SIAM J. Sci. Comput.
Vol. 44, No. 6, pp. A3490–A3514

MIONET: LEARNING MULTIPLE-INPUT OPERATORS VIA TENSOR PRODUCT[*]

PENGZHAN JIN[†], SHUAI MENG[‡], AND LU LU[§]

Abstract. As an emerging paradigm in scientific machine learning, neural operators aim to learn operators, via neural networks, that map between infinite-dimensional function spaces. Several neural operators have been recently developed. However, all the existing neural operators are only designed to learn operators defined on a single Banach space; i.e., the input of the operator is a single function. Here, for the first time, we study the operator regression via neural networks for multiple-input operators defined on the product of Banach spaces. We first prove a universal approximation theorem of continuous multiple-input operators. We also provide a detailed theoretical analysis including the approximation error, which provides guidance for the design of the network architecture. Based on our theory and a low-rank approximation, we propose a novel neural operator, MIONet, to learn multiple-input operators. MIONet consists of several branch nets for encoding the input functions and a trunk net for encoding the domain of the output function. We demonstrate that MIONet can learn solution operators involving systems governed by ordinary and partial differential equations. In our computational examples, we also show that we can endow MIONet with prior knowledge of the underlying system, such as linearity and periodicity, to further improve accuracy.

Key words. operator regression, multiple-input operators, tensor product, universal approximation theorem, neural networks, MIONet, scientific machine learning

MSC codes. 47-08, 47H99, 65D15, 68Q32, 68T07

DOI. 10.1137/22M1477751

1. Introduction.

The field of scientific machine learning (SciML) has grown rapidly in recent years, where deep learning techniques are developed and applied to solve problems in computational science and engineering [12]. As an active area of research in SciML, different methods have been developed to solve ordinary and partial differential equations (ODEs and PDEs) by parameterizing the solutions via neural networks (NNs), such as physics-informed NNs (PINNs) [48, 31, 36, 38, 47], the deep Ritz method [8], and the deep Galerkin method [41]. These methods have shown promising results in diverse applications, such as fluid mechanics [39], optics [4], systems biology [44, 5], and biomedicine [15]. However, these methods solve only one specific instance of the PDE, and hence it is necessary to train a new neural network given a new initial condition, boundary condition, or forcing term, which is computationally costly and time-consuming.

Another approach is to apply neural networks (called neural operators) to learn solution operators of PDEs, mapping from an input function v (e.g., initial condition, boundary condition, or forcing term) to the PDE solution u. This regression for the

[*]Submitted to the journal's Methods and Algorithms for Scientific Computing section February 14, 2022; accepted for publication (in revised form) July 13, 2022; published electronically November 7, 2022.
https://doi.org/10.1137/22M1477751
Funding: This work was supported by the U.S. Department of Energy (DE-SC0022953). The first author was partially supported by the China Postdoctoral Science Foundation grant 2022M710005.
[†]School of Mathematical Sciences, Peking University, Beijing 100871, China (jpz@pku.edu.cn).
[‡]Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104 USA (shuaim@seas.upenn.edu).
[§]Department of Chemical and Biomolecular Engineering, University of Pennsylvania, Philadelphia, PA 19104 USA (lulu1@seas.upenn.edu).

solution operator $\mathcal{G}$ is formulated as

$$\mathcal{G} : X \to Y, \quad v \mapsto u,$$

where X and Y are two infinite-dimensional Banach spaces of functions and $u = \mathcal{G}(v)$. We aim to learn $\mathcal{G}$ via NNs from a training dataset, i.e., some pairs $(v, \mathcal{G}(v)) \in X \times Y$. Once a neural operator is trained, obtaining a solution $\mathcal{G}(v)$ for a new instance of v requires only a forward pass of the network.

Several neural operator approaches have been recently proposed, such as deep operator network (DeepONet) [27, 28, 29] and Fourier neural operator (FNO) [21, 29], graph kernel network [22, 46], and others [34, 1, 42, 37]. Among these approaches, DeepONet has been applied and demonstrated good performance in diverse applications, such as high-speed boundary layer problems [7], multiphysics and multiscale problems of hypersonics [32] and electroconvection [2], multiscale bubble growth dynamics [23, 24], fractional derivative operators [28], stochastic differential equations [28], solar-thermal systems [35], and aortic dissection [45]. Several extensions of DeepONet have also been developed, such as Bayesian DeepONet [25], DeepONet with proper orthogonal decomposition (POD-DeepONet) [29], multiscale DeepONet [26], neural operator with coupled attention [14], and physics-informed DeepONet [43, 10].

Despite the progress of neural operators in computational tests, our theoretical understanding is lagging behind. The current theoretical analysis of neural operators focuses on approximation capability, such as the universal approximation theorems for DeepONet [3, 28] and FNO [18]. Recent theoretical works show that DeepONet [6, 20] and FNO [17] may break the curse of dimensionality (CoD) in some problems, and DeepONet can approximate the solution operators of elliptic PDEs with exponential accuracy [33]. These results demonstrate the efficient approximation capability of DeepONet and FNO. One main difference between DeepONet and FNO is that the theory of FNO requires the input function v and the output function u to be defined on the same domain [17, 18], but the theory of DeepONet does not have this restriction [3, 28, 6, 20, 33].

However, all the existing neural operators are only designed to learn operators defined on a single Banach space X, i.e., the input of the operator is a single function. The theory of universal approximation for operators has only been proved for operators defined on a single Banach space. We note that some theoretical work [18] allows the input function to be a vector-valued function, i.e., the input could be $\mathbf{v} = (v_1, v_2, \dots)$, but it still requires that all components of the input function v_i must be defined on the same domain. This limitation of the input space prohibits us from learning a wide range of useful operators, e.g., the PDE solution operator mapping from both the initial condition and boundary condition to the PDE solution, as the initial condition and boundary condition are defined on two different domains (the initial domain and the boundary domain, respectively).

To overcome this limitation, in this work we first study theoretically the approximation theory of the operator regression for a multiple-input operator $\mathcal{G}$ defined on the product of Banach spaces,

$$\mathcal{G} : X_1 \times X_2 \times \cdots \times X_n \to Y,$$

where $X_1, X_2, \dots, X_n$ are n different input Banach spaces, and Y is the output Banach space. For example, X_1 can be the function space of all initial conditions, X_2 can be the function space of all boundary conditions, and X_3 can be the function space of all forcing terms. Based on our theory, we then propose a novel neural operator, MIONet, to learn multiple-input operators. We verify in our computational results that MIONet can learn solution operators involving systems governed by ODEs and PDEs. We also discuss how we can endow MIONet with prior knowledge of the underlying system, such as linearity and periodicity, to further improve accuracy.

This paper is organized as follows. In section 2, we prove the approximation theory for multiple-input operator regression. Then we propose MIONet based on the theory in section 3. Subsequently, we test MIONet on several problems of ODEs and PDEs in section 4. Finally, section 5 summarizes this work.

2. Approximation theory.

Our goal is to learn a (typically nonlinear) operator mapping from a product of n Banach spaces (input spaces) to another Banach space (output space). These spaces are typically infinite dimensional. We first define the main notation used throughout this paper. Denote the space comprised of all the continuous maps mapping from a metric space X to a metric space Y as $C(X,Y)$, and define $C(X) := C(X,\mathbb{R})$. Let $X_1, X_2, \dots, X_n$ and Y be $n+1$ Banach spaces, and let $K_i \subset X_i$ $(i = 1,\dots,n)$ be a compact set. Then we aim to learn a continuous operator

$$\mathcal{G} : K_1 \times \cdots \times K_n \to Y, \quad (v_1,\dots,v_n) \mapsto u,$$

where $v_i \in K_i$ and $u = \mathcal{G}(v_1,\dots,v_n)$. Such $\mathcal{G}$ form the space $C(K_1 \times \cdots \times K_n, Y)$, which is studied in this paper. In the following results, Y could be any Banach space. When Y is a function space, for convenience we consider $Y = C(K_0)$ for a compact domain K_0.

In this section, we prove the approximation theory of continuous multiple-input operators by first illustrating our basic idea using the example of multilinearity on finite-dimensional spaces in section 2.1. We then introduce the techniques of Schauder basis and canonical projection for infinite-dimensional spaces in section 2.2, based on which we present the main theory of nonlinear operators in section 2.3, with more detailed analysis in section 2.4. We also provide a view of the theory through the tensor product of Banach spaces in section 2.5.

2.1. Multilinear operators defined on finite-dimensional Banach spaces.

We first use a simple case to illustrate the main idea of our theoretical approach: multilinear operators defined on finite-dimensional Banach spaces. Specifically, we consider a multilinear operator

$$\mathcal{G} : X_1 \times \cdots \times X_n \to Y,$$

where $X_1,\dots,X_n$ are Banach spaces of finite dimensions $d_1,\dots,d_n$.

Let $\{\phi_j^i\}_{j=1}^{d_i} \subset X_i$ be a basis of X_i, and thus for each $v_i \in X_i$, there exists a coordinate representation,

$$v_i = \sum_{j=1}^{d_i} \alpha_j^i \phi_j^i$$

for some vector $\alpha_i = (\alpha_1^i, \alpha_2^i, \dots, \alpha_{d_i}^i) \in \mathbb{R}^{d_i}$. Because $\mathcal{G}$ is multilinear, for any input

$(v_1, \ldots, v_n)$,

$$\begin{aligned}\mathcal{G}(v_1, \ldots, v_n) =&\mathcal{G}\left(\sum_{j_1=1}^{d_1} \alpha_{j_1}^1 \phi_{j_1}^1, \ldots, \sum_{j_n=1}^{d_n} \alpha_{j_n}^n \phi_{j_n}^n\right)\\ &= \sum_{j_1=1}^{d_1} \cdots \sum_{j_n=1}^{d_n} \mathcal{G}\left(\phi_{j_1}^1, \ldots, \phi_{j_n}^n\right) \alpha_{j_1}^1 \cdots \alpha_{j_n}^n,\end{aligned}$$

where $u_{j_1 \cdots j_n} = \mathcal{G}\left(\phi_{j_1}^1, \ldots, \phi_{j_n}^n\right) \in Y$ is the output of $\mathcal{G}$ for the input $\left(\phi_{j_1}^1, \ldots, \phi_{j_n}^n\right)$. For convenience and clarity, for $\mathbf{u} = (u_{j_1 \cdots j_n})_{d_1 \times \cdots \times d_n} \in Y^{d_1 \times \cdots \times d_n}$, we use the notation $\mathbf{u}\langle \cdots \rangle$ to represent the multilinear map

$$\mathbf{u}\langle \alpha_1, \ldots, \alpha_n \rangle := \sum_{j_1=1}^{d_1} \cdots \sum_{j_n=1}^{d_n} u_{j_1 \cdots j_n} \alpha_{j_1}^1 \cdots \alpha_{j_n}^n.$$

Hence, a multilinear operator defined on finite-dimensional Banach spaces can be represented as

$$\mathcal{G}(v_1, \ldots, v_n) = \left(\mathcal{G}\left(\phi_{j_1}^1, \ldots, \phi_{j_n}^n\right)\right)_{d_1 \times \cdots \times d_n} \langle \alpha_1, \ldots, \alpha_n \rangle. \tag{2.1}$$

Next we discuss the main idea of the approximation theory of $\mathcal{G}$, i.e., how to construct a surrogate model $\tilde{\mathcal{G}}_\theta$ (parameterized by the parameters θ) to approximate $\mathcal{G}$. We note that α_i in (2.1) can be computed directly for v_i, and thus to approximate $\mathcal{G}$, it is sufficient to approximate $\left(\mathcal{G}\left(\phi_{j_1}^1, \ldots, \phi_{j_n}^n\right)\right)_{d_1 \times \cdots \times d_n}$. We consider $Y = C(K_0)$ for a compact set $K_0 \subset \mathbb{R}^d$, and then we can construct $\tilde{\mathcal{G}}_\theta$ as

$$\begin{aligned}\tilde{\mathcal{G}}_\theta :&\mathbb{R}^{d_1} \times \cdots \times \mathbb{R}^{d_n} \to C(K_0),\\ &(\alpha_1, \ldots, \alpha_n) \mapsto \tilde{f}_\theta \langle \alpha_1, \ldots, \alpha_n \rangle,\end{aligned}$$

where $\tilde{f}_\theta \in C(K_0, \mathbb{R}^{d_1 \times \cdots \times d_n})$ is a function class parameterized by parameters θ. It is easy to show that $\tilde{\mathcal{G}}_\theta$ is multilinear and can approximate $\mathcal{G}$ arbitrarily well as long as $\tilde{f}_\theta$ approximates $\left(\mathcal{G}\left(\phi_{j_1}^1, \ldots, \phi_{j_n}^n\right)\right)_{d_1 \times \cdots \times d_n}$ well, which can be achieved by choosing $\tilde{f}_\theta$ as neural networks.

