An Operator Learning Approach via Function-valued Reproducing Kernel Hilbert Space for Differential Equations

Kaijun Bao\textsuperscript{a}, Xu Qian\textsuperscript{a,*}, Ziyuan Liu\textsuperscript{a}, Songhe Song\textsuperscript{a,b}

\textsuperscript{a}Department of Mathematics, College of Liberal Arts and Science, National University of Defense Technology, Changsha 410073, P.R. China
\textsuperscript{b}State Key Laboratory of High Performance Computing, National University of Defense Technology, Changsha 410073, China

Abstract

Much recent work has addressed the solution of a family of partial differential equations (PDEs) by computing the inverse operator map between function spaces. Toward this end, we incorporate function-valued reproducing kernel Hilbert spaces (function-valued RKHS) in our operator learning model. Motivated by recently successful neural operator: Deep operator networks (DeepONets), we use neural networks to parameterize the Hilbert-Schmidt integral operator and propose an architecture based on the representer theorem in function-valued RKHS. Experiments including the advection, KdV, burgers’, and poisson equations show that the proposed architecture has better accuracy on nonlinear PDEs and linear PDEs with a small amount of data than DeepONets. We also show that by learning the mappings between function spaces, the proposed method can find the solution of a high-resolution input after learning from lower-resolution data.

Keywords: Partial differential equation; Function-valued reproducing kernel Hilbert spaces; Operator learning; Deep operator networks.

1. Introduction

Partial differential equations (PDEs) are powerful tools for modeling the real world in fields such as aerospace and material and biomolecular dynamics, with great success from microscopic (quantum, molecular dynamics) to cryoscopic issues (ship engineering). However, two challenges remain: identifying and formulating the PDEs appropriate to modeling a specific problem, and computational efficiency when solving complicated PDE systems.

Modeling a specific problem and determining the approximate underlying PDE usually requires much prior knowledge, combined with general conservation laws, such as the law of conservation of energy, to design a predictive model. But, to acquire prior knowledge for complex systems is usually expensive, or the system is too simple and lacks information, and solving large-scale nonlinear PDEs requires significant computing resources, which can render simulations infeasible. This is reflected in the traditional numerical method for solving PDEs. Since the analytical

\textsuperscript{*}This work was supported by the National Key R&D Program of China (2020YFA0709800), the National Natural Science Foundation of China (No. 11901577, 11971481, 12071481), the Natural Science Foundation of Hunan (2021JJ20053, 2020JJ5652), the Science and Technology Innovation Program of Hunan Province (No. 2021RC3082), and the Defense Science Foundation of China (2021-1JCJQ-JJ-0538).

\textsuperscript{*}Corresponding author.

Email address: qianxu@nudt.edu.cn (Xu Qian)

Preprint submitted to Elsevier April 5, 2022
solution of PDEs is difficult, in practical engineering applications, we seek the numerical value of a solution at some discrete points in a region. Classical methods such as the finite difference, finite volume, spectrum, and finite element methods will eventually turn a PDE solution into the solution of linear equations. When the problem becomes complicated, the numerical method must be accurate enough and the discrete points dense enough, such as in turbulence problems. The distance between discrete points must usually reach the order of micrometers, but this requires significant computing resources, which renders the problem unsolvable. If the dimension of the problem increases, we only discretize two points in each dimension (this is already quite sparse), and the total number of discrete points increases exponentially, a problem also faced by traditional numerical methods, i.e., the curse of dimensionality.

Much recent work has used deep neural networks to solve PDEs, showing efficiency compared with numerical methods. One issue is to parameterize the solution of a PDE with neural networks \[1, 2, 3, 4, 5, 6, 7, 8\]. Based on automatic differentiation, we can calculate differential terms in a PDE. Substituting them in the PDE to construct a loss function, the neural network can be well trained. It is worth noting that this approach is data-independent, because the loss function is constructed based on knowledge of the underlying PDE structure, whose representative work is PINN \[1\]. The deep rize method \[2\] of Weinan E constructs a loss function via the variational form of the PDE, showing that a neural network can solve high-dimensional problems through Monte Carlo simulation. However, the neural network must be retrained when the coefficients associated with the PDEs are changed. Numerical methods also have this problem. Other work focuses on solving a series of PDEs. One issue is to discretize the coefficients and solutions of PDEs at a specific resolution \[9, 10, 11, 12\]. Based on the data, a PDE solution is transformed to the learning of a map between finite Euclidean spaces. Such an approach clearly depends on the discretization size and geometry of the training data, and it is impossible to query solutions at new points. Another approach is the neural operator \[13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23\], which is closest to the problem we investigate. DeepONets \[13\] take inputs consisting of coefficients \( f \) and locations \( x \), and generate a solution \( u \) at \( x \). Mod-net \[20\] and MWT \[18\] use the integral operator \( T[f](x) = \int_D K(x, y)f(y)dy \) to approximate solutions and use a neural network to model \( K \). We can see that the neural operator method considers the locations \( x \), and can solve a series of PDEs, but is independent of on the resolution.

