via the framework of physics-informed neural networks (PINNs) [4], the use of neural networks to represent PDE solutions has undergone rapid growth, both in terms of theory [5–8] and diverse applications in computational science and engineering [9–11]. PINNs in particular, have demonstrated remarkable power in applications including fluid dynamics [12–14], biomedical engineering [15–17], meta-material design [18,19], free boundary problems [20], Bayesian networks and uncertainty quantification [21,22], high

---

* Corresponding author.
  E-mail addresses: sifanw@sas.upenn.edu (S. Wang), wangh19@sas.upenn.edu (H. Wang), pgp@seas.upenn.edu (P. Perdikaris).

dimensional PDEs [23–25], stochastic differential equations [26], and beyond [27,28]. However, despite this early empirical success, we are still lacking a concrete mathematical understanding of the mechanisms that render such constrained neural network models effective, and, more importantly, the reasons these models can oftentimes fail. In fact, more often than not, PINNs are notoriously hard to train, especially for forward problems exhibiting high-frequency or multi-scale behavior.

Recent work by Wang et al. [6,29] has identified two fundamental weaknesses in conventional PINN formulations. The first is related to a remarkable discrepancy of convergence rate between the different terms that define a PINN loss function. As demonstrated by Wang et al. [29], the gradient flow of PINN models becomes increasingly stiff for PDE solutions exhibiting high-frequency or multi-scale behavior, often leading to unbalanced gradients during back-propagation. A subsequent analysis using the recently developed neural tangent kernel (NTK) theory [6], has revealed how different terms in a PINNs loss may dominate one another, leading to models that cannot simultaneously fit the observed data and minimize the PDE residual. These findings have motivated the development of novel optimization schemes and adaptive learning rate annealing strategies that are demonstrated to be very effective in minimizing multi-task loss functions, such as the ones routinely encountered in PINNs [6,29,30].

The second fundamental weakness of PINNs is related to spectral bias [31–33]; a commonly observed pathology of deep fully-connected networks that prevents them from learning high-frequency functions. As analyzed in [6] using NTK theory [34–37], spectral bias indeed exists in PINN models and is the leading reason that prevents them from accurately approximating high-frequency or multi-scale functions. To this end, recent work in [38–40], attempts to empirically address this pathology by introducing appropriate input scaling factors to convert the problem of approximating high frequency components of the target function to one of approximating lower frequencies. In another line of work, Tancik et al. [41] introduced Fourier feature networks which use a simple Fourier feature mapping to enhance the ability of fully-connected networks to learn high-frequency functions. Although the effectiveness of introducing appropriate feature embeddings has been illustrated in some application of PINNs [17,42], in general, they still lack a concrete mathematical justification in relation to how they potentially address spectral bias.

Building on the these recent findings, this work attempts to analyze and address the aforementioned shortcomings of PINNs, with a particular focus on designing effective models for multi-scale PDEs. To this end, we rigorously study fully-connected neural networks and PINNs through the lens of their limiting NTK, and produce novel insights into how these models fall short in presence of target functions with high-frequencies of multi-scale features. Using this analysis, we propose a family of novel architectures that can effectively mitigate spectral bias and enable the solution of problems for which current PINN approaches fail. Specifically, our main contributions can be summarized into the following points:

- We argue that spectral bias in deep neural networks in fact corresponds to "NTK eigenvector bias", and show that Fourier feature mappings can modulate the frequency of the NTK eigenvectors.
- By analyzing how the NTK eigenspace determines the type of functions a neural net can learn, we engineer new effective architectures for multi-scale problems.
- We propose a series of benchmarks for which conventional PINN models fail, and use them to demonstrate the effectiveness of the proposed methods.

The remaining of this paper is organized as follows. In Section 2, we present a brief overview of PINNs and emphasize their weakness in solving multi-scale problems. Next, we introduce the neural tangent kernel (NTK) as a theoretical tool to detect and analyze spectral bias in Section 3.1. Furthermore, we study the NTK eigensystem of Fourier feature networks and propose two novel network architectures that are efficient in handling multi-scale problems, see Sections 3.2, 3.3. We present a detailed evaluation of our proposed neural network architectures across a range of representative benchmark examples, see Section 4. Finally, in Section 5, we summarize our findings and provide a discussion on lingering limitations and promising future directions.

2. Physics-informed neural networks

In this section, we present a brief overview of physics-informed neural networks (PINNs) [4]. In general, we consider partial differential equations of the following form

$$\mathcal{N}[\boldsymbol{u}](\boldsymbol{x}) = \boldsymbol{f}(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega, \tag{2.1}$$

$$\mathcal{B}[\boldsymbol{u}](\boldsymbol{x}) = \boldsymbol{g}(\boldsymbol{x}), \quad \boldsymbol{x} \in \partial\Omega, \tag{2.2}$$

where $\mathcal{N}[\cdot]$ is a differential operator and $\mathcal{B}[\cdot]$ corresponds to Dirichlet, Neumann, Robin, or periodic boundary conditions. In addition, $\boldsymbol{u} : \overline{\Omega} \to \mathbb{R}$ describes the unknown latent quantity of interest that is governed by the PDE system of Eq. (2.1). For time-dependent problems, we consider time t as a special component of $\boldsymbol{x}$, and Ω then also contains the temporal domain. In that case, initial conditions can be simply treated as a special type of boundary condition on the spatio-temporal domain.

Following the original work of Raissi et al. [4], we proceed by approximating $\boldsymbol{u}(\boldsymbol{x})$ by a deep neural network $\boldsymbol{u_\theta}(\boldsymbol{x})$, where $\boldsymbol{\theta}$ denotes all tunable parameters of the network (e.g., weights and biases). Then, a physics-informed model can be trained by minimizing the following composite loss function

$$\mathcal{L}(\boldsymbol{\theta}) = \lambda_r \mathcal{L}_r(\boldsymbol{\theta}) + \lambda_b \mathcal{L}_{u_b}(\boldsymbol{\theta}), \tag{2.3}$$

where

$$\mathcal{L}_r(\boldsymbol{\theta}) = \frac{1}{N_r} \sum_{i=1}^{N_r} \left| \mathcal{N}[\boldsymbol{u_\theta}](x_r^i) - \boldsymbol{f}(x_r^i) \right|^2, \tag{2.4}$$

$$\mathcal{L}_b(\boldsymbol{\theta}) = \frac{1}{N_b} \sum_{i=1}^{N_b} \left| \mathcal{B}[\boldsymbol{u_\theta}](x_b^i) - \boldsymbol{g}(x_b^i) \right|^2, \tag{2.5}$$

and N_r and N_b denote the batch-sizes of training data $\{x_b^i, \boldsymbol{g}(x_b^i)\}_{i=1}^{N_b}$ and $\{x_r^i, \boldsymbol{f}(x_r^i)\}_{i=1}^{N_r}$, respectively, which are randomly sampled in the computational domain at each iteration of a gradient descent algorithm. Notice that all required gradients with respect to input variables $\boldsymbol{x}$ or parameters $\boldsymbol{\theta}$ can be efficiently computed via automatic differentiation [1]. Moreover, the parameters $\{\lambda_r, \lambda_b\}$ correspond to weight coefficients in the loss function that can effectively assign a different learning rate to each individual loss term. These weights may be user-specified or tuned automatically during network training [6,29].

Despite a series of early promising results [12,16,20], the original formulation of Raissi et al. [4] often struggles to handle multi-scale problems. As an example, let us consider a simple 1D Poisson's equation

$$\Delta u(x) = f(x), \quad x \in (0, 1) \tag{2.6}$$

subject to the boundary condition

$$u(0) = u(1) = 0$$

Here the fabricated solution we consider is

$$u(x) = \sin(2\pi x) + 0.1\sin(50\pi x)$$

and $f(x)$ can be derived using Eq. (2.6). Though this example is simple and pedagogical, it is worth noting that the solution exhibits low frequency in the macro-scale and high frequency in the micro-scale, which resembles many practical scenarios.

We represent the unknown solution $u(x)$ by a 5-layer fully-connected neural network $u_{\theta(x)}$ with 200 units per hidden layer. The parameters of the network can be learned by minimizing the following loss function

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_b(\boldsymbol{\theta}) + \mathcal{L}_r(\boldsymbol{\theta}) \tag{2.7}$$

$$= \frac{1}{N_b} \sum_{i=1}^{N_b} \left| u_{\boldsymbol{\theta}}(x_b^i) - u(x_b^i) \right|^2 + \frac{1}{N_r} \sum_{i=1}^{N_r} \left| \Delta u_{\boldsymbol{\theta}}(x_r^i) - f(x_r^i) \right|^2 \tag{2.8}$$

where the batch sizes are set to $N_b = N_r = 128$ and all training points $\{x_b^i, u(x_b^i)\}_{i=1}^{N_b}$, $\{x_r^i, f(x_r^i)\}_{i=1}^{N_r}$ are uniformly sampled for the boundary and residual collocation points at each iteration of gradient descent.

Fig. 1 summarized the results obtained by training the network for 10^7 iterations of gradient descent using the Adam optimizer [43] with default settings. We observe that the network is incapable of learning the correct solution, even after a million training iterations. In fact, it is not difficult for a conventional fully-connected neural network to approximate that function u, given sufficient data inside the computational domain. However, as shown in Fig. 1, solving high-frequency or multi-scale problems presents great challenges to PINNs. Although there has been recent efforts to elucidate the reasons PINN models may fail to train [6,29], a complete understanding of how to quantify and resolve such pathologies is still lacking. In the following sections, we will obtain insights by studying Fourier feature networks through the lens of their neural tangent kernel (NTK), and present a novel methodology to tackle multi-scale problems with PINNs.
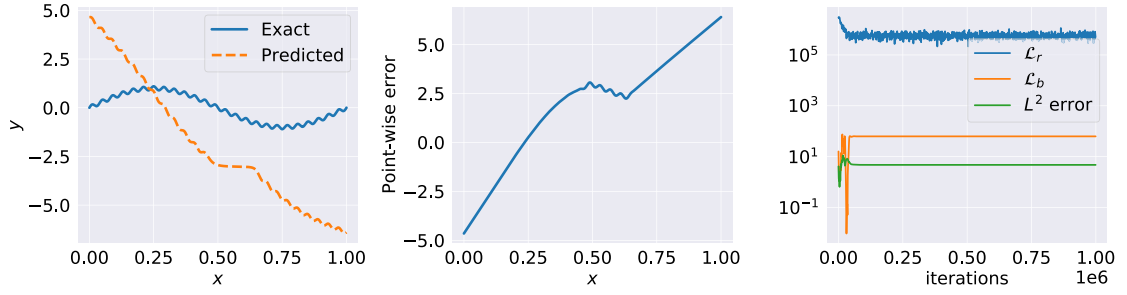
Fig. 1. *1D Poisson equation:* Results obtained by training a conventional physics-informed neural network (5-layer, 200 hidden units, tanh activations) via 10^7 iterations of gradient descent. *Left:* Comparison of the predicted and exact solutions. *Middle:* Point-wise error between the predicted and the exact solution. *Right:* Evolution of the residual loss $\mathcal{L}_r$, the boundary loss $\mathcal{L}_b$, as well as the relative L^2 error during training.

3. Methodology

3.1. Analyzing spectral bias through the lens of the neural tangent kernel

Before presenting our proposed methods in the context of PINNs, let us first start with a much simpler setting involving regression of functions using deep neural networks. To lay the foundations for our theoretical analysis, we first review the recently developed Neural Tangent Kernel (NTK) theory of Jacot et al. [34–36], and its connection to investigating spectral bias [31–33] in the training behavior of deep fully-connected networks. Let $f(x, \theta)$ be a scalar-valued fully-connected neural network (see Appendix A) with weights θ initialized by a Gaussian distribution $\mathcal{N}(0, 1)$. Given a data-set $\{X_{\text{train}}, Y_{\text{train}}\}$, where $X_{\text{train}} = (x_i)_{i=1}^N$ are inputs and $Y_{\text{train}} = (y_i)_{i=1}^N$ are the corresponding outputs, we consider a network trained by minimizing the mean square loss $\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^N |f(x_i, \theta) - y_i|^2$ using a very small learning rate η. Then, following the derivation of Jacot et al. [34,35], we can define the neural tangent kernel operator K, whose entries are given by

$$K_{ij} = K(x_i, x_j) = \left\langle \frac{\partial f(x_i, \theta)}{\partial \theta}, \frac{\partial f(x_j, \theta)}{\partial \theta} \right\rangle, \tag{3.1}$$

Strikingly, the NTK theory shows that, under gradient descent dynamics with an infinitesimally small learning rate (gradient flow), the kernel K converges to a deterministic kernel K^* and does not change during training as the width of the network grows to infinity.

Furthermore, under the asymptotic conditions stated in Lee et al. [36], we can derive that

$$\frac{df(X_{\text{train}}, \theta(t))}{dt} \approx -K \cdot (f(X_{\text{train}}, \theta(t)) - Y_{\text{train}}), \tag{3.2}$$

where $\theta(t)$ denotes the parameters of the network at iteration t and $f(X_{\text{train}}, \theta(t)) = f(x_i, \theta(t))_{i=1}^N$. Then, it directly follows that

$$f(X_{\text{train}}, \theta(t)) \approx (I - e^{-Kt}) \cdot Y_{\text{train}}. \tag{3.3}$$

Since the kernel K is positive semi-definite, we can take its spectral decomposition $K = Q^T \Lambda Q$, where Q is an orthogonal matrix whose ith column is the eigenvector q_i of K and Λ is a diagonal matrix whose diagonal entries λ_i are the corresponding eigenvalues. Since $e^{-Kt} = Q^T e^{-\Lambda t} Q$, we have

$$Q^T (f(X_{\text{train}}, \theta(t)) - Y_{\text{train}}) = -e^{\Lambda t} Q^T Y_{\text{train}}, \tag{3.4}$$

which implies

$$\begin{bmatrix} q_1^T \\ q_2^T \\ \vdots \\ q_N^T \end{bmatrix} (f(X_{\text{train}}, \theta(t)) - Y_{\text{train}}) = \begin{bmatrix} e^{-\lambda_1 t} & & & \\ & e^{-\lambda_2 t} & & \\ & & \ddots & \\ & & & e^{-\lambda_N t} \end{bmatrix} \begin{bmatrix} q_1^T \\ q_2^T \\ \vdots \\ q_N^T \end{bmatrix} Y_{\text{train}}. \tag{3.5}$$

The above equation shows that the convergence rate of $\boldsymbol{q}_i^T(f(\boldsymbol{X}_{\text{train}}, \boldsymbol{\theta}(t)) - \boldsymbol{Y}_{\text{train}})$ is determined by the ith eigenvalue λ_i. Moreover, we can decompose the training error into the eigenspace of the NTK as

$$f(\boldsymbol{X}_{\text{train}}, \boldsymbol{\theta}(t)) - \boldsymbol{Y}_{\text{train}} = \sum_{i=1}^{N} (f(\boldsymbol{X}_{\text{train}}, \boldsymbol{\theta}(t)) - \boldsymbol{Y}_{\text{train}}, \boldsymbol{q}_i)\boldsymbol{q}_i \tag{3.6}$$

$$= \sum_{i=1}^{N} \boldsymbol{q}_i^T (f(\boldsymbol{X}_{\text{train}}, \boldsymbol{\theta}(t)) - \boldsymbol{Y}_{\text{train}}) \boldsymbol{q}_i \tag{3.7}$$

$$= \sum_{i=1}^{N} \left(e^{-\lambda_i t} \boldsymbol{q}_i^T \boldsymbol{Y}_{\text{train}} \right) \boldsymbol{q}_i. \tag{3.8}$$

Clearly, the network is biased to first learn the target function along the eigendirections of neural tangent kernel with larger eigenvalues, and then the remaining components corresponding to smaller eigenvalues. A more detailed analysis on the convergence rate of different components is illustrated by Cao et al. [32]. For conventional fully-connected neural networks, the eigenvalues of the NTK shrink monotonically as the frequency of the corresponding eigenfunctions increases, yielding a significantly lower convergence rate for high frequency components of the target function [31,33]. This indeed reveals the so-called "spectral bias" [31] pathology of deep neural networks.

Since the learnability of a target function by a neural network can be characterized by the eigenspace of its neural tangent kernel, it is very natural to ask: can we engineer the eigenspace of the NTK to accelerate convergence? If this is possible, can we leverage it to help the network effectively learn different frequencies in the target function? In the next section, we will answer these questions by re-visiting the recently proposed random Fourier features embedding proposed by Tancik et al. [41].