2.2. Schauder basis and canonical projections for infinite-dimensional spaces. To deal with infinite-dimensional spaces, we introduce the Schauder basis and canonical projections. We refer the reader to [9] for more details.

DEFINITION 2.1 (Schauder basis). *Let X be an infinite-dimensional normed linear space. A sequence $\{e_i\}_{i=1}^\infty$ in X is called a Schauder basis of X if for every $x \in X$ there is a unique sequence of scalars $\{a_i\}_{i=1}^\infty$, called the coordinates of x, such that*

$$x = \sum_{i=1}^\infty a_i e_i.$$

We show two useful examples of the Schauder basis as follows.

Example 2.2 (Faber–Schauder basis of $C[0,1]$). Given distinct points $\{t_i\}_{i=1}^\infty$ which are a dense subset in $[0,1]$ with $t_1 = 0$, $t_2 = 1$, let $e_1(t) = 1$, $e_2(t) = t$, and let e_{k+1} be chosen as an element, such that $e_1, \ldots, e_k, e_{k+1}$ is a basis of the $(k+1)$-dimensional space which consists of all the piecewise linear functions with grid points $\{t_i\}_{i=1}^{k+1}$.

Example 2.3 (Fourier basis of $L^2[0,1]$). Any orthogonal basis in a separable Hilbert space is a Schauder basis.

We denote the coordinate functional of e_i by e_i^*, and thus

$$x = \sum_{i=1}^{\infty} e_i^*(x) e_i \quad \forall x \in X.$$

Then for a constant n, the canonical projection P_n is defined as

$$P_n(x) = P_n\left(\sum_{i=1}^{\infty} e_i^*(x) e_i\right) = \sum_{i=1}^{n} e_i^*(x) e_i.$$

We have the following property for P_n, according to which, we can represent points in an infinite-dimensional Banach space by finite coordinates within a sufficiently small projection error.

PROPERTY 2.4 (canonical projection). *Assume that K is a compact set in a Banach space X equipped with a Schauder basis and corresponding canonical projections P_n; then we have*

$$\lim_{n\to\infty} \sup_{x\in K} \|x - P_n(x)\| = 0.$$

Proof. The proof can be found in Appendix A. ∎

For convenience, we decompose the P_n as

$$P_n = \psi_n \circ \varphi_n,$$

where $\varphi_n : X \to \mathbb{R}^n$ and $\psi_n : \mathbb{R}^n \to X$ are defined as

$$\varphi_n(x) = \left(e_1^*(x), \dots, e_n^*(x)\right)^T, \quad \psi_n(\alpha_1, \dots, \alpha_n) = \sum_{i=1}^{n} \alpha_i e_i.$$

The $\varphi_n(x)$ are essentially the truncated coordinates for x. Moreover, sometimes we can further replace $\{e_1, \dots, e_n\}$ with an equivalent basis for the decomposition of P_n, i.e.,

$$\hat{\varphi}_n(x) = Q(e_1^*(x), \dots, e_n^*(x))^T, \quad \hat{\psi}_n(\alpha_1, \dots, \alpha_n) = (e_1, \dots, e_n) Q^{-1} (\alpha_1, \dots, \alpha_n)^T,$$

with a nonsingular matrix $Q \in \mathbb{R}^{n\times n}$. For example, when applying the Faber–Schauder basis (Example 2.2), instead of using the coordinates based on the sequence $\{e_i\}_{i=1}^{\infty}$, we use the function values evaluated at certain grid points as the coordinates, which is the same as using the linear element basis in the finite element method.

2.3. Main theorems: Approximation theory for multiple-input operators. Here, we present the main approximation theorems in Theorem 2.5 and Corollary 2.6, and the proofs will be presented afterwards.

THEOREM 2.5. *Suppose that $X_1, \dots, X_n, Y$ are Banach spaces, $K_i \subset X_i$ are compact sets, and X_i have a Schauder basis with canonical projections $P_q^i = \psi_q^i \circ \varphi_q^i$. Assume that $\mathcal{G} : K_1 \times \cdots \times K_n \to Y$ is a continuous operator; then for any $\epsilon > 0$,* *there exist positive integers p_i, q_i, continuous vector functions $\mathbf{g}_i \in C(\mathbb{R}^{q_i}, \mathbb{R}^{p_i})$, and $\mathbf{u} = (u_{j_1\cdots j_n}) \in Y^{p_1\times\cdots\times p_n}$, such that*

$$\sup_{v_i\in K_i} \left\| \mathcal{G}(v_1,\ldots,v_n) - \mathbf{u}\left\langle \mathbf{g}_1(\varphi^1_{q_1}(v_1)),\ldots,\mathbf{g}_n(\varphi^n_{q_n}(v_n))\right\rangle \right\| < \epsilon. \tag{2.2}$$

COROLLARY 2.6. *The conclusion in Theorem 2.5 can also be expressed in the following equivalent forms.*

(i) *There exist positive integers p_i, q_i, r and continuous functions $\mathbf{g}_i \in C(\mathbb{R}^{q_i}, \mathbb{R}^{p_i})$, $\mathbf{u} \in Y^r$, and $W \in \mathbb{R}^{p_1\times\cdots\times p_n\times r}$, such that*

$$\sup_{v_i\in K_i} \left\| \mathcal{G}(v_1,\ldots,v_n) - W\left\langle \mathbf{g}_1(\varphi^1_{q_1}(v_1)),\ldots,\mathbf{g}_n(\varphi^n_{q_n}(v_n)),\mathbf{u}\right\rangle \right\| < \epsilon. \tag{2.3}$$

If $\{e_i\}$ is a Schauder basis for Y, we can further have $\mathbf{u} = (e_1, e_2, \ldots, e_r)^T$.

(ii) *There exist positive integers p, q_i and continuous functions $g^i_j \in C(\mathbb{R}^{q_i})$ and $u_j \in Y$, such that*

$$\sup_{v_i\in K_i} \left\| \mathcal{G}(v_1,\ldots,v_n) - \sum_{j=1}^{p} g^1_j(\varphi^1_{q_1}(v_1))\cdots g^n_j(\varphi^n_{q_n}(v_n))\cdot u_j \right\| < \epsilon. \tag{2.4}$$

The relations between these three results are as follows. We first prove (2.2); in (2.3), we treat $\mathbf{g}_i$ and $\mathbf{u}$ in (2.2) symmetrically and combine them via a tensor; (2.4) is simply a summation of products. In fact, when Y is a space of continuous function approximated by fully connected neural networks (FNNs), (2.2) and (2.3) are technically equivalent, since W can be regarded as the final linear output layer of the FNN for approximating $\mathbf{u}$. Therefore, we design two architectures in section 3, one based on (2.2)/(2.3) and the other based on (2.4).

Next we show two special cases of $n = 1$ based on the theory above. In Example 2.7, we choose the Faber–Schauder basis as a Schauder basis. In Example 2.8, we have the universal approximation theorem for DeepONets.

Example 2.7. Assume that K is a compact set in $C[0,1]$, and $\mathcal{G} : K \to C[0,1]$ is a continuous operator; then for any $\epsilon > 0$, there exist positive integers q, r and a continuous map $\mathbf{f} : \mathbb{R}^q \to \mathbb{R}^r$, such that

$$\left\| \mathcal{G}(v) - \sum_{i=1}^{r} f_i\left(v\left(\frac{0}{q-1}\right), v\left(\frac{1}{q-1}\right), \ldots, v\left(\frac{q-1}{q-1}\right)\right)\cdot e_i \right\| < \epsilon$$

holds for all $v \in K$, where $\mathbf{f} = (f_i)$. $\{e_i\}_{i=1}^r$ are chosen as the piecewise linear functions with grid points $\frac{j}{r-1}$, and $e_i(\frac{j-1}{r-1}) = \delta_{ij}$. In fact, f_i denotes the values of $\mathcal{G}(v)$ at $\frac{i-1}{r-1}$. This example is a direct conclusion of Example 2.2 and Corollary 2.6(i).

Example 2.8 (DeepONet). As a special case, for $n = 1$ in Theorem 2.5 we obtain the universal approximation theorem for DeepONet (Theorem 2 in [28]).

2.4. Detailed analysis.

We first introduce Lemma 2.9 and Theorem 2.10, which are used to prove the main theorems in section 2.3.

LEMMA 2.9. *Suppose that $X_1, \ldots, X_n, Y$ are Banach spaces and $K_i \subset X_i$ are compact sets. Assume that $\mathcal{G} : K_1 \times\cdots\times K_n \to Y$ is a continuous operator; then for any $\epsilon > 0$, there exist positive integers p_i and continuous vector functionals $\hat{\mathbf{g}}_i \in C(X_i, \mathbb{R}^{p_i})$ and $\mathbf{u} \in Y^{p_1\times p_2\times\cdots\times p_n}$, such that*

$$\sup_{v_i\in K_i} \| \mathcal{G}(v_1,\ldots,v_n) - \mathbf{u}\langle \hat{\mathbf{g}}_1(v_1),\ldots,\hat{\mathbf{g}}_n(v_n)\rangle \| < \epsilon.$$

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

Lemma 2.9 gives the approximation theory in the original infinite-dimensional Banach spaces. Next we extend to the following result.

THEOREM 2.10. *Suppose that $X_1, \dots, X_n, Y$ are Banach spaces, $K_i \subset X_i$ are compact sets, and X_i have a Schauder basis with canonical projections P_q^i. Assume that $\mathcal{G} : K_1 \times K_2 \times \cdots \times K_n \to Y$ is a continuous operator; then for any $\epsilon > 0$, there exist positive integers p_i, continuous vector functionals $\hat{\mathbf{g}}_i \in C(X_i, \mathbb{R}^{p_i})$, and $\mathbf{u} \in Y^{p_1 \times p_2 \times \cdots \times p_n}$, such that*

$$\sup_{v_i \in K_i} \left\| \mathcal{G}(v_1, \dots, v_n) - \mathbf{u}\langle \hat{\mathbf{g}}_1(P^1_{q_1}(v_1)), \dots, \hat{\mathbf{g}}_n(P^n_{q_n}(v_n)) \rangle \right\| < \epsilon + M \sum_{i=1}^n L_i^\epsilon(q_i) \tag{2.5}$$

holds for arbitrary positive integers q_i, where

$$L_i^\epsilon(q_i) = \sup_{v_i \in K_i} \left\| \hat{\mathbf{g}}_i \circ P^i_{q_i}(v_i) - \hat{\mathbf{g}}_i(v_i) \right\|_1, \quad M = \max_{v_i \in K_i} \| \mathcal{G}(v_1, \dots, v_n) \|.$$

Note that $L_i^\epsilon(q_i) \to 0$ as $q_i \to \infty$.

