DeepONet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators

Lu Lu[1], Pengzhan Jin[2], and George Em Karniadakis[1]

[1]Division of Applied Mathematics, Brown University, Providence, RI 02912, USA
[2]LSEC, ICMSEC, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, China

Abstract

While it is widely known that neural networks are universal approximators of continuous functions, a less known and perhaps more powerful result is that a neural network with a single hidden layer can approximate accurately any nonlinear continuous operator [5]. This universal approximation theorem is suggestive of the potential application of neural networks in learning nonlinear operators from data. However, the theorem guarantees only a small approximation error for a sufficient large network, and does not consider the important optimization and generalization errors. To realize this theorem in practice, we propose deep operator networks (DeepONets) to learn operators accurately and efficiently from a relatively small dataset. A DeepONet consists of two sub-networks, one for encoding the input function at a fixed number of sensors $x_i, i = 1, \ldots, m$ (branch net), and another for encoding the locations for the output functions (trunk net). We perform systematic simulations for identifying two types of operators, i.e., dynamic systems and partial differential equations, and demonstrate that DeepONet significantly reduces the generalization error compared to the fully-connected networks. We also derive theoretically the dependence of the approximation error in terms of the number of sensors (where the input function is defined) as well as the input function type, and we verify the theorem with computational results. More importantly, we observe high-order error convergence in our computational tests, namely polynomial rates (from half order to fourth order) and even exponential convergence with respect to the training dataset size.

1   Introduction

The universal approximation theorem states that neural networks can be used to approximate any continuous function to arbitrary accuracy if no constraint is placed on the width and depth of the hidden layers [7, 11]. However, another approximation result, which is yet more surprising and has not been appreciated so far, states that a neural network with a single hidden layer can approximate accurately any nonlinear continuous *functional* (a mapping from a space of functions into the real numbers) [3, 18, 25] or (nonlinear) operator (a mapping from a space of functions into another space of functions) [5, 4].

Before reviewing the approximation theorem for operators, we introduce some notation, which will be used through this paper. Let G be an operator taking an input function u, and then $G(u)$ is the corresponding output function. For any point y in the domain of $G(u)$, the output $G(u)(y)$ is a real number. Hence, the network takes inputs composed of two parts: u and y, and outputs $G(u)(y)$ (Fig. 1A). Although our goal is to learn operators, which take a function as the input, we have to represent the input functions discretely, so that network approximations can be applied. A straightforward and simple way, in practice, is to employ the

function values at sufficient but finite many locations $\{x_1, x_2, \ldots, x_m\}$; we call these locations as "sensors" (Fig. 1A). Next, we state the following theorem due to Chen & Chen [5], see appendix for more details.

Theorem 1 (**Universal Approximation Theorem for Operator**). *Suppose that σ is a continuous non-polynomial function, X is a Banach Space, $K_1 \subset X$, $K_2 \subset \mathbb{R}^d$ are two compact sets in X and $\mathbb{R}^d$, respectively, V is a compact set in $C(K_1)$, G is a nonlinear continuous operator, which maps V into $C(K_2)$. Then for any $\epsilon > 0$, there are positive integers n, p, m, constants $c_i^k, \xi_{ij}^k, \theta_i^k, \zeta_k \in \mathbb{R}$, $w_k \in \mathbb{R}^d$, $x_j \in K_1$, $i = 1, \ldots, n$, $k = 1, \ldots, p$, $j = 1, \ldots, m$, such that*

$$\left| G(u)(y) - \sum_{k=1}^{p} \underbrace{\sum_{i=1}^{n} c_i^k \sigma \left(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k \right)}_{branch} \underbrace{\sigma(w_k \cdot y + \zeta_k)}_{trunk} \right| < \epsilon \tag{1}$$

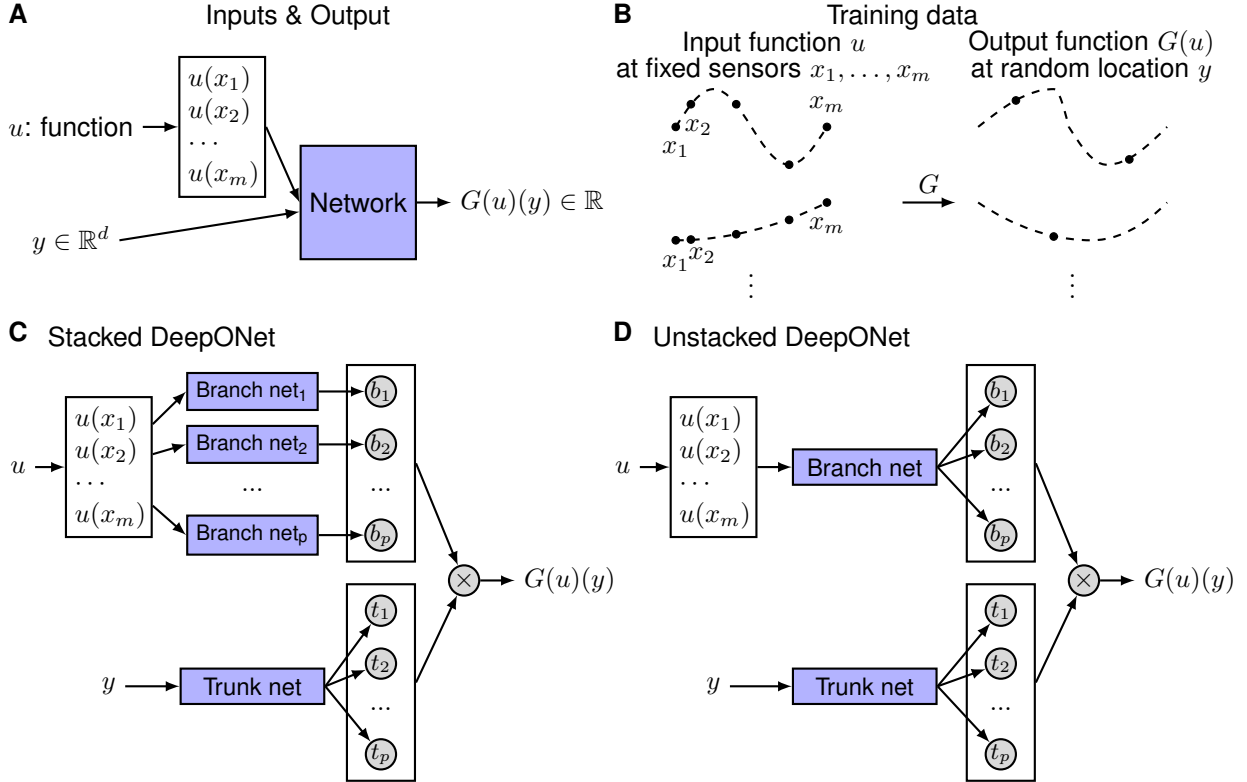*holds for all $u \in V$ and $y \in K_2$.*

Figure 1: **Illustrations of the problem setup and architectures of DeepONets.** (**A**) The network to learn an operator $G : u \mapsto G(u)$ takes two inputs $[u(x_1), u(x_2), \ldots, u(x_m)]$ and y. (**B**) Illustration of the training data. For each input function u, we require that we have the same number of evaluations at the same scattered sensors $x_1, x_2, \ldots, x_m$. However, we do not enforce any constraints on the number or locations for the evaluation of output functions. (**C**) The stacked DeepONet in Theorem 1 has one trunk network and p stacked branch networks. (**D**) The unstacked DeepONet has one trunk network and one branch network.

This approximation theorem indicates the potential application of neural networks to learn nonlinear operators from data, i.e., similar to what the deep learning community is currently doing, that is learning

functions from data. However, this theorem does not inform us how to learn operators effectively. Considering the classical image classification task as an example, the universal approximation theorem of neural networks for functions [7, 11] shows that fully-connected neural networks (FNNs) are capable to approximate the ground-truth classification function accurately, but in practice the performance of FNNs is far from networks with specific architectures, such as the widely-used convolutional neural networks (CNN) [14] or the more recent capsule neural network (CapsNet) [27]. The performance gap lies in the fact that the accuracy of NNs can be characterized by dividing the total error into three main types: approximation, optimization, and generalization [1, 17, 13, 16]. The universal approximation theorems only guarantee a small approximation error for a sufficiently large network, but they do not consider the optimization error and generalization error at all, which are equally important and often dominant contributions to the total error in practice. Useful networks should be easy to train, i.e., exhibit small optimization error, and generalize well to unseen data, i.e., exhibit small generalization error.

To demonstrate the capability and effectiveness of learning nonlinear operators by neural networks, we setup the problem as general as possible by using the weakest possible constraints on the sensors and training dataset. Specifically, the only condition required is that the sensor locations $\{x_1, x_2, \ldots, x_m\}$ are the same but not necessarily on a lattice for all input functions u, while we do not enforce any constraints on the output locations y (Fig. 1B). To learn operators accurately and efficiently, we propose a specific network architecture, the deep operator network (DeepONet), to achieve smaller total error. We will demonsrate that the DeepONet significantly improves generalization based on a design of two sub-networks, the branch net for the input function and the trunk-net for the location to evaluate the output function.