3.2. Fourier feature embeddings

Following the original formulation of Tancik et al. [41], a random Fourier mapping γ is defined as

$$\gamma(\boldsymbol{v}) = \begin{bmatrix} \cos(\boldsymbol{B}\boldsymbol{v}) \\ \sin(\boldsymbol{B}\boldsymbol{v}) \end{bmatrix}, \tag{3.9}$$

where each entry in $\boldsymbol{B} \in \mathbb{R}^{m \times d}$ is sampled from a Gaussian distribution $\mathcal{N}(0, \sigma^2)$ and $\sigma > 0$ is a user-specified hyper-parameter. Then, a Fourier features network [41] can be simply constructed using a random Fourier features mapping γ as a coordinate embedding of the inputs, followed by a conventional fully-connected neural network [41]. The proposed Fourier features embedding can be thought of as a layer with no trainable parameters that can be included in any physics-informed neural network (PINN) architecture. It is worth emphasizing that this methodology can still accommodate arbitrary domain geometries (very much like conventional PINNs), and does not adversely affect the ability of neural networks to approximate functions in high-dimensions.

As shown in [41], such a simple method can mitigate the pathology of spectral bias and enable networks to learn high frequencies more effectively, which can significantly improve the effectiveness of neural networks across many tasks including image regression, computed tomography, magnetic resonance imaging (MRI), etc.

In order to explore the deeper reasoning and understand the inner mechanisms behind this simple technique, we consider a two-layer bias-free neural network with Fourier features, i.e.

$$f(\boldsymbol{x}) = \frac{1}{\sqrt{m}} \boldsymbol{W} \cdot \begin{bmatrix} \cos(\boldsymbol{B}\boldsymbol{x}) \\ \sin(\boldsymbol{B}\boldsymbol{x}) \end{bmatrix}, \tag{3.10}$$

where $x \in \mathbb{R}^d$ is the input, $W \in \mathbb{R}^{1 \times 2m}$ is the weight matrix and $\boldsymbol{B} = [\boldsymbol{b}_1, \boldsymbol{b}_2, \dots, \boldsymbol{b}_m]^T \in \mathbb{R}^{m \times d}$ are sampled from Gaussian $\mathcal{N}(0, \sigma^2)$. Let $\{\boldsymbol{x}_i\}_{i=1}^{N}$ be input points in a compact domain C. Then, according to Eq. (3.1), the neural

tangent kernel K is given as

$$
\begin{aligned}
\boldsymbol{K}_{ij} = \boldsymbol{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) &= \frac{1}{m} \begin{bmatrix} \cos(\boldsymbol{B}\boldsymbol{x}_i) \\ \sin(\boldsymbol{B}\boldsymbol{x}_j) \end{bmatrix}^{\mathsf{T}} \cdot \begin{bmatrix} \cos(\boldsymbol{B}\boldsymbol{x}_i) \\ \sin(\boldsymbol{B}\boldsymbol{x}_j) \end{bmatrix} \\
&= \frac{1}{m} \sum_{k=1}^{m} \cos(\boldsymbol{b}_k^T \boldsymbol{x}_i) \cos(\boldsymbol{b}_k^T \boldsymbol{x}_j) + \sin(\boldsymbol{b}_k^T \boldsymbol{x}_i) \sin(\boldsymbol{b}_k^T \boldsymbol{x}_j) \\
&= \frac{1}{m} \sum_{k=1}^{m} \cos(\boldsymbol{b}_k^T (\boldsymbol{x}_i - \boldsymbol{x}_j)).
\end{aligned}
$$

To study the eigen-system of the kernel $\boldsymbol{K}$, we consider the limit of $\boldsymbol{K}$ as the number of points goes to infinity. In this limit the eigensystem of $\boldsymbol{K}$ approaches the eigen-system of the kernel function $K(\boldsymbol{x}, \boldsymbol{x}')$ which satisfies the following equation [44]

$$
\int_C K\left(\boldsymbol{x}, \boldsymbol{x}'\right) g\left(\boldsymbol{x}'\right) d\boldsymbol{x}' = \lambda g\left(\boldsymbol{x}\right), \tag{3.11}
$$

where $K(\boldsymbol{x}, \boldsymbol{x}') = \frac{1}{m} \sum_{k=1}^{m} \cos(\boldsymbol{b}_k^T (\boldsymbol{x} - \boldsymbol{x}'))$. Note that the kernel K induces a Hilbert–Schmidt integral operator $T_K : L^2(C) \to L^2(C)$

$$
T_K(g)(\boldsymbol{x}) = \int_C K\left(\boldsymbol{x}, \boldsymbol{x}'\right) g\left(\boldsymbol{x}'\right) d\boldsymbol{x}'.
$$

Also note that T_K is a compact and self-adjoint operator, which implies that the eigenfunctions exist and all eigenvalues are real. The following lemma reveals that eigenfunctions are indeed solutions to an eigenvalue problem.

Lemma 3.1. *For the kernel* $K(\boldsymbol{x}, \boldsymbol{x}') = \frac{1}{m} \sum_{k=1}^{m} \cos(\boldsymbol{b}_k^T (\boldsymbol{x} - \boldsymbol{x}'))$, *the eigenfunction* $g(\boldsymbol{x})$ *corresponding to non-zero eigenvalues satisfies the following equation*

$$
\Delta g(\boldsymbol{x}) = -\frac{1}{m} \|\boldsymbol{B}\|_F^2 g(\boldsymbol{x}) \tag{3.12}
$$

Proof. The proof can be found in Appendix B. $\square$

If we consider the Laplacian on the sphere S^{d-1} and assume that $\frac{1}{m} \|\boldsymbol{B}\|_2^2 = l(l + d - 2)$ for some positive integer l, then $g(\boldsymbol{x})$ are corresponding homogeneous harmonic polynomials of degree l [45]. However, in general, directly solving this eigenvalue problem on a complex domain is intractable.

To obtain a better understanding of the behavior of the eigenfunctions and the corresponding eigenvalues, let us consider a much simper case by setting $d = 1$ and $m = 1$. Specifically, we take the input $x \in \mathbb{R}$, the compact domain $C = [0, 1]$ and the Fourier features $\boldsymbol{B} = b \in \mathbb{R}$ are sampled from a Gaussian distribution $\mathbb{N}(0, \sigma^2)$. Then the kernel function is given by

$$
K(x, x') = \cos(b(x - x')).
$$

In this case, we can compute the exact expression of the eigenfunctions and their eigenvalues, as summarized in Proposition 3.2.

Proposition 3.2. *For the kernel function* $K(x, x') = \cos(b(x - x'))$, *the non-zero eigenvalues are given by*

$$
\lambda = \frac{1 \pm \frac{\sin b}{b}}{2}. \tag{3.13}
$$

The corresponding eigenfunctions $g(x)$ *must have the form of*

$$
g(x) = C_1 \cos(bx) + C_2 \sin(bx), \tag{3.14}
$$

where C_1 *and* C_2 *are some constants.*

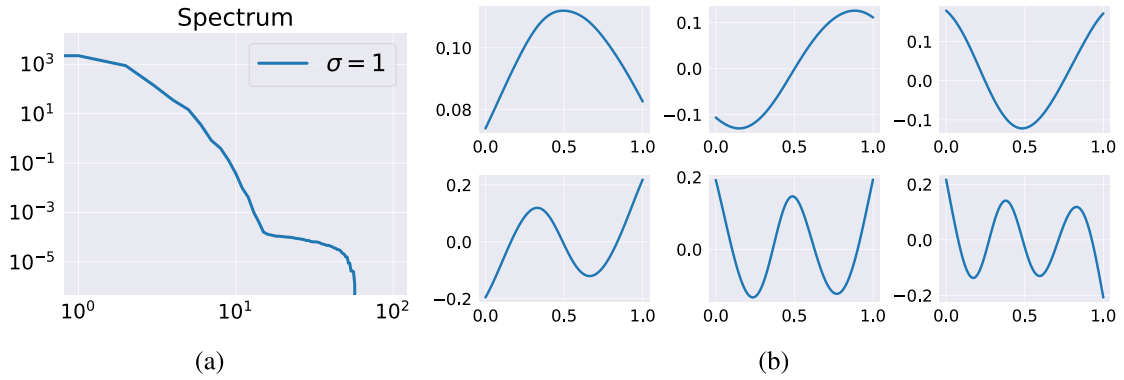**Proof.** The proof can be found in Appendix C. $\square$

Fig. 2. *NTK eigen-decomposition of a fully-connected neural network (4 layer, 100 hidden units,* tanh *activations) with Fourier features initialized by $\sigma = 1$ on 100 equally spaced points in* $[0, 1]$*: (a) The NTK eigenvalues in descending order. (b) The six leading eigenvectors of the NTK in descending order of corresponding eigenvalues.*

From this proposition, we immediately observe that the frequency of the eigenfunctions is determined by b and the gap between the eigenvalues is $\frac{\sin b}{b}$. Besides, recall that b is sampled from a Gaussian distribution $\mathcal{N}(0, \sigma^2)$, which implies that the larger the σ we choose, the higher the probability that b takes greater magnitude. Therefore, we may conclude that, for this toy model, large σ would lead to high frequency eigenfunctions, as well as narrow eigenvalues gaps. As a result, Fourier features may resolve the issue of spectral bias and enable faster convergence to high-frequency components of a target function.

Intuitively, we would expect that general fully-connected neural networks with Fourier features exhibit similar behaviors as our toy example. However, it is extremely difficult to calculate the eigenvalues and eigenfunctions for generic cases. Therefore, here we attempt to empirically verify our analysis by numerical experiments.

To this end, we first initialize two Fourier feature embeddings with $\sigma = 1, 10$, respectively, and apply them to a one-dimensional input coordinate before passing them through a 4-layer fully-connected neural network with 100 units per hidden layer. Then, we study the NTK eigendecomposition of these two networks at initialization. Figs. 2 and 3 show the visualizations of eigenfunctions and eigenvalues of the NTK computed using 100 equally spaced points in $[0, 1]$ for $\sigma = 1, 10$, respectively. One can observe that the eigenvalues corresponding to the Fourier features with $\sigma = 10$ shrink much slower than the ones corresponding to $\sigma = 1$. Moreover, comparing the eigenvectors for different σ, it is easy to see that $\sigma = 10$ results in higher frequency eigenvectors than $\sigma = 1$. This conclusion is further clarified by Fig. 4, which depicts the frequency content of the eigenvector corresponding to the largest eigenvalue, for different $\sigma \in [1, 50]$. All these observations are consistent with Proposition 3.2 and the analysis presented for the toy network with Fourier features.

Next, let us consider a simple one-dimensional target function of the form

$$f(x) = \sin(20\pi x) + \sin(2\pi x), \quad x \in [0, 1], \tag{3.15}$$

and generate the training data $\{x_i, f(x_i)\}_{i=1}^{100}$ where x_i are evenly spaced in the unit interval. We proceed by training these two networks to fit the target function using the Adam optimizer [43] with default settings for 1000 and 10,000 epochs respectively. The results for $\sigma = 1, 10$ are summarized in Figs. 5 and 6, respectively. It can be observed that low frequencies are learned first for $\sigma = 1$, which is pretty similar to the behavior of conventional fully-connected neural networks, commonly referred to as "spectral bias" [31]. Notice, however, that it is high frequencies that are learned first for $\sigma = 10$. As shown in Figs. 2 and 3, we already know that the value of σ determines the frequency of eigenvectors of the NTK. Therefore, this observation highly suggests that "spectral bias" actually corresponds to "eigenvector bias", in the sense that the leading eigenvectors corresponding to large eigenvalues of the NTK determine the frequencies which the network is biased to learn first.

Furthermore, one may note that the distribution of the eigenvalues corresponding to $\sigma = 1$ moves "outward" during training. From our experience, the movement of the eigenvalue distribution results in the movement of its NTK, as well as the parameters of the network during training. As shown in Fig. 5b), the parameters of the network barely move after a rapid change in the first few hundred epochs. This indicates that the network
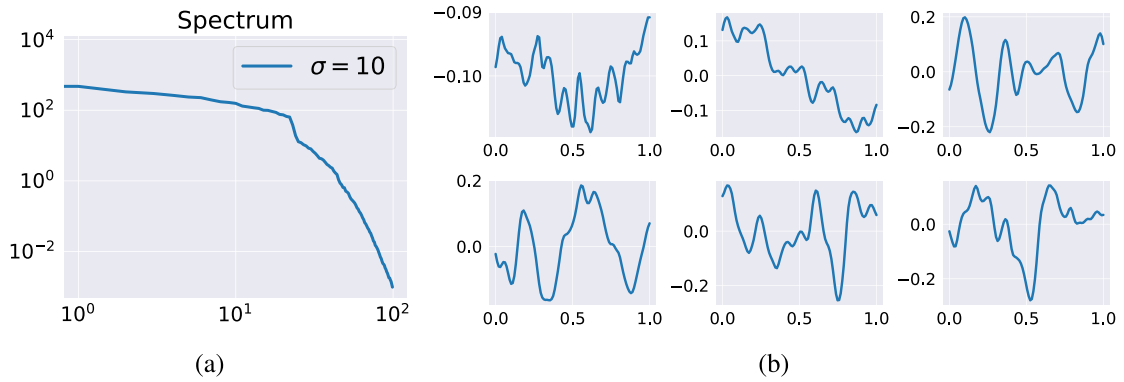
(a)   (b)

Fig. 3. *NTK eigen-decomposition of a fully-connected neural network (4 layer, 100 hidden units,* tanh *activations) with Fourier features initialized by* $\sigma = 10$ *on 100 equally spaced points in* $[0,1]$*: (a) The NTK eigenvalues in descending order. (b) The six leading eigenvectors of the NTK in descending order of corresponding eigenvalues.*
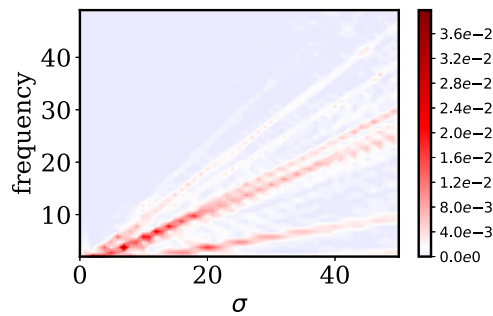

Fig. 4. Frequency domain analysis of the first leading NTK eigenvector for a fully-connected neural network (4 layer, 100 hidden units, tanh activations) with Fourier features initialized by different $\sigma \in [1, 50]$, evaluated on 100 equally spaced points in $[0, 1]$.

initialization is not suitable to fit the given target function because the parameters of the network have to move very far from initialization in order to reach a reasonable local minimum. In contrast, the distribution of the eigenvalues corresponding to $\sigma = 10$ almost keeps the NTK spectrum fixed during training. Accordingly, in the middle panel of Fig. 6b, similar behavior can be observed, but the relative change of the parameters for case of $\sigma = 10$ is much less than the case of $\sigma = 1$. This suggests that the initialization of the network is "good" and desirable local minima exist in the vicinity of the parameter initialization in the corresponding loss landscape. As shown in the top right panels of Figs. 5 and 6, the relative L^2 prediction error corresponding to $\sigma = 10$ decreases much faster than the relative L^2 error corresponding to $\sigma = 1$.

