ARTICLES
https://doi.org/10.1038/s42256-021-00302-5
nature machine intelligence

Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators

Lu Lu[1], Pengzhan Jin[2,3], Guofei Pang[2], Zhongqiang Zhang[4] and George Em Karniadakis[2]✉

It is widely known that neural networks (NNs) are universal approximators of continuous functions. However, a less known but powerful result is that a NN with a single hidden layer can accurately approximate any nonlinear continuous operator. This universal approximation theorem of operators is suggestive of the structure and potential of deep neural networks (DNNs) in learning continuous operators or complex systems from streams of scattered data. Here, we thus extend this theorem to DNNs. We design a new network with small generalization error, the deep operator network (DeepONet), which consists of a DNN for encoding the discrete input function space (branch net) and another DNN for encoding the domain of the output functions (trunk net). We demonstrate that DeepONet can learn various explicit operators, such as integrals and fractional Laplacians, as well as implicit operators that represent deterministic and stochastic differential equations. We study different formulations of the input function space and its effect on the generalization error for 16 different diverse applications.

We consider theoretical and computational issues related to operator regression and how to design deep neural networks (DNNs) that could represent accurately linear and nonlinear operators, mapping input functions into output functions. These operators can be of the explicit type, for example, the Laplace transform, or of the implicit type, for example, solution operators of partial differential equations (PDEs). For solution operators, the inputs to a DNN could be functions representing boundary conditions selected from a properly designed input space V. Implicit type operators may also describe systems for which we do not have any mathematical knowledge of their form, for example, in social dynamics, although we do not consider such cases in the present work.

Discovering equations or operators from data is not a new objective. There were multiple pioneering efforts using neural networks (NNs) in the 1990s, including work by Kevrekidis and his associates[1–3] and other groups[4], as well as efforts using different physically intuitive approaches (for example, coarse-grained operators from fine-scale data[5,6]). Although these previous efforts are not directly related to our work here, they nevertheless contain several interesting aspects that are useful in operator regression. In the literature, two types of implicit operator have been considered, that is, dynamical systems in the form of ordinary differential equations (ODEs) or in the form of PDEs. For the dynamical systems, different network architectures have been employed, including recurrent NNs (RNNs)[7], residual networks[8], neural ordinary differential equations[9] and neural jump stochastic differential equations[10]. However, they are only able to predict the evolution of a specific system (for example, a Hamiltonian system[11–14]) rather than identifying the system behaviour for new unseen input signals. In learning operators from PDEs with structured data, some works treat the input and output function as an image and then use convolutional NNs (CNNs) to learn the image-to-image mapping G (refs. [15,16]), but this approach can only be applied to particular types of problem where the locations of the points where input function u is evaluated are fixed. For unstructured data, a modified CNN based on generalized moving least squares[17] or graph kernel networks[18] can be used to approximate specific operators. However, they are not able to learn general nonlinear operators. Also, some PDEs can be parameterized by unknown coefficients[19–23] or an unknown forcing term[24,25], and then the unknown parts are identified from data. However, not all PDEs can be well parameterized. Symbolic mathematics have also been applied to represent PDEs[26,27], while accuracy and robustness still need to be addressed.

In this Article, we propose a general deep learning framework, DeepONet, to learn diverse continuous nonlinear operators. DeepONet is inspired directly by theory that guarantees small approximation error (that is, the error between the target operator and the class of neural networks of a given finite-size architecture). Moreover, the specific architectures we introduce exhibit small generalization errors (that is, the error of a neural network for previously unseen data) for diverse applications, which we study systematically herein. Proper representation of the input space V of the operator is very important. Hence, we select 16 test cases to investigate the important question of sampling the space V. These examples include integrals, Legendre transforms, fractional derivatives, nonlinear ODEs and PDEs, and stochastic ODEs and PDEs. For all examples, the proposed NNs generalize well—they predict the action of the operator on unseen functions accurately.

Our proposal of approximating functionals and nonlinear operators with NNs goes beyond the universal function approximation[28,29] and supervised data, or using the idea of physics-informed neural networks[22]. Specifically, we resort to a little known but powerful theorem, the universal operator approximation theorem[30]. This theorem states that a NN with a single hidden layer can approximate accurately any nonlinear continuous functional (a mapping from a space of functions into real numbers)[31–33] and (nonlinear) operator (a mapping from a space of functions into another space of

[1]Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA, USA. [2]Division of Applied Mathematics, Brown University, Providence, RI, USA. [3]LSEC, ICMSEC, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, China. [4]Department of Mathematical Sciences, Worcester Polytechnic Institute, Worcester, MA, USA. ✉e-mail: george_karniadakis@brown.edu

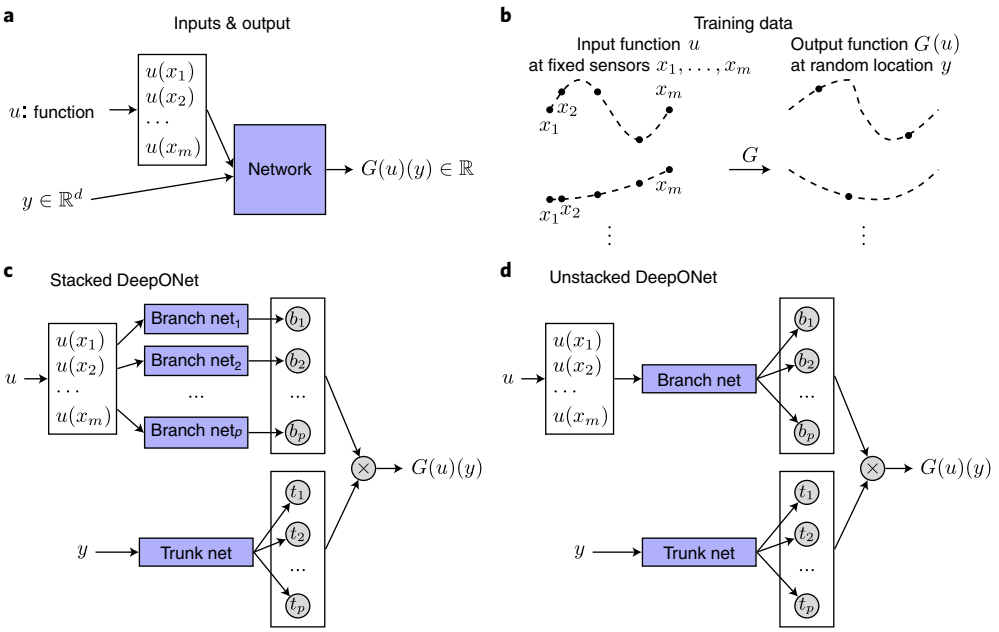

Fig. 1 | Illustrations of the problem set-up and new architectures of DeepONets that lead to good generalization. a, For the network to learn an operator $G: u \mapsto G(u)$ it takes two inputs $[u(x_1), u(x_2), \ldots, u(x_m)]$ and y. **b**, Illustration of the training data. For each input function 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 is inspired by Theorem 1, and has one trunk network and p stacked branch networks. The network constructed in Theorem 1 is a stacked DeepONet formed by choosing the trunk net as a one-layer network of width p and each branch net as a one-hidden-layer network of width n. **d**, The unstacked DeepONet is inspired by Theorem 2, and has one trunk network and one branch network. An unstacked DeepONet can be viewed as a stacked DeepONet with all the branch nets sharing the same set of parameters.

functions)[30,34]. Let G be an operator taking an input function u, with $G(u)$ being 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). In practice, we represent these input functions discretely so that network approximations can be applied. Here, we explore different representations of functions in the input space V, with the simplest one based on the function values at a sufficient but finite number of locations $\{x_1, x_2, \ldots, x_m\}$, which we call 'sensors' (Fig. 1a). There are other ways to represent a function, for example, with spectral expansions or as an image, and we demonstrate these in the 16 examples we present in this work. We envision that, in the future, such functions will be represented by other NNs.

DeepONet consists of an offline training stage followed by an online inference stage, and can be used for the real-time predictions required in critical applications such as autonomous vehicles or dynamic target identification. In the offline stage, we solve the target operator with proper input space V using classical numerical methods and then train our DNNs. Depending on the application, a DeepONet may require only one or a few hundred graphics processing unit (GPU) hours and can be trained using experimental or simulation data, or both, at various scales and levels of fidelity. The computational expense for training depends on the complexity of the operator, the quantity and quality of the data, and the network size. In this work, we assume that we have enough data and computational resources to train the model offline. In the online stage, we can use the trained network as a surrogate for online inference, which only involves a forward pass of the network, and thus it can be performed for high-dimensional models to speed up computationally expensive applications dramatically, for example, in a fraction of a second. To put this in perspective, one can look at everyday effective applications of pre-trained DNNs, for example, the recent deep learning model GPT-3[35] for language modelling that has 175 billion parameters and requires 355 GPU years (cost of approximately US$5 million) to train, but once this model is trained, it can be deployed for almost real-time inference.

DeepONet theory and network architecture

In the following, we state the theorem of Chen and Chen[30] (for more details, including definitions of variables, see Supplementary Section 1), which we further extend to deep NNs, based on which we propose the DeepONet. Subsequently, we present data generation and another theorem that relates the number and type of data with the accuracy of the input functions.

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 and m, constants c_i^k, ξ_{ij}^k, θ_i^k, $\zeta_k \in \mathbb{R}$, $w_k \in \mathbb{R}^d$, $x_j \in K_1$, $i = 1, \ldots, n$, $k = 1, \ldots, p$ and $j = 1, \ldots, m$, such that*

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

holds for all $u \in V$ and $y \in K_2$. Here, $C(K)$ is the Banach space of all continuous functions defined on K with norm $\| f \|_{C(K)} = \max_{x \in K} |f(x)|$.