We consider two types of operators, i.e., dynamic systems (e.g., in the form of ordinary differential equations, ODEs) and partial differential equations (PDEs). Dynamic systems are typically described by difference or differential equations, and identification of a nonlinear dynamic plant is a major concern in control theory. Some works [22, 33] used neural networks to identify dynamic systems, but they only considered the dynamic systems described by difference equations. Some other works [20, 24, 23, 9] predict the evolution of a specific dynamic system rather than identifying the system behavior for new unseen input signals. The network architectures they employed includes FNNs [24], recurrent neural networks (RNNs) [20], reservoir computing [20], residual networks [23], autoencoder [9], neural ordinary differential equations [6], and neural jump stochastic differential equations [12]. For identifying PDEs, some works treat the input and output function as an image, and then use CNNs to learn the image-to-image mapping [30, 34], but this approach can be only applied to the particular type of problems, where the sensors of the input function u are distributed on a equispaced grid, and the training data must include all the $G(u)(y)$ values with y also on a equispaced grid. In another approach without this restriction, PDEs are parametrized by unknown coefficients, and then only the coefficient values are identified from data [2, 26, 31, 21, 15]. Alternatively, a generalized CNN based on generalized moving least squares [28] can be used for unstructured data, but it can only approximate local operators and is not able to learn other operators like an integral operator.

The paper is organized as follows. In Section 2, we present two network architectures of DeepONet: the stacked DeepONet and the unstacked DeepONet, and then introduce the data generation procedure. In Section 3, we present a theoretical analysis on the number of sensors required to represent the input function accurately for approximating ODE operators. In Section 4, we test the performance of FNN, stacked DeepONet, and unstacked DeepONet for different examples, and demonstrate the accuracy and convergence rates of unstacked DeepONet. Finally, we conclude the paper in Section 5.

2 Methodology

2.1 Deep operator networks (DeepONets)

We focus on learning operators in a more general setting, where the only requirement for the training dataset is the consistency of the sensors $\{x_1, x_2, \ldots, x_m\}$ for input functions. In this general setting, the network inputs consist of two separate components: $[u(x_1), u(x_2), \ldots, u(x_m)]^T$ and y (Fig. 1A), and the goal is to achieve good performance by designing the network architecture. One straightforward solution is to directly employ a classical network, such as FNN, CNN or RNN, and concatenate two inputs together as the network input, i.e., $[u(x_1), u(x_2), \ldots, u(x_m), y]^T$. However, the input does not have any specific structure, and thus it is not meaningful to choose networks like CNN or RNN. Here we use FNN as the baseline model.

In high dimensional problems, y is a vector with d components, so the dimension of y does not match the dimension of $u(x_i)$ for $i = 1, 2, \ldots, m$ any more. This also prevents us from treating $u(x_i)$ and y equally, and thus at least two sub-networks are needed to handle $[u(x_1), u(x_2), \ldots, u(x_m)]^T$ and y separately. Although the universal approximation theorem (Theorem 1) does not have any guarantee on the total error, it still provides us a network structure in Eq. 1. Theorem 1 only considers a shallow network with one hidden layer, so we extend it to deep networks, which have more expressivity than shallow ones. The architecture we propose is shown in Fig. 1C, and the details are as follows. First there is a "trunk" network, which takes y as the input and outputs $[t_1, t_2, \ldots, t_p]^T \in \mathbb{R}^p$. In addition to the trunk network, there are p "branch" networks, and each of them takes $[u(x_1), u(x_2), \ldots, u(x_m)]^T$ as the input and outputs a scalar $b_k \in \mathbb{R}$ for $k = 1, 2, \ldots, p$. We merge them together as in Eq. 1:

$$G(u)(y) \approx \sum_{k=1}^{p} b_k t_k.$$

We note that the trunk network also applies activation functions in the last layer, i.e., $t_k = \sigma(\cdot)$ for $k = 1, 2, \ldots, p$, and thus this trunk-branch network can also be seen as a trunk network with each weight in the last layer parameterized by another branch network instead of the classical single variable. We also note that in Eq. 1 the last layer of each b_k branch network does not have bias. Although bias is not necessary in Theorem 1, adding bias may increase the performance by reducing the generalization error. In addition to adding bias to the branch networks, we may also add a bias $b_0 \in \mathbb{R}$ in the last stage:

$$G(u)(y) \approx \sum_{k=1}^{p} b_k t_k + b_0. \tag{2}$$

In practice, p is at least of the order of 10, and using lots of branch networks is computationally and memory expensive. Hence, we merge all the branch networks into one single branch network (Fig. 1D), i.e., a single branch network outputs a vector $[b_1, b_2, \ldots, b_p]^T \in \mathbb{R}^p$. In the first DeepONet (Fig. 1C), there are p branch networks stacked parallel, so we name it as "stacked DeepONet", while we refer to the second DeepONet (Fig. 1D) as "unstacked DeepONet". All versions of DeepONets are implemented in DeepXDE [15], a user-friendly Python library designed for scientific machine learning (`https://github.com/lululxvi/deepxde`).

DeepONet is a high-level network architecture without defining the architectures of its inner trunk and branch networks. To demonstrate the capability and good performance of DeepONet alone, we choose the simplest FNN as the architectures of the sub-networks in this study. It is possible that using convolutional layers we could further improve accuracy. However, convolutional layers usually work for square domains with $\{x_1, x_2, \ldots, x_m\}$ on a equispaced grid, so as alternative and for a more general setting we may use the "attention" mechanism [29].

Embodying some prior knowledge into neural network architectures usually induces good generalization. This inductive bias has been reflected in many networks, such as CNN for images and RNN for sequential data. The success of DeepONet even using FNN as its sub-networks is also due to its strong inductive bias. The output $G(u)(y)$ has two independent inputs u and y, and thus using the trunk and branch networks explicitly is consistent with this prior knowledge. More broadly, $G(u)(y)$ can be viewed as a function of y conditioning on u. Finding an effective way to represent the conditioning input is still an open question, and different approaches have been proposed, such as feature-wise transformations [8].

2.2 Data generation

The input signal $u(x)$ of the process plays an important role in system identification. Clearly, the input signal is the only possibility to influence the process in order to gather information about its response, and the quality of the identification signal determines an upper bound on the accuracy that in the best case can be achieved by any model. In this study, we mainly consider two function spaces: Gaussian random field (GRF) and orthogonal (Chebyshev) polynomials.

We use the mean-zero GRF:

$$u \sim \mathcal{G}(0, k_l(x_1, x_2)),$$

where the covariance kernel $k_l(x_1, x_2) = \exp(-\|x_1 - x_2\|^2/2l^2)$ is the radial-basis function (RBF) kernel with a length-scale parameter $l > 0$. The length-scale l determines the smoothness of the sampled function, and larger l leads to smoother u.

Let $M > 0$ and T_i are Chebyshev polynomials of the first kind. We define the orthogonal polynomials of degree N as:

$$V_{\text{poly}} = \left\{ \sum_{i=0}^{N-1} a_i T_i(x) : |a_i| \le M \right\}.$$

We generate the dataset from V_{poly} by randomly sampling a_i from $[-M, M]$ to get a sample of u.

After sampling u from the chosen function spaces, we solve the ODE systems by Runge-Kutta (4, 5) and PDEs by a second-order finite difference method to obtain the reference solutions. We note that one data point is a triplet $(u, y, G(u)(y))$, and thus one specific input u may appear in multiple data points with different values of y. For example, a dataset of size 10000 may only be generated from 100 u trajectories, and each evaluates $G(u)(y)$ for 100 y locations.

3 Number of sensors for identifying nonlinear dynamic systems

In this section, we investigate how many sensor points we need to achieve accuracy ε for identifying nonlinear dynamic systems. Suppose that the dynamic system is subject to the following ODE system:

$$\begin{cases} \frac{d}{dx} \boldsymbol{s}(x) = \boldsymbol{g}(\boldsymbol{s}(x), u(x), x) \\ \boldsymbol{s}(a) = \boldsymbol{s_0} \end{cases}, \tag{3}$$

where $u \in V$ (a compact subset of $C[a, b]$) is the input signal, and $\boldsymbol{s} : [a, b] \to \mathbb{R}^K$ is the solution of system (3) serving as the output signal.

Let G be the operator mapping the input u to the output $\boldsymbol{s}$, i.e., Gu satisfies

$$(Gu)(x) = \boldsymbol{s_0} + \int_a^x \boldsymbol{g}((Gu)(t), u(t), t)dt.$$

Now, we choose uniformly $m + 1$ points $x_j = a + j(b-a)/m, j = 0, 1, \cdots, m$ from $[a, b]$, and define the function $u_m(x)$ as follows:

$$u_m(x) = u(x_j) + \frac{u(x_{j+1}) - u(x_j)}{x_{j+1} - x_j}(x - x_j), \ x_j \leq x \leq x_{j+1}, \ j = 0, 1, \cdots, m - 1.$$

Denote the operator mapping u to u_m by $\mathcal{L}_m$, and let $U_m = \{\mathcal{L}_m(u)|u \in V\}$, which is obviously a compact subset of $C[a, b]$ since V is compact and continuous operator $\mathcal{L}_m$ keeps the compactness. Naturally, $W_m := V \cup U_m$ as the union of two compact sets is also compact. Then, set $W := \bigcup_{i=1}^{\infty} W_i$, and Lemma 7 points out that W is still a compact set. Since G is a continuous operator, $G(W)$ is compact in $C([a, b]; \mathbb{R}^K)$. The subsequent discussions are mainly within W and $G(W)$. For convenience of analysis, we assume that $\boldsymbol{g}(\boldsymbol{s}, u, x)$ satisfies the Lipschitz condition with respect to $\boldsymbol{s}$ and u on $G(W) \times W$, i.e., there is a constant $c > 0$ such that

$$\begin{aligned} \|\boldsymbol{g}(\boldsymbol{s_1}, u, x) - \boldsymbol{g}(\boldsymbol{s_2}, u, x)\|_2 &\leq c\|\boldsymbol{s_1} - \boldsymbol{s_2}\|_2 \\ \|\boldsymbol{g}(\boldsymbol{s}, u_1, x) - \boldsymbol{g}(\boldsymbol{s}, u_2, x)\|_2 &\leq c|u_1 - u_2| \end{aligned}.$$

Note that this condition is easy to achieve, for instance, as long as $\boldsymbol{g}$ is differentiable with respect to $\boldsymbol{s}$ and u on $G(W) \times W$.