Finally, it is worth emphasizing that Fourier feature mappings initialized by large σ do not always benefit the network, as too large value of σ may cause over-fitting. To demonstrate this point, we initialize a Fourier feature mapping with $\sigma = 10$ and pass it through the same fully-connected network. We now consider $f(x) = \sin \pi x + \sin(2\pi x)$ as the ground truth target function, from which 20 points are uniformly sampled as training data. Then we train the network to fit the target function using the Adam optimizer with default settings for 1000 epochs. As shown in Fig. 7, although all training data pairs are perfectly approximated and the training error is very small, the network interpolates the training data with high frequency oscillations, and thus fails to correctly recover the target function. One possible explanation is that the neural network approximation tends to exhibit similar frequencies as the leading eigenvectors of its NTK. Therefore, choosing an appropriate σ such that the frequency of the leading NTK eigenvectors agrees with the frequency of the target function plays an important role in the network performance, which not only accelerates the convergence speed, but also increases the prediction accuracy.
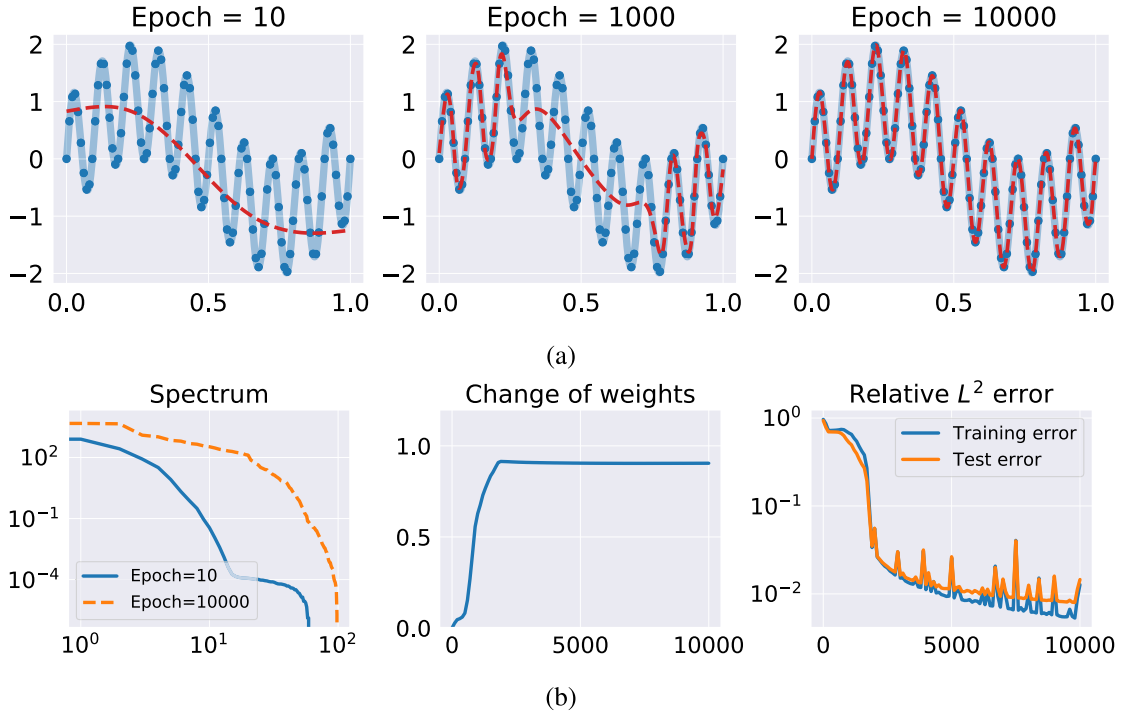
Fig. 5. *Training a network with Fourier features initialized by* $\sigma = 10$ *to fit the target function* $f(x) = \sin(20\pi x) + \sin(2\pi x)$ *for 10,000 epochs: (a)* Network prediction (dash red) against the ground truth (light blue). The network prediction exhibits high frequencies when fitting the data points during training. *(b) middle:* Relative change of the parameters $\boldsymbol{\theta}$ $\left(\frac{\|\theta(t)-\theta(0)\|_2}{|\theta(0)|_2}\right)$ of the network during training. *(b) right:* Relative L^2 training error and test error during training. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

3.3. Multi-scale Fourier feature embeddings for physics-informed neural networks

In the previous section, we presented a detailed theoretical and numerical analysis of the NTK eigen-system of a neural network for classical ℓ_2 regression problems. Building on this insight, we now draw our attention back to physics-informed neural networks for solving forward and inverse problems involving partial differential equations, whose solutions may exhibit multi-scale behavior.

To begin, we first note that the NTK of PINNs [6] is slightly more complicated than the NTK of networks used for conventional regressions. To this end, we follow the setup presented in Wang et al. [6], and consider general PDEs with appropriate boundary conditions (see Eqs. (2.1) and (2.2)) and corresponding "training data" $\{\boldsymbol{x}_b^i, g(\boldsymbol{x}_b^i)\}_{i=1}^{N_b}, \{\boldsymbol{x}_r^i, f(\boldsymbol{x}_r^i)\}_{i=1}^{N_r}$, and define the NTK of PINNs as

$$\boldsymbol{K}(t) = \begin{bmatrix} \boldsymbol{K}_{uu}(t) & \boldsymbol{K}_{ur}(t) \\ \boldsymbol{K}_{ru}(t) & \boldsymbol{K}_{rr}(t) \end{bmatrix},$$

where $\boldsymbol{K}_{ru}(t) = \boldsymbol{K}_{ur}^T(t)$ and $\boldsymbol{K}_{uu}(t) \in \mathbb{R}^{N_b \times N_b}$, $\boldsymbol{K}_{ur}(t) \in \mathbb{R}^{N_b \times N_r}$, and $\boldsymbol{K}_{rr}(t) \in \mathbb{R}^{N_r \times N_r}$, whose (i, j)th entry is given by

$$(\boldsymbol{K}_{uu})_{ij}(t) = \left\langle \frac{d\mathcal{B}[\boldsymbol{u}](\boldsymbol{x}_b^i, \boldsymbol{\theta}(t))}{d\boldsymbol{\theta}}, \frac{d\mathcal{B}[\boldsymbol{u}](\boldsymbol{x}_b^j, \boldsymbol{\theta}(t))}{d\boldsymbol{\theta}} \right\rangle \tag{3.16}$$

$$(\boldsymbol{K}_{ur})_{ij}(t) = \left\langle \frac{d\mathcal{B}[\boldsymbol{u}](\boldsymbol{x}_b^i, \boldsymbol{\theta}(t))}{d\boldsymbol{\theta}}, \frac{d\mathcal{N}[\boldsymbol{u}](\boldsymbol{x}_r^j, \boldsymbol{\theta}(t))}{d\boldsymbol{\theta}} \right\rangle \tag{3.17}$$

$$(\boldsymbol{K}_{rr})_{ij}(t) = \left\langle \frac{d\mathcal{N}[\boldsymbol{u}](\boldsymbol{x}_r^i, \boldsymbol{\theta}(t))}{d\boldsymbol{\theta}}, \frac{d\mathcal{N}[\boldsymbol{u}](\boldsymbol{x}_r^j, \boldsymbol{\theta}(t))}{d\boldsymbol{\theta}} \right\rangle. \tag{3.18}$$
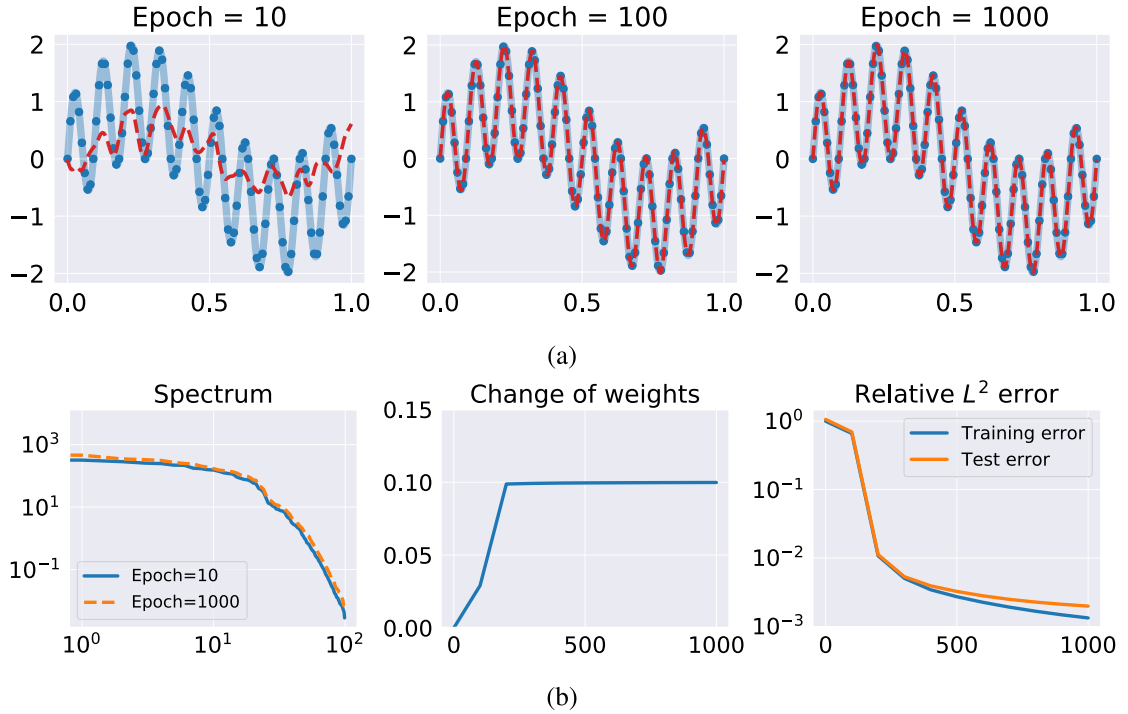
Fig. 6. *Training a network with Fourier features initialized by $\sigma = 1$ to fit the target function $f(x) = \sin(20\pi x) + \sin(2\pi x)$ for 1000 epochs: (a) Network prediction (dash red) against the ground truth (light blue). The network prediction exhibits high frequencies when fitting the data points during training. (b) left: Evolution of NTK eigenvalues during training. (b) middle: Relative change of the parameters $\boldsymbol{\theta}$ ($\frac{\|\theta(t)-\theta(0)\|_2}{\|\theta(0)\|_2}$) of the network during training. (b) right: Relative L^2 training error and test error during training. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)*

Then, the training dynamics of PINNs under gradient descent with an infinitesimally small learning rate can be characterized by the following ODE system

$$\begin{bmatrix} \frac{d\mathcal{B}[\boldsymbol{u}](x_b, \boldsymbol{\theta}(t))}{dt} \\ \frac{d\mathcal{N}[\boldsymbol{u}](x_r, \boldsymbol{\theta}(t))}{dt} \end{bmatrix} = - \begin{bmatrix} \boldsymbol{K}_{uu}(t) & \boldsymbol{K}_{ur}(t) \\ \boldsymbol{K}_{ru}(t) & \boldsymbol{K}_{rr}(t) \end{bmatrix} \cdot \begin{bmatrix} \mathcal{B}[\boldsymbol{u}](x_b, \boldsymbol{\theta}(t)) - \boldsymbol{g}(x_b) \\ \mathcal{N}[\boldsymbol{u}](x_r, \boldsymbol{\theta}(t)) - \boldsymbol{f}(x_r) \end{bmatrix}. \tag{3.19}$$

Then, the NTK framework allows us to show the following Proposition.

Proposition 3.3. *Suppose that the training dynamics of PINNs satisfies Eq. (3.19) and the spectral decompositions of $\boldsymbol{K}_{uu}(0)$ and $\boldsymbol{K}_{rr}(0)$ are*

$$\boldsymbol{K}_{uu}(0) = \boldsymbol{Q}_u^T \Lambda_u \boldsymbol{Q}_u^T \tag{3.20}$$

$$\boldsymbol{K}_{rr}(0) = \boldsymbol{Q}_r^T \Lambda_r \boldsymbol{Q}_r^T, \tag{3.21}$$

where $\boldsymbol{Q}_u$ and $\boldsymbol{Q}_r$ are orthogonal matrices consisting of eigenvectors of $\boldsymbol{K}_{uu}(0)$ and $\boldsymbol{K}_{rr}(0)$, respectively, and Λ_u and Λ_r are diagonal matrices whose entries are the eigenvalues of $\boldsymbol{K}_{uu}(0)$ and $\boldsymbol{K}_{rr}(0)$, respectively. Given the assumptions

  (i) $\boldsymbol{K}(t) \approx \boldsymbol{K}(0)$ for all $t \geq 0$.
  (ii) $\boldsymbol{K}_{uu}(0)$ and $\boldsymbol{K}_{rr}(0)$ are positive definite,

we can write $\boldsymbol{B} = \boldsymbol{Q}_r^T, \boldsymbol{K}_{ru}(0)\boldsymbol{Q}_u$, and obtain

$$\boldsymbol{Q}^T \left(\begin{bmatrix} \mathcal{B}[\boldsymbol{u}](x_b, \boldsymbol{\theta}(t)) \\ \mathcal{N}[\boldsymbol{u}](x_r, \boldsymbol{\theta}(t)) \end{bmatrix} - \begin{bmatrix} \boldsymbol{g}(x_b) \\ \boldsymbol{f}(x_r) \end{bmatrix} \right) \approx e^{-\boldsymbol{P}^T \tilde{\Lambda} \boldsymbol{P} t} \boldsymbol{Q}^T \begin{bmatrix} \boldsymbol{g}(x_b) \\ \boldsymbol{f}(x_r) \end{bmatrix}, \tag{3.22}$$
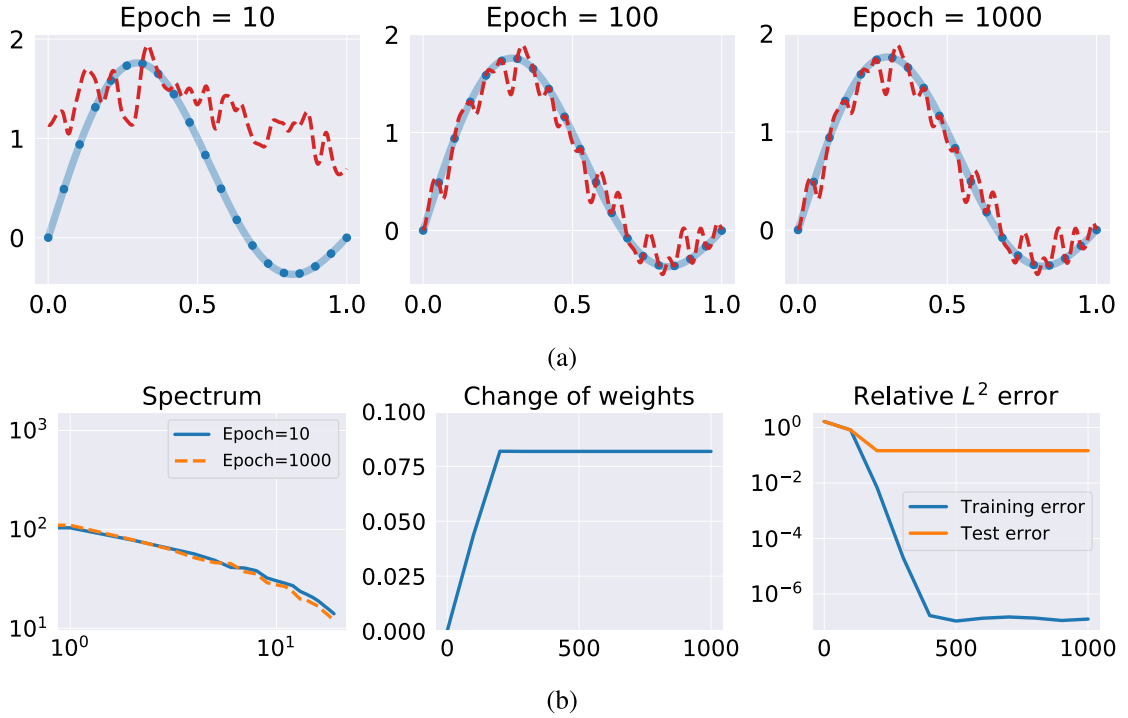
(a)

(b)

Fig. 7. *Training a network with Fourier features initialized by* $\sigma = 10$ *to fit the target function* $f(x) = \sin(\pi x) + \sin(2\pi x)$ *for 1000 epochs: (a) Network prediction (dash red) against the ground truth (light blue). The network prediction exhibits high frequencies when fitting the data points during training. (b) middle: Relative change of the parameters* θ *(* $\frac{\|\theta(t) - \theta(0)\|_2}{\|\theta(0)\|_2}$ *) of the network during training. (b) left: Evolution of NTK eigenvalues during training. (b) right: Relative* L^2 *training error and test error during training. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)*

where

$$Q = \begin{bmatrix} Q_u & 0 \\ 0 & Q_r \end{bmatrix}, \quad P = \begin{bmatrix} I & 0 \\ -B\Lambda_u^{-1} & I \end{bmatrix}, \quad \Lambda = \begin{bmatrix} \Lambda_u & 0 \\ 0 & \Lambda_r - B^T \Lambda_u^{-1} B \end{bmatrix}. \tag{3.23}$$

The above proposition reveals that, under some assumptions, the resulting NTK eigen-system of PINNs is determined by the eigenvectors of K_{uu} and K_{rr}. Furthermore, as shown in [6], during training via gradient descent with an infinitesimally small learning rate, infinitely wide PINNs models effectively behave as a kernel method, with the NTK as their limiting kernel. This implies that, under the assumption that NTK matrix is invertible, an infinitely (or sufficiently) wide PINN is also equivalent to a kernel regression. However, from the authors' experience, the NTK of PINNs is always degenerate which means that we may not be able to casually perform kernel regression predictions in practice, unless additional regularization is introduced [6]. Understanding the behavior of this eigen-system derived from the NTK of PINNs should be at the core of future extensions of this line of research.