The network constructed in equation (1) and the meanings of the hyperparameters n, p and m are depicted in Fig. 1c. This

approximation theorem is indicative of the potential application of neural networks to learn nonlinear operators from data, that is, similar to a standard NN where we learn functions from data. However, this theorem does not inform us how to learn operators efficiently. The overall accuracy of NNs can be characterized by dividing the total error into three main types: approximation, optimization and generalization errors[36–38]. The universal approximation theorem only guarantees a small approximation error for a sufficiently large network, but it does not consider the important optimization and generalization errors at all, which, in practice, are often dominant contributions to the total error. Useful networks should be easy to train (that is, to exhibit small optimization error) and generalize well to unseen data (that is, to exhibit small generalization error).

To demonstrate the capability and effectiveness of learning nonlinear operators using NNs, we set up the problem to be 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). However, even this constraint can be lifted, for example, by interpolating u on a common set of sensor locations or by projecting u to a set of basis functions and then use the coefficients as a representation of u. Heavily inspired by Theorem 1 and its extension Theorem 2 (see below), we propose a specific new network architecture, the deep operator network (DeepONet), to achieve small total errors. We will demonstrate that, unlike fully connected neural networks (FNNs) and residual neural networks (ResNets)[39], DeepONet substantially improves generalization based on a design of two subnetworks, the branch net for the input function and the trunk net for the locations to evaluate the output function. The key point is that we discover a new operator G as a NN, which is able to make inferences for quantities of interest given new and unseen data. If we wish to further interpret the type of operator G using the familiar classical calculus, we can project the results of $G(u)(y)$ onto a dictionary containing first- or higher-order derivatives, gradients, Laplacians and so on, as is done currently with existing regression techniques[19].

DeepONet architecture. 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 (we do not require the sensor locations to be equispaced). 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, ResNet, CNN or RNN, and concatenate two inputs together as the network input, that is, $[u(x_1), u(x_2), \ldots, u(x_m), y]^T$. However, in general, the input does not have any specific structure, and thus we use FNN and ResNet as the baseline models. To compare DeepONets with additional models, we also consider CNN or RNN as the baselines in a few examples for specific problems and datasets.

In high-dimensional problems, y is a vector with d components, so the dimension of y no longer matches the dimension of $u(x_i)$ for $i=1, 2, \ldots, m$. This also prevents us from treating $u(x_i)$ and y equally, and thus at least two subnetworks 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 with a network structure constructed in equation (1). Inspired by this network, the architecture we propose is shown in Fig. 1c. 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 then merge them together as in equation (1):

$$G(u)(y) \approx \sum_{k=1}^{p} \underbrace{b_k(u(x_1), u(x_2), \ldots, u(x_m))}_{\text{branch}} \underbrace{t_k(y)}_{\text{trunk}}$$

The network constructed in Theorem 1 is equivalent to our proposed network on choosing the trunk net as a one-layer network of width p and each branch net as a one-hidden-layer network of width n. Hence, we essentially replace the shallow networks in Theorem 1 with deep networks in Fig. 1c to gain expressivity. We note that the trunk network also applies activation functions to the last layer, that is, $t_k = \sigma(\cdot)$ 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. In equation (1) we also note that the last layer of each b_k branch network does not have bias. Although bias is not included in Theorem 1, adding bias may increase the performance by reducing the generalization error (Fig. 2). In addition to adding bias to the branch networks, we also add a bias $b_0 \in \mathbb{R}$ in the last stage: $G(u)(y) \approx \sum_{k=1}^{p} b_k t_k + b_0$.

In practice, p is at least of order 10, and using lots of branch networks is inefficient. Hence, we merge all the branch networks into one single branch network (Fig. 1d), that is, 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 in parallel, so we name it 'stacked DeepONet', while we refer to the second DeepONet (Fig. 1d) as 'unstacked DeepONet'. An unstacked DeepONet can be viewed as a stacked DeepONet with all the branch nets sharing the same set of parameters. We note that our proposed DeepONets are also universal approximators for nonlinear operators, because none of our modifications decrease the network expressivity compared to the network in Theorem 1. All versions of DeepONets are implemented in DeepXDE[22], a user-friendly Python library designed for scientific machine learning. The loss function we use is the mean squared error (m.s.e.) between the true value of $G(u)(y)$ and the network prediction for the input $([u(x_1), u(x_2), \ldots, u(x_m)], y)$.

We developed two versions of DeepONets by extending the network architecture in Theorem 1, and next we prove that the proposed DeepONets are also universal approximators for operators in Theorem 2. Although Theorem 1 only considers shallow networks, Theorem 2 allows different branch/trunk networks.

Theorem 2 (Generalized Universal Approximation Theorem for Operator). *Suppose that 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 $C(K_1)$. Assume that $G: V \to C(K_2)$ is a nonlinear continuous operator. Then, for any $\epsilon > 0$, there exist positive integers m, p, continuous vector functions $\mathbf{g}: \mathbb{R}^m \to \mathbb{R}^p$, $\mathbf{f}: \mathbb{R}^d \to \mathbb{R}^p$, and $x_1, x_2, \ldots, x_m \in K_1$, such that*

$$\left| G(u)(y) - \left\langle \underbrace{\mathbf{g}(u(x_1), u(x_2), \cdots, u(x_m))}_{\text{branch}}, \underbrace{\mathbf{f}(y)}_{\text{trunk}} \right\rangle \right| < \epsilon$$

holds for all $u \in V$ and $y \in K_2$, where $\langle \cdot, \cdot \rangle$ denotes the dot product in $\mathbb{R}^p$. Furthermore, the functions $\mathbf{g}$ and $\mathbf{f}$ can be chosen as diverse classes of neural networks, which satisfy the classical universal approximation theorem of functions, for example, (stacked/unstacked) fully connected neural networks, residual neural networks and convolutional neural networks.

Proof. The proof can be found in Supplementary Section 2.

As stated in Theorem 2, DeepONet is a high-level network architecture without defining the architectures of its inner trunk and

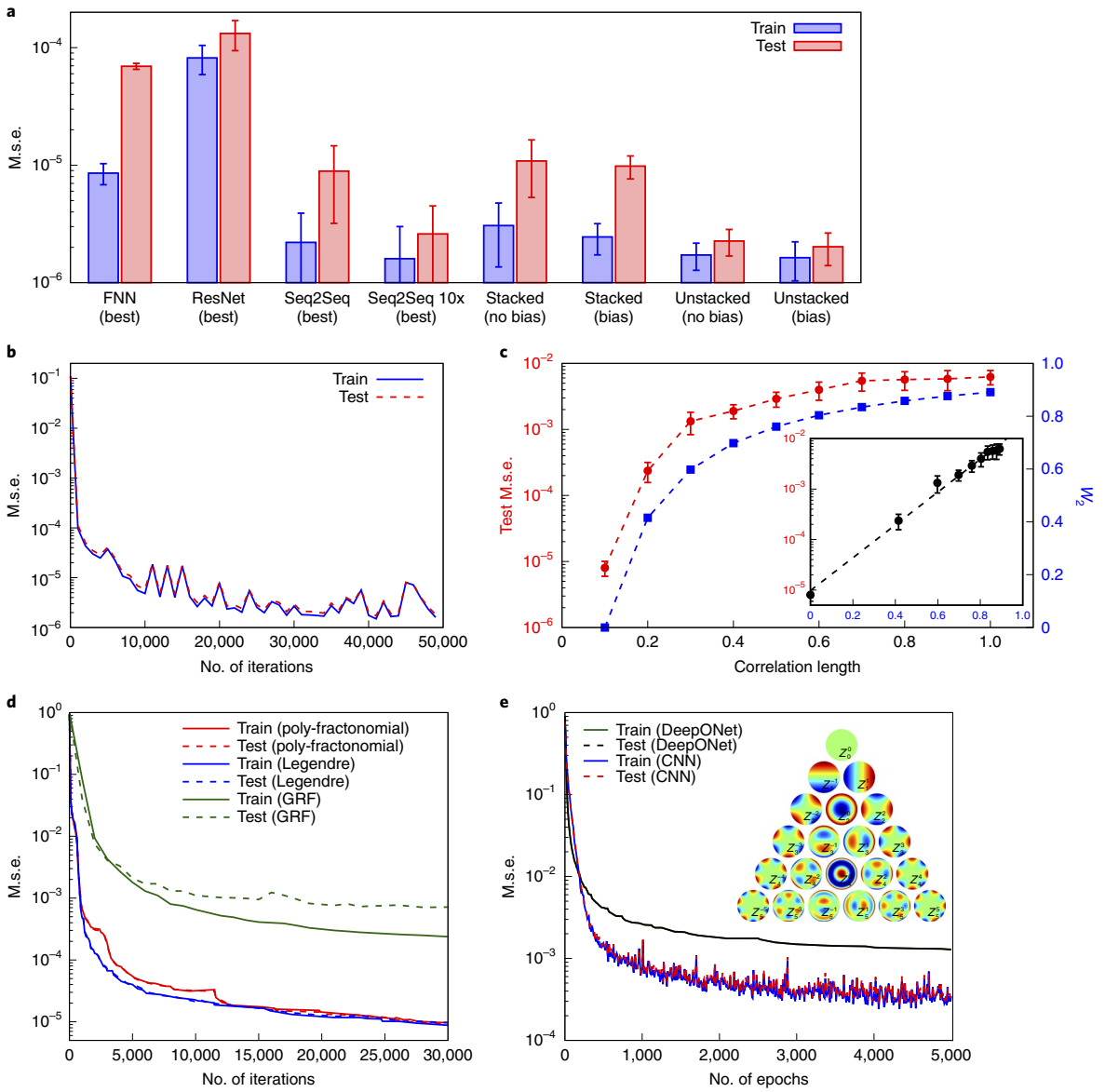

Fig. 2 | Learning explicit operators using different *V* spaces and different network architectures. a, Errors of different network architectures trained to learn the antiderivative operator (linear case). The training/test errors of stacked/unstacked DeepONets with/without bias compared with the best test error and the corresponding training error of FNNs, ResNets and Seq2Seq models. The 'Seq2Seq 10×' is a Seq2Seq model with 10 times more training data points. The error bars show the one standard deviation from 10 runs with different training/test data and network initialization. **b**, The training trajectory of an unstacked DeepONet with bias (m.s.e., mean squared error). **c**, The error (mean and standard deviation) tested on the space of Gaussian random fields (GRFs) with the correlation length $l = 0.1$ for DeepONets trained with GRF spaces of different correlation length l (red curve). The 2-Wasserstein metric between the GRF of $l = 0.1$ and a GRF of different correlation length l is shown as a blue curve. The test error grows exponentially with respect to the W_2 metric (inset). **d**, Learning the Caputo fractional derivative: poly-fractonomials versus Legendre versus GRF. **e**, Learning the fractional Laplacian on a disk. The V space consists of the Zernike polynomials.

branch networks. To demonstrate the capability and good performance of DeepONet alone, we choose the simplest FNN for the architectures of the subnetworks in this study. If the input function has a certain structure, then it is possible that by using some specialized layers we could further improve the accuracy. For example, if $\{x_1, x_2, \ldots, x_m\}$ are on an equispaced grid, then we can use convolutional layers in the branch net. Other network designs may also be considered in DeepONets, for example, the 'attention' mechanism[40].