For $u \in V, u_m \in U_m$, there exists a constant $\kappa(m, V)$ depending on m and compact space V, such that

$$\max_{x \in [a,b]} |u(x) - u_m(x)| \leq \kappa(m, V), \quad \kappa(m, V) \to 0 \ as \ m \to \infty. \tag{4}$$

When V is GRF with RBF kernel, we have $\kappa(m, V) \sim \frac{1}{m^2 l^2}$, see Appendix C for the proof. Based on the these concepts, we have the following theorem.

Theorem 2. *Suppose that m is a positive integer making $c(b-a)\kappa(m, V)e^{c(b-a)}$ less than ε, then for any $d \in [a, b]$, there exist $\mathcal{W}_1 \in \mathbb{R}^{n \times (m+1)}, b_1 \in \mathbb{R}^{m+1}, \mathcal{W}_2 \in \mathbb{R}^{K \times n}, b_2 \in \mathbb{R}^K$, such that*

$$\|(Gu)(d) - (\mathcal{W}_2 \cdot \sigma(\mathcal{W}_1 \cdot [u(x_0) \quad \cdots \quad u(x_m)]^T + b_1) + b_2)\|_2 < \varepsilon$$

holds for all $u \in V$.

Proof. The proof can be found in Appendix B. □

4 Simulation results

In this section, we first show that DeepONets have better performance than FNNs due to the smaller generalization error even in the easiest linear problem, and then demonstrate the capability of DeepONets for three nonlinear ODE and PDE problems. In all problems, we use the Adam optimizer with learning rate 0.001, and the number of iterations is chosen to guarantee the training is convergent. The other parameters and network sizes are listed in Tables 1 and 2, unless otherwise stated. The codes of all examples are published in GitHub (`https://github.com/lululxvi/deepxde`).

4.1 A simple 1D dynamic system

A 1D dynamic system is described by

$$\frac{ds(x)}{dx} = g(s(x), u(x), x), \quad x \in [0, 1],$$

with an initial condition $s(0) = 0$. Our goal is to predict $s(x)$ over the whole domain $[0, 1]$ for any $u(x)$.

Table 1: **Default parameters for each problem, unless otherwise stated.** We note that one data point is a triplet $(u, y, G(u)(y))$, and thus one specific input u may generate multiple data points with different values of y.

Case	u space	# Sensors m	# Training	# Test	# Iterations	Other parameters
4.1.1	GRF ($l = 0.2$)	100	10000	100000	50000	
4.1.2	GRF ($l = 0.2$)	100	10000	100000	100000	
4.2	GRF ($l = 0.2$)	100	10000	100000	100000	$k = 1, T = 1$
4.3	GRF ($l = 0.2$)	100		1000000	500000	

Table 2: **DeepONet size for each problem, unless otherwise stated.**

Case	Network type	Trunk depth	Trunk width	Branch depth	Branch width
4.1	Stacked/Unstacked	3	40	2	40
4.2	Unstacked	3	40	2	40
4.3	Unstacked	3	100	2	100

4.1.1 Linear case: $g(s(x), u(x), x) = u(x)$

We first consider a linear problem by choosing $g(s(x), u(x), x) = u(x)$, which is equivalent to learning the antiderivative operator

$$G : u(x) \mapsto s(x) = \int_0^x u(\tau)d\tau.$$

As the baseline, we train FNNs to learn the antiderivative operator. To obtain the best performance of FNNs, we grid search the three hyperparameters: depth from 2 to 4, width from 10 to 2560, and learning rate from 0.0001 to 0.01. The mean squared error (MSE) of the test dataset with learning rate 0.01, 0.001, and 0.0001 are shown in Fig. 2. Although we only choose depth up to 4, the results show that increasing the depth further does not improve the error. Among all these hyperparameters, the smallest test error $\sim 10^{-4}$ is obtained for the network with depth 2, width 2560, and learning rate 0.001. We observe that when the network is small, the training error is large and the generalization error (the difference between test error and training error) is small, due to small expressivity. When the network size increases, the training error decreases, but the generalization error increases. We note that FNN has not reached the overfitting region, where the test error increases.

Compared to FNNs, DeepONets have much smaller generalization error and thus smaller test error. Here we do not aim to find the best hyperparameters, and only test the performance of the two DeepONets listed in Table 2. The training trajectory of an unstacked DeepONet with bias is shown in Fig. 3A, and the generalization error is negligible. We observe that for both stacked and unstacked DeepONets, adding bias to branch networks and Eq. (2) reduces both training and test errors (Fig. 3B); DeepONets with bias also have smaller uncertainty, i.e., more stable for training from random initialization (Fig. 3B). Compared to stacked DeepONets, although unstacked DeepONets have larger training error, the test error is smaller, due to the smaller generalization error. Therefore, unstacked DeepONets with bias achieve the best performance. In addition, unstacked DeepONets have fewer number of parameters than stacked DeepONets, and thus can be trained faster using much less memory.
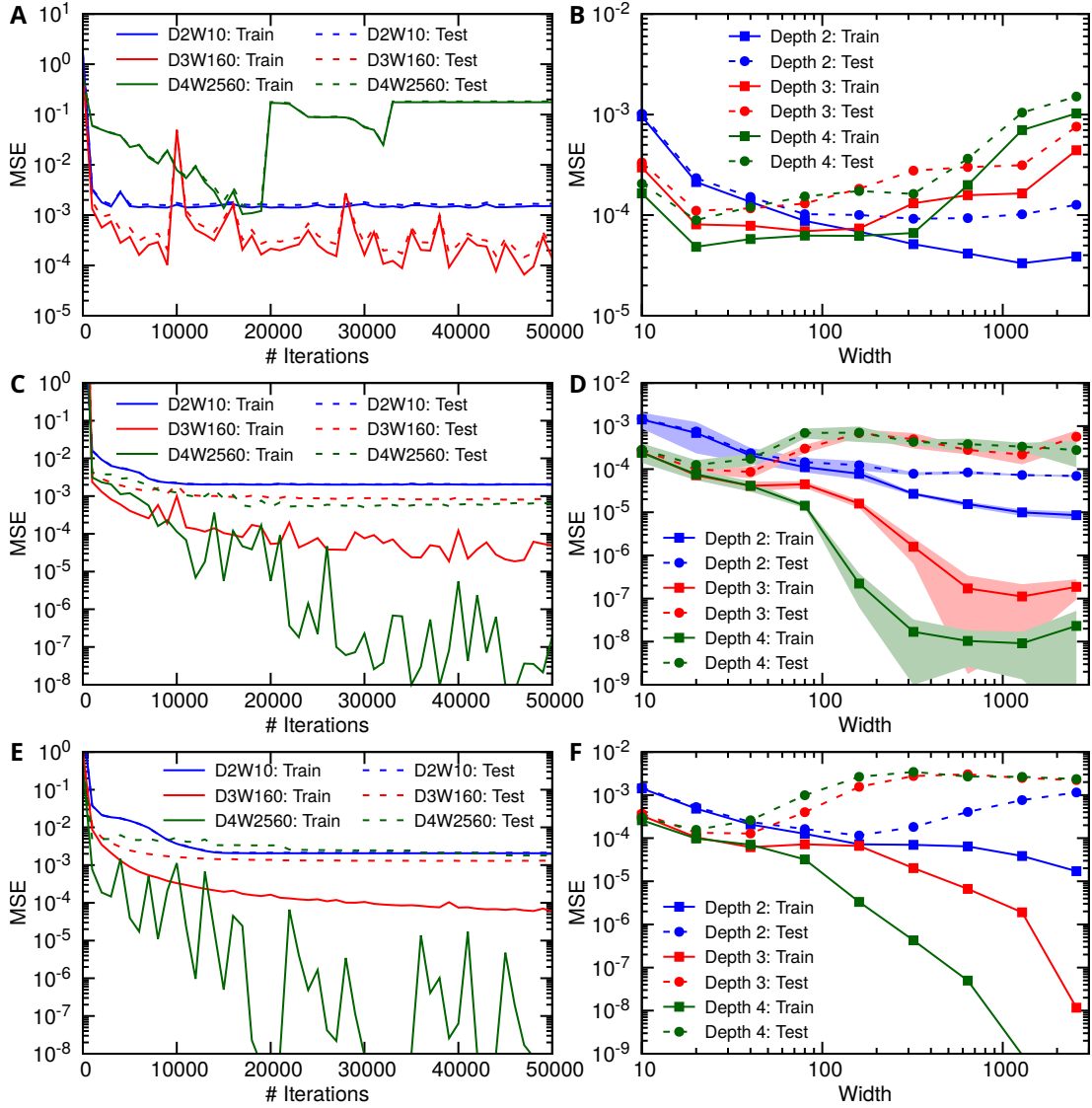
Figure 2: **Errors of FNNs trained to learn the antiderivative operator (linear case).** (**A** and **B**) Learning rate 0.01. (**C** and **D**) Learning rate 0.001. (**E** and **F**) Learning rate 0.0001. (A, C, E) The solid and dash lines are the training error and test error during the training process, respectively. The blue, red and green lines represent FNNs of size (depth 2, width 10), (depth 3, width 160), and (depth 4, width 2560), respectively. (B, D, F) The blue, red and green lines represent FNNs of depth 2, 3 and 4, respectively. The shaded regions in (D) are the one-standard-derivation (SD) from 10 runs with different training/test data and network initialization. For clarity, only the SD of learning rate 0.001 is shown (Fig. D). The number of sensors for u is $m = 100$.
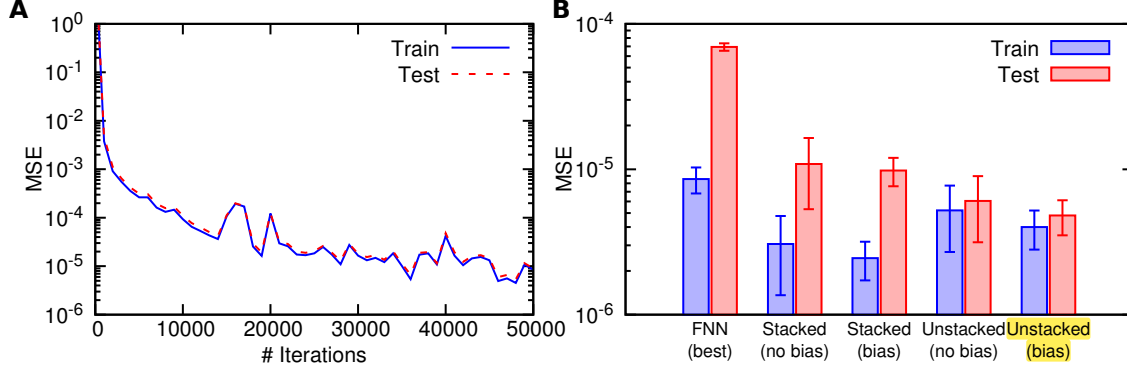
Figure 3: **Errors of DeepONets trained to learn the antiderivative operator (linear case).** (**A**) The training trajectory of an unstacked DeepONet with bias. (**B**) The training/test error for stacked/unstacked DeepONets with/without bias compared to the best error of FNNs. The error bars are the one-standard-derivation from 10 runs with different training/test data and network initialization.