We treat the problem of learning a PDE as an operator learning problem. For a well-posed PDE, given a coefficient, boundary condition, and source term, there exists a unique solution, which means the PDE can be completely represented by the operator mapping coefficients, boundary conditions, or source terms to solutions. This indicates that solving a PDE can be accomplished by learning from data, but we do not need to know its underlying knowledge. Function-valued reproducing kernel Hilbert spaces \[24\], which are induced by a unique operator-valued kernel, offer a special form given by the representer theorem to approximate an operator, and the error between the approximate solution given by this approach and the exact solution can be estimated. This indicates that with an appropriate operator-valued kernel and growth of the data, the approximate solution will converge to the exact one at a fixed rate. Inspired by DeepONets, we take the operator-valued kernel as Hilbert-Schmidt integral operator \[24\]. By using neural networks to parameterize it and combining with the representer theorem, we introduce the
function-valued RKHS-based model, a novel operator learning method. Our main contributions are as follows: i) based on the architecture of DeepONets and the representer theorem, we develop a neural network model that efficiently learns the operator map; ii) we demonstrate the applicability of our model on the one-dimensional dataset of nonlinear burgers’ and KdV equations and our model performs better than DeepONets; iii) in linear case with a small amount of data, we show that the proposed model can efficiently learn the operator but DeepONets cannot; and iv) we show that our model is a mesh-independent method that can find the solution to a high-resolution input after learning from lower-resolution data.

2. Methodology

In this section, we will first transform the solution of PDEs to the learning of the operator between two infinite-dimensional spaces. Then, we propose a neural network architecture based on the representer theorem in function-valued RKHS and Hilbert-Schmidt integral operator.

2.1. Problem Setting

We aim to solve PDEs by approximating the operator between two infinite-dimensional spaces. The PDE we consider takes the form

\[
(L_\alpha u)(x) = f(x), \quad x \in D \\
u(x) = g(x), \quad x \in \partial D,
\]

where \( L \) is a differential operator, and \( \alpha \) involves terms that determine the governing equation of the PDE. We assume that PDE (1) is well-posed, which means that there exists an operator mapping from the space constructed by functions \( \alpha, f, g \) to the space constructed by solution \( u \). We actually fix two terms in functions \( \alpha, f, g \), and the PDE can be considered as the operator mapping the remaining term to the solution. We propose to solve the PDE by constructing a parametric map to approximate this operator. Without loss of generality, we fix functions \( \alpha, g \).

Let \( G^\dagger \) be the operator mapping source term \( f \) to solution \( u \) in PDE (1), and \( \mathcal{G} \) the parametric map taking \( f \) and \( \theta \in \Theta \) as input, where \( \Theta \) represents some finite-dimensional parameter spaces. We expect there is a \( \theta^\dagger \in \Theta \) such that \( \mathcal{G}(\cdot, \theta^\dagger) \approx G^\dagger \).