Proof. The proof can be found in Appendix C. ∎

Theorem 2.10 immediately derives Theorem 2.5 and Corollary 2.6 as shown in Appendices D and E. In (2.5), the first part of error "ϵ" is due to operator approximation, while the second part of error "$M \sum_{i=1}^n L_i^\epsilon(q_i)$" is due to the projection to finite-dimensional space. We note that L_i^ϵ depends on ϵ, and thus when ϵ is small, which makes L_i^ϵ converge more slowly, a large value of q_i is needed.

Next we show further analysis of these results, which also provides guidance for the design of the network architectures in section 3.

COROLLARY 2.11 (effect of a bias). *In Theorem* 2.10, *if $Y = C(K_0)$ for compact K_0 in a Banach space X_0, we take an additional bias $b \in \mathbb{R}$, such that* (2.5) *becomes*

$$\sup_{\substack{v_i \in K_i \\ y \in K_0}} \left| \mathcal{G}(v_1, \dots, v_n)(y) - \mathbf{f}(y) \left\langle \hat{\mathbf{g}}_1(P^1_{q_1}(v_1)), \dots, \hat{\mathbf{g}}_n(P^n_{q_n}(v_n)) \right\rangle - b \right| < \epsilon + M \sum_{i=1}^n L_i^\epsilon(q_i),$$

where $\mathbf{f} \in C(K_0, \mathbb{R}^{p_1 \times p_2 \times \cdots \times p_n})$ and

$$M = \frac{1}{2} \left(\max_{v_i \in K_i, y \in K_0} \mathcal{G}(v_1, \dots, v_n)(y) - \min_{v_i \in K_i, y \in K_0} \mathcal{G}(v_1, \dots, v_n)(y) \right).$$

Proof. Replace $\mathcal{G}(v_1, \dots, v_n)$ by $\mathcal{G}(v_1, \dots, v_n) - b$ in Theorem 2.10, where

$$b = \frac{1}{2} \left(\max_{v_i \in K_i, y \in K_0} \mathcal{G}(v_1, \dots, v_n)(y) + \min_{v_i \in K_i, y \in K_0} \mathcal{G}(v_1, \dots, v_n)(y) \right). \qquad ∎$$

Corollary 2.11 suggests adding a bias, which makes the constant M smaller and thus decreases the error. In addition, we explore more characteristics of Theorem 2.5 for learning multiple operators.

COROLLARY 2.12 (approximation theory for multiple operators). *Suppose that $X_1, \dots, X_n, Y_1, \dots, Y_m$ are Banach spaces, $K_i \subset X_i$ are compact sets, and X_i have a Schauder basis with canonical projections $P_q^i = \psi_q^i \circ \varphi_q^i$. Assume that $\mathcal{G}_j : K_1 \times \cdots \times K_n \to Y_j$ are continuous operators, then for any $\epsilon > 0$ the following hold:*

(i) *There exist positive integers* p_i, q_i *and continuous vector functions* $\mathbf{g}_i \in C(\mathbb{R}^{q_i}, \mathbb{R}^{p_i})$ *and* $\mathbf{u}_j \in Y_j^{p_1 \times p_2 \times \cdots \times p_n}$*, such that*

$$\sup_{v_i \in K_i} \left\| \mathcal{G}_j(v_1, \ldots, v_n) - \mathbf{u}_j \langle \mathbf{g}_1(\varphi^1_{q_1}(v_1)), \ldots, \mathbf{g}_n(\varphi^n_{q_n}(v_n)) \rangle \right\| < \epsilon, \quad 1 \le j \le m.$$

(ii) *There exist positive integers* p_i, q_i, r_j *and continuous functions* $\mathbf{g}_i \in C(\mathbb{R}^{q_i}, \mathbb{R}^{p_i})$, $\mathbf{u}_j \in Y_j^{r_j}$*, and* $W_j \in \mathbb{R}^{p_1 \times \cdots \times p_n \times r_j}$*, such that*

$$\sup_{v_i \in K_i} \left\| \mathcal{G}_j(v_1, \ldots, v_n) - W_j \langle \mathbf{g}_1(\varphi^1_{q_1}(v_1)), \ldots, \mathbf{g}_n(\varphi^n_{q_n}(v_n)), \mathbf{u}_j \rangle \right\| < \epsilon, \quad 1 \le j \le m.$$

If $\{e^j_k\}$ *is a Schauder basis for* Y_j*, we can further have* $\mathbf{u}_j = (e^j_1, e^j_2, \ldots, e^j_{r_j})^T$.

(iii) *There exist positive integers* p, q_i *and continuous functions* $g^i_k \in C(\mathbb{R}^{q_i})$ *and* $u^j_k \in Y_j$*, such that*

$$\sup_{v_i \in K_i} \left\| \mathcal{G}_j(v_1, \ldots, v_n) - \sum_{k=1}^{p} g^1_k(\varphi^1_{q_1}(v_1)) \cdots g^n_k(\varphi^n_{q_n}(v_n)) \cdot u^j_k \right\| < \epsilon, \quad 1 \le j \le m.$$

Proof. Replace Y by $Y_1 \times \cdots \times Y_m$ in Theorem 2.5. □

We list Corollary 2.12 here to emphasize that multiple operators defined on the same product space can share the same $\mathbf{g}_i$ or g^i_j, which indicates a practical approximation method for operators mapping from $X_1 \times \cdots \times X_n$ to $Y_1 \times \cdots \times Y_m$.

COROLLARY 2.13 (linear case). *In Theorem* 2.5 *and Corollary* 2.6*, if* $\mathcal{G}$ *is linear with respect to* v_i*, then linear* $\mathbf{g}_i$ *and* g^i_j *are sufficient.*

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

By Corollary 2.13, if we know the operator is linear with respect to v_i; then we can choose $\mathbf{g}_i$ or g^i_j as linear maps in practice to make the learning procedure easier and generalize better.

PROPERTY 2.14. *In Theorem* 2.5 *and Corollary* 2.6*, if* $\mathbf{g}_i = W_i \cdot \mathbf{h}_i$ *for* $W_i \in \mathbb{R}^{p_i \times h_i}$ *and* $\mathbf{h}_i \in C(\mathbb{R}^{q_i}, \mathbb{R}^{h_i})$*, then* (2.2) *and* (2.3) *can be rewritten as*

(i) $\sup_{v_i \in K_i} \left\| \mathcal{G}(v_1, \ldots, v_n) - \tilde{\mathbf{u}} \langle \mathbf{h}_1(\varphi^1_{q_1}(v_1)), \ldots, \mathbf{h}_n(\varphi^n_{q_n}(v_n)) \rangle \right\| < \epsilon,$

(ii) $\sup_{v_i \in K_i} \left\| \mathcal{G}(v_1, \ldots, v_n) - \tilde{W} \langle \mathbf{h}_1(\varphi^1_{q_1}(v_1)), \ldots, \mathbf{h}_n(\varphi^n_{q_n}(v_n)), \mathbf{u} \rangle \right\| < \epsilon,$

respectively, for a new $\tilde{\mathbf{u}} \in Y^{h_1 \times \cdots \times h_n}$ *and a new* $\tilde{W} \in \mathbb{R}^{h_1 \times \cdots \times h_n \times r}$.

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

Property 2.14 shows that the linear output layers of $\mathbf{g}_i$ are allowed to be removed without loss of universality. For example, when $\mathbf{g}_i$ are chosen as FNNs, we can eliminate the redundant parameters of the last linear layer.

COROLLARY 2.15 (universal approximation theorem for functions). *Assume that* $f : K_1 \times \cdots \times K_n \to \mathbb{R}$ *is a continuous function for compact* $K_i \subset \mathbb{R}^{q_i}$ *and* σ *is an activation function which satisfies the requirements for the approximation theorem of fully connected neural networks; then for any* $\epsilon > 0$ *the following hold:*

(i) *There exist integers* p_i*, weights* $W_i \in \mathbb{R}^{p_i \times q_i}$, $W \in \mathbb{R}^{p_1 \times \cdots \times p_n}$*, and biases* $b_i \in \mathbb{R}^{p_i}$*, such that*

$$\| f - W \langle \sigma(W_1(\cdot) + b_1), \ldots, \sigma(W_n(\cdot) + b_n) \rangle \|_{C(K_1 \times \cdots \times K_n)} < \epsilon.$$

(ii) *There exist integer p, weights $W_i \in \mathbb{R}^{p_i \times q_i}$, $w_j^i \in \mathbb{R}^{1\times p_i}$, and biases $b_i \in \mathbb{R}^{p_i}$, such that*

$$\left\| f - \sum_{j=1}^{p} \left(w_j^1 \sigma(W_1(\cdot) + b_1)\right) \cdots \left(w_j^n \sigma(W_n(\cdot) + b_n)\right) \right\|_{C(K_1 \times \cdots \times K_n)} < \epsilon.$$

Proof. We take $X_i = \mathbb{R}^{q_i}$ and $Y = \mathbb{R}$ in Lemma 2.9. Then the corollary can be obtained by the universal approximation theorem for fully connected neural networks with one hidden layer and Property 2.14. ∎

When $n = 1$, Corollary 2.15 degenerates to the classical universal approximation theorem for fully connected neural networks with one hidden layer. Generally speaking, $V_1 \times \cdots \times V_n$ can be regarded as a compact $V \in \mathbb{R}^{q_1+\cdots+q_n}$, and thus f can also be approximated by FNNs. Compared to FNNs, the two new architectures in Corollary 2.15 divide the input components into several different groups.

Remark 2.16 (tensor neural network). Here, we introduce the tensor neural network (TNN) as a general function approximator mapping from $\mathbb{R}^d$ to $\mathbb{R}^m$. Assume that $d = q_1 + \cdots + q_n$ for positive integers $q_1, \ldots, q_n$; then a TNN $\Phi : \mathbb{R}^d \to \mathbb{R}^m$ can be written as

$$\Phi(x_1, \ldots, x_n) \coloneqq W(\Psi_1(x_1) \odot \cdots \odot \Psi_n(x_n)), \quad x_i \in \mathbb{R}^{q_i}, \quad 1 \le i \le n, \tag{2.6}$$

where $\odot$ is the Hadamard product (i.e., elementwise product), $\Psi_i : \mathbb{R}^{q_i} \to \mathbb{R}^p$ are standard neural networks (such as FNNs), and $W \in \mathbb{R}^{m\times p}$ is a trainable weight matrix. The approximation property for TNN can be obtained via setting $X_i = \mathbb{R}^{q_i}$ and $Y = \mathbb{R}^m$ in Theorem 2.5 and Corollary 2.6. A bias can also be added to (2.6).

2.5. View through the tensor product of Banach spaces.

We have presented all the main theorems and related analysis. Here, we provide another view of the theory through the tensor product of Banach spaces; this section is optional, and the reader's understanding of the subsequent content will not be hindered if it is skipped. We also refer the reader to [40] for more details.