4.1.2 Nonlinear case: $g(s(x), u(x), x) = -s^2(x) + u(x)$

Next we consider a nonlinear problem with $g(s(x), u(x), x) = -s^2(x) + u(x)$. Because $s(x)$ may explode for certain u, we compute the test MSE by removing the 1‰ worst predictions. During the network training, the training MSE and test MSE of both stacked and unstacked DeepONets decrease, but the correlation between training MSE and test MSE of unstacked DeepONets is tighter (Fig. 4A), i.e., smaller generalization error. This tight correlation between training and test MSE of unstacked DeepONets is also observed across multiple runs with random training dataset and network initialization (Fig. 4B). Moreover, the test MSE and training MSE of unstacked DeepONets follow almost a linear correlation

$$\text{MSE}_{\text{test}} \approx 10 \times \text{MSE}_{\text{train}} - 10^{-4}.$$

Fig. 4C shows that unstacked DeepONets have smaller test MSE due to smaller generalization error. DeepONets work even for out-of-distribution predictions, see three examples of the prediction in Fig. 5. In the following study, we will use unstacked DeepONets.

4.2 Gravity pendulum with an external force

The motion of a gravity pendulum with an external force is described as

$$\frac{ds_1}{dt} = s_2,$$
$$\frac{ds_2}{dt} = -k \sin s_1 + u(t),$$

with an initial condition $\mathbf{s}(0) = \mathbf{0}$, and k is determined by the acceleration due to gravity and the length of the pendulum. This problem is characterized by three factors: (1) k, (2) maximum prediction time T, and (3) input function space. The accuracy of learned networks is determined by four factors: (1) the number of sensor points m; (2) training dataset size; (3) network architecture, (4) optimizer. We investigate the effects of these factors on the accuracy.

4.2.1 Number of sensors

The number of sensors required to distinguish two input functions depends on the value of k, prediction time T, and the input function space. For the case with $k = 1$, $T = 1$ and $l = 0.2$, when the number of sensors
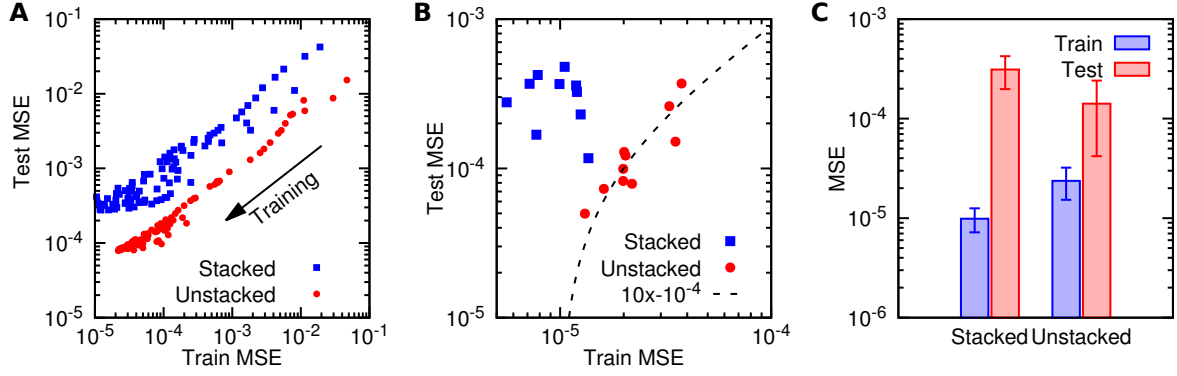
Figure 4: **Nonlinear ODE: unstacked DeepONets have smaller generalization error and smaller test MSE than stacked DeepONets.** (**A**) The correlation between the training MSE and the test MSE of one stacked/unstacked DeepONet during the training process. (**B**) The correlation between the final training MSE and test MSE of stacked/unstacked DeepONets in 10 runs with random training dataset and network initialization. The training and test MSE of unstacked DeepONets follow a linear correlation (black dash line). (**C**) The mean and one-standard-derivation of data points in B.
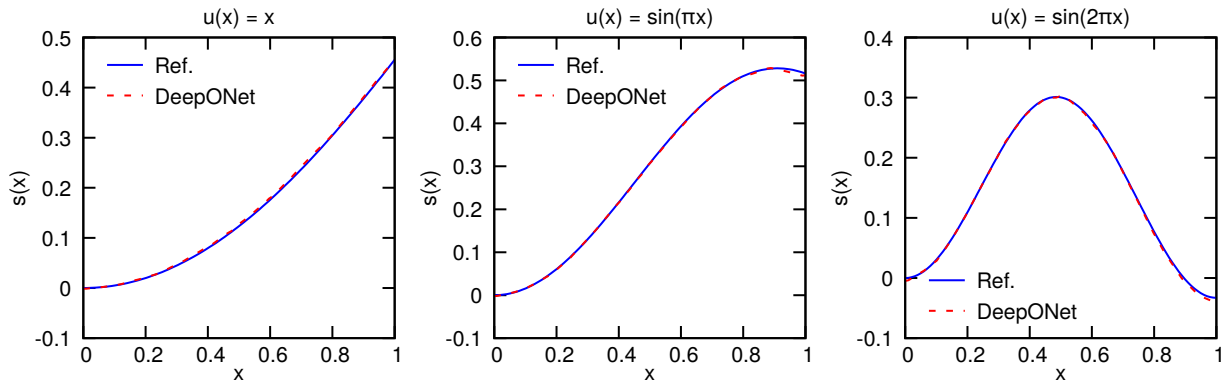

Figure 5: **Nonlinear ODE: predictions of a trained unstacked DeepONet on three out-of-distribution input signals.** The blue and red lines represent the reference solution and the prediction of a DeepONet.

m is small, the error decays exponentially as we increase the number of sensors, Fig. 6A):

$$\text{MSE} \propto 4.6^{-\#\text{sensors}}.$$

When m is already large, the effect of increasing m is negligible. The transition occurs at ~ 10 sensors, as indicated by the arrow.