Let \( \mathcal{F} \) and \( \mathcal{U} \) be Banach spaces of functions and suppose we have observations \( \{f_i, u_i\}_{i=1}^n \), where \( f_i \sim \mu \) is an i.i.d. sequence from the probability measure \( \mu \) supported on \( \mathcal{F} \), and \( u_i \) is the corresponding solution, which belongs to \( \mathcal{U} \). To determine \( \theta^\dagger \), a natural framework is to define a cost functional \( C : \mathcal{U} \times \mathcal{U} \to \mathbb{R} \) and seek a minimizer of the problem,

\[
\min_{\theta \in \Theta} \mathbb{E}_{f \sim \mu}[C(G(f, \theta), u)].
\]

To numerically work with \( f, u \), we assume pointwise evaluations of functions. Let \( P_k \) be a \( k \)-point discretization of domain \( D \), and assume observations \( f_i|_{P_k}, u_i|_{P_k} \) for a finite collection of input-output pairs indexed by \( i \).
2.2. Function-valued RKHS-based Model

Function-valued RKHS, which has the property that the representer theorem provides a special form for approximating an operator, demonstrates that the generalization error only depends on the reproducing kernel. Then, inspired by the architecture of DeepONets, we take Hilbert-Schmidt integral operator as the reproducing kernel and propose our model.

![Figure 1: Architecture of function-valued RKHS-based model.](image)

We first consider the supervised learning in view of an operator. Given data \( \{ f_i, G^\dagger(f_i) \}_{i=1}^n \), we consider the following regression estimator:

\[
\hat{G}_n = \arg\min_{G \in \mathcal{H}_k} (\mathcal{R}_n(G) + \lambda \| G \|^2_{\mathcal{H}_k})
\]

\[
\mathcal{R}_n(G) = \frac{1}{n} \sum_{i=1}^n \| G(f_i) - G^\dagger(f_i) \|_U^2.
\]

(3)

where \( \mathcal{H}_k \) is a Hilbert space. Based on the data, \( \hat{G}_n \) is the optimal approximation of \( G^\dagger \), but we wish to know if \( \hat{G}_n \) can perform well on the whole input space. The performance is characterized by the generalization error, which takes the form

\[
\mathbb{E}_{f \sim \mu}[\| \hat{G}_n(f) - G^\dagger(f) \|_U^2].
\]

(4)

Let \( \hat{G} = \arg\min_{G \in \mathcal{H}_k} (\mathcal{R}(G) + \lambda \| G \|^2_{\mathcal{H}_k}) \), where \( \mathcal{R}(G) = \mathbb{E}_{f \sim \mu}[\| G(f) - G^\dagger(f) \|_U^2] \). The generalization error can be decomposed as

\[
\mathbb{E}_{f \sim \mu}[\| \hat{G}_n(f) - G^\dagger(f) \|_U^2] \leq \mathbb{E}_{f \sim \mu}[\| \hat{G}_n(f) - \hat{G}(f) \|_U^2] + \mathbb{E}_{f \sim \mu}[\| \hat{G}(f) - G^\dagger(f) \|_U^2].
\]

(5)
By taking $\mathcal{H}_k$ as the function-valued RKHS, which is induced by a reproducing kernel $K$ and following the representer theorem, we have

$$\hat{G}_n = \sum_{i=1}^{n} K(f_i, \cdot)A_i,$$

where $A_i(x)$ are in $\mathcal{U}$, and $K$ is a nonnegative operator-valued function. For the first term in equation (5), the approximate theorem in function-valued RKHS shows that with the growth of data, $\hat{G}_n(f)$ can converge to $\hat{G}(f)$ at a fixed rate. Then the second term is what matters, but it will vanish if $G^\dagger$ belongs to the function-valued RKHS $H_k$, which means that it is determined by the reproducing kernel $K$. Hence, based on equation (6), we parameterize the reproducing kernel $K$ and propose the following neural network architecture.

Inspired by the successful application of DeepONets, we consider the Hilbert-Schmidt integral operator, which has a similar architecture. It takes form:

$$K(f_i, \cdot)A_i = k_1(f_i, \cdot) \int_D k_2(x, y)A_i(y)dy,$$

where $k_1$ and $k_2$ are scalar-valued kernel functions. By using neural networks $K_1, K_2$ to model $k_1, k_2$, we observe that $K_1$ and $K_2$ play the same role as the branch networks and the trunk networks in DeepONets. Next, we propose an architecture following an encoder-decoder framework. The neural networks $K_1, K_2$ encode the input to a $d$-dimensional vector field. We proceed to equation (6) in the $d$-dimensional vector field, and finally use a decoder to project back to the scalar field of interest.

We denote:

$$a \& b = [a_1 b_1, a_2 b_2, \cdots, a_d b_d]^T$$

where $a = [a_1, a_2, \cdots, a_d]^T \in \mathbb{R}^d$ and $b = [b_1, b_2, \cdots, b_d]^T \in \mathbb{R}^d$.