Recall that we aim to learn a continuous operator

$$\mathcal{G} \in C(K, Y), \quad K = K_1 \times \cdots \times K_n,$$

where K_i is a compact set in a Banach space X_i. By the injective tensor product, we have

$$C(K, Y) \cong C(K)\hat{\otimes}_\varepsilon Y,$$

with the canonical linear map defined as

$$\begin{aligned} J : C(K) \otimes Y &\to C(K, Y), \\ \sum_{j=1}^{p} f_j \otimes u_j &\mapsto \sum_{j=1}^{p} f_j \cdot u_j \end{aligned}$$

for representation $\mu = \sum_{j=1}^{p} f_j \otimes u_j \in C(K) \otimes Y$. Here, $C(K)\hat{\otimes}_\varepsilon Y$ is the completion of $C(K) \otimes Y$ with the injective norm $\varepsilon(\mu) = \sup_{v\in K} \left\| \sum_{j=1}^{p} f_j(v) u_j \right\|$, and we have the isometric isomorphism between $C(K)\hat{\otimes}_\varepsilon Y$ and $C(K, Y)$; for convenience it is still denoted as J. Then for $\mathcal{G} \in C(K, Y)$ and any $\epsilon > 0$, there exists $\sum_{j=1}^{p} f_j \otimes u_j$ such that

$$\left\| \mathcal{G} - \sum_{j=1}^{p} f_j \cdot u_j \right\|_{C(K,Y)} = \varepsilon \left(J^{-1}\mathcal{G} - \sum_{j=1}^{p} f_j \otimes u_j \right) < \epsilon. \tag{2.7}$$

Furthermore, by repeating the decomposition

$$\begin{aligned} C(K_1 \times \cdots \times K_n) &\cong C(K_1 \times \cdots \times K_{n-1}, C(K_n)) \\ &\cong C(K_1 \times \cdots \times K_{n-1}) \hat{\otimes}_\varepsilon C(K_n), \end{aligned}$$

we obtain

$$C(K_1 \times K_2 \times \cdots \times K_n, Y) \cong C(K_1) \hat{\otimes}_\varepsilon C(K_2) \hat{\otimes}_\varepsilon \cdots \hat{\otimes}_\varepsilon C(K_n) \hat{\otimes}_\varepsilon Y. \tag{2.8}$$

Similar to (2.7), we have

$$\left\| \mathcal{G} - \sum_{j=1}^{p} f_j^1 \cdot f_j^2 \cdots f_j^n \cdot u_j \right\|_{C(K,Y)} < \epsilon \tag{2.9}$$

for some $f_j^i \in C(K_i)$ and $u_j \in Y$. Note that $C(K_i)$, as a continuous function space on compact K_i, has a Schauder basis, denoted as $\{g_k^i\}_{k=1}^{\infty}$. Let $f_j^i = \sum_{k=1}^{\infty} \alpha_{jk}^i g_k^i$; then there exist positive integers p_i such that

$$\|\mathcal{G} - \mathbf{u}\langle \mathbf{g}_1, \ldots, \mathbf{g}_n \rangle\|_{C(K,Y)} < \epsilon \tag{2.10}$$

for $\mathbf{g}_i = \left(g_1^i, \ldots, g_{p_i}^i\right)^T$, $\mathbf{u} = \left(\sum_{j=1}^{p} \alpha_{jk_1}^1 \cdots \alpha_{jk_n}^n u_j\right)_{p_1 \times \cdots \times p_n}$. Furthermore, if Y has a Schauder basis $\{e_k\}$ and $u_j = \sum_{k=1}^{\infty} \beta_{jk} e_k$, there exists an r such that

$$\|\mathcal{G} - W\langle \mathbf{g}_1, \ldots, \mathbf{g}_n, \mathbf{e} \rangle\|_{C(K,Y)} < \epsilon \tag{2.11}$$

for $\mathbf{e} = (e_1, \ldots, e_r)^T$ and $W = \left(\sum_{j=1}^{p} \alpha_{jk_1}^1 \cdots \alpha_{jk_n}^n \beta_{jk_{n+1}}\right) \in \mathbb{R}^{p_1 \times \cdots \times p_n \times r}$. Hence, we have also obtained the three approximation formulas (2.9)–(2.11) corresponding to (2.2)–(2.4), with the additional information that the components of $\mathbf{g}_i$ and $\mathbf{e}$ are all basis functions.

Next we analyze the complexity of the approximation in terms of tensor rank. The operator $\mathcal{G}$ in (2.11) is discretely represented by a tensor $W \in \mathbb{R}^{p_1 \times \cdots \times p_n \times r}$, since $\mathbf{g}_i$ and $\mathbf{e}$ are fixed bases. The number of parameters of W grows exponentially with respect to n, so directly using W in computation is too expensive for large n. However, if we rewrite W as

$$W = \left(\sum_{j=1}^{p} \alpha_{jk_1}^1 \cdots \alpha_{jk_n}^n \beta_{jk_{n+1}} \right) = \sum_{j=1}^{p} \mathbf{a}_j^1 \otimes \cdots \otimes \mathbf{a}_j^n \otimes \mathbf{b}_j$$

for $\mathbf{a}_j^i = (\alpha_{jk}^i) \in \mathbb{R}^{p_i}$, $\mathbf{b}_j = (\beta_{jk}) \in \mathbb{R}^r$, then we have

$$W\langle \mathbf{g}_1, \ldots, \mathbf{g}_n, \mathbf{e} \rangle = \sum_{j=1}^{p} f_j^1 \cdot f_j^2 \cdots f_j^n \cdot u_j \tag{2.12}$$

for $f_j^i = \sum_{k=1}^{p_i} \alpha_{jk}^i g_k^i$, $u_j = \sum_{k=1}^{r} \beta_{jk} e_k$. Here, W in (2.12) is a tensor of rank at most p. From this point of view, (2.9) also gives a low-rank approximation by the tensor

$$\mu = \sum_{j=1}^{p} f_j^1 \otimes \cdots \otimes f_j^n \otimes u_j$$

of rank at most p. We note that it is usually difficult to determine the rank of high-order tensors, which is NP-hard [11], but in some cases there exist some relationships between the dimension of W and its rank p. For example, if $W \in \mathbb{R}^{p_1 \times p_2 \times p_3}$, then the rank of W has an upper bound [16, 19]:

$$\operatorname{rank}(W) \leq \min\{p_1p_2, p_1p_3, p_2p_3\}.$$

In short, we have a more general viewpoint for our results. Assume that $Y = C(K_0)$ for a compact set K_0 in a Banach space X_0; then, depending on the level of decomposition applied, we have the following cases:

$$C(K,Y) = \begin{cases} C(K_1 \times \cdots \times K_n \times K_0) & \text{(standard NN)}, \quad (2.13)\\ C(K_1 \times \cdots \times K_n)\hat{\otimes}_\varepsilon C(K_0) & \text{(DeepONet)}, \quad (2.14)\\ C(K_1)\hat{\otimes}_\varepsilon \cdots \hat{\otimes}_\varepsilon C(K_n)\hat{\otimes}_\varepsilon C(K_0) & \text{(MIONet)}, \quad (2.15)\end{cases}$$

where $K = K_1 \times \cdots \times K_n$. We discuss how the three different representations lead to different network architectures as follows.

- In (2.13), we first combine all the inputs from all the spaces and then pass them into a function (i.e., a machine learning model) to approximate $\mathcal{G}$. When we restrict the model to be a neural network, it is a standard NN, such as FNN, residual neural network (ResNet), convolutional neural network (CNN), etc.
- In (2.14), we first split the input and output spaces, and have one model for the input space and one model for the output space, and then we combine them. When both models are standard NNs, this leads to the same architecture as DeepONet [28]. Therefore, DeepONet with this simple extension can handle the multiple-input case via treating $X_1 \times \cdots \times X_n$ as a single Banach space. However, this treatment will lose the structure of the product space, which leads to a large generalization error, as we demonstrate in our numerical experiments.
- In (2.15), we split all the spaces with one model for each space, and then combine them to compute the output. This leads to our proposed MIONet in section 3, where each model is a standard NN.

3. Operator regression methods.

Based on our theory, we propose a new neural operator, MIONet, for learning multiple-input operator regression.

3.1. Network architectures.

The architectures of MIONet are designed based on Theorem 2.5 and Corollary 2.6 with $Y = C(K_0)$ for a compact set $K_0 \subset \mathbb{R}^d$. We design two slightly different versions of MIONet according to different formulas as follows.

MIONet (high-rank). We first construct the architecture according to (2.2) and (2.3). Note that the architecture induced by (2.3) is technically equivalent to (2.2) as we discussed in section 2.3. Specifically, we use $\mathbf{f} \in C(K_0, \mathbb{R}^{p_1 \times \cdots \times p_n})$ to denote the $\mathbf{u}$ in (2.2), and we approximate $\mathbf{g}_i$ and $\mathbf{f}$ by independent neural networks denoted by

$\tilde{\mathbf{g}}_i$ (called branch net i) and $\tilde{\mathbf{f}}$ (called trunk net). We also add a trainable bias $b \in \mathbb{R}$ according to Corollary 2.11. Then the network is

$$\tilde{\mathcal{G}}(v_1, \dots, v_n)(y) = \underbrace{\tilde{\mathbf{f}}(y)}_{\text{trunk}} \left\langle \underbrace{\tilde{\mathbf{g}}_1(\varphi^1_{q_1}(v_1))}_{\text{branch}_1}, \dots, \underbrace{\tilde{\mathbf{g}}_n(\varphi^n_{q_n}(v_n))}_{\text{branch}_n} \right\rangle + b. \tag{3.1}$$

MIONet has n independent branch nets and one trunk net. The ith branch net $\tilde{\mathbf{g}}_i$ encodes the input function v_i, and the trunk net $\tilde{\mathbf{f}}$ encodes the input y. The output tensor of the trunk net has a high rank as we discussed in section 2.5. We note that the last linear layer of each branch net can be removed by Property 2.14 to reduce the number of parameters.

MIONet (low-rank; the default version). We then construct MIONet according to (2.4). Specifically, $\mathbf{g}_i = (g^i_1, \dots, g^i_p)^T \in C(\mathbb{R}^{q_i}, \mathbb{R}^p)$ and $\mathbf{f} = (u_1, \dots, u_p)^T \in C(K_0, \mathbb{R}^p)$ are approximated by neural networks $\tilde{\mathbf{g}}_i$ (called branch net i) and $\tilde{\mathbf{f}}$ (called trunk net). Then the network (Figure 1) is

$$\tilde{\mathcal{G}}(v_1, \dots, v_n)(y) = \mathcal{S}\left(\underbrace{\tilde{\mathbf{g}}_1(\varphi^1_{q_1}(v_1))}_{\text{branch}_1} \odot \cdots \odot \underbrace{\tilde{\mathbf{g}}_n(\varphi^n_{q_n}(v_n))}_{\text{branch}_n} \odot \underbrace{\tilde{\mathbf{f}}(y)}_{\text{trunk}} \right) + b, \tag{3.2}$$

where $\odot$ is the Hadamard product, $\mathcal{S}$ is the summation of all the components of a vector, and $b \in \mathbb{R}$ is a trainable bias. This MIONet is a low-rank version of the MIONet (high-rank) above, which greatly reduces the number of parameters of the trunk net. Furthermore, if the output function is also defined on a product space, e.g., $C(K_0) = C(K_{01} \times K_{02})$, we can choose to further decompose it into $C(K_{01}) \hat{\otimes}_\varepsilon C(K_{02})$, and the corresponding MIONet can be built similarly. As a special case, if the image space Y is finite dimensional, we show the corresponding low-rank MIONet in Appendix H.