To predict s for a longer time, more sensors are required (Fig. 6B). For example, predicting until $T = 5$ requires ~ 25 sensors. If the function u is less smooth corresponding to smaller l, it also requires more sensors (Fig. 6C). However, the number of sensors is not sensitive to k (Fig. 6D). Although it is hard to quantify the exact dependency of m on T and l, by fitting the computational results we show that

$$m \propto \sqrt{T} \quad \text{and} \quad m \propto l^{-1}.$$

In Theorem 2, m should be large enough to make $Te^{cT}\kappa(m, V)$ small. In Appendix C we show theoretically that $m \propto l^{-1}$ for the GRF function space with RBF kernel, which is consistent with our computational result here. Te^{cT} in the bound is loose compared to the computational results.
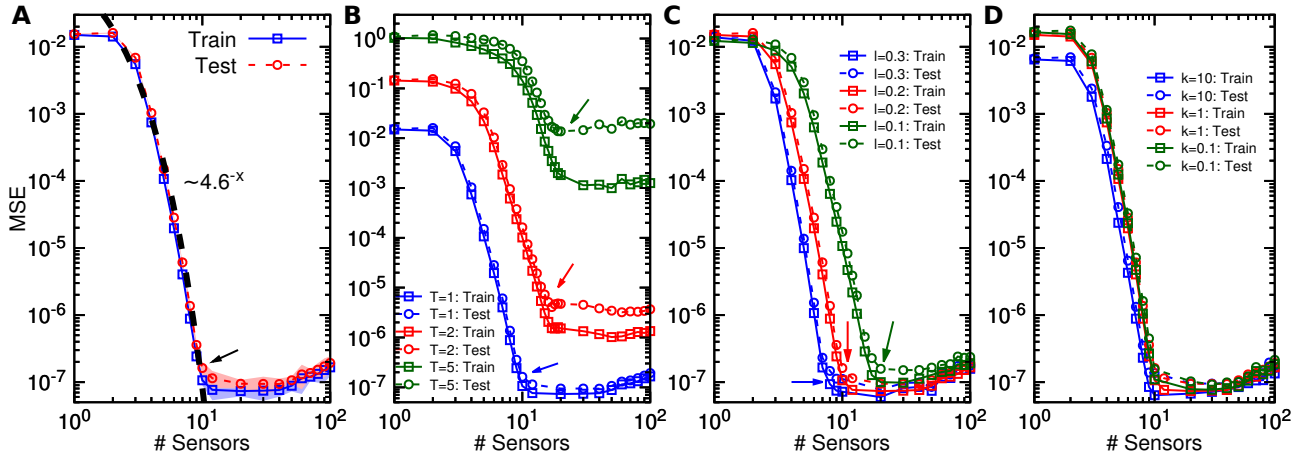

Figure 6: **Gravity pendulum: required number of sensors for different T, k and l.** (**A**) Training MSE (square symbols) and test MSE (circle symbols) decrease as the number of sensors increases in the case $k = 1$, $T = 1$ and $l = 0.2$. Training and test MSE versus the number of sensors in different conditions of (**B**) T, (**C**) l, and (**D**) k. For clarity, the SD is not shown. The arrow indicates where the rate of the error decay diminishes.

4.2.2 Error tendency and convergence

Here we investigate the error tendency under different conditions, including prediction time, network size, and training dataset size. We first observe that both the training and test errors grow exponentially with the maximum prediction time (Fig. 7A):

$$\text{MSE} \propto 8^T.$$

We note that the error is not the error at time T but the average error over the domain $[0, T]$, because we predict the $s(t)$ for any $t \in [0, T]$. As shown in Fig. 6B, 100 sensor points are sufficient for $T = 5$, and thus the possible reasons for increased error are: (1) the network is too small, and (2) training data is not sufficient. Because the error in $T = 1$ is already very small, to leverage the error, we use $T = 3$ in the following experiments. By varying the network width, we can observe that there is a best width to achieve

the smallest error (Fig. 7B). It is reasonable that increasing the width from 1 to 100 would decrease the error, but the error would instead increase when the width further increases. This could be due to the increased optimization error, and a better learning rate may be used to find a better network.

To examine the effect of training dataset size, we choose networks of width 100 to eliminate the network size effect. The training, test, and generalization errors using different dataset size are shown in Fig. 7C and D, and we have

$$
\mathrm{MSE_{test}} \propto \begin{cases} e^{-x/2000}, & \text{for small dataset} \\ x^{-0.5}, & \text{for large dataset} \end{cases}, \quad \mathrm{MSE_{gen}} \propto \begin{cases} e^{-x/2000}, & \text{for small dataset} \\ x^{-1}, & \text{for large dataset} \end{cases},
$$

where x is the number of training data points. It is surprising that test error and generalization error have exponential convergence for training dataset size $< 10^4$. Even for large dataset, the convergence rate of x^{-1} for the generalization error is still higher than the classical $x^{-0.5}$ in the learning theory [19]. This fast convergence confirms the exceptional performance of DeepONets, especially in the region of small dataset.
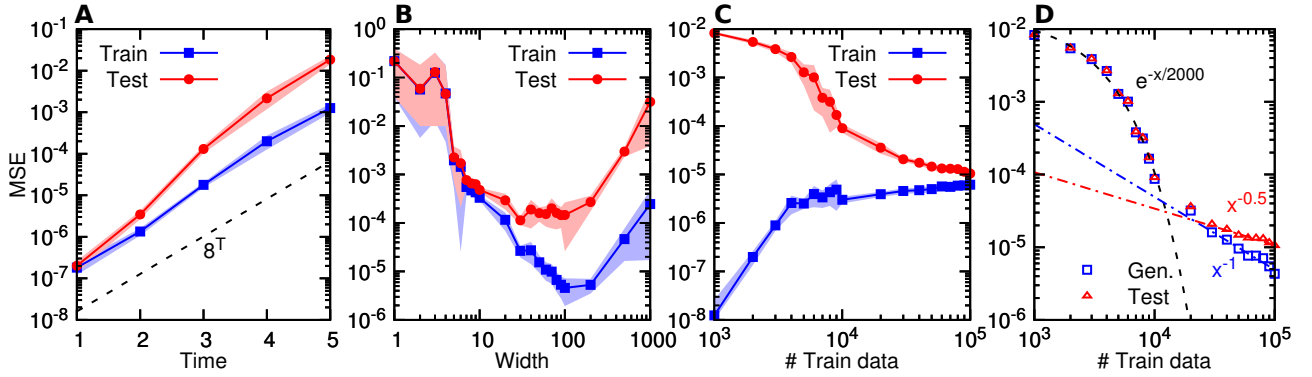

Figure 7: **Gravity pendulum: error tendency and convergence.** (**A**) Both training error (blue) and test error (red) increase fast for long-time prediction if we keep the network size fixed at width 40 and the training data size fixed at 10000 points. (**B**) Training and test errors first decrease and then increase, when network width increases ($T = 3$). (**C**) Test error decreases when more training data are used ($T = 3$, width 100). (**D**) Test error and generalization error have exponential convergence for small training dataset, and then converge with rate $x^{-0.5}$ and x^{-1}, respectively ($T = 3$, width 100).

4.2.3 Input function spaces

Next we investigate the effect of different function spaces, including GRF with different length scale l and the space of Chebyshev polynomials with different number of bases. For a fixed sensor number, we observe that there exists a critical length scale, around where the training and test errors change rapidly, see the arrows in Fig. 8A and B. This sharp transition around the critical value is also observed in the space of Chebyshev polynomials with different number of bases (Fig. 8B). The relations between the critical values and the sensor numbers are

$$
l \propto m^{-1} \quad \text{and} \quad \#\mathrm{Bases} \propto \sqrt{m}.
$$

The inverse relation between l and m is consistent with the theoretical results in Appendix C and the computational results in Section 4.2.1. Therefore, when the function space complexity increases, one may increase m to capture the functions well.
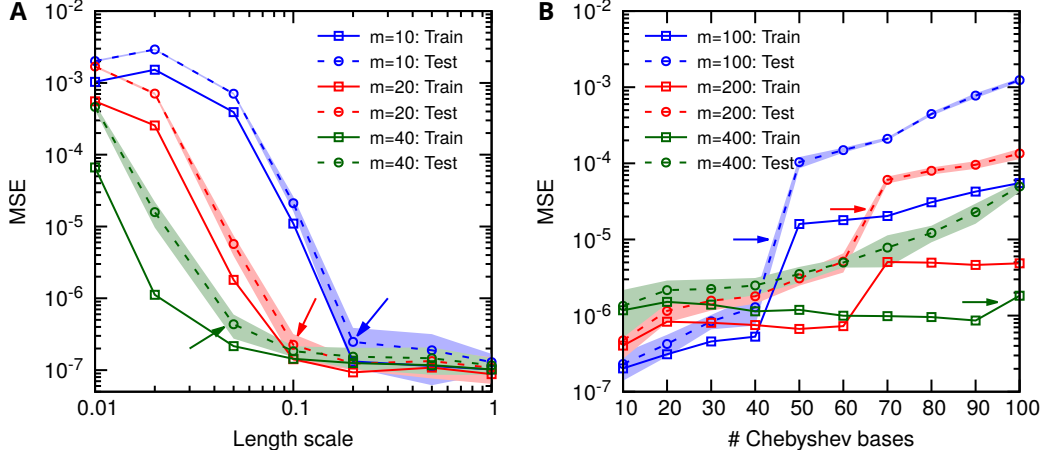
Figure 8: **Gravity pendulum: errors for different input function spaces.** Training error (solid line) and test error (dash line) for (**A**) GRF function space with different length scale l. The colors correspond to the number of sensors as shown in the inset. (**B**) Chebyshev polynomials for different number of basis functions.

4.3 Diffusion-reaction system with a source term

A diffusion-reaction system with a source term $u(x)$ is described by

$$\frac{\partial s}{\partial t} = D\frac{\partial^2 s}{\partial x^2} + ks^2 + u(x), \quad x \in [0,1], t \in [0,1],$$

with zero initial/boundary conditions, where $D = 0.01$ is the diffusion coefficient, and $k = 0.01$ is the reaction rate. We use DeepONets to learn the operator mapping from $u(x)$ to the PDE solution $s(x,t)$. In the previous examples, for each input u, we only use one random point of $s(x)$ for training, and instead we may also use multiple points of $s(x)$. To generate the training dataset, we solve the diffusion-reaction system using a second-order implicit finite difference method on a 100 by 100 grid, and then for each s we randomly select P points out of these $10000 = 100 \times 100$ grid points (Fig. 9A). Hence, the dataset size is equal to the product of P by the number of u samples. We confirm that the training and test datasets do not include the data from the same s.