Given data $\{f_i|\rho_k, u_i|\rho_k\}|_{i=1}^{n}$ and $f$ in PDE (1), $u$ is the corresponding solution of $f$. Then we approximate $u$ by

$$v(x) = \sum_{i=1}^{n} K_1(f|\rho_k, f_i|\rho_k) \odot \int_D K_2(x, y) \odot A_i(x)dy$$

$$u^{app}(x) = Wv(x) + b,$$

where $W \in \mathbb{R}^{1 \times d}, b \in \mathbb{R}$, and $K_1, K_2$ project the input to $\mathbb{R}^d$. Fig. 1 shows the architecture of our model.

It is worth noting that in practice, the integral in (9) can not be calculated analytically, and we often represent it by numerical integration. We use the Gauss-Chebyshev integral, and the calculation becomes

$$v(x) = \sum_{i=1}^{n} K_1(f|\rho_k, f_i|\rho_k) \odot \sum_{y \in S_G} \omega_y K_2(x, y) \odot A_i(x)$$

$$u^{app}(x) = Wv(x) + b,$$
where \( S_G \subset D \) consists of fixed integration points determined by the Gauss-Chebyshev integral method, and \( \omega_y \) are corresponding coefficients.

3. Numerical Examples

We evaluate our model on several typical PDE datasets. As mentioned above, we solve the PDEs in high resolution by a traditional numerical method and obtain our training data by downsampling. The numerical results show that our method can efficiently solve PDEs based on learning the operator. We demonstrate our model’s good generalization properties even with a small amount of training data, and show that the proposed method exhibits mesh-independent properties, is trained at low resolutions, and can generalize at high resolutions.

Unless stated otherwise, in our model, the architectures of neural networks \( K_1 \) and \( K_2 \) are 128-256-256-128 and 128-128-128-128, respectively, with a ReLU activation function, and they encode inputs to a 32-dimensional vector field. \( K_1 \) is a convolutional neural network, and we first use one layer to enhance the input channel to 32 without an activation function. The kernel size is taken as 32, and we use padding to keep the channel unchanged. \( K_2 \) is a feedforward neural network (FNN). We choose \( m = 10 \) for Chebyshev polynomials, \( |S_G| = 10 \) in one-dimensional PDEs, and \( |S_G| = 100 \) in two-dimensional PDEs for Gauss-Chebyshev integration. For all examples, we train our model in 4000 epochs using the Adam optimizer with an initial learning rate (LR) of 0.0002. Applying polynomial decay to the LR, we set the power of the polynomial to 0.5, and the LR will decay to \( 2 \times 10^{-8} \) in 200 epochs. To efficiently train our model, we increase the LR after each 200 epochs, and then decrease it. Due to the structural similarity, we choose DeepONets as our benchmark model. Meanwhile, we also take the traditional neural networks into account. NN: a point-wise feed-forward neural network. CNN: a fully convolutional neural network.

The PDEs we consider include the one-dimensional advection equation, burgers’ equation, KdV equation, and two-dimensional poisson equation. Advection is one of the most important processes in atmospheric motion, and advection terms are included in the motion equation, heat flux equation, and water vapor equation in the atmospheric motion equations. Burgers’ equation is a fundamental PDE in various fields of applied mathematics, involving many fields of applied mathematics such as fluid mechanics, nonlinear acoustics, and gas dynamics. The KdV equation, as proposed by Dutch mathematicians Korteweg and de Vries, describes nonlinear shallow water waves in fluid mechanics. The Poisson equation is commonly found in electrostatics, mechanical engineering, and theoretical physics.

3.1. Advection Equation

We first consider the one-dimensional advection equation,

\[
\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} = 0, \quad x \in (0, 1), \quad t \in [0, 1] \\
u(0, t) = u(1, t).
\]
Our task is to use the initial conditions \( u(x,0) \) to give the solution at \( u(x,t) \) at time \( t=1 \). We use Gaussian random fields to generate initial conditions with periodic boundary conditions, numerically solve the equation at high resolution, and downsample the final training data from the high-resolution numerical solution. The Gaussian random field is chosen as \( N(0,625(-\Delta + 5^2 I)^{-4}) \), and we use Chebfun to solve the equation in \( 2^{11} \) resolution.