Embodying some prior knowledge into neural network architectures usually induces good generalization. This inductive bias has been reflected in many networks, such as CNNs for images and RNNs for sequential data. The success of DeepONet, even when using FNNs for its subnetworks, 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 conditioned on u, and thus DeepONets can be viewed as a conditional model, where the embedding of u (the output of the branch net) and the embedding of y (the output of the trunk net) are merged at the end through a dot product operation. Finding an effective way to represent the conditioning input and merge the embeddings is still an open question, and different approaches have been proposed, such as feature-wise transformations[41].

Data generation. The input function $u(x)$ plays an important role in learning operators. In this study, we mainly consider the following function spaces: Gaussian random fields (GRFs), spectral representations and formulating the input functions as images (for more details see Supplementary Section 3). We note that one data point is a triplet $(u, y, G(u)(y))$, and thus a specific input u may appear in multiple data points with different values of y. For example, a dataset of size 10,000 may only be generated from 100 u trajectories, and each evaluates $G(u)(y)$ at 100 different y locations. Moreover, the number and locations of y could be different for different u. In our dataset, for each u we randomly select P different y points in the domain of $G(u)$, and thus the total number of data points is equal to $P\times\#u$.

In DeepONets, we use $[u(x_1), u(x_2), \ldots, u(x_m)]$ as the input of the branch net to represent $u(x)$, and we should estimate how many sensors m are required to achieve a good accuracy ε. We consider the following ODE system:

$$\text{Problem 1} \qquad \begin{cases} \frac{\mathrm{d}}{\mathrm{d}x}\mathbf{s}(x) = \mathbf{g}(\mathbf{s}(x), u(x), x) \\ \mathbf{s}(a) = \mathbf{s}_0 \end{cases} \tag{2}$$

where $u\in V$ (a compact subset of $C[a, b]$) is the input signal, and $\mathbf{s} : [a, b] \to \mathbb{R}^K$ is the solution of system (2) serving as the output signal.

Let G be the operator mapping the input u to the output $\mathbf{s}$, that is, $G(u)$ satisfies

$$G(u)(x) = \mathbf{s}_0 + \int_a^x \mathbf{g}(G(u)(t), u(t), t)\mathrm{d}t$$

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

$$\begin{aligned} u_m(x) &= u(x_j) + \tfrac{u(x_{j+1})-u(x_j)}{x_{j+1}-x_j}(x - x_j),\ x_j \le x \le x_{j+1},\ j \\ &= 0,\ 1,\ \ldots, m-1 \end{aligned}$$

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 a compact subset of $C[a, b]$, since V is compact and continuous operator $\mathcal{L}_m$ keeps the compactness. Obviously, $W_m := V\cup U_m$, as the union of two compact sets is also compact. Then, set $W := \bigcup_{i=1}^{\infty} W_i$, and the lemma in Supplementary Section 5 points out that W is still a compact set. Because G is a continuous operator, $G(W)$ is compact in $C([a, b];\mathbb{R}^K)$. Let $\mathcal{B}[G(W)]$ and $\mathcal{B}[W]$ be the unions of the ranges of the functions in $G(W)$ and W, respectively, and then $\mathcal{B}[G(W)]$ and $\mathcal{B}[W]$ are compact due to the compactness of $G(W)$ and W. For convenience of analysis, we assume that $\mathbf{g}(\mathbf{s}, u, x)$ satisfies the Lipschitz condition with respect to $\mathbf{s}$ and u on $\mathcal{B}[G(W)] \times \mathcal{B}[W]$, that is, there is a constant $c>0$ such that

$$\begin{aligned} \|\ \mathbf{g}(\mathbf{s}_1, u, x) - \mathbf{g}(\mathbf{s}_2, u, x)\|_2 &\le c\ \|\ \mathbf{s}_1 - \mathbf{s}_2\|_2 \\ \|\ \mathbf{g}(\mathbf{s}, u_1, x) - \mathbf{g}(\mathbf{s}, u_2, x)\|_2 &\le c|u_1 - u_2| \end{aligned}$$

Note that this condition is easy to achieve, for example as long as $\mathbf{g}$ is differentiable with respect to $\mathbf{s}$ and u on $\mathcal{B}[G(W)] \times \mathcal{B}[W]$.

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

$$\max_{x\in[a,b]}|u(x) - u_m(x)| \le \kappa(m, V), \quad \kappa(m, V) \to 0 \text{ as } m \to \infty \tag{3}$$

When V is a GRF with the Gaussian kernel, we have $\kappa(m, V) \sim \frac{1}{m^{2l^2}}$ (for the proof see Supplementary Section 4). Based on the these concepts, we have the following theorem.

Theorem 3. *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*

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

holds for all $u\in V$.

Proof. The proof can be found in Supplementary Section 5.

Results and discussion

To demonstrate the capability and efficiency of DeepONets, we will learn 16 different operators via DeepONets, including diverse linear/nonlinear explicit and implicit operators. Specifically, the four explicit operators include the integration, Legendre transform, one-dimensional (1D) fractional derivative, and 2D fractional Laplacian. The implicit operators include eight operators of deterministic ODEs (nonlinear ODE and the gravity pendulum) and PDEs (diffusion-reaction, advection and advection-diffusion) and four operators of two stochastic differential equations for both pathwise solutions and statistics. For the ODEs and PDEs, the input function of the operators could be the boundary conditions, initial conditions or forcing terms. We have carefully selected these diverse tests to probe the generalization error of DeepONet, investigate proper representations of the input space V including non-compact spaces, compare different DNN architectures and consider both explicit and implicit mathematical operators. All details of these tests are provided in the Supplementary Information to assist the readers to readily reproduce our results. In ongoing work, we have employed DeepONets to simulate various multiscale and multiphysics systems, and we will report this work in future publications.

We first show how to learn explicit operators, and demonstrate small generalization error for different representations of the discrete input space V. We then present how to learn implicit operators. The parameter values for all examples are listed in Supplementary Section 7, and verification of the Hölder continuity of all the explicit and implicit operators considered in this study is presented in Supplementary Section 18.

Learning explicit operators. First, we consider a pedagogical example described by

$$\frac{\mathrm{d}s(x)}{\mathrm{d}x} = g(s(x), u(x), x), \quad x \in (0, 1]$$

with an initial condition (IC) $s(0)=0$. Our goal is to predict $s(x)$ over the whole domain $[0, 1]$ for any $u(x)$. We first consider a linear problem by choosing $g(s(x), u(x), x)=u(x)$, which is equivalent to learning the antiderivative operator:

$$\begin{aligned} \text{Problem 1.A} \qquad \tfrac{\mathrm{d}s(x)}{\mathrm{d}x} &= u(x) \text{ and } G : u(x) \mapsto s(x) \\ &= s_0 + \textstyle\int_0^x u(\tau)\mathrm{d}\tau,\ x \in [0, 1] \end{aligned} \tag{4}$$

We train FNNs and ResNets to learn the antiderivative operator, where the network input is a concatenation of $u(x)$ and y and the output is $G(u)(y)$. To obtain the best performance of FNNs, we grid-search three hyperparameters: depth from 2 to 4, width from 10 to 2,560 and learning rate from 0.0001 to 0.01. The m.s.e.s of the test dataset with learning rates 0.01, 0.001 and 0.0001 are shown in Supplementary Fig. 1. Although we only choose depth up to 4, the results show that increasing the depth further does not improve the test error. Among all these hyperparameters, the smallest test error of $\sim 7\times10^{-5}$ is obtained for the network with depth 2, width 2,560 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

the small expressivity. When the network size increases, the training error decreases, but the generalization error increases. We stop the training before FNNs reach the overfitting region, where the test error increases. Similarly, for ResNets, we grid-search the two hyperparameters: the number of residual blocks from 1 to 5 and width from 10 to 320. We note that one residual block includes two dense layers with a shortcut connection, and each ResNet has one hidden layer before the first residual block and one hidden layer after the last residual block[39], and thus a ResNet has in total (3 + 2# residual block) layers. We chose a learning rate of 0.001 based on previous results for FNNs. Among all these hyperparameters, the smallest test error of $\sim 1 \times 10^{-4}$ is obtained for the ResNet with one residual block and width 20. In addition to FNNs and ResNets, we also considered sequence-to-sequence (Seq2Seq) models[42–44] using the long short-term memory (LSTM) or the gated recurrent units (GRUs) with attention mechanism as the baseline. More details of Seq2Seq are presented in Supplementary Section 16.

Compared to FNNs, ResNets and Seq2Seq, DeepONets have much smaller generalization error and thus smaller test error (Fig. 2a and Supplementary Table 6). Seq2Seq is better than FNNs and ResNets, and has performance similar to stacked DeepONets—for ~10 times more data points, the performance of Seq2Seq is close to unstacked DeepONets. We note that the datasets used for DeepONets and Seq2Seq are not exactly the same, because Seq2Seq requires that the output function evaluations have to be on a dense grid ($P = 100$), while DeepONet does not have this constraint and uses a dataset of $P = 1$ only. Unstacked DeepONets have performance similar to Seq2Seq if we train DeepONets on the same dense dataset, which is not required for DeepONets as they work with sparse datasets. Moreover, the training of Seq2Seq is much more expensive (more than seven times) than the training of DeepONets (Supplementary Table 4).