We investigate the error tendency with respect to (w.r.t.) the number of u samples and the value of P. When we use 100 random u samples, the test error decreases first as P increases (Fig. 9B), and then saturates due to other factors, such as the finite number of u samples and fixed neural network size. We observe a similar error tendency but with less saturation as the number of u samples increases with P fixed (Fig. 9C). In addition, in this PDE problem the DeepONet is able to learn from a small dataset, e.g., a DeepONet can reach the test error of $\sim 10^{-5}$ when it is only trained with 100 u samples ($P = 1000$). We recall that we test on 10000 grid points, and thus on average each location point only has $100 \times 1000/10000 = 10$ training data points.

Before the error saturates, the rates of convergence w.r.t. both P and the number of u samples obey a polynomial law in the most of the range (Figs. 10A and B). The rate of convergence w.r.t. P depends on the number of u samples, and more u samples induces faster convergence until it saturates (the blue line in Fig. 10C). Similarly, the rate of convergence w.r.t. the number of u samples depends on the value of P (the red line in Fig. 10C). In addition, in the initial range of the convergence, we observe an exponential convergence (Figs. 10D and E) as in Section 4.2.2. The coefficient $1/k$ in the exponential convergence $e^{-x/k}$ also depends on the number of u samples or the value of P (Fig. 10F). It is reasonable that the convergence rate in Figs. 10C and F increases with the number of u samples or the value of P, because the total number
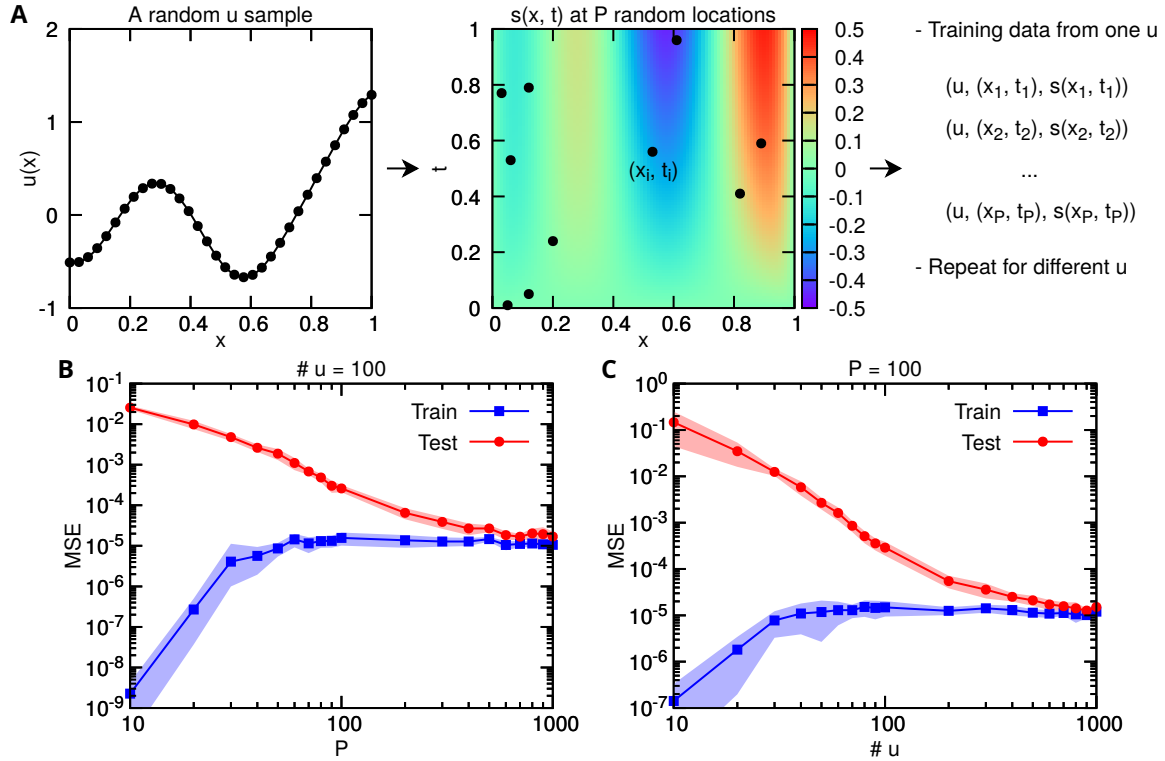
13

Figure 9: **Learning a diffusion-reaction system.** (**A**) (left) An example of a random sample of the input function $u(x)$. (middle) The corresponding output function $s(x,t)$ at P different (x,t) locations. (right) Pairing of inputs and outputs at the training data points. The total number of training data points is the product of P times the number of samples of u. (**B**) Training error (blue) and test error (red) for different values of the number of random points P when 100 random u samples are used. (**C**) Training error (blue) and test error (red) for different number of u samples when $P = 100$. The shaded regions denote one-standard-derivation.

of training data points is equal to $P \times \#u$. However, by fitting the points, it is surprising that there is a clear tendency in the form of either $\ln(x)$ or e^{-x}, which we cannot fully explain yet, and hence more theoretical and computational investigations are required.
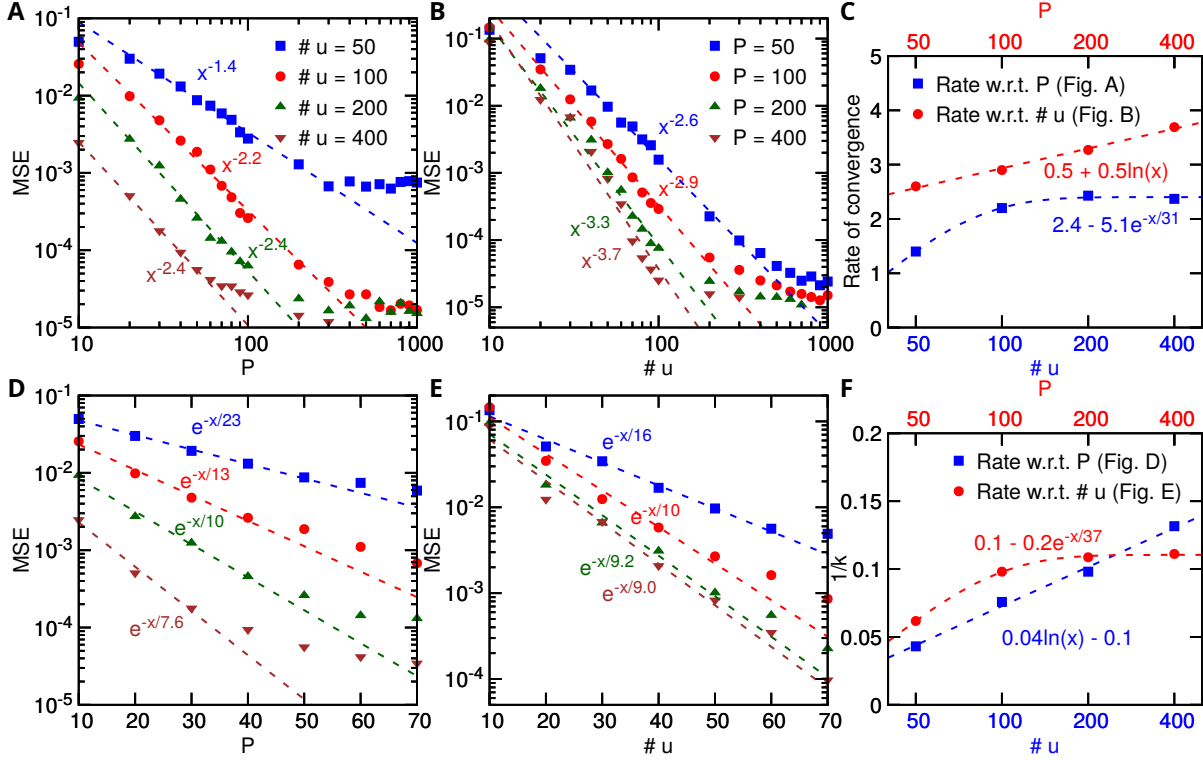

Figure 10: **Error convergence rates for different number of training data points.** (**A**) Convergence of test error with respect to P for different number of u samples. (**B**) Convergence of test error with respect to the number of u samples for different values of P. (**C**) The polynomial rates of convergence versus the number of u samples or the values of P. (**D**) Exponential convergence of test error with respect to P for different number of u samples. (**E**) Exponential convergence of test error with respect to the number of u samples for different values of P. (**F**) The coefficient $1/k$ in the exponential convergence $e^{-x/k}$ versus the number of u samples or the values of P.

5 Conclusion

In this paper, we formulate the problem of learning operators in a more general setup, and propose DeepONets to learn nonlinear operators. In DeepONets, we first construct two sub-networks to encode input functions and location variables separately, and then merge them together to compute the output. We test DeepONets on four ordinary/partial differential equation problems, and show that DeepONets can achieve small generalization errors by employing this inductive bias. In our simulations, we study systematically the effects on the test error of different factors, including the number of sensors, maximum prediction time, the complexity of the space of input functions, training dataset size, and network size. We observe different order polynomial and even exponential error convergence with respect to the training dataset size. To the best of our knowledge, this is the first time exponential convergence is observed in deep learning. Moreover, we derive theoretically the dependence of approximation error on different factors, which is consistent with our computational results.

Despite the aforementioned achievements, more work should be done both theoretically and computationally. For example, there have not been any theoretical results of network size for operator approximation, similar to the bounds of width and depth for function approximation [10]. We also do not understand theoretically yet why DeepONets can induce small generalization errors. On the other hand, in this paper we use fully-connected neural networks for the two sub-networks, but as we discussed in Section 2.1, we can also employ other network architectures, such as convolutional neural networks or "attention" mechanism. These modifications may improve further the accuracy of DeepONets.