The experimental results of the advection equation are shown in Fig. 2 and Fig. 3. Given a new initial condition, Fig. 2 compares the approximate solution generated by our model with the exact solution in a different resolution. Fig. 3 shows the training error history in the view of \( L_2 \) error.

Our model is trained in \( 2^8 \) resolution and the result in Fig. 2 (a) shows that the predicted solution fits accurately the exact solution. In addition, we evaluate our model at \( 2^9 \) resolution. Fig. 2 (b) validates that the proposed method is able to find the solution of a high-resolution input after learning from lower-resolution data, which correctly learns the operator about the PDE. Due to the scheme, which we choose for LR, the variation trend of loss has large fluctuations. Meanwhile, it is seen from Fig. 3 that after 1000 epochs, the training loss is able to reach \( 10^{-5} \) periodically. Consequently, the trained model demonstrates good generalization capability. Moreover, Our approach can achieve 1.7% relative \( L_2 \) error on the test dataset.
3.2. Burgers’ Equation

The one-dimensional Burgers’ equation is a PDE whose applications include modeling the one-dimensional flow of a viscous fluid. It takes the form

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

\[u_0(x) = u(x, t = 0).\]

The task for the neural operator is to learn the mapping of initial condition \(u(x, t = 0)\) to the solutions at \(u(x, t = 1)\). The initial condition \(u_0(x)\) is generated according to \(u_0 \sim u_0 \sim \mu\), where \(\mu = \mathcal{N}(0, 625(-\Delta + 25I)^{-4})\) with periodic boundary conditions. We set the viscosity to \(v = 0.01\) and solve the equation using a split-step method, where the heat equation part is solved exactly in Fourier space, and the part is advanced again in Fourier space, using a very fine forward Euler method. We solve on a spatial mesh with resolution \(2^{12}\), and use this dataset to downsample other resolutions.

![Figure 4: Burgers’ equation: Comparison of true solution and learned model solution in two inputs.](image)

The results of our experiments are shown in Fig. 4, Fig. 5, Table 1 and Table 2. In Fig. 4, given new initial conditions, we compare the approximate solution generated by our model with the exact solution. Fig. 4 and Table 1 compare our model and benchmark models in view of the relative \(L_2\) error. Table 2 shows the results of our model trained in some resolutions and generalized at various input resolutions. The results show that the predicted solution by our approach simulates exact solution accurately, and obtains the lowest relative error compared to any of the benchmarks. Further, with the increasing resolution, we see that the relative error of our model gradually decreases, which learns better in high resolution than in low resolution, while other models increase with resolution. Our model also demonstrates the property of mesh-independence that when the model is trained in a special resolution, the relative error is invariant with the testing resolution.
3.3. Korteweg-de Vries (KdV) Equation

The one-dimensional KdV equation takes the form

$$\frac{\partial u}{\partial t} = -0.5u \frac{\partial u}{\partial x} - \frac{\partial^3 u}{\partial x^3} \in (0, 1), \quad t \in (0, 1]$$

$$u_0(x) = u(x, t = 0).$$

(15)

As mentioned above, we still solve the equation by learning the mapping of initial condition \(u(x, t = 0)\) to the solutions \(u(x, t)\) at \(t = 1\). The initial condition \(u_0(x)\) is generated according to a random field \([25]\) with a fluctuating parameter \(\lambda\), and we take \(\lambda = 0.25\). We solve on a spatial mesh with resolution \(2^{12}\), and use this dataset to downsample other resolutions to train our model.

Fig. 6, Fig. 7, Table 3, and Table 4 show the results of our experiments. In Fig. 6, we compare the approximate solution generated by our model with the exact solution, given a new initial condition. Fig. 7 and Table 3 compare the relative \(L_2\) error for our model and benchmark models. Table 4 shows the results for our model trained in some resolutions and generalized at various input resolutions.