Here, we do not aim to find the best hyperparameters of DeepONets, and only test the performance of the stacked and unstacked DeepONets listed in Supplementary Table 3. One training trajectory of an unstacked DeepONet with bias is shown in Fig. 2b, and the generalization error is negligible. We observe that for both stacked and unstacked DeepONets, adding bias to branch networks reduces both training and test errors (Fig. 2a). DeepONets with bias also have smaller uncertainty; that is, they are more stable for training from random initialization (Fig. 2a). Compared to stacked DeepONets, unstacked DeepONets have smaller test error due to the smaller generalization error. Therefore, unstacked DeepONets with bias achieve the best performance. In addition, unstacked DeepONets have fewer parameters than stacked DeepONets, and thus can be trained more quickly using much less memory. We also provide examples of the Legendre transform in Supplementary Section 10 and nonlinear ODEs in Supplementary Section 11, as well as more details on the accuracy and architecture. In the following study, we will use unstacked DeepONets.

When we train a neural network, we usually generate a training dataset from the same space as the functions that we will predict, namely, interpolation. We have shown that DeepONets perform very well for interpolation (Fig. 2a,b). Next, we will test DeepONets for the situation where the training dataset and test dataset are not sampled from the same function space. Specifically, the input functions in the training dataset are sampled from the space of a GRF with the covariance kernel $k_l(x_1, x_2) = \exp(-\| x_1 - x_2 \|^2 / 2l^2)$, where l is the length-scale parameter. After the network is trained, instead of testing functions also in this space, we will always use the functions sampled from the space of the GRF with $l = 0.1$ for testing. To quantify the difference between two GRF spaces of different correlation lengths, we use the 2-Wasserstein (W_2) metric[45] to measure their distance (details are provided in Supplementary Section 6). When l is large, the W_2 metric is large (blue curve, Fig. 2c), and the test error is also large (red curve, Fig. 2c). When l is smaller, the W_2 metric and test error become smaller. If $l = 0.1$, the training and test spaces are the same, and thus the W_2 metric is 0. It is interesting that the test m.s.e. grows exponentially with respect to the W_2 metric (inset, Fig. 2c).

In addition to the aforementioned integral operator, we consider integro-differential operators, namely, fractional differential operators, to demonstrate the flexibility of DeepONets to learn more complicated operators. The first fractional differential operator we learn is the 1D Caputo fractional derivative[46]:

Problem 2

$$G(u)(y, \alpha) : u(x) \mapsto s(y, \alpha) = \frac{1}{\Gamma(1-\alpha)} \int_0^y (y - \tau)^{-\alpha} u'(\tau) \mathrm{d}\tau,$$

$$y \in [0, 1], \alpha \in (0, 1)$$

where α and $u'(\cdot)$ are the fractional order and first derivative of u, respectively. The domain of the output function now includes two variables y and α. We concatenate y and α to form an augmented $\hat{y} = [y, \alpha]^T$ and then feed $\hat{y}$ to the trunk net. We consider the influence of different V spaces on the generalization error of DeepONets. These spaces include two orthogonal polynomial spaces spanned by poly-factonomials[47] and Legendre polynomials, as well as the GRF. More details are provided in Supplementary Section 17. Figure 2d shows the generalization errors for the three different V spaces. We see that small generalization errors are achieved for all of the cases. The generalization error for the GRF is slightly larger than those for orthogonal polynomial spaces.

The second fractional differential operator we learn is the 2D Riesz fractional Laplacian[48]:

Problem 3

$$G(u)(y, \alpha) : u(x) \mapsto s(y, \alpha)$$

$$= \frac{2^\alpha \Gamma(1 + \frac{\alpha}{2})}{\pi |\Gamma(-\frac{\alpha}{2})|} \times \text{p.v.} \int_{\mathbb{R}^2} \frac{u(y) - u(\tau)}{\|y - \tau\|_2^{2+\alpha}} \mathrm{d}\tau, \quad \alpha \in (0, 2)$$

where 'p.v.' means principle value. The input and output functions are both assumed to be identically zero outside of a unit disk centred at the origin. The 2D fractional Laplacian reduces to standard Laplacian Δ as the fractional order α goes to two. For learning this operator, we specify the V space to be the orthogonal space spanned by the Zernike polynomials[49], which are commonly used to generate or approximate functions defined on a unit disk. Figure 2e (inset) displays the first 21 Zernike polynomials (for a more detailed description of the polynomial expansions see Supplementary Section 3).

We consider two different NN architectures: unstacked DeepONets versus CNNs. In this problem, we generated a dataset such that the input and output functions are evaluated on an equispaced grid, such that a CNN can also be used. For the DeepONet, in a similar manner as for handling the 1D Caputo derivative case, we feed the augmented $\hat{y} = [y, \alpha]$ to the trunk net. For the CNN architecture, we rearrange the values of input and output functions to 2D images in which 'pixel' (or function) values are attached to a lattice in the polar coordinate. We first utilize a CNN as an encoder to extract the features of the input image, which reduces the high-dimensional input space to a low-dimensional latent space, and then we employ another CNN as a decoder to map the vector in the latent space to the output space. To accommodate the extra parameter α, we set the image consisting of values of $G(y, \alpha_k)$ for kth α as the kth channel of the output image. As such, we obtain a multi-channel output image. We observe from Fig. 2e that both architectures yield small generalization errors. Moreover, the CNN architecture exhibits slightly higher accuracy than the DeepONet. This is because the former sufficiently takes advantage of the spatial structure of the training data. Nevertheless, DeepONet is more flexible than CNN for unstructured data, as we commented in the

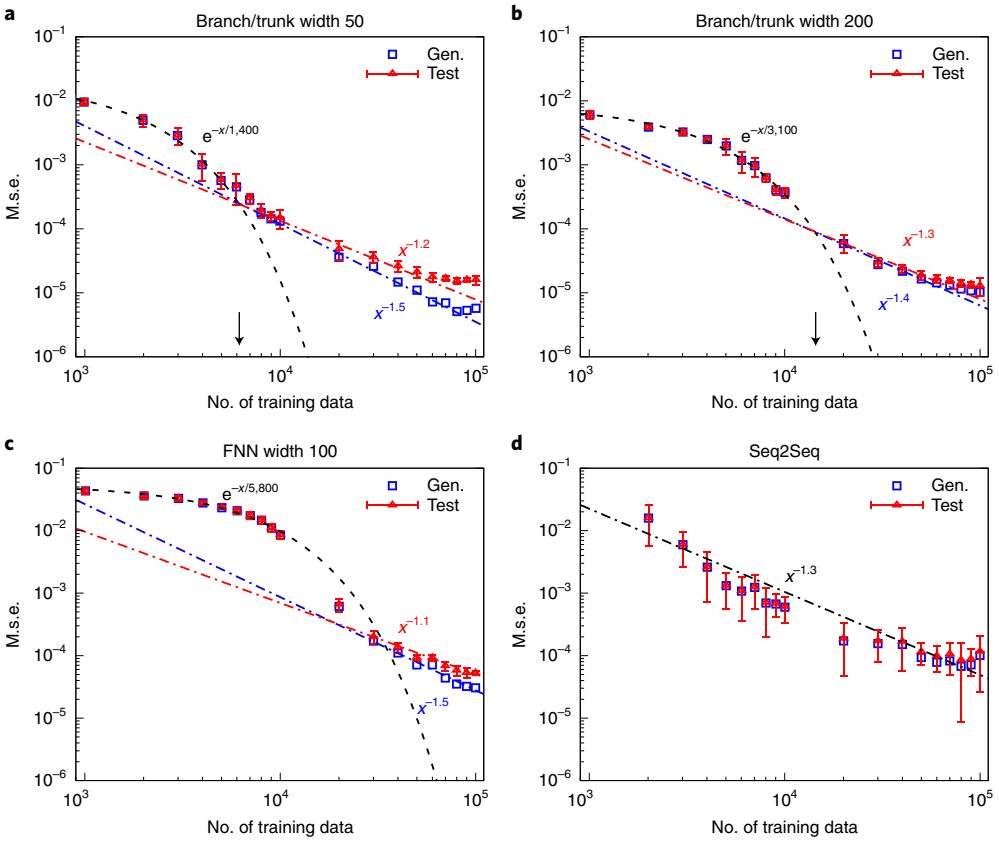

Fig. 3 | Fast learning of implicit operators in a nonlinear pendulum ($k=1$ and $T=3$). a,**b**, The test and generalization errors of DeepONets have exponential convergence for small training datasets, and then converge with polynomial rates. The transition point from exponential to polynomial (indicated by the arrow) convergence depends on the width (branch/trunk width of 50 in **a** and 200 in **b**), and a bigger network has a later transition point. **c**, FNNs also have an initial exponential error decay, but with much larger error and much slower convergence speed. **d**, Seq2Seq models have a roughly polynomial convergence rate, and the test errors have a large variation for different runs. x is the number of training data.

preceding paragraphs. More details are presented in Supplementary Section 17.

DeepONet learns fast. An important question for the effectiveness of DeepONet is how fast it learns new operators. We investigate this question by learning a system of nonlinear ODEs first and subsequently a nonlinear PDE. First, we consider the motion of a gravity pendulum with an external force described by

$$\text{Problem 1.B} \qquad \frac{\mathrm{d}s_1}{\mathrm{d}t} = s_2, \qquad \frac{\mathrm{d}s_2}{\mathrm{d}t} = -k\sin s_1 + u(t)$$

with an initial condition $\mathbf{s}(0)=\mathbf{0}$, and k 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 and (4) optimizer. We investigate the effects of all these factors on the accuracy of DeepONet in Supplementary Section 12 and verify our analysis on the number of sensors, but here we focus on the convergence rate of the training process.