As mentioned in Section 2, PINNs often struggle in solving multi-scale problems. Unlike conventional regression tasks, there is generally no or just a handful data points provided for PINNs inside the computational domain. This is similar to the case illustrated in Fig. 7 where the network would fit the target function biases towards its preferred frequencies, which are determined by the eigenvectors of its NTK. Consequently, PINNs using fully-connected networks would learn the solutions and their PDE residuals with the lowest frequency first, due to "spectral bias". We believe that this may be one of the fundamental reasons that cause failure of PINNs in learning high-frequency or multi-scale solutions of PDEs.

Inspired by our analysis and observations of Fourier features in Section 3.2, we present a novel network architecture to handle multi-scale problems. As illustrated in Fig. 8a, we apply multiple Fourier feature embeddings initialized with different σ to input coordinates before passing these embedded inputs through the same fully-connected neural network and finally concatenate the outputs with a linear layer. The detailed forward pass is
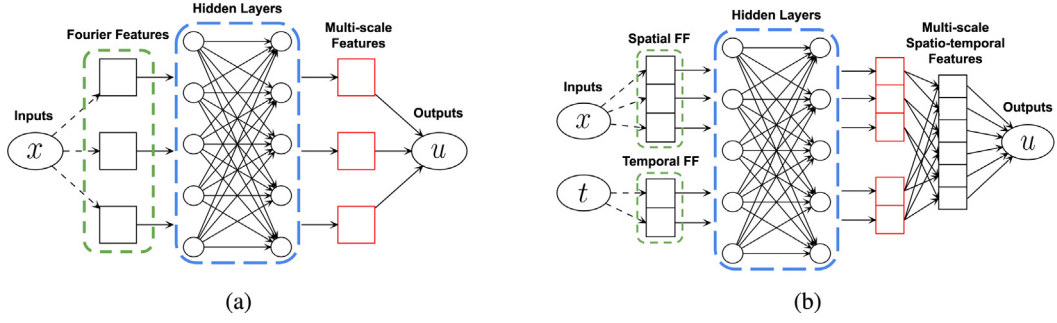
Fig. 8. *(a) Multi-scale Fourier feature architecture:* multiple Fourier feature embeddings (initialized with different σ) are applied to input coordinates and then passed through the same fully-connected neural network, before the outputs are finally concatenated with a linear layer. *(b) Spatio-temporal multi-scale Fourier feature architecture:* Multi-scale Fourier feature architecture: multiple Fourier feature embeddings (initialized with different σ) are separately applied to spatial and temporal input coordinates and then passed through the same fully-connected neural network. Merging of the spatial and temporal outputs is performed using a point-wise multiplication layer, before obtaining the final outputs through a linear layer.

defined as follows:

$$\boldsymbol{\gamma}^{(i)}(\boldsymbol{x}) = \begin{bmatrix} \cos(2\pi \boldsymbol{B}^{(i)} \boldsymbol{x}) \\ \sin(2\pi \boldsymbol{B}^{(i)} \boldsymbol{x}) \end{bmatrix}, \quad \text{for } i = 1, 2, \ldots, M \tag{3.24}$$

$$\boldsymbol{H}_1^{(i)} = \phi(\boldsymbol{W}_1 \cdot \boldsymbol{\gamma}^{(i)}(\boldsymbol{x}) + \boldsymbol{b}_1), \quad \text{for } i = 1, 2, \ldots, M \tag{3.25}$$

$$\boldsymbol{H}_\ell^{(i)} = \phi(\boldsymbol{W}_\ell \cdot \boldsymbol{H}_{\ell-1}^{(i)} + \boldsymbol{b}_\ell), \quad \text{for } \ell = 2, \ldots, L, i = 1, 2, \ldots, M \tag{3.26}$$

$$\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{W}_{L+1} \cdot \left[\boldsymbol{H}_L^{(1)}, \boldsymbol{H}_L^{(2)}, \ldots, \boldsymbol{H}_L^{(M)} \right] + \boldsymbol{b}_{L+1}, \tag{3.27}$$

where $\boldsymbol{\gamma}^{(i)}$ and ϕ denote Fourier feature mappings and activation functions, respectively, and each entry in $\boldsymbol{B}^{(i)} \in \mathbb{R}^{m \times d}$ is sampled from $\mathcal{N}(0, \sigma_i)$, and is held fixed during model training (i.e. $\boldsymbol{B}^{(i)}$ are not trainable parameters, as in [41]). Notice that the weights and the biases of this architecture are essentially the same as in a standard fully-connected neural network. Here, we underline that the choice of σ_i is problem-dependent and typical values can be 1, 20, 50, 100, etc.

To better understand the motivation behind this architecture, suppose that $\boldsymbol{f}_{\boldsymbol{\theta}}$ is a approximation of a given target function f whose Fourier decomposition is

$$f(x) = \sum_{k=-\infty}^{\infty} \hat{f}_k e^{ikx}, \tag{3.28}$$

where $\hat{f}_k$ is the Fourier coefficient corresponding to the wave-number k [46]. Note that $\boldsymbol{f}_{\boldsymbol{\theta}}$ is simply a linear combination of $\{\boldsymbol{H}_L^{(i)}\}_{i=1}^M$, which has some degree of consistency with Eq. (3.28). Moreover, we emphasize again that, for networks with Fourier features, σ determines the frequency that the networks prefer to learn. As a result, if we just employ networks with one Fourier feature embedding, then whatever the value of σ is used to initialize Fourier feature mappings, will yield slower convergence to the rest frequency components, except for the preferable frequencies determined by the choice of σ. Therefore, it is reasonable to embed inputs to several Fourier feature mappings with different σ and concatenate them through a linear layer after the forward propagation such that all frequency components can be learned with the same convergence rate.

For time-dependent problems, multi-scale behavior may exist not only across spatial directions but also across time. Thus, we present another novel multi-scale Fourier feature architecture to tackle multi-scale problems in spatio-temporal domains. Specifically, the feed-forward pass of the network is now defined as

$$\boldsymbol{\gamma}_{\boldsymbol{x}}^{(i)}(\boldsymbol{x}) = \begin{bmatrix} \cos(2\pi \boldsymbol{B}_{\boldsymbol{x}}^{(i)} \boldsymbol{x}) \\ \sin(2\pi \boldsymbol{B}_{\boldsymbol{x}}^{(i)} \boldsymbol{x}) \end{bmatrix}, \quad \boldsymbol{H}_{\boldsymbol{x},1}^{(i)} = \phi(\boldsymbol{W}_1 \cdot \boldsymbol{\gamma}_{\boldsymbol{x}}^{(i)}(\boldsymbol{x}) + \boldsymbol{b}_1), \quad \text{for } i = 1, 2, \ldots, M_x, \tag{3.29}$$

$$\boldsymbol{\gamma}_t^{(j)}(t) = \begin{bmatrix} \cos(2\pi \boldsymbol{B}_t^{(j)} t) \\ \sin(2\pi \boldsymbol{B}_t^{(j)} t) \end{bmatrix}, \quad \boldsymbol{H}_{t,1}^{(j)} = \phi(\boldsymbol{W}_1 \cdot \boldsymbol{\gamma}_t^{(j)}(t) + \boldsymbol{b}_1), \quad \text{for } j = 1, 2, \ldots, M_t, \tag{3.30}$$

Table 1

Training time (s) for performing 10 stochastic gradient iterations for the different numerical examples and neural network architectures considered in this section. Reported timings are measured on a Lenovo X1 Carbon ThinkPad laptop with an Intel Core i5 2.3 GHz processor and 8 Gb of RAM memory.

Computational cost (s)	Plain	MFF	ST-MFF
Poisson equation	0.03	0.03	–
Heat equation	0.04	0.05	0.08
Wave equation	0.13	0.28	0.37
Gray–Scott equation	2.39	–	4.24

$$\boldsymbol{H}_{x,\ell}^{(i)} = \phi(\boldsymbol{W}_\ell \cdot \boldsymbol{H}_{x,\ell-1}^{(i)} + \boldsymbol{b}_\ell), \quad \text{for } \ell = 2, \ldots, L \text{ and } i = 1, 2, \ldots, M_x, \tag{3.31}$$

$$\boldsymbol{H}_{t,\ell}^{(j)} = \phi(\boldsymbol{W}_\ell \cdot \boldsymbol{H}_{t,\ell-1}^{(j)} + \boldsymbol{b}_\ell), \quad \text{for } \ell = 2, \ldots, L \text{ and } j = 1, 2, \ldots, M_t, \tag{3.32}$$

$$\boldsymbol{H}_L^{(i,j)} = \boldsymbol{H}_{x,L}^{(i)} \odot \boldsymbol{H}_{t,L}^{(j)}, \quad \text{for } i = 1, 2, \ldots, M_x \text{ and } j = 1, 2, \ldots, M_t, \tag{3.33}$$

$$\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}, t) = \boldsymbol{W}_{L+1} \cdot \left[\boldsymbol{H}_L^{(1,1)}, \ldots, \boldsymbol{H}_L^{(M_x, M_t)} \right] + \boldsymbol{b}_{L+1}, \tag{3.34}$$

where $\boldsymbol{\gamma}_x^{(i)}$ and $\boldsymbol{\gamma}_t^{(j)}$ denote spatial and temporal Fourier feature mappings, respectively, and $\odot$ represents the point-wise multiplication. Here each entry of $\boldsymbol{B}_x^{(i)}$ and $\boldsymbol{B}_t^{(j)}$ is sampled from $\mathcal{N}(0, \sigma_i^x)$ and $\mathcal{N}(0, \sigma_j^t)$, respectively, and are held fixed during model training. A visualization of this architecture is presented in Fig. 8b. One key difference from Fig. 8a is that we apply separate Fourier feature embeddings to spatial and temporal input coordinates before passing the embedded inputs through the same fully-connected network. Another key difference is that we merge spatial outputs $\boldsymbol{H}_{x,L}^{(i)}$ and temporal outputs $\boldsymbol{H}_{t,L}^{(j)}$ using point-wise multiplication and passing them through a linear layer. Heuristically, this architecture is consistent with the Fourier spectral method [46], i.e., given a function $f(\boldsymbol{x}, t)$, using Fourier series we may rewrite it as

$$f(\boldsymbol{x}, t) = \sum_{k=-\infty}^{\infty} \hat{f}_k(t) e^{ik\boldsymbol{x}} \tag{3.35}$$

where $\hat{f}_k(t)$ is the Fourier coefficient corresponding to the wave-number k at time t.

It is also worth noting that both proposed architectures neither introduce any additional trainable parameters compared to conventional PINN models, nor do they require significantly more floating point operations to evaluate their forward or backward pass. Therefore they can be used as drop-in replacements to conventional fully-connected architectures with no sacrifices to computational efficiency (see Table 1). In Section 4, we will validate the effectiveness of the proposed architectures through a series of systematic numerical experiments.

4. Results

In this section we demonstrate the performance of the proposed architectures in solving forward and inverse multi-scale problems. Throughout all benchmarks, we employ hyperbolic tangent activation functions and initialize the network using the Glorot normal scheme [47]. All networks are trained via stochastic gradient descent using the Adam optimizer [43] with defaulting settings. Particularly, we employ exponential learning rate decay with a decay-rate of 0.9 every 1000 training iterations. Table 1 summarizes the computational cost for each numerical example in this section, in terms of training time (in sec) for performing 10 stochastic gradient iterations on a Lenovo X1 Carbon ThinkPad laptop with an Intel Core i5 2.3 GHz processor and 8 Gb of RAM memory. All results presented in this section can be reproduced using our publicly available codes at https://github.com/Predicti veIntelligenceLab/MultiscalePINNs.

4.1. 1D Poisson equation

We begin with a pedagogical example involving the one-dimensional (1D) Poisson equation benchmark described in Section 2 and examine the performance of the proposed architectures.

We begin by approximating the latent solution $u(x)$ with the proposed multi-scale Fourier feature architecture (Fig. 8a). Specifically, we apply two Fourier feature mappings to 1D input coordinates, respectively, before passing
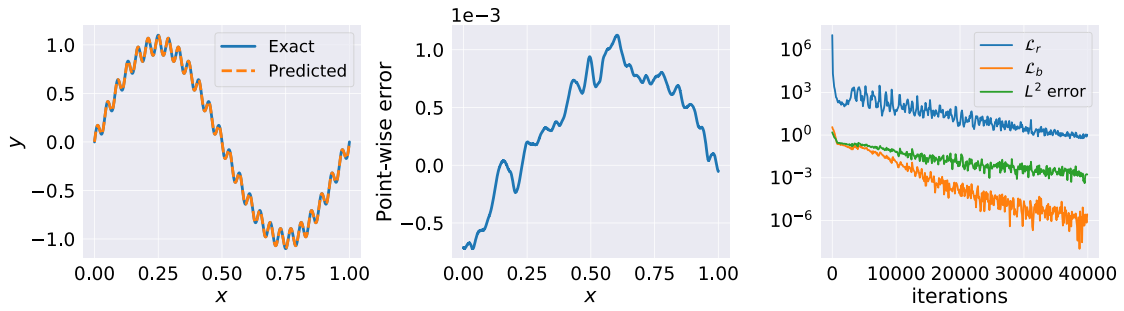
Fig. 9. *1D Poisson equation:* Results obtained by training a fully-connected network (2-layer, 100 hidden units, tanh activations) with the proposed multi-scale Fourier feature mappings via 40,000 iterations of gradient descent. *Left:* Comparison of the predicted and exact solutions. The relative L^2 error is $1.36e{-}03$. *Middle:* Point-wise error between the predicted and the exact solution. *Right:* Evolution of the residual loss $\mathcal{L}_r$, the boundary loss $\mathcal{L}_b$, as well as the relative L^2 error during training.
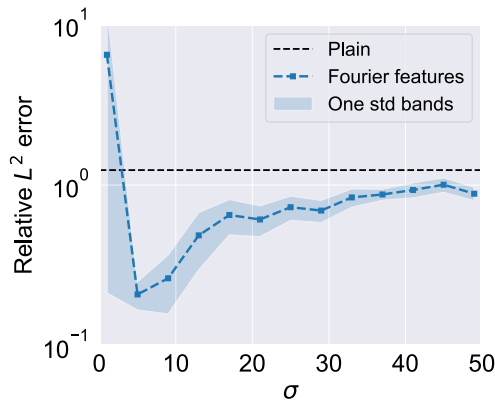
Fig. 10. *1D Poisson equation:* Relative L^2 errors of predicted solutions averaged over 10 independent trials by training a plain fully-connected neural network (2-layer, 100 hidden units, tanh activations), as well as the same network with single Fourier feature mapping initialized by different $\sigma \in [1, 50]$.

them through a 2-layer fully-connected neural network with 100 units per hidden layer, and then concatenating them through a linear layer. Particularly, these Fourier feature mappings are initialized with $\sigma_1 = 1$ and $\sigma_2 = 10$, respectively. We train the network by minimizing the loss function (2.7) in Section 2 under the exactly same hyper-parameter settings. Fig. 9 summarizes our results of the predicted solution to the 1D Poisson equation after 40,000 training iterations. One can see that the predicted solution obtained using the proposed architecture achieves excellent agreement with the exact solution, yielding a $1.36e{-}03$ prediction error measured in the relative L^2-norm.