It is worth noting that our model also obtains the lowest relative error compared to any of the benchmarks.
and still maintains a small relative error compared to Burgers’ equation. The predicted solution fits accurately the exact solution, while the benchmarks perform poorly for large fluctuations and even the CNN does not converge at all. At various input resolutions, our approach is able to achieve the same degree of relative error. Meanwhile, we again observe the invariance of the error with respect to the resolution. These results show that our approach is mesh-independent and has the property of stability.

| Resolutions | $s = 64$ | $s = 128$ | $s = 256$ | $s = 512$ |
|-------------|----------|----------|----------|----------|
| NN          | 0.3512   | 0.3824   | 0.4325   | 0.4874   |
| CNN         | -        | -        | -        | -        |
| DeepONets   | 0.2011   | 0.1957   | 0.2669   | 0.2244   |
| RKHS-based  | **0.0391** | **0.0300** | **0.0394** | **0.0387** |

Table 3: Benchmarks on KdV equation at various input resolution.
Table 4: KdV equation: Resolutions in training and testing.

| Resolutions | $s' = 64$ | $s' = 128$ | $s' = 256$ | $s' = 512$ |
|-------------|------------|------------|------------|------------|
| $s = 64$    | 0.0391     | 0.0399     | 0.0404     | 0.0406     |
| $s = 128$   | 0.0291     | 0.0300     | 0.0310     | 0.0315     |
| $s = 256$   | 0.0396     | 0.0395     | 0.0394     | 0.0394     |
| $s = 512$   | 0.0386     | 0.0388     | 0.0387     | 0.0387     |

3.4. Poisson equation

We consider the poisson equation in the case with a small amount of data [20],

$$-\triangle u(x) = f(x), x \in D$$
$$u(x) = 0, x \in \partial D.$$ (16)

Consider the 2D case, in which the source function $f$ is generated by $-a(x^2 - x + y^2 - y)$, i.e.,

$$\partial_{xx} u + \partial_{yy} u = -a(x^2 - x + y^2 - y), (x, y) \in D$$
$$u = 0, (x, y) \in \partial D,$$ (17)

where $D = [0, 1]^2$, and the constant $a$ controls the source term. Obviously, the analytical solution is $u = \frac{a}{2} x(x - 1)y(y - 1)$.

In this experiment, the settings of the training process are different from those mentioned above. During training, we choose a set of source functions, i.e., we sample 10 source functions $f$ from a selected region, which is implemented by sampling control parameter $a$ uniformly from $\{10k\}_{k=1}^{20}$, and take $P_k$ as $64 \times 64$ isometric grid points. For each epoch, we randomly sample 200 points in $[0, 1]^2$ and sample 100 points in each line of the boundary. It is worth noting that in previous experiments, the training points were also $P_k$. Finally, we test the performance of our well-trained model on $a = 15$. To visualize its performance, we show the exact and predicted solutions on $101 \times 101$ isometric grid points. To more intuitively compare these solutions, we calculate their difference at each point and indicate the error by colors, as shown in Fig. 8. The relative $L_2$ error is $1.5\%$.

We also test DeepONets on a two-dimensional Poisson equation with the similar training process, as we sample $K = 10$ source functions $f$ from the selected region and evaluate them on $64 \times 64$ isometric grid points. For each epoch, we randomly sample 200 points in $[0, 1]^2$ and 100 points in each line of the boundary. We observe that the testing relative $L_2$ error is $62.3\%$. Moreover, we evaluate them on $101 \times 101$ isometric grid points and also use the grid points as training points to train Deeponets, NN and CNN. The testing relative $L_2$ error is as follows, DeepONets: $0.26\%$, NN: $4.6\%$ and CNN: $8.6\%$. The results show that when the data is sufficient, DeepONets achieves a better performance than other models. However, in the case of a small amount of data, our model may be a better choice.
4. Conclusion

In this study, we transformed the solution of PDEs to the problem of data-driven learning of the operator that maps between two function spaces. Motivated by the fundamental properties of the function-valued RKHS, we proposed an architecture following the form given by the representer theorem, whose kernel was constructed by a Hilbert-Schmidt integral operator. We showed that the proposed model performed well on several kinds of PDEs, even solutions with high fluctuation. Numerical experiments showed that our approach was mesh-independent, with the ability to find the solution of a high-resolution input after learning from lower-resolution data, and had the property of stability. Another advantage of our model was that it only used a small amount of data to train, but had good generalization.