6   Acknowledgments

References

[1] L. Bottou and O. Bousquet. The tradeoffs of large scale learning. In *Advances in Neural Information Processing Systems*, pages 161–168, 2008.

[2] S. L. Brunton, J. L. Proctor, and J. N. Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016.

[3] T. Chen and H. Chen. Approximations of continuous functionals by neural networks with application to dynamic systems. *IEEE Transactions on Neural Networks*, 4(6):910–918, 1993.

[4] T. Chen and H. Chen. Approximation capability to functions of several variables, nonlinear functionals, and operators by radial basis function neural networks. *IEEE Transactions on Neural Networks*, 6(4):904–910, 1995.

[5] T. Chen and H. Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks*, 6(4):911–917, 1995.

[6] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pages 6571–6583, 2018.

[7] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

[8] V. Dumoulin, E. Perez, N. Schucher, F. Strub, H. d. Vries, A. Courville, and Y. Bengio. Feature-wise transformations. *Distill*, 2018. https://distill.pub/2018/feature-wise-transformations.

[9] N. B. Erichson, M. Muehlebach, and M. W. Mahoney. Physics-informed autoencoders for Lyapunov-stable fluid flow prediction. *arXiv preprint arXiv:1905.10866*, 2019.

[10] B. Hanin. Universal function approximation by deep neural nets with bounded width and ReLU activations. *arXiv preprint arXiv:1708.02691*, 2017.

[11] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[12] J. Jia and A. R. Benson. Neural jump stochastic differential equations. *arXiv preprint arXiv:1905.10403*, 2019.

[13] P. Jin, L. Lu, Y. Tang, and G. E. Karniadakis. Quantifying the generalization error in deep learning in terms of data distribution and neural network smoothness. *arXiv preprint arXiv:1905.11427*, 2019.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[15] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis. DeepXDE: A deep learning library for solving differential equations. *arXiv preprint arXiv:1907.04502*, 2019.

[16] L. Lu, Y. Shin, Y. Su, and G. E. Karniadakis. Dying ReLU and initialization: Theory and numerical examples. *arXiv preprint arXiv:1903.06733*, 2019.

[17] L. Lu, Y. Su, and G. E. Karniadakis. Collapse of deep and narrow neural nets. *arXiv preprint arXiv:1808.04947*, 2018.

[18] H. N. Mhaskar and N. Hahm. Neural networks for functional approximation and system identification. *Neural Computation*, 9(1):143–159, 1997.

[19] M. Mitzenmacher and E. Upfal. *Probability and computing: randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.

[20] G. Neofotistos, M. Mattheakis, G. D. Barmparis, J. Hizanidis, G. P. Tsironis, and E. Kaxiras. Machine learning with observers predicts complex spatiotemporal behavior. *arXiv preprint arXiv:1807.10758*, 2018.

[21] G. Pang, L. Lu, and G. E. Karniadakis. fPINNs: Fractional physics-informed neural networks. *SIAM Journal on Scientific Computing*, 41(4):A2603–A2626, 2019.

[22] J. C. Patra, R. N. Pal, B. Chatterji, and G. Panda. Identification of nonlinear dynamic systems using functional link artificial neural networks. *IEEE transactions on systems, man, and cybernetics, part b (cybernetics)*, 29(2):254–262, 1999.

[23] T. Qin, K. Wu, and D. Xiu. Data driven governing equations approximation using deep neural networks. *Journal of Computational Physics*, 2019.

[24] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Multistep neural networks for data-driven discovery of nonlinear dynamical systems. *arXiv preprint arXiv:1801.01236*, 2018.

[25] F. Rossi and B. Conan-Guez. Functional multi-layer perceptron: A non-linear tool for functional data analysis. *Neural Networks*, 18(1):45–60, 2005.

[26] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz. Data-driven discovery of partial differential equations. *Science Advances*, 3(4):e1602614, 2017.

[27] S. Sabour, N. Frosst, and G. E. Hinton. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems*, pages 3856–3866, 2017.

[28] N. Trask, R. G. Patel, B. J. Gross, and P. J. Atzberger. GMLS-Nets: A framework for learning from unstructured data. *arXiv preprint arXiv:1909.05371*, 2019.

[29] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[30] N. Winovich, K. Ramani, and G. Lin. ConvPDE-UQ: Convolutional neural networks with quantified uncertainty for heterogeneous elliptic partial differential equations on varied domains. *Journal of Computational Physics*, 2019.

[31] D. Zhang, L. Lu, L. Guo, and G. E. Karniadakis. Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems. *Journal of Computational Physics*, 397:108850, 2019.

[32] Z. Zhang and G. E. Karniadakis. *Numerical methods for stochastic partial differential equations with white noise*. Springer, 2017.

[33] H. Zhao and J. Zhang. Nonlinear dynamic system identification using pipelined functional link artificial recurrent neural network. *Neurocomputing*, 72(13-15):3046–3054, 2009.

[34] Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, and P. Perdikaris. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics*, 394:56–81, 2019.

A   Neural networks to approximate nonlinear operators

We list in Table 3 the main symbols and notations that are used throughout this paper.

Table 3: Notations.

X	a Banach space with norm $\|\cdot\|_X$
$\mathbb{R}^d$	Euclidean space of dimension d
K	a compact set in a Banach space
$C(K)$	Banach space of all continuous functions defined on K with norm $\|f\|_{C(K)} = \max_{x \in K} |f(x)|$
V	a compact set in $C(K)$
$u(x)$	an input function/signal
$s(x)$	an output function/signal
f	a function or functional
G	an operator
(TW)	all the Tauber-Wiener functions
σ	an activation function
$\{x_1, x_2, \ldots, x_m\}$	m sensor points
n, p	neural network size hyperparameters in Theorems 4 and 5
N	number of basis functions
M	value of some upper bound

Let $C(K)$ denote the Banach space of all continuous functions defined on a compact set $K \subset X$ with sup-norm $\|f\|_{C(K)} = \max_{x \in K} |f(x)|$, where X is a Banach space. We first review the definition and sufficient condition of Tauber-Wiener (TW) functions [5], and the definition of continuous operator.

Definition 1 (**TW**). *If a function $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ (continuous or discontinuous) satisfies that all the linear combinations $\sum_{i=1}^{N} c_i \sigma(\lambda_i x + \theta_i)$, $\lambda_i \in \mathbb{R}$, $\theta_i \in \mathbb{R}$, $c_i \in \mathbb{R}$, $i = 1, 2, \ldots, N$, are dense in every $C([a, b])$, then σ is called a Tauber-Wiener (TW) function.*

Theorem 3 (**Sufficient condition for TW**). *Suppose that σ is a continuous function, and $\sigma \in S'(\mathbb{R})$ (tempered distributions), then $\sigma \in (TW)$, if and only if σ is not a polynomial.*

It is easy to verify that all the activation functions we used nowadays, such as sigmoid, $\tanh$ and ReLU, are TW functions.

Definition 2 (**Continuity**). *Let G be an operator between topological spaces X and Y. We call G continuous if for every $\epsilon > 0$, there exists a constant $\delta > 0$ such that*

$$\|G(x) - G(y)\|_Y < \epsilon$$

for all $x, y \in X$ satisfying $\|x - y\|_X < \delta$.

We recall the following two main theorems of approximating nonlinear continuous functionals and operators due to Chen & Chen [5].

Theorem 4 (**Universal Approximation Theorem for Functional**). *Suppose that $\sigma \in (TW)$, X is a Banach Space, $K \subset X$ is a compact set, V is a compact set in $C(K)$, f is a continuous functional defined on V, then for any $\epsilon > 0$, there are a positive integer n, m points $x_1, \ldots, x_m \in K$, and real constants c_i, θ_i, ξ_{ij}, $i = 1, \ldots, n$, $j = 1, \ldots, m$, such that*

$$\left| f(u) - \sum_{i=1}^{n} c_i \sigma \left(\sum_{j=1}^{m} \xi_{ij} u(x_j) + \theta_i \right) \right| < \epsilon$$

holds for all $u \in V$.

Theorem 5 (**Universal Approximation Theorem for Operator**). *Suppose that $\sigma \in (TW)$, X is a Banach Space, $K_1 \subset X$, $K_2 \subset \mathbb{R}^d$ are two compact sets in X and $\mathbb{R}^d$, respectively, V is a compact set in $C(K_1)$, G is a nonlinear continuous operator, which maps V into $C(K_2)$, then for any $\epsilon > 0$, there are positive integers n, p, m, constants $c_i^k, \xi_{ij}^k, \theta_i^k, \zeta_k \in \mathbb{R}$, $w_k \in \mathbb{R}^d$, $x_j \in K_1$, $i = 1, \ldots, n$, $k = 1, \ldots, p$, $j = 1, \ldots, m$, such that*

$$\left| G(u)(y) - \sum_{k=1}^{p} \sum_{i=1}^{n} c_i^k \sigma \left(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k \right) \sigma(w_k \cdot y + \zeta_k) \right| < \epsilon$$

holds for all $u \in V$ and $y \in K_2$.

Both Theorems 4 and 5 require the compactness of the function space. In fact, the function space like $C([0, 1])$ is "too large" for real applications, so it is suitable to consider a smaller space with compactness property. The necessary and sufficient conditions of the compactness are given by the following Arzelà–Ascoli Theorem.

Definition 3 (**Uniform Boundedness**). *Let V be a family of real-valued functions on the set K. We call V uniformly bounded if there exists a constant M such that*

$$|f(x)| \leq M$$

for all $f \in V$ and all $x \in K$.