Furthermore, we aim to demonstrate that the proposed architecture outperforms the conventional PINNs, as well as the PINNs with a single Fourier feature mapping. To this end, we first train a plain a conventional physics-informed network using exactly the same hyper-parameters and take the resulting relative L^2 error as our baseline. Next, we train the same network with a conventional Fourier feature mapping, considering different initializations for σ in the range $[1, 50]$, and report the resulting relative L^2 errors over 10 independent trials in Fig. 10. It can be observed that either plain PINNs or PINNs with vanilla Fourier features fail to attain good prediction accuracy, yielding errors ranging from 10% to above 100% in the relative L^2-norm. In particular, we visualize the results obtained using the same network with conventional Fourier features [41] initialized by $\sigma = 1$ and $\sigma = 50$, respectively. As shown in Figs. 11a and 11b, the network with conventional Fourier features initialized by $\sigma = 1$ tends to capture the low frequency components of the solution while the one initialized by $\sigma = 50$ successfully captures the high frequency oscillations but ignores the low frequency components.
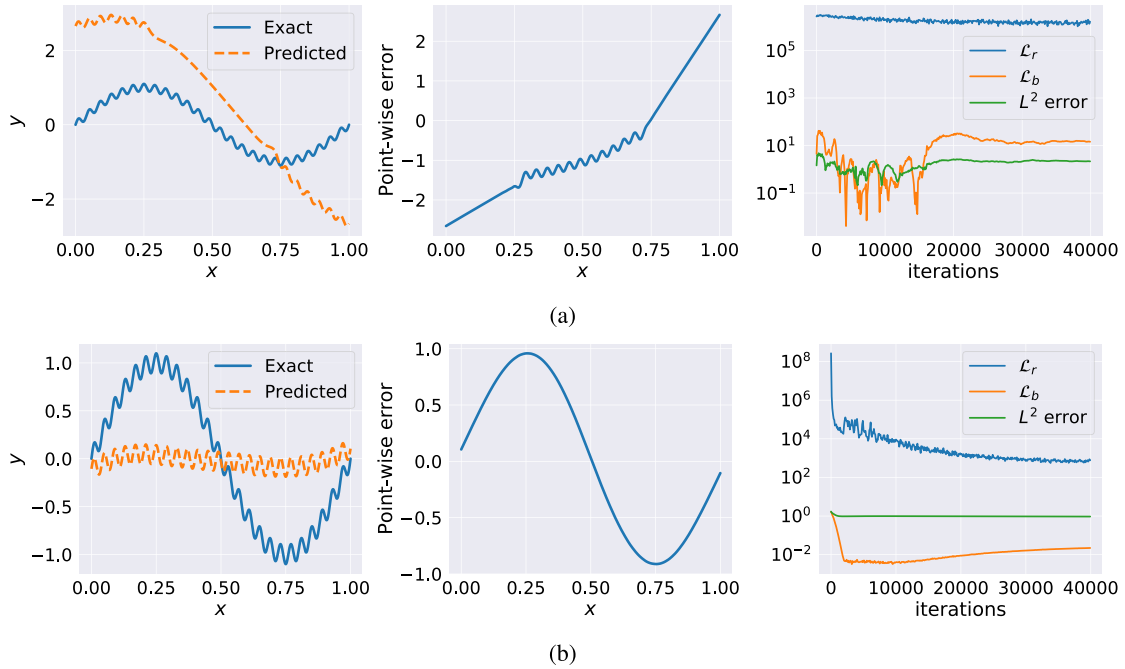
(a)

(b)

Fig. 11. *1D Poisson equation:* *(a)* Results obtained by training a fully-connected network (2-layer, 100 hidden units, tanh activations) with a single Fourier feature mapping initialized by $\sigma = 1$ via 40,000 iterations of gradient descent. *Left:* Comparison of the predicted and exact solutions. *Middle:* Point-wise error between the predicted and the exact solution. *Right:* Evolution of the residual loss $\mathcal{L}_r$, the boundary loss $\mathcal{L}_b$, as well as the relative L^2 error during training. *(b)* Results obtained by training the same network with single Fourier feature mapping initialized by $\sigma = 50$ via 40,000 iterations of gradient descent. *Left:* Comparison of the predicted and exact solutions. *Middle:* Point-wise error between the predicted and the exact solution. *Right:* Evolution of the residual loss $\mathcal{L}_r$, the boundary loss $\mathcal{L}_b$, as well as the relative L^2 error during training.

4.2. High frequencies in a heat equation

To demonstrate the necessity and effectiveness of the proposed spatio-temporal multi-scale Fourier feature architecture (ST-MFF), let us consider the one-dimensional heat equation taking the form

$$u_t = \frac{1}{(500\pi)^2} u_{xx}, \quad (x, t) \in (0, 1) \times (0, 1) \tag{4.1}$$

$$u(x, 0) = \sin(500\pi x), \quad x \in [0, 1] \tag{4.2}$$

$$u(0, t) = u(1, t) = 0, \quad t \in [0, 1]. \tag{4.3}$$

The exact solution $u(x, t)$ for this benchmark is given by

$$u(x, t) = e^{-t} \sin(500\pi x). \tag{4.4}$$

As the solution is mainly dominated by a single high frequency in the spatial domain, it suffices to employ just single Fourier feature mapping in the network architecture. We proceed by approximating the latent variable $u(x, t)$ with the network $u_\theta(x)$ using the proposed spatio-temporal architecture (Fig. 8b). Specifically, we embed the spatial and temporal input coordinates through two separate Fourier features initialized with $\sigma = 200$ and $\sigma = 1$, respectively, and pass the embedded inputs though a 3-layer fully-connected neural network with 100 neurons per hidden layer and finally merge them according to Eqs. (3.31)–(3.34). The network is trained by minimizing the following loss

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_{bc}(\boldsymbol{\theta}) + \mathcal{L}_{ic}(\boldsymbol{\theta}) + \mathcal{L}_r(\boldsymbol{\theta}) \tag{4.5}$$

$$= \frac{1}{N_{bc}} \sum_{i=1}^{N_{bc}} \left| u_\theta(x_{bc}^i, t_{bc}^i) - u(x_{bc}^i, t_{bc}^i) \right|^2 + \frac{1}{N_{ic}} \sum_{i=1}^{N_{ic}} \left| u_\theta(x_{ic}^i, t_{ic}^i) - u(x_{ic}^i, t_{ic}^i) \right|^2 \tag{4.6}$$

$$+ \frac{1}{N_r} \sum_{i=1}^{N_r} \left| \frac{\partial u_\theta}{\partial t}(x_r^i, t_r^i) - \frac{\partial^2 u_\theta}{\partial x^2}(x_r^i, t_r^i) \right|^2, \tag{4.7}$$
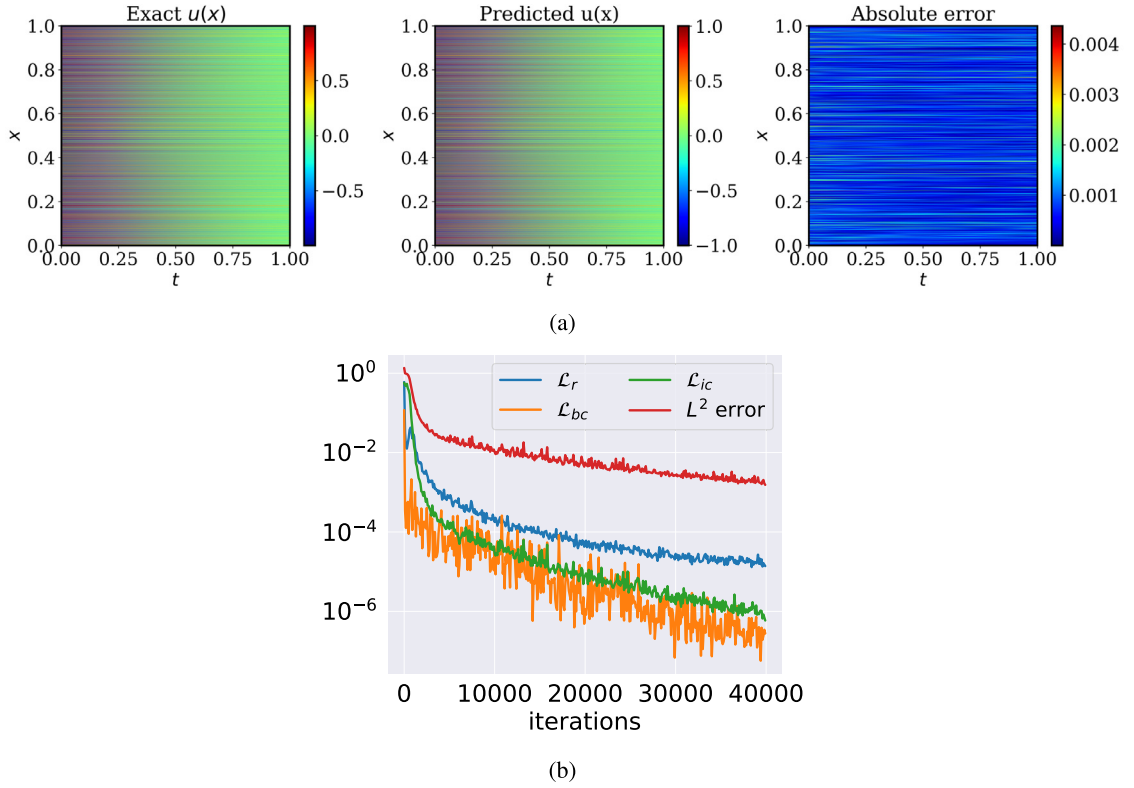
(a)

(b)

Fig. 12. *1D heat equation: (a)* Exact solution versus the predicted solution by training a fully-connected network (3-layer, 100 hidden units, tanh activations) with the spatio-temporal Fourier feature mappings via 40,000 iterations of gradient descent. The relative L^2 error is $1.78e{-}03$. *(b)* Evolution of the different terms in the loss function, as well as the relative L^2 error during training.

where we set the batch sizes to $N_{bc} = N_{ic} = N_r = 128$. The predictions of the trained model against the exact solution along with the point-wise absolute error between them are presented in Fig. 12a. This figure indicates that our spatio-temporal multi-scale Fourier feature architecture is able to accurately capture the high frequency oscillations, leading to a $1.78e{-}03$ prediction error measured in the relative L^2-norm. To the best of author's knowledge, this is the first time that PINNs can be effective in solving a time-dependent problem exhibiting such extremely high frequencies.

Next, we test the performance of the multi-scale Fourier feature architecture. To this end, we represent the unknown solution $u(x, t)$ by the same network (3-layer, 100 hidden units) with single Fourier feature mapping initialized using different $\sigma \in [1, 1000]$. A visual assessment of the resulting relative L^2 error, as well as the baseline obtained by the conventional PINNs are shown in Fig. 13. As suggested in this figure, all reported relative L^2 errors are around 100%, which implies that both conventional PINNs and the multi-scale Fourier feature architecture are incapable of learning high frequency components of solutions in the spatio-temporal domain.

4.3. Wave propagation

In this example, we aim to demonstrate that employing appropriate architectures solely cannot guarantee accurate predictions. To this end, we consider one-dimensional wave equation taking the form

$$u_{tt}(x, t) - 100u_{xx}(x, t) = 0, \quad (x, t) \in (0, 1) \times (0, 1) \tag{4.8}$$

$$u(0, t) = u(1, t) = 0, \quad t \in [0, 1] \tag{4.9}$$

$$u(x, 0) = \sin(\pi x) + \sin(2\pi x), \quad x \in [0, 1] \tag{4.10}$$

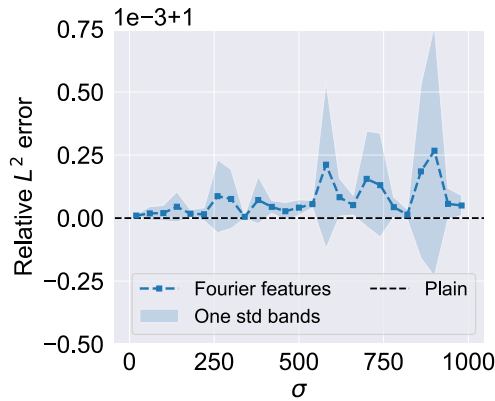$$u_t(x, 0) = 0, \quad x \in [0, 1]. \tag{4.11}$$

16

Fig. 13. *1D heat equation:* Relative L^2 errors of predicted solutions averaged over 10 independent trials by training a plain fully-connected neural network (3-layer, 100 hidden units, tanh activations), as well as the same network with single Fourier feature mapping initialized by different $\sigma \in [1, 1000]$. We can see that the resulting relative L^2 errors of all trained models are greater than 100%, which indicates that both the conventional PINNs and the plain Fourier feature networks completely fail to learn the correct solution.

By d'Alembert's formula [45], the solution $u(x, t)$ is given by

$$u(x, t) = \sin(\pi x)\cos(10\pi t) + \sin(2\pi x)\cos(20\pi t). \tag{4.12}$$

To handle the multi-scale behavior in both spatial and temporal directions, we employ the spatio-temporal architecture (Fig. 8b) to approximate the latent solution $u(x, t)$. Specifically, we apply two separate Fourier feature mappings initialized by $\sigma = 1, 10$ respectively to temporal coordinates t and apply one Fourier feature mapping initialized by $\sigma = 1$ to spatial coordinates x. Then we pass all featurized spatial and temporal inputs coordinated through a 3-layer fully-connected neural network with 200 units per hidden layer and concatenate network outputs using Eqs. (3.33)–(3.34). In particular, we treat the initial condition (4.10) as a special boundary condition on the spatio-temporal domain Ω. Then Eqs. (4.9) and (4.10) can be summarized as

$$u(\boldsymbol{x}) = g(\boldsymbol{x}), \quad x \in \partial \Omega$$

Then, the network can be trained by minimizing the following loss function

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_u(\boldsymbol{\theta}) + \mathcal{L}_{u_t}(\boldsymbol{\theta}) + \mathcal{L}_r(\boldsymbol{\theta}) \tag{4.13}$$

$$= \frac{1}{N_u}\sum_{i=1}^{N_u}|u(\boldsymbol{x}_u^i, \boldsymbol{\theta}) - g(\boldsymbol{x}_u^i)|^2 + \frac{1}{N_{u_t}}\sum_{i=1}^{N_{u_t}}|u_t(\boldsymbol{x}_{u_t}^i, \boldsymbol{\theta})|^2 + \frac{1}{N_r}\sum_{i=1}^{N_r}|\frac{\partial^2 u_{\boldsymbol{\theta}}}{\partial t^2}(\boldsymbol{x}_r^i) - 100\frac{\partial^2 u_{\boldsymbol{\theta}}}{\partial x^2}(\boldsymbol{x}_r^i)|^2, \tag{4.14}$$

where the batch sizes are set to $N_u = N_{u_t} = N_r = 360$ and all data points $\{\boldsymbol{x}_u^i, g(\boldsymbol{x}_u^i)\}_{i=1}^{N_u}$, $\{\boldsymbol{x}_{u_t}^i\}_{i=1}^{N_{u_t}}$ and $\{\boldsymbol{x}_r^i\}_{i=1}^{N_r}$ are uniformly sampled from the appropriate regions in the computational domain at each iteration of gradient descent.

Fig. 14 presents a comparison of the exact and predicted solution obtained after 40,000 iterations of gradient descent. It is evident that the PINN model completely fails to learn the correct solution. This illustrated the fact that even if we choose an appropriate network architecture to approximate the latent PDE solution, there might be some other issues that lead PINNs to fail. Wang et al. [6] found that multi-scale problems are more likely to cause a large discrepancy in the convergence rate of different terms contributing to the total training error, which may lead to severe training difficulties of PINNs in practice. Such a discrepancy can be further justified in Fig. 14b, from which one can see that the loss $\mathcal{L}_r$ and loss $\mathcal{L}_{u_t}$ decrease much faster than the loss $\mathcal{L}_u$ during training. We believe that this may be a fundamental reason behind the collapse of physics-informed neural networks, and their inability to yield accurate predictions for this specific example.