References

[1] M. Raissi, P. Perdikaris, G. E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, Journal of Computational physics 378 (2019) 686–707.
[2] B. Yu, et al., The deep ritz method: a deep learning-based numerical algorithm for solving variational problems, Communications in Mathematics and Statistics 6 (1) (2018) 1–12.

[3] J. Sirignano, K. Spiliopoulos, Dgm: A deep learning algorithm for solving partial differential equations, Journal of computational physics 375 (2018) 1339–1364.

[4] Y. Zang, G. Bao, X. Ye, H. Zhou, Weak adversarial networks for high-dimensional partial differential equations, Journal of Computational Physics 411 (2020) 109409.

[5] S. Dong, Z. Li, Local extreme learning machines and domain decomposition for solving linear and nonlinear partial differential equations, Computer Methods in Applied Mechanics and Engineering 387 (2021) 114129.

[6] J. Yu, L. Lu, X. Meng, G. E. Karniadakis, Gradient-enhanced physics-informed neural networks for forward and inverse pde problems, arXiv preprint arXiv:2111.02801 (2021).

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

[8] S. Wang, H. Wang, P. Perdikaris, On the eigenvector bias of fourier feature networks: From regression to solving multi-scale pdes with physics-informed neural networks, Computer Methods in Applied Mechanics and Engineering 384 (2021) 113938.

[9] J. Adler, O. Öktem, Solving ill-posed inverse problems using iterative deep neural networks, Inverse Problems 33 (12) (2017) 124007.

[10] S. Bhatnagar, Y. Afshar, S. Pan, K. Duraisamy, S. Kaushik, Prediction of aerodynamic flow fields using convolutional neural networks, Computational Mechanics 64 (2) (2019) 525–545.

[11] X. Guo, W. Li, F. Iorio, Convolutional neural networks for steady flow approximation, in: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, 2016, pp. 481–490.

[12] Y. Zhu, N. Zabaras, Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification, Journal of Computational Physics 366 (2018) 415–447.

[13] L. Lu, P. Jin, G. E. Karniadakis, Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators, arXiv preprint arXiv:1910.03193 (2019).

[14] S. Wang, H. Wang, P. Perdikaris, Learning the solution operator of parametric partial differential equations with physics-informed deeponets, Science advances 7 (40) (2021) eabi8605.

[15] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Neural operator: Graph kernel network for partial differential equations, arXiv preprint arXiv:2003.03485 (2020).
[16] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, A. Stuart, K. Bhattacharya, A. Anandkumar, Multipole graph neural operator for parametric partial differential equations, Advances in Neural Information Processing Systems 33 (2020) 6755–6766.

[17] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Fourier neural operator for parametric partial differential equations, arXiv preprint arXiv:2010.08895 (2020).

[18] G. Gupta, X. Xiao, P. Bogdan, Multiwavelet-based operator learning for differential equations, Advances in Neural Information Processing Systems 34 (2021).

[19] K. Wu, D. Xiu, Data-driven deep learning of partial differential equations in modal space, Journal of Computational Physics 408 (2020) 109307.

[20] L. Zhang, T. Luo, Y. Zhang, Z.-Q. J. Xu, Z. Ma, Mod-net: A machine learning approach via model-operator-data network for solving pdes, arXiv preprint arXiv:2107.03673 (2021).

[21] F. Alet, A. K. Jeewajee, M. B. Villalonga, A. Rodriguez, T. Lozano-Perez, L. Kaelbling, Graph element networks: adaptive, structured computation and memory, in: International Conference on Machine Learning, PMLR, 2019, pp. 212–222.

[22] L. Lu, H. He, P. Kasimbeg, R. Ranade, J. Pathak, One-shot learning for solution operators of partial differential equations, arXiv preprint arXiv:2104.05512 (2021).

[23] K. Bhattacharya, B. Hosseini, N. B. Kovachki, A. M. Stuart, Model reduction and neural networks for parametric pdes, arXiv preprint arXiv:2005.03180 (2020).

[24] H. Kadri, E. Duflos, P. Preux, S. Camu, A. Rakotomamonjy, J. Audiffren, Operator-valued kernels for learning from functional response data, Journal of Machine Learning Research 17 (20) (2016) 1–54.

[25] S. Filip, A. Javeed, L. N. Trefethen, Smooth random functions, random odes, and gaussian processes, SIAM Review 61 (1) (2019) 185–205.