The test and generalization errors of DeepONets with different trunk/branch width are shown in Fig. 3a,b. It is surprising that the test and generalization errors have exponential convergence for small training dataset size. Even for a large dataset, the polynomial convergence rates are still higher than the classical $x^{-0.5}$ in learning theory[50]. This fast convergence reveals that DeepONets learn exponentially fast, especially in the region of a small dataset. Moreover, the transition point from exponential to polynomial convergence depends on the network size, and a larger exponential regime can be accomplished with a sufficiently large network, but the smaller network has higher accuracy for a small dataset. We cannot yet explain theoretically this convergence behaviour, but we speculate that it may be related to the theory of information bottleneck[51].

We have observed a fast exponential and polynomial error convergence for DeepONets. For comparison we checked the error decay of two baseline models (FNN and Seq2Seq) with the same set-up. We chose an FNN of depth 3 and width 100, which has size similar to that of the DeepONet with branch/trunk width of 50 in Fig. 3a. Details for the Seq2Seq are provided in Supplementary Section 16. We also observed a similar initial exponential converge of FNNs (Fig. 3c), but, compared with DeepONets, FNNs have a much larger test error and much slower convergence speed. However, Seq2Seq models do not exhibit the exponential convergence regime and, overall, the test errors are larger with larger variation.

Next, we learn an implicit operator in the form of a nonlinear diffusion-reaction PDE with a source term $u(x)$ described by

$$\text{Problem 4} \qquad \frac{\partial s}{\partial t} = D\frac{\partial^2 s}{\partial x^2} + ks^2 + u(x), \quad x \in (0,1), t \in (0,1]$$

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 used one random

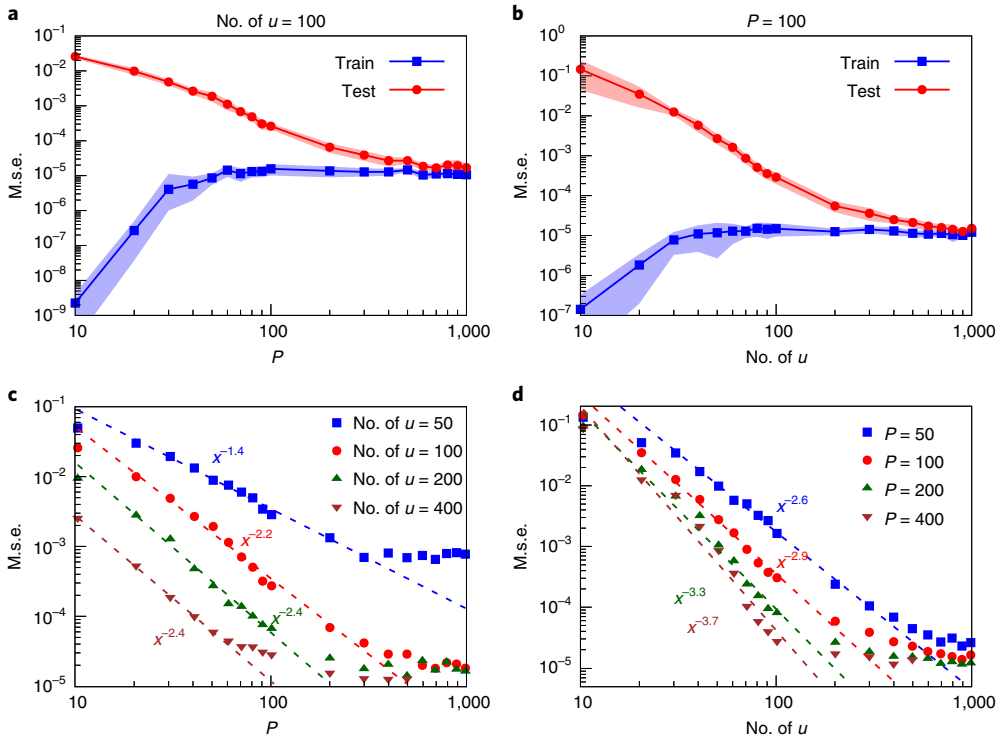

Fig. 4 | Fast learning of implicit operators in a diffusion-reaction system. a,**b**, Comparison of training (blue) and testing (red) errors for different values of the number of random points P when 100 random u samples are used (**a**) and for different numbers of u samples when $P=100$ (**b**). The shaded regions denote one standard deviation. **c**,**d**, The algebraic decay of test errors in terms of the number of sampling points P and the number of input functions $u(x, t)$: convergence of test error with respect to P for different numbers of u samples (**c**) and with respect to the number of u samples for different values of P (**d**).

point of $s(x)$ for training; 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 100×100 grid, and then for each s we randomly select P points from these 10,000($=100\times100$) grid points (Supplementary Fig. 8). Hence, the dataset size is equal to the product of P and 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 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. 4a), 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. 4b). In addition, in this PDE problem the DeepONet is able to learn from a small dataset; for example, a DeepONet can reach a test error of $\sim10^{-5}$ when it is only trained with 100 u samples ($P=1{,}000$). We recall that we test on 10,000 grid points and thus, on average, each location point only has $100\times1{,}000/10{,}000=10$ training data points. For comparison, we train ResNets of different sizes with the same set-up, and the test error is $\sim10^{-1}$ due to the large generalization error (Supplementary Table 7).

Before the error saturates, the rates of convergence with respect to both P and the number of u samples obey a polynomial law in most of the range (Fig. 4c,d). The rate of convergence versus P depends on the number of u samples, and more u samples induce faster convergence until saturation (blue line, Supplementary Fig. 9d). Similarly, the rate of convergence versus the number of u samples depends on the value of P (red line, Supplementary Fig. 9d). In addition, in the initial range of the convergence, we observe an exponential convergence (Supplementary Fig. 9a,b). The coefficient $1/k$ in the exponential convergence $e^{-x/k}$ also depends on the number of u samples or the value of P (Supplementary Fig. 9c). It is reasonable that the convergence rate presented in Supplementary Fig. 9c,d increases with the number of u samples or the value of P, because the total number 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} (Supplementary Fig. 9c,d), which we cannot fully explain yet, and hence more theoretical and computational investigations are required.

Here, when studying the convergence of DeepONets, we fix either $\#u$ or P and increase the other one. We show that the same exponential and polynomial convergence behaviour is also observed (Supplementary Fig. 10) when we keep $P=\#u$ to generate the training dataset; that is, the number of training data points is equal to $P\times\#u=P^2=(\#u)^2$. We also provide examples of the advection equation in Supplementary Section 14 and the advection-diffusion equation in Supplementary Section 15.

Learning stochastic operators. Next, we demonstrate that we can learn high-dimensional operators, so here we consider a stochastic ODE and a stochastic PDE and present our main findings.

Consider the population growth model

$$\text{Problem 5}\qquad \mathrm{d}y(t;\omega)=k(t;\omega)y(t;\omega)\mathrm{d}t,\quad t\in(0,1]\text{ and }\omega\in\Omega \tag{5}$$

with $y(0)=1$. Here, Ω is the random space. The randomness comes from the coefficient $k(t;\omega)$. Here, $k(t;\omega)$ is modelled as a Gaussian random process such that

$$k(t;\omega)\sim\mathcal{GP}(k_0(t),\ \mathrm{Cov}(t_1,t_2))$$

where the mean $k_0(t)=0$ and the covariance function is $\mathrm{Cov}(t_1,t_2)=\sigma^2\exp(-\|t_1-t_2\|^2/2l^2)$. We choose $\sigma=1$, and the correlation length l is in the range $[1, 2]$.

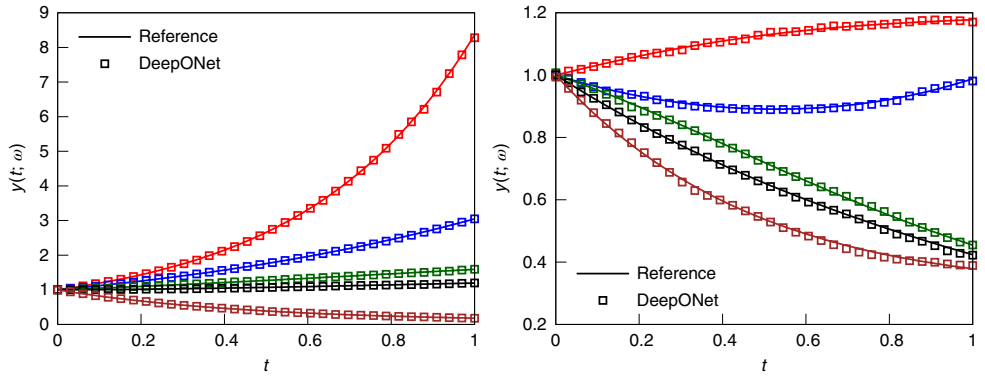

Fig. 5 | DeepONet prediction for a stochastic ODE. The DeepONet prediction (symbols) is very close to the reference solution for 10 different random samples (five in each panel) from $k(x; \omega)$ with $l = 1.5$.

We use DeepONet to learn the operator mapping from $k(t; \omega)$ of different correlation lengths to the solution $y(t; \omega)$. We note that we do not assume we know the covariance function for training DeepONet. The main differences between this example and the previous examples are as follows: (1) here, the input of the branch net is a random process instead of a function and (2) the input of the trunk net contains both physical spaces and random spaces. Specifically, to handle the random process as the input, we employ the Karhunen–Loève (KL) expansion:

$$k(t;\omega) \approx \sum_{i=1}^{N} \sqrt{\lambda_i} e_i(t) \xi_i(\omega)$$

where N is the number of retained modes, λ_i and $e_i(t)$ are the ith largest eigenvalue and its associated normalized eigenfunction of the covariance function, respectively, and $\xi_1, \ldots, \xi_N$ are independent standard Gaussian random variables. Then, the input of the branch net is the N eigenfunctions scaled by the eigenvalues:

$$[\sqrt{\lambda_1} e_1(t), \sqrt{\lambda_2} e_2(t), \ldots, \sqrt{\lambda_N} e_N(t)] \in \mathbb{R}^{N \times m}$$

where $\sqrt{\lambda_i} e_i(t) = \sqrt{\lambda_i}[e_i(t_1), e_i(t_2), \ldots, e_i(t_m)] \in \mathbb{R}^m$, and the input of the trunk net is $[t, \xi_1, \xi_2, \ldots, \xi_N] \in \mathbb{R}^{N+1}$. We note that, by using the KL expansion, the randomness is only in ξ_i and becomes the input of DeepONet. However, this problem remains challenging because its dimension is very high; the input of the branch net contains N orthogonal functions instead of one function, and the dimension of the input of the trunk net is the sum of the dimensions of the physical space and random space.