To address this a training pathology, we employ the adaptive weights algorithm proposed by Wang et al. [6] to train the same network with the same spatio-temporal Fourier feature mappings under the same exactly hyper-parameter settings. As shown in Fig. 15, the results demonstrate excellent agreement between the predicted and the exact solution with relative L^2 error within 0.1%. Furthermore, we test the performance of standard fully-connected network and the proposed multi-scale Fourier feature architecture, with or without the adaptive weights algorithm
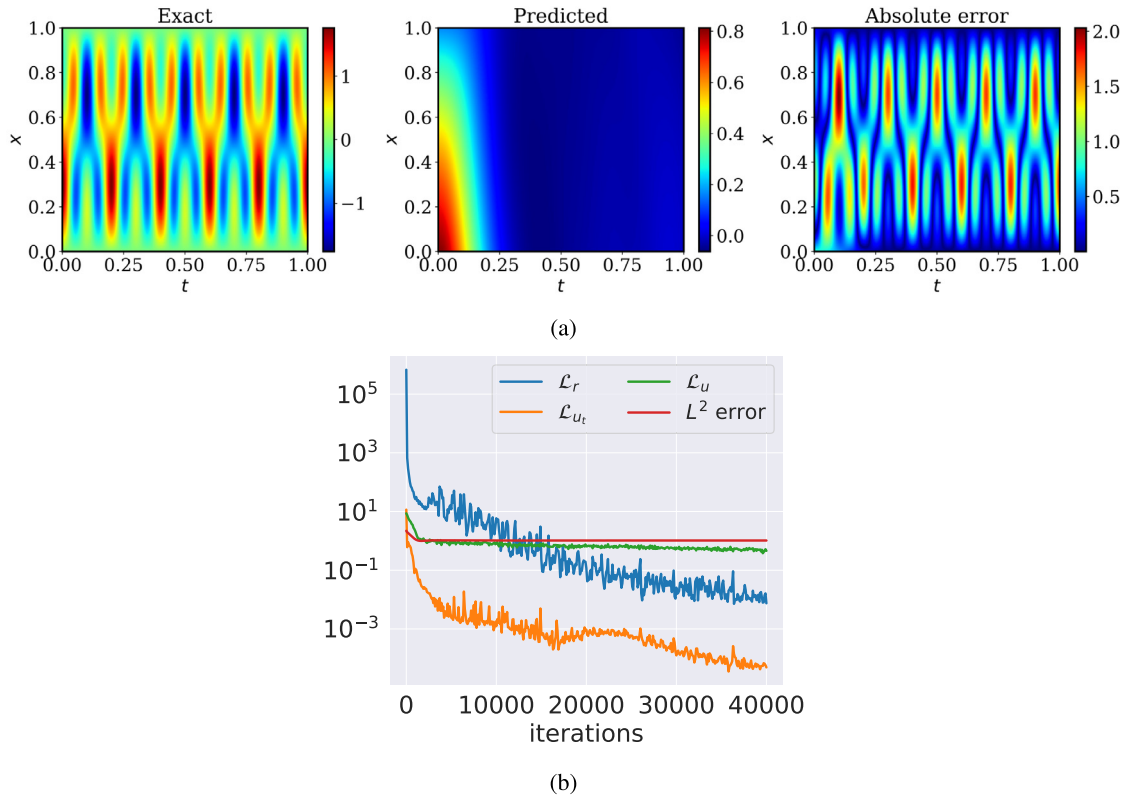
(a)

(b)

Fig. 14. *1D Wave equation: (a)* Exact solution versus the predicted solution by training a fully-connected network (3-layer, 200 hidden units, tanh activations) with the spatio-temporal Fourier feature mappings via 40,000 iterations of gradient descent. The relative L^2 error is 1.03e+00. *(b)* Evolution of the different terms in the loss function, as well as the relative L^2 error during training.

Table 2
1D wave equation: Relative L^2 errors of the predicted solutions obtained using conventional fully-connected networks (plain), multi-scale Fourier features architecture (MFF) and spatio-temporal multi-scale Fourier features architecture (ST-MFF).

Method	Architecture		
	Plain	MFF	ST-MFF
No adaptive weights	1.01e+00	1.03e+00	1.02e+00
With adaptive weights	8.77e−01	1.00e+00	9.83e−04

and the resulting relative L^2 errors are summarized in Table 2. For all results shown in the table, we employ a 3-layer fully-connected network with 200 units per hidden layer as a backbone. To obtain the result of the multi-scale Fourier features architecture, we jointly embed the spatio-temporal coordinates (x, t) using two separate Fourier feature mappings initialized with $\sigma = 1, 10$, respectively. We observe that only the proposed spatio-temporal Fourier feature architecture trained with the adaptive weights algorithm of Wang et al. [6] is capable of achieving an accurate approximation of the true solution, which highly suggests the necessity of combining the proposed architecture with an appropriate optimization scheme for training.

4.4. Reaction–diffusion dynamics in a two-dimensional Gray–Scott model

Our final example aims to highlight the ability of the proposed methods to handle inverse problems. Let us consider a two-dimensional Gray–Scott model [48] that describes two non-real chemical species U, V reacting and transforming to each other. This model is governed by a coupled system of reaction–diffusion equations taking the
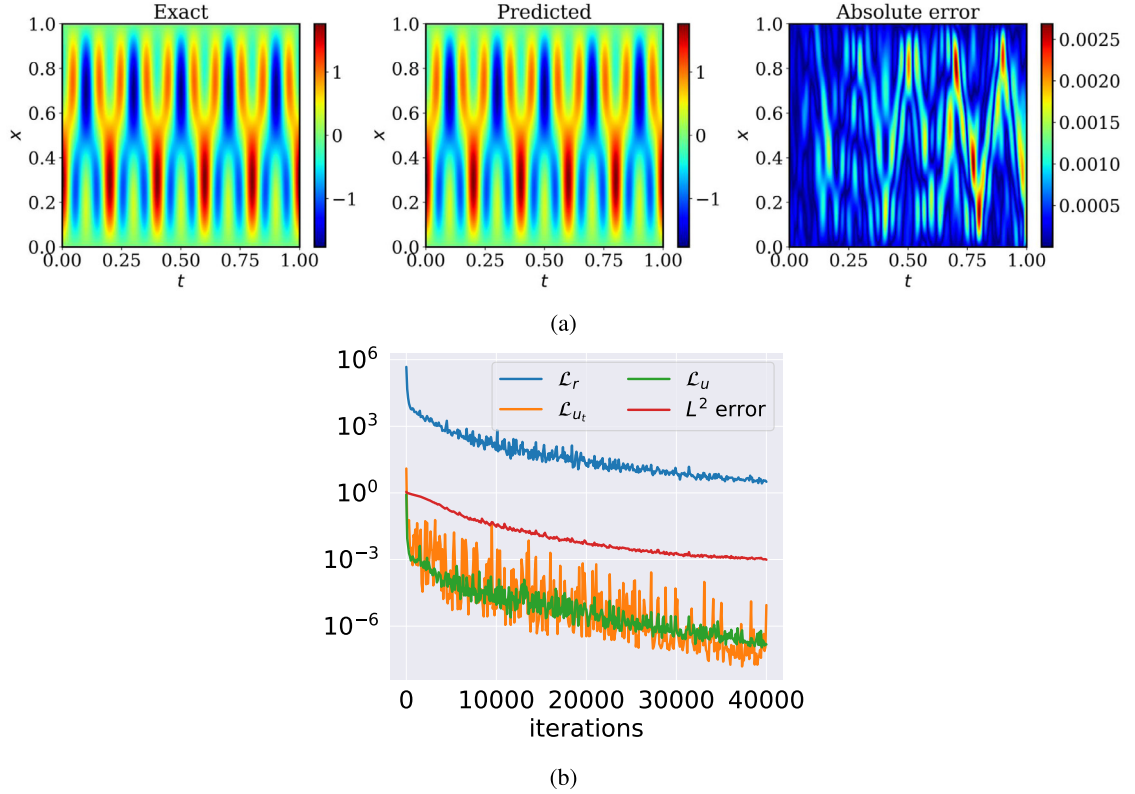
(a)

(b)

Fig. 15. *1D wave equation: (a)* Exact solution versus the predicted solution by training a fully-connected network (3-layer, 200 hidden units, tanh activations) with the spatio-temporal Fourier feature mappings using adaptive weights algorithm [6] via 40,000 iterations of gradient descent. The relative L^2 error is $9.83e{-}04$. *(b)* Evolution of the different terms in the loss function, as well as the relative L^2 error during training.

form

$$u_t = \varepsilon_1 \Delta u + b(1 - u) - uv^2 \tag{4.15}$$

$$v_t = \varepsilon_2 \Delta v - dv + uv^2 \tag{4.16}$$

where u, v represent the concentrations of U, V respectively and ϵ_1, ϵ_2 are their corresponding diffusion rates.

We generate a data-set containing a direct numerical solution of the two-dimensional Gray–Scott equations with 40,000 spatial points and 401 temporal snapshots. Specifically, we take $b = 0.04, d = 0.1, \epsilon_1 = 2e{-}5, \epsilon_2 = 1e{-}5$ and, assuming periodic boundary conditions, we start from an initial condition

$$u(x, y, 0) = 1 - \exp(-80(x + 0.05)^2 + (y + 0.02)^2), \quad (x, y) \in [-1, 1] \times [-1, 1]$$

$$v(x, y, 0) = \exp(-80(x - 0.05)^2 + (y - 0.02)^2), \quad (x, y) \in [-1, 1] \times [-1, 1],$$

and integrate the equations up to the final time $t = 4000$. Synthetic training data for this example are generated using the Chebfun package [49] with a spectral Fourier discretization and a fourth-order stiff time-stepping scheme [50] with time-step size of 0.5. Temporal snapshots of the solution are saved every $\Delta t = 10$. From this data-set, we create a smaller training subset by collecting the data points from time $t = 3500$ to $t = 4000$ (50 snapshots in total) as our training data. A representative snapshot of the numerical solution is presented in Fig. 16. As illustrated in this figure, the solution exhibits complex spatial patterns with a non-trivial frequency content.

Given training data $\{(x^i, y^i, t^i), (u^i, v^i)\}_{i=1}^{N}$ and assuming that b, d are known, we are interested in predicting the latent concentration fields u, v, as well as inferring the unknown diffusion rates ϵ_1, ϵ_2. To this end, we represent
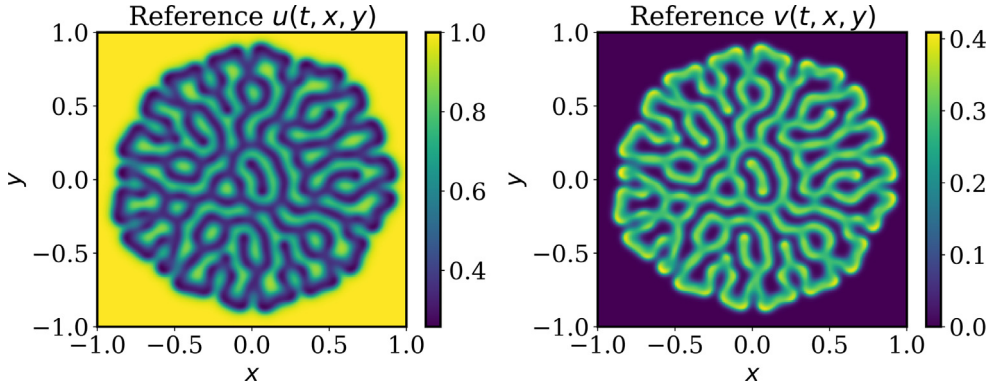
Fig. 16. *2D Gray–Scott equation:* Representative snapshots of the ground truth concentration fields u, v at $t = 3500$.

the latent variables u, v by a deep network employing the proposed spatio-temporal Fourier feature architecture

$$(x, y, t) \xrightarrow{f_\theta} (u_\theta, v_\theta) \tag{4.17}$$

To be precise, we map temporal coordinates t by a Fourier feature embedding with $\sigma = 1$ and map spatial coordinates (x, y) by another Fourier feature embedding with $\sigma = 30$. Then we pass the embedded inputs through a 9-layer fully-connected neural network with 100 neurons per hidden layer. The corresponding loss function is given by

$$\mathcal{L}(\theta) = \mathcal{L}_u(\theta) + \mathcal{L}_v(\theta) + \mathcal{L}_{r^u}(\theta) + \mathcal{L}_{r^v}(\theta) \tag{4.18}$$

$$= \frac{1}{N_u} \sum_{i=1}^{N_u} \left| u_\theta(x^i, y^i, y^i) - u^i \right|^2 + \frac{1}{N_v} \sum_{i=1}^{N_v} \left| v_\theta(x^i, y^i, y^i) - v^i \right|^2 \tag{4.19}$$

$$+ \frac{1}{N_{r^u}} \sum_{i=1}^{N_{r^u}} \left| r_\theta^u(x_r^i, y_r^i, t_r^i) \right|^2 + \frac{1}{N_{r^v}} \sum_{i=1}^{N_{r^v}} \left| r_\theta^v(x_r^i, y_r^i, t_r^i) \right|^2, \tag{4.20}$$

where the PDE residuals are defined as

$$r_\theta^u = \frac{\partial u_\theta}{\partial t} - \epsilon_1 \Delta u_\theta - b(1 - u_\theta) + u_\theta v_\theta^2 \tag{4.21}$$

$$r_\theta^v = \frac{\partial v_\theta}{\partial t} - \epsilon_2 \Delta v_\theta + d v_\theta - u_\theta v_\theta^2. \tag{4.22}$$

Here we choose batch sizes $N_u = N_v = N_{r^u} = N_{r^v} = 1000$ where all data points along with collocation points are randomly sampled at each iteration of gradient descent. Particularly, since the diffusion rates are strictly positive and generally very small, we parameterize ϵ_1, ϵ_2 by exponential functions, i.e $\epsilon_i = e^{\alpha_i}$ for $i = 1, 2$ where α_i's are trainable parameters initialized by -10.

We train the network by minimizing the above loss function via 120,000 iterations of gradient descent. Figs. 17a and 17b present the comparisons of reconstructed concentrations u, v against the ground truth functions at the final time $t = 4000$. The results show excellent agreement between the predictions and the numerical estimations. This is further validated by the relative L^2-norm of error results shown in Fig. 17c. Moreover, the evolution of inferred diffusion rates during training, as well as the final predictions are presented in Fig. 17d and Table 3 respectively, which show good agreement with the exact values.

To compare these results against the performance of a conventional PINNs model [4], we also train the same fully-connected neural network (9-layer, width 100) under the same hyper-parameter settings. The results of this experiment are summarized in Fig. 18. Evidently, conventional PINNs are incapable of accurately learning the concentrations u, v, as well as inferring the unknown diffusion rates under the current setting.
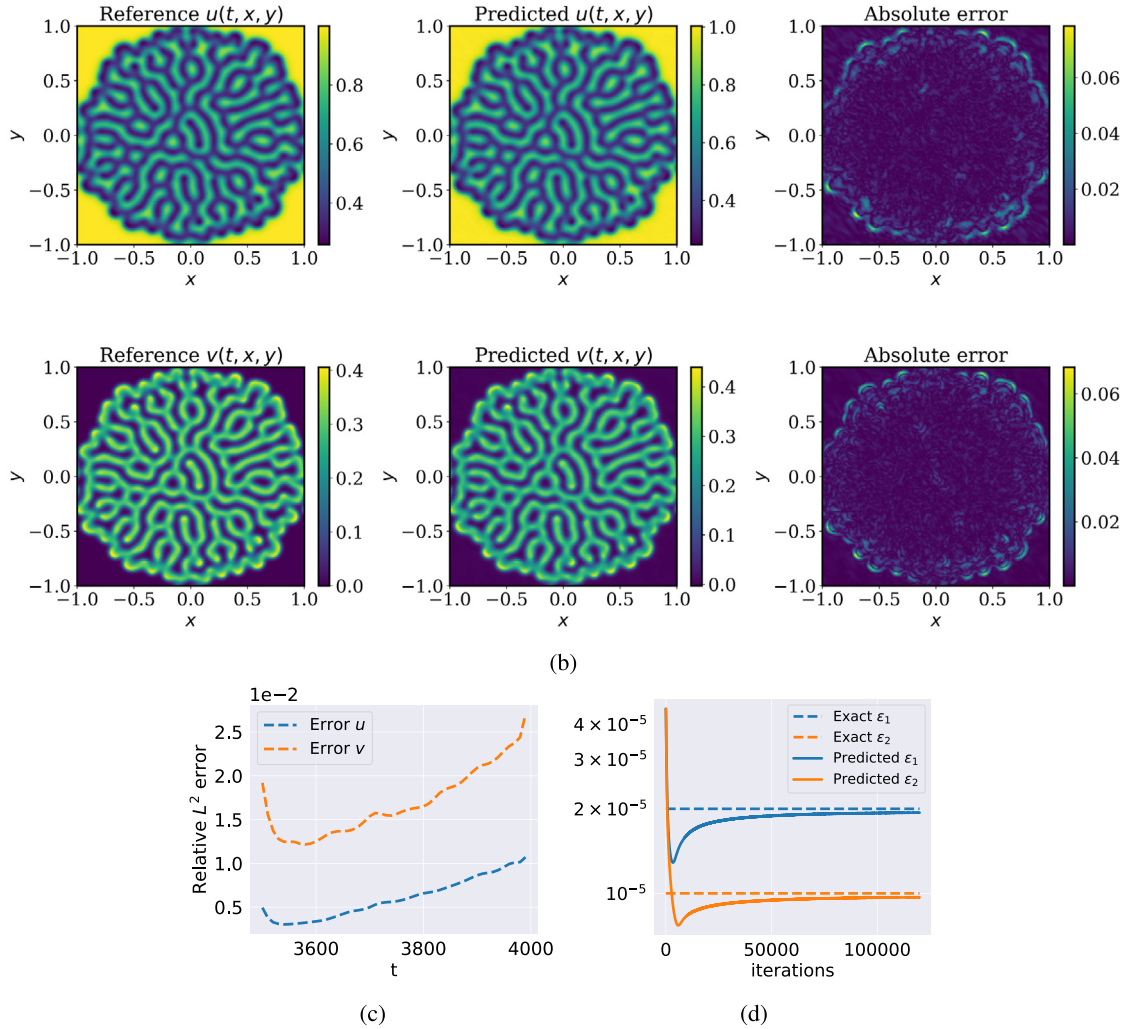
Fig. 17. *2D Gray–Scott equation:* Results obtained by training a fully-connected network (9 layers, 100 hidden units, tanh activations) with the spatio-temporal Fourier feature mappings after 120,000 iterations of gradient descent. (a) (b) Numerical estimations versus the predicted concentration fields u, v respective at the final time $t = 4000$. (c) Relative L^2 errors between the model predictions and the corresponding exact concentration fields for each snapshot $t \in [3500, 4000]$. (d) Evolution of the inferred diffusion rates ϵ_1, ϵ_2 during training.