The universal approximation theorem for MIONet can be easily obtained in Theorem 3.1 from Theorem 2.5 and Corollary 2.6.

THEOREM 3.1 (universal approximation theorem for MIONet). *Under the setting of Theorem* 2.5 *with* $Y = C(K_0)$, *for any* $\epsilon > 0$ *there exists a MIONet (low-/high-rank)* $\tilde{\mathcal{G}}$, *such that*

$$\left\| \mathcal{G} - \tilde{\mathcal{G}} \right\|_{C(K_1 \times \cdots \times K_n, C(K_0))} < \epsilon.$$

Proof. We consider $\mathcal{F}$ mapping from $C(\varphi^1_{q_1}(K_1), \mathbb{R}^p) \times \cdots \times C(\varphi^n_{q_n}(K_n), \mathbb{R}^p) \times C(K_0, \mathbb{R}^p)$ to $C(K_1 \times \cdots \times K_n, C(K_0))$ as

$$\mathcal{F}(\mathbf{g}_1, \dots, \mathbf{g}_n, \mathbf{f})(v_1, \dots, v_n) \coloneqq \mathcal{S}\left(\mathbf{g}_1(\varphi^1_{q_1}(v_1)) \odot \cdots \odot \mathbf{g}_n(\varphi^n_{q_n}(v_n)) \odot \mathbf{f}\right).$$

It is easy to verify that $\mathcal{F}$ is continuous. Then the universal approximation theorem of MIONet can be easily obtained from Theorem 2.5, Corollary 2.6, and the approximation property of neural networks. ∎

Connections to DeepONet. Our proposed MIONet is related to DeepONet. When there is only one input function, i.e., $n = 1$, MIONet becomes DeepONet with one branch net and one trunk net.

Remark. In this work, we mainly consider MIONet (low-rank), as MIONet (high-rank) is computationally expensive. We note that MIONet is a high-level architecture, where the neural networks $\tilde{\mathbf{g}}_i$ and $\tilde{\mathbf{f}}$ can be chosen as any valid NNs, such as FNN,

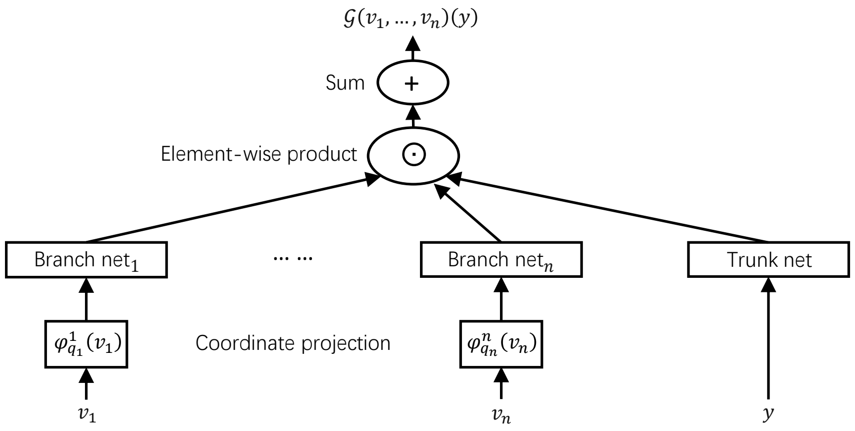

FIG. 1. *Architecture of MIONet. All the branch nets and the trunk net have the same number of outputs, which are merged together via the Hadamard product and a subsequent summation.*

ResNet, CNN, etc., depending on the specific problem. All the techniques developed for DeepONet in [29] can be directly used for MIONet. For example, we can encode the periodicity in the trunk net to ensure that the predict functions from MIONet are always periodic. We refer the reader to [29] for more details.

3.2. Other computational details.

Data. One data point in the dataset is comprised of input functions and their corresponding output function, i.e., $(v_1, \ldots, v_n, \mathcal{G}(v_1, \ldots, v_n))$. In the first step of MIONet, we project the input functions v_i onto finite-dimensional spaces as stated in (3.1) and (3.2), which can be done separately before the network training. Hence, the network input in practice is $(\varphi^1_{q_1}(v_1), \ldots, \varphi^n_{q_n}(v_n))$, and then the dataset takes the form

$$\mathcal{T} = \left\{ \left(\underbrace{\varphi^1_{q_1}(v^k_1)}_{\text{branch}_1}, \ldots, \underbrace{\varphi^n_{q_n}(v^k_n)}_{\text{branch}_n}, \underbrace{y_k}_{\text{trunk}}, \underbrace{\mathcal{G}\left(v^k_1, \ldots, v^k_n\right)(y_k)}_{\text{output}} \right) \right\}_{k=1}^{N}, \tag{3.3}$$

where $y_k \in K_0 \subset \mathbb{R}^d$ is a single point location in the domain of the output function.

Training. For a training dataset $\mathcal{T}$, in this study we use a standard mean squared error (MSE),

$$\text{MSE} = \frac{1}{N} \sum_{k=1}^{N} |s_k - \tilde{\mathcal{G}}(v^k_1, \ldots, v^k_n)(y_k)|^2.$$

We also provides alternative losses via numerical integration in Appendix I.

Inference. For new input functions $v_1, \ldots, v_n$, the prediction is simply given by $\tilde{\mathcal{G}}(v_1, \ldots, v_n)$. We note that $\tilde{\mathcal{G}}(v_1, \ldots, v_n)$ is a function given by neural networks, which can be evaluated at arbitrary points without interpolation.

4. Numerical results. To demonstrate the capability of MIONet, we learn three different operators of ODEs and PDEs. In the experiments, we directly evaluate the function values at uniform grid points as the input of branch nets, i.e., each φ takes 100 equidistant sampling points in $[0, 1]$ for each input function. The branch and trunk nets are all chosen as fully connected neural networks (FNNs) unless noted otherwise. Each branch or trunk net has the same number of neurons (i.e., width) for each layer. The activation in all networks is set to ReLU. We train all the networks by the Adam optimizer [13]. To evaluate the performance of the networks, we compute the L^2 relative error of the predictions, and for each case five independent training trials are performed to compute the mean error and the standard deviation. The code in this study is implemented by using the library DeepXDE [30] and is publicly available from the GitHub repository https://github.com/lu-group/mionet.

MIONet is the first neuron operator designed for multiple inputs, and there is no other network that can be directly compared to MIONet. In order to compare the performance of MIONet and DeepONet, we concatenate all the input functions together as a single input of DeepONet branch net as discussed in section 2.5. Specifically, if we assume that $\{e_j^i\}$ is the Schauder basis of X_i, then $\{(e_j^1, 0, \dots, 0)\} \cup \{(0, e_j^2, 0, \dots, 0)\} \cup \cdots \cup \{(0, \dots, 0, e_j^n)\}$ is, in fact, a Schauder basis of the Banach space $X_1 \times \cdots \times X_n$.

The hyperparameters of DeepONet and MIONet are chosen as follows. For each case, we perform a grid search for the depth and width to find the best accuracy of DeepONet. Then we set the depth of MIONet to be the same as DeepONet and manually tune the width, so it is possible that MIONet can achieve even better accuracy than results reported below.

4.1. An ODE system. We first consider a nonlinear ODE system,

$$\frac{du_1}{dt} = u_2, \quad \frac{du_2}{dt} = -f_1(t)\sin(u_1) + f_2(t), \quad t \in [0, 1],$$

with an initial condition $u_1(0) = u_2(0) = 0$. We learn the operator mapping from f_1 and f_2 to one of the ODE solutions u_1:

$$\mathcal{G} : (f_1, f_2) \mapsto u_1.$$

To generate the dataset, f_1 and f_2 are both sampled from a Gaussian random field (GRF)

$$\mathcal{GP}(0, k_l(x_1, x_2)),$$

where the covariance kernel $k_l(x_1, x_2) = \exp(-\|x_1 - x_2\|^2/2l^2)$ is the Gaussian kernel with a length-scale parameter l. Here, we choose $l = 0.2$. We set the number of functions in the training set to 1000 and in the test set to 100,000, and for each couple of (f_1, f_2) we get the numerical solution of u_1 at 100 equidistant grid points in $[0, 1]$. We train the networks for 100,000 epochs with learning rate 0.001.

MIONet has the L^2 relative error of 1.69% (Table 1), which outperforms DeepONet with almost the same number of parameters (2.41%). We also perform a grid search for the depth and width to find the best accuracy of DeepONet; the best accuracy is 2.26%, which is still worse than MIONet.

TABLE 1
MIONet and DeepONet for an ODE system. DeepONet (same size) has the same number of parameters as MIONet. DeepONet (best) is the best result chosen from depth 2–5 *and width* 100–400.

	Depth	Width	No. of parameters	L^2 relative error
MIONet	2	200	161K	$1.69 \pm 0.13\%$
DeepONet (same size)	2	312	161K	$2.41 \pm 0.27\%$
DeepONet (best)	2	300	151K	$2.26 \pm 0.14\%$

4.2. A diffusion-reaction system. We consider a nonlinear diffusion-reaction system

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(D(x)\frac{\partial u}{\partial x}\right) + ku^2 + g(x), \quad x \in [0,1],\ t \in [0,1],$$

with zero initial and boundary conditions, where $D(x) = 0.01(|f(x)|+1)$ and $k = 0.01$ is the reaction rate. We learn the operator

$$\mathcal{G} : (D, g) \mapsto u.$$

In the dataset, f and g are generated by GRF with length scale 0.2. We set the number of couples of (D, g) in the training dataset to 1000 and in the test dataset to 5000, and for each couple we solve u in a grid with 100×100 equidistant points. We train each case for 100,000 epochs with learning rate 0.001.

The error of MIONet is significantly less than that of DeepONet of similar size and also that of the best DeepONet (Table 2). In Figure 2, we show an example of the inputs and the corresponding PDE solution. We also show the prediction and pointwise error of DeepONet and MIONet.

TABLE 2
MIONet and DeepONet for a diffusion-reaction system. DeepONet (same size) has the same number of parameters as MIONet. DeepONet (best) is the best DeepONet result chosen from depth 2–5 *and width* 100–400.

	Depth	Width	Parameters	L^2 relative error
MIONet	2	200	161K	$1.97 \pm 0.11\%$
DeepONet (same size)	2	312	161K	$5.25 \pm 0.38\%$
DeepONet (best)	2	400	242K	$5.18 \pm 0.11\%$

4.3. An advection-diffusion system. We consider an advection-diffusion system

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

with the periodic boundary condition and the initial condition $u_0(x) = u(x,0) = f_1(\sin^2(\pi x))$, where $D(x) = 0.05|f_2(\sin^2(\pi x)| + 0.05$ is the diffusion coefficient. We aim to learn the operator

$$\mathcal{G} : (D, u_0) \mapsto u.$$

In the dataset, f_1 and f_2 are sampled from a GRF with the length scale 0.5. The training/test dataset consists of 1000 couples of (D, u_0). For each (D, u_0), we solve the solution u numerically in a grid of size 100×100 and randomly select 100 values of u out of the 10,000 grid points. We train each case for 100,000 epochs with learning rate 0.0002.