We choose $N = 5$, which is sufficient to conserve 99.9% stochastic energy. We train a DeepONet with a dataset of 10,000 different $k(t; \omega)$ with l randomly sampled in $[1, 2]$, and for each $k(t; \omega)$ we use only one realization. The test m.s.e. is $8.0 \times 10^{-5} \pm 3.4 \times 10^{-5}$. As an example, the prediction for 10 different random samples from $k(t; \omega)$ with $l = 1.5$ is shown in Fig. 5, and the average L^2 relative error is ~0.5%.

Next, we consider the following elliptic problem with multiplicative noise:

$$\text{Problem 6} \quad -\operatorname{div}(e^{b(x;\omega)} \nabla u(x;\omega)) = f(x), \quad x \in (0, 1) \text{ and } \omega \in \Omega$$

with Dirichlet boundary conditions $u(0) = u(1) = 0$. The randomness comes from the diffusion coefficient $e^{b(x;\omega)}$, and $b(x;\omega) \sim \mathcal{GP}(b_0(x), \operatorname{Cov}(x_1, x_2))$, where the mean $b_0(x) = 0$ and the covariance function is $\operatorname{Cov}(x_1, x_2) = \sigma^2 \exp(-\| x_1 - x_2 \|^2 / 2l^2)$. In this example, we choose a constant forcing term $f(x) = 10$, and we set the standard deviation as $\sigma = 0.1$ and correlation length in the range $[1, 2]$.

We use DeepONet to learn the operator mapping from $b(x; \omega)$ of different correlation lengths to the solution $u(x; \omega)$. We train a DeepONet with a dataset of 10,000 different $b(x; \omega)$ with l randomly sampled in $[1, 2]$, and for each $b(x, \omega)$ we use only one realization. The test m.s.e. is $2.0 \times 10^{-3} \pm 1.7 \times 10^{-3}$. As an example, the prediction for 10 different random samples from $b(x; \omega)$ with $l = 1.5$ is presented in Fig. 6a, and the average L^2 relative error is ~1.1%. We have a relative larger error in this stochastic elliptic problem and, to reduce the error further, we can remove outliers of the random variables $\{\xi_1, \xi_2, \ldots, \xi_N\}$ by clipping them to a bounded domain. For example, if the random variables are clipped to $[-3.1, 3.1]$, which is sufficient large to keep 99% probability space, then the test m.s.e. is reduced to $9.4 \times 10^{-4} \pm 3.0 \times 10^{-4}$.

In the aforementioned stochastic ODE and PDE problems, we have used DeepONets to predict the pathwise solutions. Next, we will apply DeepONets to predict statistical averages of the stochastic solution, for example, the mean and standard deviation. We use the same set-up as in Problem 6, except that we consider a larger range $[0.2, 2]$ for the correlation length. We also employ the KL expansion to handle the random process, and we keep the input of the branch net the same as in the previous problems, but for the trunk net the input is only t without random variables ξ_i. Because we consider a smaller correlation length, $N = 8$ modes are required to conserve 99.9% stochastic energy. We show that DeepONets can predict the mean and standard deviation accurately, and as an example the predictions for three values of the correlation length are shown in Fig. 6b,c.

In summary, the above method for stochastic ODEs and PDEs can be used for any colour noise where the KL expansion can be employed. For stochastic ODEs/PDEs with white noise, a different approach may be required.

Conclusions

We have formulated the problem of learning operators in a general set-up and have proposed DeepONets to learn diverse linear/nonlinear explicit and implicit operators. In DeepONets, we first construct two subnetworks to encode input functions and location variables separately, and then merge them together to compute the output. We test DeepONets on learning explicit operators, including fractional operators, as well as implicit operators in the form of deterministic and stochastic ODEs and PDEs. Our two main findings are that the generalization error is small and that the training and testing errors decay quickly with respect to the training data size. In fact, we observed exponentially fast learning for sufficiently large networks and transition to standard convergence for large datasets. In our simulations, we systematically studied the effects on the test error of different factors, including the number of sensors, maximum prediction time, the complexity of the space of

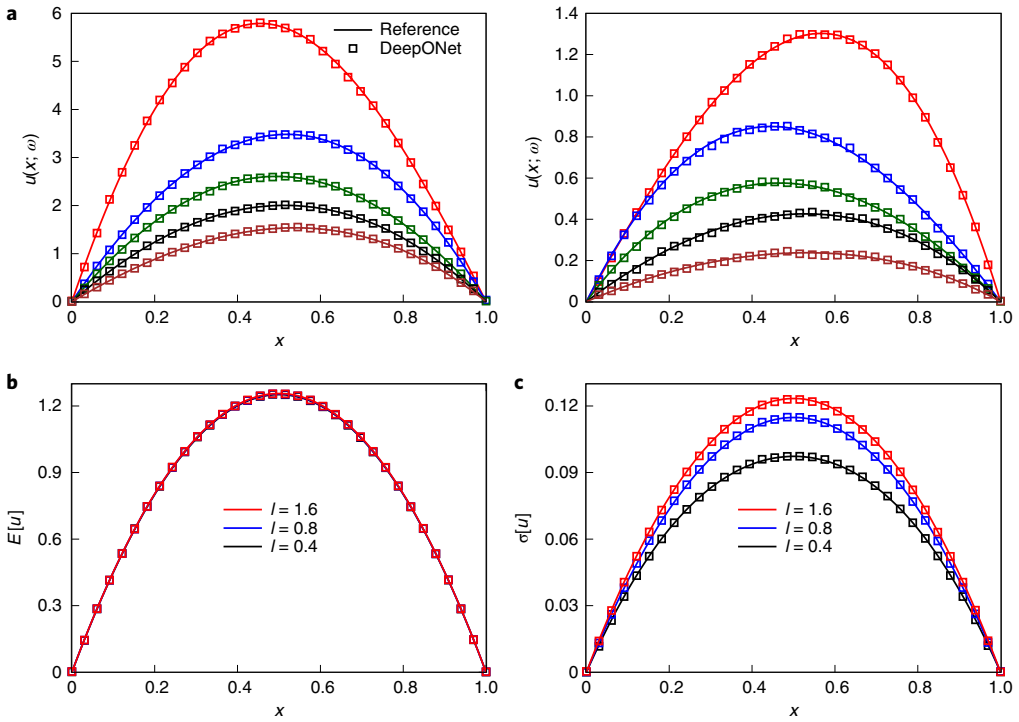

Fig. 6 | DeepONet prediction for a stochastic elliptic equation. a, The DeepONet prediction (symbols) is very close to the reference solution for 10 different random samples (five in each panel) from $b(x;\omega)$ with $l=1.5$. **b**,**c**, The DeepONet prediction (symbols) of the mean $E[u]$ (**b**) and standard deviation $\sigma[u]$ (**c**) of the solution for $k(x;\omega)$ with $l=0.4$ (black), 0.8 (blue) and 1.6 (red). $E[u]$ of different correlation lengths collapse to the same line.

input functions, training dataset size and network size. Moreover, we derived, theoretically, the dependence of approximation error on different factors, which is consistent with our computational results. In this study, all our training datasets are obtained from numerical solvers; that is, the trained DeepONet is a surrogate of the numerical solver. Hence, DeepONets cannot be better than the numerical solvers in terms of accuracy. However, as we show in Supplementary Table 5, the computational cost of running inference of DeepONet is substantially lower than for the numerical solver.

More broadly, a DeepONet could represent a multiscale operator trained with data spanning several orders of magnitude in spatio-temporal scales, for example, trained by data from the molecular, mesoscopic and continuum regimes in fluid mechanics or other multiscale problems. We could also envision other types of composite DNN for developing multiphysics operators, for example, in electro-convection involving the evolution of the flow field and concentration fields of anions and cations due to continuous variation of imposed electric potentials. Indeed, in ongoing work, we have developed an extension of DeepONet for simulating this electro-convection multiphysics problem[52], where we show that DeepONet is substantially faster than a spectral element solver. We have obtained a similar speed-up and high accuracy in hypersonics for learning the aerodynamics coupled with the finite rate chemistry of multiple species. Learning such multiscale and multiphysics nonlinear operators will simplify computational modelling and will facilitate very fast predictions of complex dynamics for unseen new parameter values and new input functions (boundary/initial conditions/excitation) with good accuracy, if the generalization error is bounded.

Despite the reported progress in this first Article, more work should be done both theoretically and computationally. Because the training dataset size in DeepONets is a product of the number of input functions u and the number of evaluation locations y for $G(u)$, training DeepONets for operator approximation is much more computationally intensive than training NNs for function approximation. Hence, more research is needed in speeding up the training process and in formulating efficient offline training strategies, including perhaps transfer learning approaches. Here, we have employed known operators to evaluate the accuracy of DeepONet systematically; however, the real strength of DeepONet is that it can discover new operators that are trained by multi-fidelity data or by heterogeneous sources of experimental data and simulation data. We could also endow the operator G with prior knowledge, for example, translational and rotational invariances as in CNNs for imaging[53]. On the theoretical side, there have not been any results on the network size for operator approximation, similar to the bounds of width and depth for function approximation[54]. We also do not yet understand theoretically why DeepONets can induce small generalization errors. On the other hand, in this Article we use fully connected neural networks for the two subnetworks, but, as we discussed in the Materials and methods, if the network input has a certain structure, we can also employ other network architectures, such as CNNs or the 'attention' mechanism. These modifications may improve further the accuracy of DeepONets, and the example of using a CNN with encoders and decoders for learning the fractional Laplacian presented here is a first such indication.

In summary, we have formulated a new framework for deep NN, based on the new Theorem 2, to learn linear and nonlinear operators implicitly as NNs. In theory, all the operators of the classical integer calculus, but also of fractional calculus, can be represented by carefully trained NNs as well as other transforms and even theorems, such as the Gauss theorem. In future work, we will present our ongoing simulation experiments in predicting very complex dynamics of multiscale and multiphysics operators, accurately and very quickly.