Table 3
2D Gray–Scott equation: Exact diffusion rates versus the inferred diffusion rates after training.

Parameters	Exact	Learned	Relative L^2 error
ϵ_1	$2e{-}05$	$1.95e{-}05$	3.00%
ϵ_2	$1e{-}05$	$9.70e{-}06$	8.06%

5. Discussion

In this work, we study Fourier feature networks through the lens of their limiting neural tangent kernel, and show that Fourier feature mappings determine the frequency of the eigenvectors of the resulting NTK. This analysis sheds light into mechanisms that introduce spectral bias in the training of deep neural networks, and suggests possible avenues for overcoming this fundamental limitation. Specific to the context of physics-informed neural networks, our analysis motivates the design of two novel network architectures to tackle forward and inverse problems involving time-dependent PDEs with solutions that exhibit complex multi-scale spatio-temporal features. To gain further
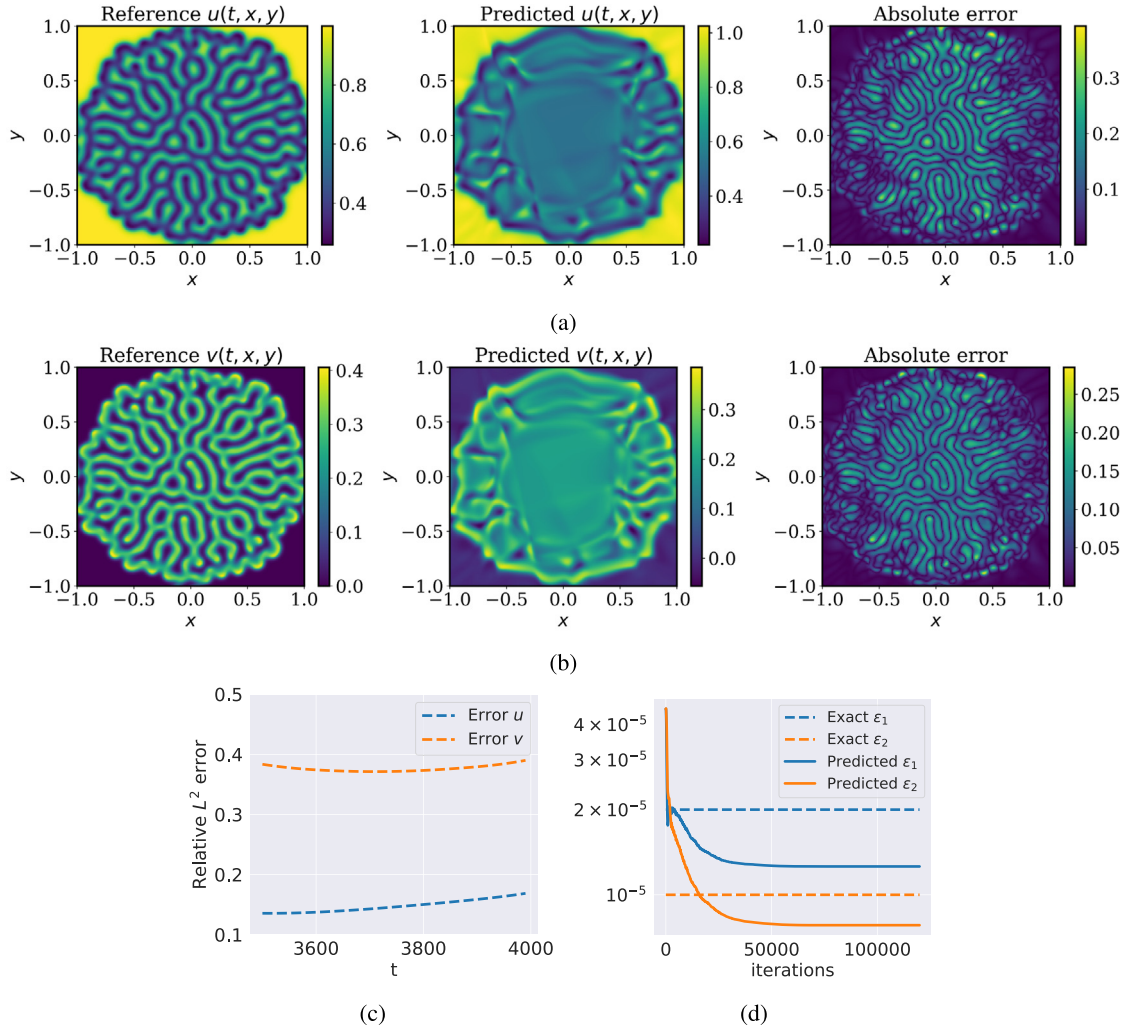
Fig. 18. *2D Gray–Scott equation:* Results obtained by training a fully-connected network (9 layers, 100 hidden units, tanh activations) after 120,000 iterations of gradient descent. (a)(b) Numerical estimations versus the predicted concentration fields u, v respective at the final time $t = 4000$. (c) Relative L^2 errors between the model predictions and the corresponding exact concentration fields for each snapshot $t \in [3500, 4000]$. (d) Evolution of the inferred diffusion rates ϵ_1, ϵ_2 during training.

insight, we propose a series of benchmarks for which conventional PINN approaches fail, and demonstrate the effectiveness of the proposed methods under these challenging settings. Taken together, the developments presented in this work provide a principled way of analyzing the performance of PINN models, and enable the design of a new generation of architectures and training algorithms that introduce significant improvements both in terms of training speed and generalization accuracy, especially for multi-scale PDEs for which current PINN models struggle.

Despite this progress, we must admit that we are still at the very early stages of tackling realistic multi-scale and multi-physics problems with PINNs. One main limitation of the proposed architectures is that we have to carefully choose the appropriate number of Fourier feature mappings and their scale, such that the frequency of the NTK eigenvectors and the target function are roughly matched to each other. In other words, the proposed architectures require some prior knowledge regarding the frequency distribution of the target PDE solution. However, this kind of information may not be accessible for some forward problems, especially for more complex dynamical systems involving the fast transitions of frequencies such as the Kuramoto–Sivashinsky equation [51,52], or the Navier–Stokes equations in the turbulent regime. In the case where the frequency distribution of the target solution is unknown, two to three sets of Fourier features corresponding to different frequency bands are generally sufficient

to obtain good results. These Fourier feature embeddings can be consecutively initialized using $\sigma = 10^n$ for $n = 0, 1, 2, \ldots$, etc. Also, one may tune σ by monitoring the loss function during training. Furthermore, this issue could be mitigated for some inverse problems where we may perform some spectral analysis on the training data to determine the appropriate number and scale of the Fourier feature mappings.

There are also many open questions worth considering as future research directions. From a theoretical standpoint, we numerically verify that the frequency of Fourier feature mappings determines the frequency of the NTK eigenvectors. Can we rigorously establish a theory for general networks? Besides, what is the behavior of the NTK eigensystem of PINNs? What is the difference between the resulting eigensystem of PINNs and conventional neural networks? From a practical standpoint, we observe that the eigenvalue distribution moves outward (see Fig. 5b) when choosing an inappropriate scale of Fourier feature mappings, which implies that the parameters of the network have to move far away from their initialization to find a good local minimum. Thus, it is natural to ask how to initialize PINNs such that the desirable local minima exist in the vicinity of the parameter initialization in the corresponding loss landscape? Moreover, can we design other useful feature embeddings that aim to handle different scenarios (e.g., shocks, boundary layers, etc.)? We believe that answering these questions not only paves a new way to better understand PINNs and their training dynamics, but also opens a new door for developing scientific machine learning algorithms with provable convergence guarantees, as needed for many critical applications in computational science and engineering.

Declaration of competing interest

Acknowledgments

Appendix A. Definition of fully-connected neural networks

A scalar-valued fully-connected neural network with L hidden layers is defined recursively as follows

$$\boldsymbol{f}^{(0)}(\boldsymbol{x}) = \boldsymbol{x} \tag{A.1}$$

$$\boldsymbol{f}^{(h)}(\boldsymbol{x}) = \frac{1}{\sqrt{d_h}} \boldsymbol{W}^{(h)} \cdot \boldsymbol{g}^{(h)} + \boldsymbol{b}^{(h)} \in \mathbb{R}^{d_{h+1}}, \tag{A.2}$$

$$\boldsymbol{g}^{(h)}(\boldsymbol{x}) = \sigma(\boldsymbol{W}^{(h-1)} \boldsymbol{f}^{(h-1)}(\boldsymbol{x}) + \boldsymbol{b}^{(h-1)}), \tag{A.3}$$

for $h = 1, \ldots, L$, where $\boldsymbol{W}^{(h)} \in \mathbb{R}^{d_{h+1} \times d_h}$ are weight matrices and $\boldsymbol{b}^{(h)} \in \mathbb{R}^{d_{h+1}}$ are bias vectors in the hth hidden layer, and $\sigma : \mathbb{R} \to \mathbb{R}$ is a coordinate-wise smooth activation function. The final output of the neural network is given by

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{f}^{(L)}(\boldsymbol{x}) = \frac{1}{\sqrt{d_L}} \boldsymbol{W}^{(L)} \cdot \boldsymbol{g}^{(L)}(\boldsymbol{x}) + \boldsymbol{b}^{(L)}, \tag{A.4}$$

where $\boldsymbol{W}^{(L)} \in \mathbb{R}^{1 \times d_L}$ and $\boldsymbol{b}^{(L)} \in \mathbb{R}$ are the weight and bias parameters of the last layer. Here, $\boldsymbol{\theta} = \{\boldsymbol{W}^{(0)}, \boldsymbol{b}^{(0)}, \ldots, \boldsymbol{W}^{(L)}, \boldsymbol{b}^{(L)}\}$ denotes all parameters of the network and are initialized as independent and identically distributed (i.i.d.) Gaussian random variables $\mathcal{N}(0, 1)$. We remark that such a parameterization is known as the "NTK parameterization" following the original work of Jacot et al. [34].

Appendix B. Proof of Lemma 3.1

Proof. Recall that $K(x, x') = \frac{1}{m} \sum_{k=1}^{m} \cos(b_k^T(x - x'))$, where $x = (x_1, \ldots, x_l, \ldots, x_d)$. By Eq. (3.11), we have

$$\frac{1}{m} \sum_{k=1}^{m} \int_C \cos(b_k^T(x - x'))g(x') \, dx' = \lambda g(x). \tag{B.1}$$

For $l = 1, 2, \ldots, d$, taking derivatives with respect to x_l gives

$$-\frac{1}{m} \sum_{k=1}^{m} b_{kl} \int_C \sin(b_k^T(x - x'))g(x') \, dx' = \lambda \frac{\partial g(x)}{\partial x_l}. \tag{B.2}$$

Again taking derivatives with respect to x_l yields

$$-\frac{1}{m} \sum_{k=1}^{m} b_{kl}^2 \int_C \cos(b_k^T(x - x'))g(x') \, dx' = \lambda \frac{\partial^2 g(x)}{\partial x_l^2}. \tag{B.3}$$

Summing over l gives

$$-\frac{1}{m} \sum_{k=1}^{m} \sum_{l=1}^{d} b_{kl}^2 \int_C \cos(b_k^T(x - x'))g(x') \, dx' = \lambda \sum_{l=1}^{d} \frac{\partial^2 g(x)}{\partial x_l^2}.$$

Note that $\|B\|_F^2 = \sum_{k,l} b_{kl}^2$ and $\Delta = \sum_l \partial_{x_l}^2$. Therefore, we have

$$-\frac{1}{m} \|B\|_F^2 \lambda g(x) = \lambda \Delta g(x). \tag{B.4}$$

When $\lambda \neq 0$, the above equation is equivalent to

$$\Delta g(x) = -\frac{1}{m} \|B\|_F^2 g(x). \tag{B.5}$$

This concludes the proof. □

Appendix C. Proof of Proposition 3.2

Proof. Suppose that $g(x)$ is an eigenfunction of the integral operator $K(x, x') = \cos(b(x - x'))$ with respect to the non-zero eigenvalue λ, i.e.

$$\int_0^1 \cos(b(x - x'))g(x') \, dx' = \lambda g(x). \tag{C.1}$$

By Lemma 3.1, first we know that $g(x)$ satisfies

$$g''(x) = -b^2 g(x). \tag{C.2}$$

Note that this is an ODE and thus $g(x)$ must have the form of

$$g(x) = C_1 \cos(bx) + C_2 \sin(bx), \tag{C.3}$$

where C_1, C_2 are some constants.

Next, we compute the corresponding eigenvalue. Substituting the expression of g into Eq. (C.1) we get

$$\int_0^1 \cos(b(x - x')) \left[C_1 \cos(bx') + C_2 \sin(bx') \right] dx' = \lambda \left[C_1 \cos(bx') + C_2 \sin(bx') \right].$$

Then,

$$LHS = \int_0^1 \cos(b(x - x')) \left[C_1 \cos(bx') + C_2 \sin(bx') \right] dx'$$

$$= \int_0^1 \cos(bx)\cos(bx') + \sin(bx)\sin(bx') \left[C_1 \cos(bx') + C_2 \sin(bx') \right] dx'$$

$$= \cos(bx) \int_0^1 \cos(bx') \left[C_1 \cos(bx') + C_2 \sin(bx') \right] dx' + \sin(bx) \int_0^1 \sin(bx') \left[C_1 \cos(bx') \right.$$
$$\left. + C_2 \sin(bx') \right] dx'$$
$$= \cos(bx)[C_1 I_1 + C_2 I_2] + \sin(bx)[C_1 I_2 + C_2 I_3],$$

where

$$I_1 = \int_0^1 \cos(bx') \cos(bx') dx' = \frac{1}{2} + \frac{\sin(2b)}{4b},$$

$$I_2 = \int_0^1 \cos(bx') \sin(bx') dx' = \frac{1 - \cos(2b)}{4b},$$

$$I_3 = \int_0^1 \sin(bx') \sin(bx') dx' = \frac{1}{2} - \frac{\sin(2b)}{4b}.$$

Since $LHS = RHS$, we have

$$[C_1 I_1 + C_2 I_2] \cos(bx) + [C_1 I_2 + C_2 I_3] \sin(bx) = \lambda \left[C_1 \cos(bx) + C_2 \sin(bx) \right], \tag{C.4}$$

which follows

$$C_1 I_1 + C_2 I_2 = \lambda C_1 \tag{C.5}$$
$$C_1 I_2 + C_2 I_3 = \lambda C_2. \tag{C.6}$$

Let $A = \begin{bmatrix} I_1 & I_2 \\ I_2 & I_3 \end{bmatrix}$ and $v = \begin{bmatrix} C_1 \\ C_2 \end{bmatrix}$. Then, the above linear system can be written as

$$Av = \lambda v.$$

This means that the eigenvalue of the kernel function $K(x, x')$ is determined by the eigenvalue of the matrix A. The characteristic polynomial is given by

$$\det(\lambda I - A) = (\lambda - I_1)(\lambda - I_3) - I_2^2 = \lambda^2 - (I_1 + I_3)\lambda + I_1 I_3 - I_2^2.$$

Note that $I_1 + I_3 = 1$, and

$$I_1 I_3 - I_2^2 = \left(\frac{1}{2} + \frac{\sin(2b)}{4b} \right) \left(\frac{1}{2} - \frac{\sin(2b)}{4b} \right) - \left(\frac{1 - \cos(2b)}{4b} \right)^2$$

$$= \frac{1}{4} - \frac{\sin^2(2b)}{16b^2} - \frac{1 - 2\cos(2b) - \cos^2(2b)}{16b^2}$$

$$= \frac{1}{4} - \frac{2 - 2\cos(2b)}{16b^2}$$

$$= \frac{1}{4} - \frac{\sin^2(b)}{4b^2}.$$

Hence we have

$$\det(\lambda I - A) = \lambda^2 - \lambda + \frac{1}{4} - \frac{\sin^2(b)}{4b^2}.$$