Here, we show how to encode the prior information of this problem. Since the operator $\mathcal{G}$ is linear with respect to the initial condition u_0, we choose the branch net for u_0 in MIONet to be a linear network, i.e., a linear layer without bias. Moreover, because the solution u is periodic with respect to x, we decompose the single trunk net to two independent networks, one for x and one for t. For the trunk net of x, we apply a periodic layer as the input of FNN [29]:

$$\text{Trunk}(x) = \text{FNN}(\cos(2\pi x), \sin(2\pi x), \cos(4\pi x), \sin(4\pi x)), \quad x \in \mathbb{R}.$$

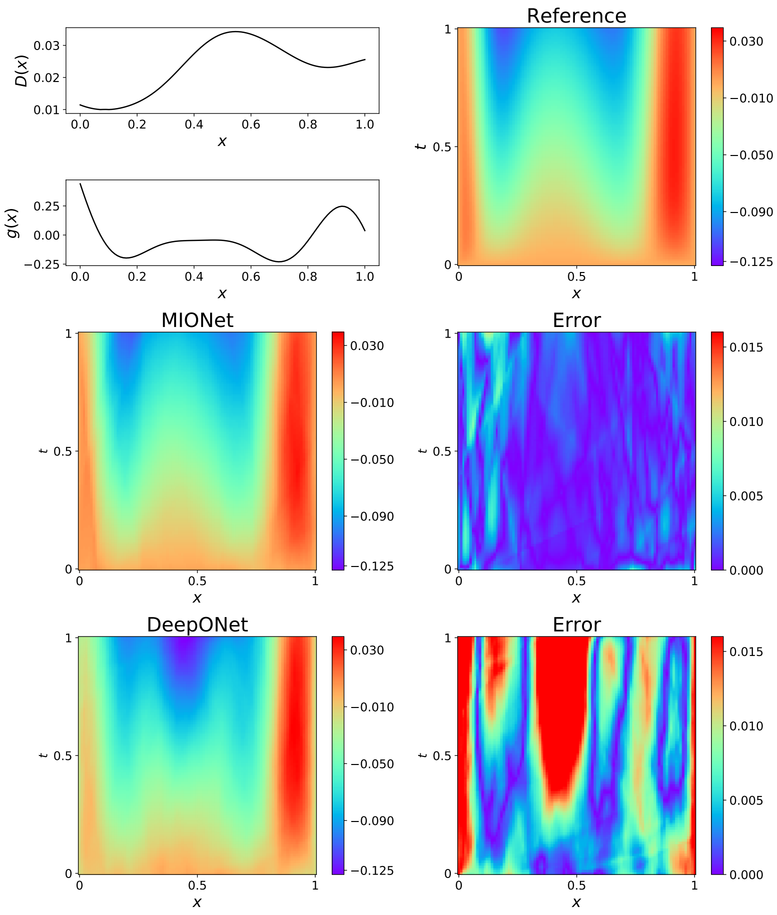

FIG. 2. *Example of the diffusion-reaction system. (Top) Examples of the input functions (left) and the reference solution (right). (Middle) MIONet prediction and corresponding absolute error. (Bottom) DeepONet prediction and corresponding absolute error.*

It is easy to check that by using these cos and sin features, MIONet is automatically periodic with respect to x. We present the illustration of the modified MIONet in Figure 3.

We show the accuracy of different networks in Table 3. MIONet performs significantly better than DeepONet (same size) and DeepONet (best). By encoding the periodicity information, MIONet (periodic) obtains the smallest prediction error. An example of prediction of MIONet (periodic) is shown in Figure 4.

We also investigate how the dataset size affects the accuracy of MIONet and DeepONet. We train the MIONet (periodic) and DeepONet (best) architectures in Table 3 for dataset sizes ranging from 200 to 4000, and the results are shown in Figure 5. The prediction errors of both MIONet and DeepONet monotonically decrease with the increase of dataset size. MIONet always performs better than DeepONet, but for a larger dataset, the difference becomes smaller.

5. Conclusions. In this study, we aim to learn an operator mapping from a product of multiple Banach spaces to another Banach space. Our main contribution

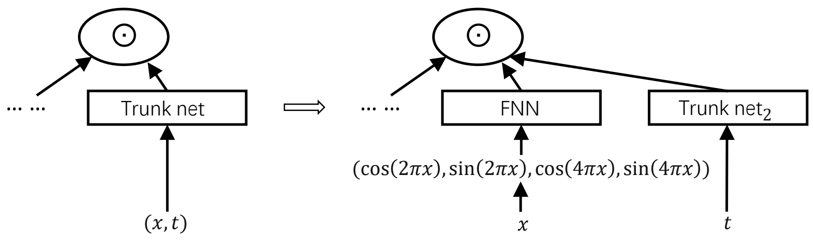

Fig. 3. *Architecture of the modified MIONet for the advection-diffusion system. There are two trunk nets, one for x and one for t. The trunk net of x has a periodic layer.*

Table 3
MIONet and DeepONet for the advection-diffusion system. MIONet (periodic) has a periodic layer for the trunk net of x. MIONet, MIONet (periodic), and DeepONet (same size) have the same number of parameters. DeepONet (best) is the best DeepONet chosen from depth 2–5 and width 100–400.

	Depth	Width	Parameters	L^2 relative error
MIONet	3	300	422K	$1.98 \pm 0.07\%$
MIONet (periodic)	3	248	422K	$1.29 \pm 0.09\%$
DeepONet (same size)	3	343	424K	$7.83 \pm 0.49\%$
DeepONet (best)	3	300	332K	$7.70 \pm 0.69\%$

is that for the first time, we provide universal approximation theorems for multiple-input operator regression based on the tensor product of Banach spaces. Based on the theory and a low-rank tensor approximation, we propose a new network architecture, MIONet, which consists of multiple branch nets for encoding the input functions and one trunk net for encoding the domain of the output function. To show the effectiveness of MIONet, we have performed three experiments including an ODE system, a diffusion-reaction system, and an advection-diffusion system. We also show that it is flexible to customize MIONet to encode the prior knowledge.

In future work, more experiments should be done to test the performance of MIONet on diverse problems. Moreover, MIONet can be viewed as an extension of DeepONet from a single branch net to multiple branch nets, and thus recent developments and extensions of DeepONet (see the discussion in the introduction) can be directly applied to MIONet. For example, similar to DeepONet with proper orthogonal decomposition (POD-DeepONet) [29], we can employ POD in MIONet to develop POD-MIONet. We can also embed physics into the loss function [43, 10] of MIONet to develop physics-informed MIONet. These techniques will further improve the accuracy and efficiency of MIONet.

Appendix A. Proof of Property 2.4.

Since X is a Banach space, P_n is uniformly bounded by the basis constant C. For any $\epsilon > 0$, we choose finite points $\{x_i\}_{i=1}^k \subset K$, such that the union of open balls $\cup_{i=1}^k B(x_i, \delta)$ covers K, where $\delta = \frac{\epsilon}{2(1+C)}$. There exists a large integer $m \in \mathbb{N}^*$ such that $\|x_i - P_n(x_i)\| < \frac{\epsilon}{2}$ holds for all $1 \leq i \leq k$ and $n \geq m$. When $n \geq m$, for any $x \in K$, assume that $x \in B(x_j, \delta)$;

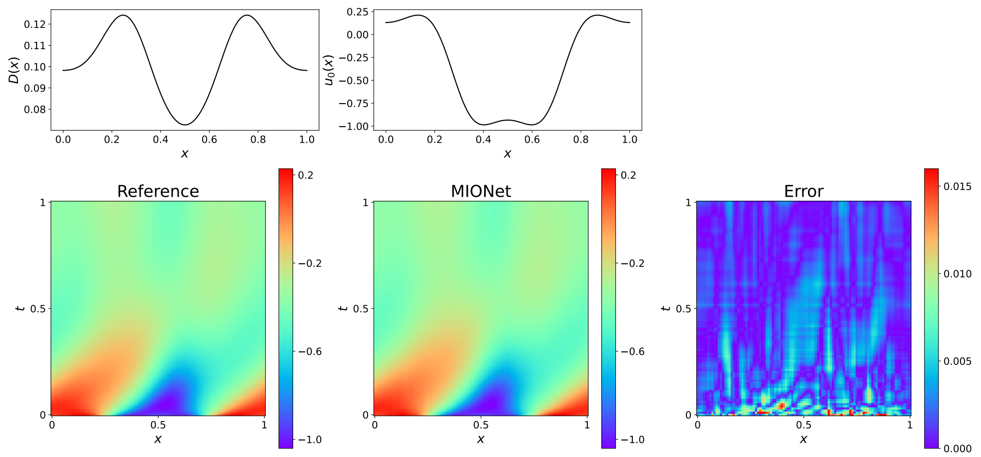

FIG. 4. *Prediction of MIONet (periodic) for the advection-diffusion system.*

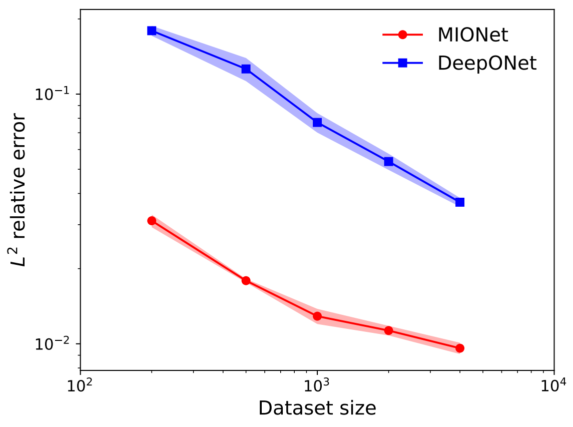

FIG. 5. *L^2 relative errors of MIONet and DeepONet for different dataset sizes. MIONet (periodic) and DeepONet (best) architectures in Table 3 are applied to the advection-diffusion system.*

then

$$\begin{aligned}\|x - P_n(x)\| &= \|(I - P_n)(x - x_j) + x_j - P_n(x_j)\|\\ &\leq \|I - P_n\| \cdot \|x - x_j\| + \|x_j - P_n(x_j)\|\\ &< (1 + C) \cdot \frac{\epsilon}{2(1 + C)} + \frac{\epsilon}{2}\\ &= \epsilon. \end{aligned}$$

□

Appendix B. Proof of Lemma 2.9. As $\mathcal{G}$ is uniformly continuous on $K_1 \times \cdots \times K_n$, there exists a $\delta > 0$ such that $\|\mathcal{G}(v_1, \ldots, v_n) - \mathcal{G}(v_1', \ldots, v_n')\| < \epsilon$ holds for all $v_i, v_i' \in K_i$, $\|v_i - v_i'\| < \delta$, $1 \leq i \leq n$. Due to the compactness of K_i, we can choose

$\{\nu_j^i\}_{j=1}^{p_i} \subset K_i$ such that

$$\bigcup_{j=1}^{p_i} B(\nu_j^i, \delta) \supset K_i, \tag{B.1}$$

where $B(\nu_j^i, \delta)$ denotes the open ball centered at ν_j^i with radius δ, $1 \leq i \leq n$. Now define $\tilde{\mathbf{g}}_i : X_i \to \mathbb{R}^{p_i}$ as

$$\tilde{\mathbf{g}}_i(x) = (\mathrm{ReLU}(\delta - \|x - \nu_1^i\|), \mathrm{ReLU}(\delta - \|x - \nu_2^i\|), \dots, \mathrm{ReLU}(\delta - \|x - \nu_{p_i}^i\|))^T,$$

and $\hat{\mathbf{g}}_i(x) : X_i \to \mathbb{R}^{p_i}$ as

$$\hat{\mathbf{g}}_i(x) = \frac{\tilde{\mathbf{g}}_i(x)}{\|\tilde{\mathbf{g}}_i(x)\|_1 + d(x, K_i)},$$

where $\mathrm{ReLU}(x) := \max(x, 0)$, and $d(x, K_i) := \inf_{x' \in K_i} \|x - x'\|$ represents the distance between x and K_i. $\hat{\mathbf{g}}_i$ is, in fact, the normalization of $\tilde{\mathbf{g}}_i$ on K_i, and the condition (B.1) guarantees that $\hat{\mathbf{g}}_i(x)$ is well defined everywhere, i.e., $\|\tilde{\mathbf{g}}_i(x)\|_1$ is nonzero on K_i while $d(x, K_i)$ is nonzero outside K_i. Moreover, define $\mathbf{u} \in Y^{p_1 \times p_2 \times \cdots \times p_n}$ as

$$\mathbf{u} = (\mathcal{G}(\nu_{j_1}^1, \nu_{j_2}^2, \dots, \nu_{j_n}^n))_{p_1 \times p_2 \times \cdots \times p_n}.$$

We will show that the constructed $\hat{\mathbf{g}}_i$ and $\mathbf{u}$ are what we need.