Reporting Summary. Further information on research design is available in the Nature Research Reporting Summary linked to this Article.

Data availability

All the datasets in the study were generated directly from the code.

Code availability

The code used in the study is publicly available from the GitHub repository https://github.com/lululxvi/deeponet[55].

Received: 14 April 2020; Accepted: 25 January 2021;
Published online: 18 March 2021

References

1. Rico-Martinez, R., Krischer, K., Kevrekidis, I. G., Kube, M. C. & Hudson, J. L. Discrete- vs. continuous-time nonlinear signal processing of Cu electrodissolution data. *Chem. Eng. Commun.* **118**, 25–48 (1992).
2. Rico-Martinez, R., Anderson, J. S. & Kevrekidis, I. G. Continuous-time nonlinear signal processing: a neural network based approach for gray box identification. In *Proc. IEEE Workshop on Neural Networks for Signal Processing* 596–605 (IEEE, 1994).
3. González-García, R., Rico-Martínez, R. & Kevrekidis, I. G. Identification of distributed parameter systems: a neural net based approach. *Comput. Chem. Eng.* **22**, S965–S968 (1998).
4. Psichogios, D. C. & Ungar, L. H. A hybrid neural network-first principles approach to process modeling. *AIChE J.* **38**, 1499–1511 (1992).
5. Kevrekidis, I. G. et al. Equation-free, coarse-grained multiscale computation: enabling mocroscopic simulators to perform system-level analysis. *Commun. Math. Sci.* **1**, 715–762 (2003).
6. Weinan, E. *Principles of Multiscale Modeling* (Cambridge Univ. Press, 2011).
7. Ferrandis, J., Triantafyllou, M., Chryssostomidis, C. & Karniadakis, G. Learning functionals via LSTM neural networks for predicting vessel dynamics in extreme sea states. Preprint at https://arxiv.org/pdf/1912.13382.pdf (2019).
8. Qin, T., Chen, Z., Jakeman, J. & Xiu, D. Deep learning of parameterized equations with applications to uncertainty quantification. Preprint at https://arxiv.org/pdf/1910.07096.pdf (2020).
9. Chen, T. Q., Rubanova, Y., Bettencourt, J. & Duvenaud, D. K. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems* 6571–6583 (NIPS, 2018).
10. Jia, J. & Benson, A. R. Neural jump stochastic differential equations. Preprint at https://arxiv.org/pdf/1905.10403.pdf (2019).
11. Greydanus, S., Dzamba, M. & Yosinski, J. Hamiltonian neural networks. In *Advances in Neural Information Processing Systems* 15379–15389 (NIPS, 2019).
12. Toth, P. et al. Hamiltonian generative networks. Preprint at https://arxiv.org/pdf/1909.13789.pdf (2019).
13. Zhong, Y. D., Dey, B. & Chakraborty, A. Symplectic ODE-Net: learning Hamiltonian dynamics with control. Preprint at https://arxiv.org/pdf/1909.12077.pdf (2019).
14. Chen, Z., Zhang, J., Arjovsky, M. & Bottou, L. Symplectic recurrent neural networks. Preprint at https://arxiv.org/pdf/1909.13334.pdf (2019).
15. Winovich, N., Ramani, K. & Lin, G. ConvPDE-UQ: convolutional neural networks with quantified uncertainty for heterogeneous elliptic partial differential equations on varied domains. *J. Comput. Phys.* **394**, 263–279 (2019).
16. Zhu, Y., Zabaras, N., Koutsourelakis, P.-S. & Perdikaris, P. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *J. Comput. Phys.* **394**, 56–81 (2019).
17. Trask, N., Patel, R. G., Gross, B. J. & Atzberger, P. J. GMLS-Nets: a framework for learning from unstructured data. Preprint at https://arxiv.org/pdf/1909.05371.pdf (2019).
18. Li, Z. et al. Neural operator: graph kernel network for partial differential equations. Preprint at https://arxiv.org/pdf/2003.03485.pdf (2020).
19. Rudy, S. H., Brunton, S. L., Proctor, J. L. & Kutz, J. N. Data-driven discovery of partial differential equations. *Sci. Adv.* **3**, e1602614 (2017).
20. Zhang, D., Lu, L., Guo, L. & Karniadakis, G. E. Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems. *J. Comput. Phys.* **397**, 108850 (2019).
21. Pang, G., Lu, L. & Karniadakis, G. E. fPINNs: fractional physics-informed neural networks. *SIAM J. Sci. Comput.* **41**, A2603–A2626 (2019).
22. Lu, L., Meng, X., Mao, Z. & Karniadakis, G. E. DeepXDE: a deep learning library for solving differential equations. *SIAM Rev.* **63**, 208–228 (2021).
23. Yazdani, A., Lu, L., Raissi, M. & Karniadakis, G. E. Systems biology informed deep learning for inferring parameters and hidden dynamics. *PLoS Comput. Biol.* **16**, e1007575 (2020).
24. Chen, Y., Lu, L., Karniadakis, G. E. & Negro, L. D. Physics-informed neural networks for inverse problems in nano-optics and metamaterials. *Opt. Express* **28**, 11618–11633 (2020).
25. Holl, P., Koltun, V. & Thuerey, N. Learning to control PDEs with differentiable physics. Preprint at https://arxiv.org/pdf/2001.07457.pdf (2020).
26. Lample, G. & Charton, F. Deep learning for symbolic mathematics. Preprint at https://arxiv.org/pdf/1912.01412.pdf (2019).
27. Charton, F., Hayat, A. & Lample, G. Deep differential system stability—learning advanced computations from examples. Preprint at https://arxiv.org/pdf/2006.06462.pdf (2020).
28. Cybenko, G. Approximation by superpositions of a sigmoidal function. *Math. Control Signals Syst.* **2**, 303–314 (1989).
29. Hornik, K., Stinchcombe, M. & White, H. Multilayer feedforward networks are universal approximators. *Neural Networks* **2**, 359–366 (1989).
30. Chen, T. & Chen, H. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Trans. Neural Networks* **6**, 911–917 (1995).
31. Chen, T. & Chen, H. Approximations of continuous functionals by neural networks with application to dynamic systems. *IEEE Trans. Neural Networks* **4**, 910–918 (1993).
32. Mhaskar, H. N. & Hahm, N. Neural networks for functional approximation and system identification. *Neural Comput.* **9**, 143–159 (1997).
33. Rossi, F. & Conan-Guez, B. Functional multi-layer perceptron: a non-linear tool for functional data analysis. *Neural Networks* **18**, 45–60 (2005).
34. Chen, T. & Chen, H. Approximation capability to functions of several variables, nonlinear functionals, and operators by radial basis function neural networks. *IEEE Trans. Neural Networks* **6**, 904–910 (1995).
35. Brown, T. B. et al. Language models are few-shot learners. Preprint at https://arxiv.org/pdf/2005.14165.pdf (2020).
36. Lu, L., Su, Y. & Karniadakis, G. E. Collapse of deep and narrow neural nets. Preprint at https://arxiv.org/pdf/1808.04947.pdf (2018).
37. Jin, P., Lu, L., Tang, Y. & Karniadakis, G. E. Quantifying the generalization error in deep learning in terms of data distribution and neural network smoothness. *Neural Networks* **130**, 85–99 (2020).
38. Lu, L., Shin, Y., Su, Y. & Karniadakis, G. E. Dying ReLU and initialization: theory and numerical examples. *Commun. Comput. Phys.* **28**, 1671–1706 (2020).
39. He, K., Zhang, X., Ren, S. & Sun, J. Deep residual learning for image recognition. In *Proc. 2016 IEEE Conference on Computer Vision and Pattern Recognition* 770–778 (IEEE, 2016).
40. Vaswani, A. et al. Attention is all you need. In *Advances in Neural Information Processing Systems* 5998–6008 (NIPS, 2017).
41. Dumoulin, V. et al. Feature-wise transformations. *Distill* https://distill.pub/2018/feature-wise-transformations (2018).
42. Sutskever, I., Vinyals, O. & Le, Q. V. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems* 3104–3112 (NIPS, 2014).
43. Bahdanau, D., Cho, K. & Bengio. Y. Neural machine translation by jointly learning to align and translate. Preprint at https://arxiv.org/pdf/1409.0473.pdf (2014).
44. Britz, D., Goldie, A., Luong, M. & Le, Q. Massive exploration of neural machine translation architectures. Preprint at https://arxiv.org/pdf/1703.03906.pdf (2017).
45. Gelbrich, M. On a formula for the l^2 Wasserstein metric between measures on Euclidean and Hilbert spaces. *Math. Nachrichten* **147**, 185–203 (1990).
46. Podlubny, I. *Fractional Differential Equations*: *An Introduction to Fractional Derivatives*, *Fractional Differential Equations*, *to Methods of their Solution and Some of their Applications* (Elsevier, 1998).
47. Zayernouri, M. & Karniadakis, G. E. Fractional Sturm–Liouville Eigen-problems: theory and numerical approximation. *J. Comput. Phys.* **252**, 495–517 (2013).
48. Lischke, A. et al. What is the fractional Laplacian? A comparative review with new results. *J. Comput. Phys.* **404**, 109009 (2020).
49. Born, M. & Wolf, E. *Principles of Optics*: *Electromagnetic Theory of Propagation*, *Interference and Diffraction of Light* (Elsevier, 2013).
50. Mitzenmacher, M. & Upfal, E. *Probability and Computing*: *Randomization and Probabilistic Techniques in Algorithms and Data Analysis* (Cambridge Univ. Press, 2017).
51. Shwartz-Ziv, R. & Tishby, N. Opening the black box of deep neural networks via information. Preprint at https://arxiv.org/pdf/1703.00810.pdf (2017).
52. Cai, S., Wang, Z., Lu, L., Zaki, T. A. & Karniadakis, G. E. DeepM&Mnet: inferring the electroconvection multiphysics fields based on operator approximation by neural networks. Preprint at https://arxiv.org/pdf/2009.12935.pdf (2020).
53. Tai, K. S., Bailis, P. & Valiant, G. Equivariant transformer networks. Preprint at https://arxiv.org/pdf/1901.11399.pdf (2019).
54. Hanin, B. Universal function approximation by deep neural nets with bounded width and ReLU activations. Preprint at https://arxiv.org/pdf/1708.02691.pdf (2017).
55. Lu, L. DeepONet https://doi.org/10.5281/zenodo.4319385 (13 December 2020).