Then the eigenvalues are

$$\lambda_1 = \frac{1 + \frac{\sin b}{b}}{2}, \quad \lambda_2 = \frac{1 - \frac{\sin b}{b}}{2}$$

which completes the proof. $\square$

Appendix D. Proof of Proposition 3.3

Proof. By assumption (i) in Proposition 3.3, we immediately obtain

$$\begin{bmatrix} \frac{d\mathcal{B}[\boldsymbol{u}](\boldsymbol{x}_b, \boldsymbol{\theta}(t))}{dt} \\ \frac{d\mathcal{N}[\boldsymbol{u}](\boldsymbol{x}_r, \boldsymbol{\theta}(t))}{dt} \end{bmatrix} \approx - \begin{bmatrix} \boldsymbol{K}_{uu}(0) & \boldsymbol{K}_{ur}(0) \\ \boldsymbol{K}_{ru}(0) & \boldsymbol{K}_{rr}(0) \end{bmatrix} \cdot \begin{bmatrix} \mathcal{B}[\boldsymbol{u}](\boldsymbol{x}_b, \boldsymbol{\theta}(t)) - \boldsymbol{g}(\boldsymbol{x}_b) \\ \mathcal{N}[\boldsymbol{u}](\boldsymbol{x}_r, \boldsymbol{\theta}(t)) - \boldsymbol{f}(\boldsymbol{x}_r) \end{bmatrix}, \tag{D.1}$$

which implies

$$\begin{bmatrix} \mathcal{B}[\boldsymbol{u}](\boldsymbol{x}_b, \boldsymbol{\theta}(t)) \\ \mathcal{N}[\boldsymbol{u}](\boldsymbol{x}_r, \boldsymbol{\theta}(t)) \end{bmatrix} - \begin{bmatrix} \boldsymbol{g}(\boldsymbol{x}_b) \\ \boldsymbol{f}(\boldsymbol{x}_r) \end{bmatrix} \approx \left(I - e^{-\boldsymbol{K}(0)t} \right) \cdot \begin{bmatrix} \boldsymbol{g}(\boldsymbol{x}_b) \\ \boldsymbol{f}(\boldsymbol{x}_r) \end{bmatrix} - \begin{bmatrix} \boldsymbol{g}(\boldsymbol{x}_b) \\ \boldsymbol{f}(\boldsymbol{x}_r) \end{bmatrix} \tag{D.2}$$

$$\approx -e^{-\boldsymbol{K}(0)t} \cdot \begin{bmatrix} \boldsymbol{g}(\boldsymbol{x}_b) \\ \boldsymbol{f}(\boldsymbol{x}_r) \end{bmatrix}. \tag{D.3}$$

By assumption (ii) in Proposition 3.3, $\boldsymbol{K}_{uu}(0)$ $\boldsymbol{K}_{rr}(0)$ are positive definite, and there exist orthogonal matrices $\boldsymbol{Q}_u$ and $\boldsymbol{Q}_r$ such that

$$\boldsymbol{K}_{uu}(0) = \boldsymbol{Q}_u^T \Lambda_u \boldsymbol{Q}_u^T, \tag{D.4}$$
$$\boldsymbol{K}_{rr}(0) = \boldsymbol{Q}_r^T \Lambda_r \boldsymbol{Q}_r^T, \tag{D.5}$$

where Λ_u and Λ_r are diagonal matrices whose entries are eigenvalues of $\boldsymbol{K}_{uu}(0)$ and $\boldsymbol{K}_{rr}(0)$, respectively. We remark that Λ_u and Λ_r are invertible since all eigenvalues are strictly positive.

Now let $\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{Q}_u & 0 \\ 0 & \boldsymbol{Q}_r \end{bmatrix}$ to obtain

$$\boldsymbol{Q}^T \boldsymbol{K}(0) \boldsymbol{Q} = \begin{bmatrix} \boldsymbol{Q}_u^T & 0 \\ 0 & \boldsymbol{Q}_r^T \end{bmatrix} \begin{bmatrix} \boldsymbol{K}_{rr}(0) & \boldsymbol{K}_{ur}(0) \\ \boldsymbol{K}_{ur}^T(0) & \boldsymbol{K}_{rr}(0) \end{bmatrix} \begin{bmatrix} \boldsymbol{Q}_u & 0 \\ 0 & \boldsymbol{Q}_r \end{bmatrix} \tag{D.6}$$

$$= \begin{bmatrix} \Lambda_u & \boldsymbol{Q}_u^T \boldsymbol{K}_{ur}(0) \boldsymbol{Q}_r \\ \boldsymbol{Q}_r^T \boldsymbol{K}_{ru}(0) \boldsymbol{Q}_u & \Lambda_r \end{bmatrix} := \tilde{\Lambda}. \tag{D.7}$$

Furthermore, letting $\boldsymbol{B} = \boldsymbol{Q}_r^T \boldsymbol{K}_{ru}(0) \boldsymbol{Q}_u$ and $\boldsymbol{P} = \begin{bmatrix} \boldsymbol{I} & 0 \\ -\boldsymbol{B}\Lambda_u^{-1} & \boldsymbol{I} \end{bmatrix}$, gives

$$\tilde{\Lambda} = \boldsymbol{P}^T \begin{bmatrix} \Lambda_u & 0 \\ 0 & \Lambda_r - \boldsymbol{B}^T \Lambda_u^{-1} \boldsymbol{B} \end{bmatrix} \boldsymbol{P} = \boldsymbol{P}^T \Lambda \boldsymbol{P}. \tag{D.8}$$

Therefore, we obtain

$$\boldsymbol{Q}^T \left(\begin{bmatrix} \mathcal{B}[\boldsymbol{u}](\boldsymbol{x}_b, \boldsymbol{\theta}(t)) \\ \mathcal{N}[\boldsymbol{u}](\boldsymbol{x}_r, \boldsymbol{\theta}(t)) \end{bmatrix} - \begin{bmatrix} \boldsymbol{g}(\boldsymbol{x}_b) \\ \boldsymbol{f}(\boldsymbol{x}_r) \end{bmatrix} \right) \approx e^{-\boldsymbol{P}^T \Lambda \boldsymbol{P} t} \boldsymbol{Q}^T \begin{bmatrix} \boldsymbol{g}(\boldsymbol{x}_b) \\ \boldsymbol{f}(\boldsymbol{x}_r) \end{bmatrix}. \tag{D.9}$$

This concludes the proof. □

References

[1] Atılım Güneş Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, Jeffrey Mark Siskind, Automatic differentiation in machine learning: a survey, J. Mach. Learn. Res. 18 (1) (2017) 5595–5637.
[2] Dimitris C. Psichogios, Lyle H. Ungar, A hybrid neural network-first principles approach to process modeling, AIChE J. 38 (10) (1992) 1499–1511.
[3] Isaac E. Lagaris, Aristidis Likas, Dimitrios I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Netw. 9 (5) (1998) 987–1000.
[4] Maziar Raissi, Paris Perdikaris, George E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.
[5] Yeonjong Shin, Jérôme Darbon, George Em. Karniadakis, On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type PDEs, 2020.
[6] Sifan Wang, Xinling Yu, Paris Perdikaris, When and why PINNs fail to train: A neural tangent kernel perspective, 2020, arXiv preprint arXiv:2007.14527.
[7] Tao Luo, Haizhao Yang, Two-layer neural networks for partial differential equations: Optimization and generalization theory, 2020, arXiv preprint arXiv:2006.15733.
[8] Yeonjong Shin, Zhongqiang Zhang, George Em. Karniadakis, Error estimates of residual minimization using neural networks for linear PDEs, 2020, arXiv preprint arXiv:2010.08019.
[9] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, Anima Anandkumar, Fourier neural operator for parametric partial differential equations, 2020, arXiv preprint arXiv:2010.08895.
[10] Yinhao Zhu, Nicholas Zabaras, Phaedon-Stelios Koutsourelakis, Paris Perdikaris, Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data, J. Comput. Phys. 394 (2019) 56–81.
[11] Luning Sun, Han Gao, Shaowu Pan, Jian-Xun Wang, Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data, Comput. Methods Appl. Mech. Engrg. 361 (2020) 112732.
[12] Maziar Raissi, Alireza Yazdani, George Em. Karniadakis, Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations, Science 367 (6481) (2020) 1026–1030.

[13] Xiaowei Jin, Shengze Cai, Hui Li, George Em. Karniadakis, Nsfnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations, 2020, arXiv preprint arXiv:2003.06496.

[14] Brandon Reyes, Amanda A. Howard, Paris Perdikaris, Alexandre M. Tartakovsky, Learning unknown physics of non-Newtonian fluids, 2020, arXiv preprint arXiv:2009.01658.

[15] Francisco Sahli Costabal, Yibo Yang, Paris Perdikaris, Daniel E. Hurtado, Ellen Kuhl, Physics-informed neural networks for cardiac activation mapping, Front. Phys. 8 (2020) 42.

[16] Georgios Kissas, Yibo Yang, Eileen Hwuang, Walter R. Witschey, John A. Detre, Paris Perdikaris, Machine learning in cardiovascular flows modeling: Predicting arterial blood pressure from non-invasive 4D flow MRI data using physics-informed neural networks, Comput. Methods Appl. Mech. Engrg. 358 (2020) 112623.

[17] Alireza Yazdani, Lu Lu, Maziar Raissi, George Em. Karniadakis, Systems biology informed deep learning for inferring parameters and hidden dynamics, PLoS Comput. Biol. 16 (11) (2020) e1007575.

[18] Zhiwei Fang, Justin Zhan, Deep physical informed neural networks for metamaterial design, IEEE Access 8 (2019) 24506–24513.

[19] Yuyao Chen, Lu Lu, George Em. Karniadakis, Luca Dal Negro, Physics-informed neural networks for inverse problems in nano-optics and metamaterials, Opt. Express 28 (8) (2020) 11618–11633.

[20] Sifan Wang, Paris Perdikaris, Deep learning of free boundary and Stefan problems, 2020, arXiv preprint arXiv:2006.05311.

[21] Liu Yang, Xuhui Meng, George Em. Karniadakis, B-pinns: Bayesian physics-informed neural networks for forward and inverse pde problems with noisy data, 2020, arXiv preprint arXiv:2003.06097.

[22] Yibo Yang, Mohamed Aziz Bhouri, Paris Perdikaris, Bayesian differential programming for robust systems identification under uncertainty, Proc. R. Soc. A: Math. Phys. Eng. Sci. 476 (2243) (2020) 20200290.

[23] Jiequn Han, Arnulf Jentzen, E. Weinan, Solving high-dimensional partial differential equations using deep learning, Proc. Natl. Acad. Sci. 115 (34) (2018) 8505–8510.

[24] Maziar Raissi, Forward-backward stochastic neural networks: Deep learning of high-dimensional partial differential equations, 2018, arXiv preprint arXiv:1804.07010.

[25] Sharmila Karumuri, Rohit Tripathy, Ilias Bilionis, Jitesh Panchal, Simulator-free solution of high-dimensional stochastic elliptic partial differential equations using deep neural networks, J. Comput. Phys. 404 (2020) 109120.

[26] Yibo Yang, Paris Perdikaris, Adversarial uncertainty quantification in physics-informed neural networks, J. Comput. Phys. 394 (2019) 136–152.

[27] Lu Lu, Pengzhan Jin, George Em. Karniadakis, DeepONet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators, 2019, arXiv preprint arXiv:1910.03193.

[28] Han Gao, Luning Sun, Jian-Xun Wang, PhyGeoNet: Physics-informed geometry-adaptive convolutional neural networks for solving parametric PDEs on irregular domain, 2020, arXiv preprint arXiv:2004.13145.

[29] Sifan Wang, Yujun Teng, Paris Perdikaris, Understanding and mitigating gradient pathologies in physics-informed neural networks, 2020, arXiv preprint arXiv:2001.04536.

[30] Ravi G. Patel, Indu Manickam, Nathaniel A. Trask, Mitchell A. Wood, Myoungkyu Lee, Ignacio Tomas, Eric C. Cyr, Thermodynamically consistent physics-informed neural networks for hyperbolic systems, 2020, arXiv preprint arXiv:2012.05343.

[31] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, Aaron Courville, On the spectral bias of neural networks, in: International Conference on Machine Learning, 2019, pp. 5301–5310.

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

[33] Basri Ronen, David Jacobs, Yoni Kasten, Shira Kritchman, The convergence rate of neural networks for learned functions of different frequencies, in: Advances in Neural Information Processing Systems, 2019, pp. 4761–4771.

[34] Arthur Jacot, Franck Gabriel, Clément Hongler, Neural tangent kernel: Convergence and generalization in neural networks, in: Advances in Neural Information Processing Systems, 2018, pp. 8571–8580.

[35] Sanjeev Arora, Simon S. Du, Wei Hu, Zhiyuan Li, Russ R. Salakhutdinov, Ruosong Wang, On exact computation with an infinitely wide neural net, in: Advances in Neural Information Processing Systems, 2019, pp. 8141–8150.

[36] Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, Jeffrey Pennington, Wide neural networks of any depth evolve as linear models under gradient descent, in: Advances in Neural Information Processing Systems, 2019, pp. 8572–8583.

[37] Francis Williams, Matthew Trager, Claudio Silva, Daniele Panozzo, Denis Zorin, Joan Bruna, Gradient dynamics of shallow univariate relu networks, 2019, arXiv preprint arXiv:1906.07842.

[38] Bo Wang, Wenzhong Zhang, Wei Cai, Multi-scale deep neural network (MscaleDNN) methods for oscillatory stokes flows in complex domains, 2020, arXiv preprint arXiv:2009.12729.

[39] Xi-An Li, Zhi-Qin John Xu, Lei Zhang, A DNN-based algorithm for multi-scale elliptic problems, 2020, arXiv preprint arXiv:2009.14597.

[40] Ziqi Liu, Wei Cai, Zhi-Qin John Xu, Multi-scale deep neural network (MscaleDNN) for solving Poisson-Boltzmann equation in complex domains, 2020, arXiv preprint arXiv:2007.11207.

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

[42] Lu Lu, Raphael Pestourie, Wenjie Yao, Zhicheng Wang, Francesc Verdugo, Steven G. Johnson, Physics-informed neural networks with hard constraints for inverse design, 2021, arXiv preprint arXiv:2102.04626.

[43] Diederik P. Kingma, Jimmy Ba, Adam: A method for stochastic optimization, 2014, arXiv preprint arXiv:1412.6980.

[44] John Shawe-Taylor, Christopher K.I. Williams, Nello Cristianini, Jaz Kandola, On the eigenspectrum of the gram matrix and the generalization error of kernel-PCA, IEEE Trans. Inform. Theory 51 (7) (2005) 2510–2522.

[45] L.C. Evans and American Mathematical Society, Partial Differential Equations, in: Graduate Studies in Mathematics, American Mathematical Society, 1998.

[46] Jan S. Hesthaven, Sigal Gottlieb, David Gottlieb, Spectral Methods for Time-Dependent Problems, Vol. 21, Cambridge University Press, 2007.

[47] Xavier Glorot, Yoshua Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Proceedings of the thirteenth international conference on artificial intelligence and statistics, 2010, pp. 249–256.

[48] Peter Gray, Stephen K. Scott, Chemical oscillations and instabilities: non-linear chemical kinetics, 1990.

[49] Tobin A. Driscoll, Nicholas Hale, Lloyd N. Trefethen, Chebfun guide, 2014.

[50] Steven M. Cox, Paul C. Matthews, Exponential time differencing for stiff systems, J. Comput. Phys. 176 (2) (2002) 430–455.

[51] G.I. Sivashinsky, Nonlinear analysis of hydrodynamic instability in laminar flames—i. derivation of basic equations, AcAau 4 (11) (1977) 1177–1206.

[52] Yoshiki Kuramoto, Diffusion-induced chaos in reaction systems, Progr. Theoret. Phys. Suppl. 64 (1978) 346–367.

[53] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, Qiao Zhang, JAX: composable transformations of Python+NumPy programs, 2018.

[54] John D. Hunter, Matplotlib: A 2d graphics environment, IEEE Ann. Hist. Comput. 9 (03) (2007) 90–95.

[55] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, et al., Array programming with numpy, Nature 585 (7825) (2020) 357–362.