Denote $\hat{\mathbf{g}}_i = (\hat{\mathbf{g}}_i^1, \dots, \hat{\mathbf{g}}_i^{p_i})^T$, and define $A_i[v] = \{j \in \{1, 2, \dots, p_i\} | \|\nu_j^i - v\| < \delta\}$ for $v \in K_i$. Given any $v_i \in K_i$, we have

$$\begin{aligned}
&\|\mathcal{G}(v_1, \dots, v_n) - \mathbf{u}\langle \hat{\mathbf{g}}_1(v_1), \dots, \hat{\mathbf{g}}_n(v_n)\rangle\| \\
&= \left\| \mathcal{G}(v_1, \dots, v_n) - \sum_{j_1, \dots, j_n} \mathcal{G}(\nu_{j_1}^1, \dots, \nu_{j_n}^n) \cdot \hat{\mathbf{g}}_1^{j_1}(v_1) \cdots \hat{\mathbf{g}}_n^{j_n}(v_n) \right\| \\
&= \left\| \sum_{j_1, \dots, j_n} (\mathcal{G}(v_1, \dots, v_n) \cdot \hat{\mathbf{g}}_1^{j_1}(v_1) \cdots \hat{\mathbf{g}}_n^{j_n}(v_n) - \mathcal{G}(\nu_{j_1}^1, \dots, \nu_{j_n}^n) \cdot \hat{\mathbf{g}}_1^{j_1}(v_1) \cdots \hat{\mathbf{g}}_n^{j_n}(v_n)) \right\| \\
&= \left\| \sum_{j_1, \dots, j_n} (\mathcal{G}(v_1, \dots, v_n) - \mathcal{G}(\nu_{j_1}^1, \dots, \nu_{j_n}^n)) \cdot \hat{\mathbf{g}}_1^{j_1}(v_1) \cdots \hat{\mathbf{g}}_n^{j_n}(v_n) \right\| \\
&= \left\| \sum_{\substack{j_i \in A_i[v_i] \\ 1 \leq i \leq n}} (\mathcal{G}(v_1, \dots, v_n) - \mathcal{G}(\nu_{j_1}^1, \dots, \nu_{j_n}^n)) \cdot \hat{\mathbf{g}}_1^{j_1}(v_1) \cdots \hat{\mathbf{g}}_n^{j_n}(v_n) \right\| \\
&\leq \sum_{\substack{j_i \in A_i[v_i] \\ 1 \leq i \leq n}} \left\| \mathcal{G}(v_1, \dots, v_n) - \mathcal{G}(\nu_{j_1}^1, \dots, \nu_{j_n}^n) \right\| \cdot \hat{\mathbf{g}}_1^{j_1}(v_1) \cdots \hat{\mathbf{g}}_n^{j_n}(v_n) \\
&< \sum_{\substack{j_i \in A_i[v_i] \\ 1 \leq i \leq n}} \epsilon \cdot \hat{\mathbf{g}}_1^{j_1}(v_1) \cdots \hat{\mathbf{g}}_n^{j_n}(v_n) \\
&= \epsilon.
\end{aligned}$$

∎

Appendix C. Proof of Theorem 2.10. For any $\epsilon > 0$, there exist $p_i, \hat{\mathbf{g}}_i, \mathbf{u}$ as defined in the proof of Lemma 2.9, such that

$$\sup_{v_i \in K_i} \|\mathcal{G}(v_1, \dots, v_n) - \mathbf{u}\langle \hat{\mathbf{g}}_1(v_1), \dots, \hat{\mathbf{g}}_n(v_n)\rangle\| < \epsilon.$$

Denote $M = \max_{v_i \in K_i} \|\mathcal{G}(v_1, \dots, v_n)\|$, and then for positive integers q_i,

$$\begin{aligned}
&\left\|\mathbf{u}\langle \hat{\mathbf{g}}_1 \circ P^1_{q_1}(v_1), \dots, \hat{\mathbf{g}}_n \circ P^n_{q_n}(v_n)\rangle - \mathbf{u}\langle \hat{\mathbf{g}}_1(v_1), \dots, \hat{\mathbf{g}}_n(v_n)\rangle\right\| \\
&= \left\|\sum_{i_1,\dots,i_n} \mathcal{G}(\nu^1_{i_1}, \nu^2_{i_2}, \dots, \nu^n_{i_n}) \cdot \left(\hat{\mathbf{g}}^{i_1}_1 \circ P^1_{q_1}(v_1) \cdots \hat{\mathbf{g}}^{i_n}_n \circ P^n_{q_n}(v_n) - \hat{\mathbf{g}}^{i_1}_1(v_1) \cdots \hat{\mathbf{g}}^{i_n}_n(v_n)\right)\right\| \\
&\le M \sum_{i_1,\dots,i_n} \left|\hat{\mathbf{g}}^{i_1}_1 \circ P^1_{q_1}(v_1) \cdots \hat{\mathbf{g}}^{i_n}_n \circ P^n_{q_n}(v_n) - \hat{\mathbf{g}}^{i_1}_1(v_1) \cdots \hat{\mathbf{g}}^{i_n}_n(v_n)\right| \\
&= M \sum_{i_1,\dots,i_n} \left|\sum_{k=1}^n \left(\prod_{j=1}^{k-1} \hat{\mathbf{g}}^{i_j}_j \circ P^j_{q_j}(v_j) \cdot \prod_{j=k+1}^n \hat{\mathbf{g}}^{i_j}_j(v_j) \cdot \left(\hat{\mathbf{g}}^{i_k}_k \circ P^k_{q_k}(v_k) - \hat{\mathbf{g}}^{i_k}_k(v_k)\right)\right)\right| \\
&\le M \sum_{i_1,\dots,i_n} \sum_{k=1}^n \left(\prod_{j=1}^{k-1} \hat{\mathbf{g}}^{i_j}_j \circ P^j_{q_j}(v_j) \cdot \prod_{j=k+1}^n \hat{\mathbf{g}}^{i_j}_j(v_j) \cdot \left|\hat{\mathbf{g}}^{i_k}_k \circ P^k_{q_k}(v_k) - \hat{\mathbf{g}}^{i_k}_k(v_k)\right|\right) \\
&= M \sum_{k=1}^n \sum_{i_1,\dots,i_n} \left(\prod_{j=1}^{k-1} \hat{\mathbf{g}}^{i_j}_j \circ P^j_{q_j}(v_j) \cdot \prod_{j=k+1}^n \hat{\mathbf{g}}^{i_j}_j(v_j) \cdot \left|\hat{\mathbf{g}}^{i_k}_k \circ P^k_{q_k}(v_k) - \hat{\mathbf{g}}^{i_k}_k(v_k)\right|\right) \\
&\le M \sum_{k=1}^n \sum_{i_k} \left|\hat{\mathbf{g}}^{i_k}_k \circ P^k_{q_k}(v_k) - \hat{\mathbf{g}}^{i_k}_k(v_k)\right| \\
&= M \sum_{k=1}^n \left\|\hat{\mathbf{g}}_k \circ P^k_{q_k}(v_k) - \hat{\mathbf{g}}_k(v_k)\right\|_1.
\end{aligned}$$

Note that $\sum_i \hat{\mathbf{g}}^i_j(x) \in [0,1]$ for all $x \in X_j$. Therefore,

$$\begin{aligned}
&\left\|\mathcal{G}(v_1, \dots, v_n) - \mathbf{u}\langle \hat{\mathbf{g}}_1(P^1_{q_1}(v_1)), \dots, \hat{\mathbf{g}}_n(P^n_{q_n}(v_n))\rangle\right\| \\
&\le \|\mathcal{G}(v_1, \dots, v_n) - \mathbf{u}\langle \hat{\mathbf{g}}_1(v_1), \dots, \hat{\mathbf{g}}_n(v_n)\rangle\| \\
&\quad + \left\|\mathbf{u}\langle \hat{\mathbf{g}}_1(P^1_{q_1}(v_1)), \dots, \hat{\mathbf{g}}_n(P^n_{q_n}(v_n))\rangle - \mathbf{u}\langle \hat{\mathbf{g}}_1(v_1), \dots, \hat{\mathbf{g}}_n(v_n)\rangle\right\| \\
&< \epsilon + M \sum_{k=1}^n \left\|\hat{\mathbf{g}}_k \circ P^k_{q_k}(v_k) - \hat{\mathbf{g}}_k(v_k)\right\|_1 \\
&\le \epsilon + M \sum_{k=1}^n L^\epsilon_k(q_k).
\end{aligned}$$

∎

Appendix D. Proof of Theorem 2.5. By Theorem 2.10, for any $\epsilon > 0$, there exist positive integers p_i, q_i and continuous vector functionals $\hat{\mathbf{g}}_i \in C(X_i, \mathbb{R}^{p_i})$ and $\mathbf{u} \in Y^{p_1 \times p_2 \times \cdots \times p_n}$, such that

$$\sup_{v_i \in K_i} \left\|\mathcal{G}(v_1, \dots, v_n) - \mathbf{u}\langle \hat{\mathbf{g}}_1(P^1_{q_1}(v_1)), \dots, \hat{\mathbf{g}}_n(P^n_{q_n}(v_n))\rangle\right\| < \epsilon.$$

Now we define

$$\mathbf{g}_i = \hat{\mathbf{g}}_i \circ \psi^i_{q_i}$$

and obtain this theorem. ∎

Appendix E. Proof of Corollary 2.6.

(2.2)⇒(2.4): Denoting $\mathbf{g}_i = (g^i_j)$, we have

$$\mathbf{u}\langle \mathbf{g}_1, \dots, \mathbf{g}_n\rangle = \sum_{j_1,\cdots,j_n} g^1_{j_1}\cdots g^n_{j_n} u_{j_1\cdots j_n},$$

which is indeed the form (2.4) by rearrangement and relabeling of the summation.

(2.4)⇒(2.3): Denoting $\mathbf{g}_i = (g^i_j)$, $\mathbf{u} = (u_j)$, we then have

$$\sum_{j=1}^{p} g^1_j \cdots g^n_j u_j = \sum_{j_1,\dots,j_{n+1}} \delta_{j_1\cdots j_{n+1}} g^1_{j_1}\cdots g^n_{j_n} u_{j_{n+1}} = (\delta_{j_1\cdots j_{n+1}})\langle \mathbf{g}_1, \dots, \mathbf{g}_n, \mathbf{u}\rangle,$$

where $\delta_{j_1\cdots j_{n+1}}$ is equal to 1 if $j_1 = \cdots = j_{n+1}$; otherwise, it is 0. Moreover, if u_j is approximated by $\tilde{u}_j = \sum_{k=1}^r \alpha^j_k e_k$, denote $\tilde{\mathbf{u}} = (\tilde{u}_j)$, $\mathbf{e} = (e_j)$, and we then have

$$(\delta_{j_1\cdots j_{n+1}})\langle \mathbf{g}_1, \dots, \mathbf{g}_n, \tilde{\mathbf{u}}\rangle = W\langle \mathbf{g}_1, \dots, \mathbf{g}_n, \mathbf{e}\rangle,$$

where $W = (\sum_{j_{n+1}} \delta_{j_1\cdots j_{n+1}} \alpha^{j_{n+1}}_k)$.