Definition 4 (**Equicontinuity**). *Let V be a family of real-valued functions on the set K. We call V equicontinuous if for every $\epsilon > 0$, there exists a $\delta > 0$ such that*

$$|f(x) - f(y)| < \epsilon$$

for all $f \in V$ and all $x, y \in K$ satisfying $|x - y| < \delta$.

Theorem 6 (**Arzelà–Ascoli Theorem**). *Let X be a Banach space, and $K \subset X$ be a compact set. A subset V of $C(K)$ is pre-compact (has compact closure) if and only if V is uniformly bounded and equicontinuous.*

B Number of sensors for identifying nonlinear dynamic systems

Lemma 7. $W := \bigcup_{i=1}^{\infty} W_i$ *is compact.*

Proof. At first, we prove that W is pre-compact. For any $\varepsilon > 0$, by (4), there exists an m_0 such that

$$\|u - \mathcal{L}_m(u)\|_C < \frac{\varepsilon}{4}, \quad \forall u \in V, \forall m > m_0.$$

Since W_{m_0} is a compact set subject to equicontinuity, there exists a $\delta > 0$ such that

$$|x - y| < \delta \Rightarrow |u(x) - u(y)| < \frac{\varepsilon}{2}, \quad \forall u \in W_{m_0}, \forall x, y \in [a, b].$$

Now for all $u \in W$ and all $x, y \in [a, b], |x - y| < \delta$, if $u \in W_{m_0}$, naturally we have $|u(x) - u(y)| < \frac{\varepsilon}{2} < \varepsilon$, otherwise, $u \in \bigcup_{i=m_0+1}^{\infty} U_i$. Suppose that $u = \mathcal{L}_m(v), m > m_0, v \in V$, then there holds

$$
\begin{aligned}
|u(x) - u(y)| =& |u(x) - v(x) + v(x) - v(y) + v(y) - u(y)| \\
\leq& |\mathcal{L}_m(v)(x) - v(x)| + |v(x) - v(y)| + |\mathcal{L}_m(v)(y) - v(y)| \\
\leq& 2\|\mathcal{L}_m(v) - v\|_C + |v(x) - v(y)| \\
<& 2 \cdot \frac{\varepsilon}{4} + \frac{\varepsilon}{2} = \varepsilon,
\end{aligned}
$$

which shows the quicontinuity of W. In addition, it is obvious that W is uniformly bounded, so that we know W is pre-compact by applying the Arzelà–Ascoli Theorem.

Next we show that W is close. Let $\{w_i\}_{i=1}^{\infty} \subset W$ be a sequence which converges to a $w_0 \in C[a, b]$. If there exists an m such that $\{w_i\} \subset W_m$, then $w_0 \in W_m \subset W$. Otherwise, there is a subsequence $\{\mathcal{L}_{i_n}(v_{i_n})\}$ of $\{w_i\}$ such that $v_{i_n} \in V$ and $i_n \to \infty$ as $n \to \infty$. Then we have

$$
\begin{aligned}
\|v_{i_n} - w_0\|_C =& \|v_{i_n} - \mathcal{L}_{i_n}(v_{i_n}) + \mathcal{L}_{i_n}(v_{i_n}) - w_0\|_C \\
\leq& \|v_{i_n} - \mathcal{L}_{i_n}(v_{i_n})\|_C + \|\mathcal{L}_{i_n}(v_{i_n}) - w_0\|_C \\
\leq& \kappa(i_n, V) + \|\mathcal{L}_{i_n}(v_{i_n}) - w_0\|_C,
\end{aligned}
$$

which implies that $w_0 = \lim_{n \to \infty} v_{i_n} \in V \subset W$. $\qquad\square$

Next we show the proof of Theorem 2.

Proof. For $u \in V$ and $u_m \in U_m$, according to the bound (4) and the Lipschitz condition, we can derive that

$$
\begin{aligned}
\|(Gu)(d) - (Gu_m)(d)\|_2 &\leq c \int_a^d \|(Gu)(t) - (Gu_m)(t)\|_2 dt + c \int_a^d |u(t) - u_m(t)| dt \\
&\leq c \int_a^d \|(Gu)(t) - (Gu_m)(t)\|_2 dt + c(b - a)\kappa(m, V).
\end{aligned}
$$

By Gronwall inequality, we have

$$\|(Gu)(d) - (Gu_m)(d))\|_2 \le c(b-a)\kappa(m,V)e^{c(b-a)}.$$

Define $S_m = \{(u(x_0), u(x_1), \cdots, u(x_m)) \in \mathbb{R}^{m+1} | u \in V\}$ which is a compact set in $\mathbb{R}^{m+1}$, and there is a bijective mapping between S_m and U_m. Furthermore, we define a vector-valued function on S_m by

$$\varphi(u(x_0), u(x_1), \cdots, u(x_m)) = (Gu_m)(d).$$

For any $\varepsilon > 0$, make m large enough so that $c(b-a)\kappa(m,V)e^{c(b-a)} < \varepsilon$. By universal approximation theorem of neural network for high-dimensional functions, there exist $\mathcal{W}_1 \in \mathbb{R}^{n \times (m+1)}, b_1 \in \mathbb{R}^{m+1}, \mathcal{W}_2 \in \mathbb{R}^{K \times n}, b_2 \in \mathbb{R}^K$, such that

$$\|\varphi(u(x_0), \cdots, u(x_m)) - (\mathcal{W}_2 \cdot \sigma(\mathcal{W}_1 \cdot [u(x_0) \quad \cdots \quad u(x_m)]^T + b_1) + b_2)\|_2 < \varepsilon - c(b-a)\kappa(m,V)e^{c(b-a)}.$$

Hence we have

$$
\begin{aligned}
&\|(Gu)(d) - (\mathcal{W}_2 \cdot \sigma(\mathcal{W}_1 \cdot [u(x_0) \quad \cdots \quad u(x_m)]^T + b_1) + b_2)\|_2 \\
\le& \|(Gu)(d) - (Gu_m)(d)\|_2 + \|(Gu_m)(d) - (\mathcal{W}_2 \cdot \sigma(\mathcal{W}_1 \cdot [u(x_0) \quad \cdots \quad u(x_m)]^T + b_1) + b_2)\|_2 \\
<& c(b-a)\kappa(m,V)e^{c(b-a)} + \varepsilon - c(b-a)\kappa(m,V)e^{c(b-a)} = \varepsilon.
\end{aligned}
$$

In summary, by choosing the value of m so that it makes $c(b-a)\kappa(m,V)e^{c(b-a)}$ less than ε is sufficient to achieve accuracy ε. $\qquad\square$

C Gaussian random field with the radial-basis function kernel

Suppose that $X(t) \sim \mathcal{G}(0, \exp(-\frac{|x-y|^2}{l^2}))$. Then

$$X(t) = \sqrt{2}(\pi)^{\frac{1}{4}} \int_{\mathbb{R}^+} (l)^{\frac{1}{2}} \cos(\omega t) \exp(-\frac{l^2\omega^2}{8}) dW(\omega) - \sqrt{2}(\pi)^{\frac{1}{4}} \int_{\mathbb{R}^+} (l)^{\frac{1}{2}} \sin(\omega t) \exp(-\frac{l^2\omega^2}{8}) dB(\omega),$$

where W and B are independent standard Brownian motions [32]. Apply the change of variable $\lambda = l\omega$ and write $X(t)$ as

$$X(t) = \sqrt{2}(\pi)^{\frac{1}{4}} \int_{\mathbb{R}^+} \cos(\frac{\lambda}{l}t) \exp(-\frac{\lambda^2}{8}) dW(\lambda) - \sqrt{2}(\pi)^{\frac{1}{4}} \int_{\mathbb{R}^+} \sin(\frac{\lambda}{l}t) \exp(-\frac{\lambda^2}{8}) dB(\lambda).$$

Applying a linear interpolation Π_1 on the interval $[t_i, t_{i+1}]$, then

$$
\begin{aligned}
\mathbb{E}[(X(t) - \Pi_1 X(t))^2] &= 2(\pi)^{\frac{1}{2}} \int_{\mathbb{R}^+} ((I - \Pi_1)\cos(\frac{\lambda}{l}t))^2 \exp(-\frac{\lambda^2}{4}) d\lambda \\
&\quad + 2(\pi)^{\frac{1}{2}} \int_{\mathbb{R}^+} ((I - \Pi_1)\sin(\frac{\lambda}{l}t))^2 \exp(-\frac{\lambda^2}{4}) d\lambda \\
&\le (\pi)^{\frac{1}{2}}(t_{i+1} - t_i)^4 \int_{\mathbb{R}^+} (\frac{\lambda}{l})^4 \exp(-\frac{\lambda^2}{4}) d\lambda \\
&= 24\pi \frac{(t_{i+1} - t_i)^4}{l^4},
\end{aligned}
$$

where we recalled the error estimate of the linear interpolation on $[a, b]$ (by Taylor's expansion)

$$|(I - \Pi_1)g(t)| = \frac{1}{2}(b-a)^2 |f''(\xi)|,$$

where ξ lies in between a and b. Then by the Borel-Cantelli lemma, we have

$$|X(t) - \Pi_1 X(t)| \leq C \frac{(t_{i+1} - t_i)^{2-\epsilon}}{l^2}, \quad \epsilon > 0,$$

where C is an absolute value of a Gaussian random variable with a finite variance. Therefore, taking a piecewise linear interpolation of $X(t)$ with m points will lead to convergence with order $\mathcal{O}(\frac{1}{m^2 l^2})$.