Acknowledgements
This work was supported by the DOE PhILMs project (no. DE-SC0019453) and DARPA-CompMods grant no. HR00112090062.

Author contributions
L.L. and G.E.K. designed the study based on G.E.K.'s original idea. L.L. developed DeepONet architectures. L.L., P.J. and Z.Z. developed the theory. L.L. performed the experiments for the integral, nonlinear ODE, gravity pendulum and stochastic ODE/PDE operators. L.L. and P.J. performed the experiments for the Legendre transform, diffusion-reaction, advection and advection-diffusion PDEs. G.P. performed the experiments for fractional operators. L.L., P.J., G.P., Z.Z. and G.E.K. wrote the manuscript. G.E.K. supervised the project.

Competing interests
The authors declare no competing interests.

Additional information
Supplementary information The online version contains supplementary material available at https://doi.org/10.1038/s42256-021-00302-5.

Correspondence and requests for materials should be addressed to G.E.K.

Peer review information *Nature Machine Intelligence* thanks Irana Higgins, Jian-Xun Wang and the other, anonymous, reviewer(s) for their contribution to the peer review of this work.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

nature research

Corresponding author(s): George Em Karniadakis

Last updated by author(s): Dec 13, 2020

Reporting Summary

Nature Research wishes to improve the reproducibility of the work that we publish. This form provides structure for consistency and transparency in reporting. For further information on Nature Research policies, see Authors & Referees and the Editorial Policy Checklist.

Statistics

For all statistical analyses, confirm that the following items are present in the figure legend, table legend, main text, or Methods section.

n/a	Confirmed
☒	☐ The exact sample size (*n*) for each experimental group/condition, given as a discrete number and unit of measurement
☒	☐ A statement on whether measurements were taken from distinct samples or whether the same sample was measured repeatedly
☐	☒ The statistical test(s) used AND whether they are one- or two-sided *Only common tests should be described solely by name; describe more complex techniques in the Methods section.*
☒	☐ A description of all covariates tested
☒	☐ A description of any assumptions or corrections, such as tests of normality and adjustment for multiple comparisons
☐	☒ A full description of the statistical parameters including central tendency (e.g. means) or other basic estimates (e.g. regression coefficient) AND variation (e.g. standard deviation) or associated estimates of uncertainty (e.g. confidence intervals)
☒	☐ For null hypothesis testing, the test statistic (e.g. *F*, *t*, *r*) with confidence intervals, effect sizes, degrees of freedom and *P* value noted *Give P values as exact values whenever suitable.*
☒	☐ For Bayesian analysis, information on the choice of priors and Markov chain Monte Carlo settings
☒	☐ For hierarchical and complex designs, identification of the appropriate level for tests and full reporting of outcomes
☒	☐ Estimates of effect sizes (e.g. Cohen's *d*, Pearson's *r*), indicating how they were calculated

Our web collection on statistics for biologists contains articles on many of the points above.

Software and code

Policy information about availability of computer code

Data collection	Python 3, TensorFlow
Data analysis	Python 3, TensorFlow

For manuscripts utilizing custom algorithms or software that are central to the research but not yet described in published literature, software must be made available to editors/reviewers. We strongly encourage code deposition in a community repository (e.g. GitHub). See the Nature Research guidelines for submitting code & software for further information.

Data

Policy information about availability of data

All manuscripts must include a data availability statement. This statement should provide the following information, where applicable:

- Accession codes, unique identifiers, or web links for publicly available datasets
- A list of figures that have associated raw data
- A description of any restrictions on data availability

All the datasets in the study are generated directly from the code.

Field-specific reporting

Please select the one below that is the best fit for your research. If you are not sure, read the appropriate sections before making your selection.

☐ Life sciences ☐ Behavioural & social sciences ☐ Ecological, evolutionary & environmental sciences

For a reference copy of the document with all sections, see nature.com/documents/nr-reporting-summary-flat.pdf

Life sciences study design

All studies must disclose on these points even when the disclosure is negative.

Sample size	*Describe how sample size was determined, detailing any statistical methods used to predetermine sample size OR if no sample-size calculation was performed, describe how sample sizes were chosen and provide a rationale for why these sample sizes are sufficient.*
Data exclusions	*Describe any data exclusions. If no data were excluded from the analyses, state so OR if data were excluded, describe the exclusions and the rationale behind them, indicating whether exclusion criteria were pre-established.*
Replication	*Describe the measures taken to verify the reproducibility of the experimental findings. If all attempts at replication were successful, confirm this OR if there are any findings that were not replicated or cannot be reproduced, note this and describe why.*
Randomization	*Describe how samples/organisms/participants were allocated into experimental groups. If allocation was not random, describe how covariates were controlled OR if this is not relevant to your study, explain why.*
Blinding	*Describe whether the investigators were blinded to group allocation during data collection and/or analysis. If blinding was not possible, describe why OR explain why blinding was not relevant to your study.*

Behavioural & social sciences study design

All studies must disclose on these points even when the disclosure is negative.

Study description	*Briefly describe the study type including whether data are quantitative, qualitative, or mixed-methods (e.g. qualitative cross-sectional, quantitative experimental, mixed-methods case study).*
Research sample	*State the research sample (e.g. Harvard university undergraduates, villagers in rural India) and provide relevant demographic information (e.g. age, sex) and indicate whether the sample is representative. Provide a rationale for the study sample chosen. For studies involving existing datasets, please describe the dataset and source.*
Sampling strategy	*Describe the sampling procedure (e.g. random, snowball, stratified, convenience). Describe the statistical methods that were used to predetermine sample size OR if no sample-size calculation was performed, describe how sample sizes were chosen and provide a rationale for why these sample sizes are sufficient. For qualitative data, please indicate whether data saturation was considered, and what criteria were used to decide that no further sampling was needed.*
Data collection	*Provide details about the data collection procedure, including the instruments or devices used to record the data (e.g. pen and paper, computer, eye tracker, video or audio equipment) whether anyone was present besides the participant(s) and the researcher, and whether the researcher was blind to experimental condition and/or the study hypothesis during data collection.*
Timing	*Indicate the start and stop dates of data collection. If there is a gap between collection periods, state the dates for each sample cohort.*
Data exclusions	*If no data were excluded from the analyses, state so OR if data were excluded, provide the exact number of exclusions and the rationale behind them, indicating whether exclusion criteria were pre-established.*
Non-participation	*State how many participants dropped out/declined participation and the reason(s) given OR provide response rate OR state that no participants dropped out/declined participation.*
Randomization	*If participants were not allocated into experimental groups, state so OR describe how participants were allocated to groups, and if allocation was not random, describe how covariates were controlled.*

Ecological, evolutionary & environmental sciences study design

All studies must disclose on these points even when the disclosure is negative.

Study description	*Briefly describe the study. For quantitative data include treatment factors and interactions, design structure (e.g. factorial, nested, hierarchical), nature and number of experimental units and replicates.*
Research sample	*Describe the research sample (e.g. a group of tagged Passer domesticus, all Stenocereus thurberi within Organ Pipe Cactus National Monument), and provide a rationale for the sample choice. When relevant, describe the organism taxa, source, sex, age range and any manipulations. State what population the sample is meant to represent when applicable. For studies involving existing datasets, describe the data and its source.*
Sampling strategy	*Note the sampling procedure. Describe the statistical methods that were used to predetermine sample size OR if no sample-size calculation was performed, describe how sample sizes were chosen and provide a rationale for why these sample sizes are sufficient.*
Data collection	*Describe the data collection procedure, including who recorded the data and how.*

Timing and spatial scale	*Indicate the start and stop dates of data collection, noting the frequency and periodicity of sampling and providing a rationale for these choices. If there is a gap between collection periods, state the dates for each sample cohort. Specify the spatial scale from which the data are taken*
Data exclusions	*If no data were excluded from the analyses, state so OR if data were excluded, describe the exclusions and the rationale behind them, indicating whether exclusion criteria were pre-established.*
Reproducibility	*Describe the measures taken to verify the reproducibility of experimental findings. For each experiment, note whether any attempts to repeat the experiment failed OR state that all attempts to repeat the experiment were successful.*
Randomization	*Describe how samples/organisms/participants were allocated into groups. If allocation was not random, describe how covariates were controlled. If this is not relevant to your study, explain why.*
Blinding	*Describe the extent of blinding used during data acquisition and analysis. If blinding was not possible, describe why OR explain why blinding was not relevant to your study.*

Did the study involve field work? ☐ Yes ☐ No

Field work, collection and transport

Field conditions	*Describe the study conditions for field work, providing relevant parameters (e.g. temperature, rainfall).*
Location	*State the location of the sampling or experiment, providing relevant parameters (e.g. latitude and longitude, elevation, water depth).*
Access and import/export	*Describe the efforts you have made to access habitats and to collect and import/export your samples in a responsible manner and in compliance with local, national and international laws, noting any permits that were obtained (give the name of the issuing authority, the date of issue, and any identifying information).*
Disturbance	*Describe any disturbance caused by the study and how it was minimized.*

Reporting for specific materials, systems and methods

We require information from authors about some types of materials, experimental systems and methods used in many studies. Here, indicate whether each material, system or method listed is relevant to your study. If you are not sure if a list item applies to your research, read the appropriate section before selecting a response.

Materials & experimental systems

n/a	Involved in the study
☒	☐ Antibodies
☒	☐ Eukaryotic cell lines
☒	☐ Palaeontology
☒	☐ Animals and other organisms
☒	☐ Human research participants
☒	☐ Clinical data

Methods

n/a	Involved in the study
☒	☐ ChIP-seq
☒	☐ Flow cytometry
☒	☐ MRI-based neuroimaging