(2.3)⇒(2.2): Denote $W = (w_{j_1\cdots j_{n+1}})$, $\mathbf{u} = (u_j)$, so

$$W\langle \mathbf{g}_1, \dots, \mathbf{g}_n, \mathbf{u}\rangle = \tilde{\mathbf{u}}\langle \mathbf{g}_1, \dots, \mathbf{g}_n\rangle,$$

where $\tilde{\mathbf{u}} = (\sum_{j_{n+1}} w_{j_1\cdots j_{n+1}} u_{j_{n+1}})$. ∎

Appendix F. Proof of Corollary 2.13. Without loss of generality, assume that $\mathcal{G}$ is linear with respect to v_1; that is, there is a continuous operator defined on $X_1 \times K_2 \times \cdots \times K_n$ which is linear with respect to v_1 and equal to $\mathcal{G}$ limited on $K_1 \times \cdots \times K_n$, and for convenience we still denote it as $\mathcal{G}$. Suppose that $\{e_i\}, \{e^*_i\}$ are the Schauder basis and coordinate functionals of X_1. For $\epsilon > 0$, according to the continuity of $\mathcal{G}$ and Property 2.4, there exists a positive integer q_1 such that

$$\sup_{v_i \in K_i} \left\| \mathcal{G}(v_1, v_2, \dots, v_n) - \mathcal{G}(P^1_{q_1}(v_1), v_2, \dots, v_n)\right\| < \frac{\epsilon}{2}.$$

Denote $M = \max_{v_1 \in K_1, 1\le j \le q_1} |e^*_j(v_1)|$. Now define continuous operators $\mathcal{G}_j : K_2 \times \cdots \times K_n \to Y$ as

$$\mathcal{G}_j(v_2, \dots, v_n) = \mathcal{G}(e_j, v_2, \dots, v_n), \quad 1 \le j \le q_1.$$

Then by Corollary 2.12, there exist positive integers p_i, q_i and continuous vector functions $\mathbf{g}_i \in C(\mathbb{R}^{q_i}, \mathbb{R}^{p_i})$, $\mathbf{u}_j = (u^j_{k_2\cdots k_n}) \in Y^{p_2\times\cdots\times p_n}$, $2 \le i \le n$, $1 \le j \le q_1$, such that

$$\sup_{v_i \in K_i} \left\| \mathcal{G}_j(v_2, \dots, v_n) - \mathbf{u}_j\langle \mathbf{g}_2(\varphi^2_{q_2}(v_2)), \dots, \mathbf{g}_n(\varphi^n_{q_n}(v_n))\rangle\right\| < \frac{\epsilon}{2q_1 M}, \quad j = 1, \dots, q_1.$$

Let $p_1 = q_1$, $\mathbf{u} = (u^{k_1}_{k_2\cdots k_n}) \in Y^{p_1\times\cdots\times p_n}$, $\mathbf{g}_1 : \mathbb{R}^{q_1} \to \mathbb{R}^{q_1}$ be the identity map; then

$$\begin{aligned}
&\left\|\mathcal{G}(P^1_{q_1}(v_1), v_2, \ldots, v_n) - \mathbf{u}\langle \mathbf{g}_1(\varphi^1_{q_1}(v_1)), \ldots, \mathbf{g}_n(\varphi^n_{q_n}(v_n))\rangle\right\|\\
&= \left\|\sum_{j=1}^{q_1} e^*_j(v_1)\mathcal{G}(e_j, v_2, \ldots, v_n) - \sum_{j=1}^{q_1} e^*_j(v_1)\mathbf{u}_j\langle \mathbf{g}_2(\varphi^2_{q_2}(v_2)), \ldots, \mathbf{g}_n(\varphi^n_{q_n}(v_n))\rangle\right\|\\
&\le \sum_{j=1}^{q_1} |e^*_j(v_1)| \cdot \left\|\mathcal{G}(e_j, v_2, \ldots, v_n) - \mathbf{u}_j\langle \mathbf{g}_2(\varphi^2_{q_2}(v_2)), \ldots, \mathbf{g}_n(\varphi^n_{q_n}(v_n))\rangle\right\|\\
&< q_1 \cdot M \cdot \frac{\epsilon}{2q_1 M} = \frac{\epsilon}{2}.
\end{aligned}$$

Therefore,

$$\sup_{v_i\in K_i} \left\|\mathcal{G}(v_1, \ldots, v_n) - \mathbf{u}\langle \mathbf{g}_1(\varphi^1_{q_1}(v_1)), \ldots, \mathbf{g}_n(\varphi^n_{q_n}(v_n))\rangle\right\| < \frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon,$$

where $\mathbf{g}_1$ is linear. The proofs for the other two cases are similar. ∎

Appendix G. Proof of Property 2.14. Assume that $W_i = (w^i_j)_{p_i} = (w^i_{jk})_{p_i\times h_i}$, $\mathbf{h}_i = (H^i_j)_{h_i}$; then for $\mathbf{u} = (u_{j_1\cdots j_n}) \in Y^{p_1\times p_2\times\cdots\times p_n}$,

$$\begin{aligned}
\mathbf{u}\langle \mathbf{g}_1, \ldots, \mathbf{g}_n\rangle &= \sum_{j_1,\ldots,j_n} u_{j_1\cdots j_n}(w^1_{j_1}\mathbf{h}_1)\cdots(w^n_{j_n}\mathbf{h}_n)\\
&= \sum_{j_1,\ldots,j_n} u_{j_1\cdots j_n}\left(\sum_k w^1_{j_1k}H^1_k\right)\cdots\left(\sum_k w^n_{j_nk}H^n_k\right)\\
&= \sum_{k_1,\ldots,k_n}\left(\sum_{j_1,\ldots,j_n} u_{j_1\cdots j_n}w^1_{j_1k_1}\cdots w^n_{j_nk_n}\right)H^1_{k_1}\cdots H^n_{k_n}\\
&= \tilde{\mathbf{u}}\langle \mathbf{h}_1, \ldots, \mathbf{h}_n\rangle,
\end{aligned}$$

where $\tilde{\mathbf{u}} = (\sum_{j_1,\ldots,j_n} u_{j_1\cdots j_n}w^1_{j_1k_1}\cdots w^n_{j_nk_n})_{h_1\times\cdots\times h_n}$. Briefly speaking, the linear output layers of $\mathbf{g}_i$ can be merged into $\mathbf{u}$. The proof for the other case is similar. ∎

Appendix H. MIONet for finite-dimensional image space. Remark 2.16 for tensor neural networks also gives the approximation theorem for the operators projecting onto both finite-dimensional domain and image space. Given data points $(v_1, \ldots, v_n, \mathcal{G}(v_1, \ldots, v_n))$, we first transform them into a training set

$$\{\varphi^1_{q_1}(v^k_1), \ldots, \varphi^n_{q_n}(v^k_n), \varphi^Y_m(\mathcal{G}(v^k_1, \ldots, v^k_n))\}_{k=1}^N$$

by determining basis elements $\{e_i\}_{i=1}^m$ for Y with $\varphi^Y_m(x) = (e^*_1(x), \ldots, e^*_m(x))^T$. Then the loss function can be written as

$$\mathrm{MSE} = \frac{1}{mN}\sum_{k=1}^N \left\|\varphi^Y_m(\mathcal{G}(v^k_1, \ldots, v^k_n)) - W\left(\tilde{\mathbf{g}}_1(\varphi^1_{q_1}(v^k_1)) \odot \cdots \odot \tilde{\mathbf{g}}_n(\varphi^n_{q_n}(v^k_n))\right) - b\right\|_2^2,$$

where $\tilde{\mathbf{g}}_i : \mathbb{R}^{q_i} \to \mathbb{R}^p$ are neural networks to be trained. $W \in \mathbb{R}^{m\times p}$ and $b \in \mathbb{R}^m$ are trainable weights and bias, respectively. After training, we make prediction by

$$\tilde{\mathcal{G}}(v_1, \ldots, v_n) = (e_1, \ldots, e_m) \cdot \left(W\left(\tilde{\mathbf{g}}_1(\varphi^1_{q_1}(v_1)) \odot \cdots \odot \tilde{\mathbf{g}}_n(\varphi^n_{q_n}(v_n))\right) + b\right).$$

Appendix I. Loss function via numerical integration.

Suppose that $Y = C[0,1]$. For $\mathcal{T} = \{v_1^k, \ldots, v_n^k, \mathcal{G}(v_1^k, \ldots, v_n^k)\}_{k=1}^N$, the general loss function can be computed as

$$\mathcal{L}(\mathcal{T}) = \frac{1}{N} \sum_{k=1}^{N} \mathbf{I}(\mathcal{G}(v_1^k, \ldots, v_n^k) - \tilde{\mathcal{G}}(v_1^k, \ldots, v_n^k)),$$

where $\mathbf{I}(\cdot)$ is a numerical integration. For example, for $\mathbf{x}_k$ uniformly sampled on $[0,1]$ $(0 = x_0 < \cdots < x_m = 1)$, we have the following choices of $\mathbf{I}(\cdot)$:

- rectangle rule:

$$\mathbf{I}_{rec}(f) = \frac{1}{m} \left(\sum_{k=1}^{m} f\left(\frac{x_{k-1} + x_k}{2} \right) \right),$$

- trapezoidal rule:

$$\mathbf{I}_{tra}(f) = \frac{1}{m} \left(\sum_{k=1}^{m} \frac{f(x_{k-1}) + f(x_k)}{2} \right),$$

- Monte Carlo integration:

$$\mathbf{I}_{mon}(f) = \frac{1}{m} \left(\sum_{k=1}^{m} f(\mathbf{x}_k) \right).$$

For a high-dimensional integration, Monte Carlo integration usually performs better.

REFERENCES

[1] K. Bhattacharya, B. Hosseini, N. B. Kovachki, and A. M. Stuart, *Model Reduction and Neural Networks for Parametric PDEs*, preprint, https://arxiv.org/abs/2005.03180, 2020.

[2] S. Cai, Z. Wang, L. Lu, T. A. Zaki, and G. E. Karniadakis, *DeepM&Mnet: Inferring the electroconvection multiphysics fields based on operator approximation by neural networks*, J. Comput. Phys., 436 (2021), 110296.

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

[4] Y. Chen, L. Lu, G. E. Karniadakis, and L. Dal Negro, *Physics-informed neural networks for inverse problems in nano-optics and metamaterials*, Optics Express, 28 (2020), pp. 11618–11633.

[5] M. Daneker, Z. Zhang, G. E. Karniadakis, and L. Lu, *Systems Biology: Identifiability Analysis and Parameter Identification via Systems-Biology Informed Neural Networks*, preprint, https://arxiv.org/abs/2202.01723, 2022.

[6] B. Deng, Y. Shin, L. Lu, Z. Zhang, and G. E. Karniadakis, *Convergence Rate of DeepONets for Learning Operators Arising from Advection-Diffusion Equations*, preprint, https://arxiv.org/abs/2102.10621, 2021.

[7] P. C. Di Leoni, L. Lu, C. Meneveau, G. Karniadakis, and T. A. Zaki, *DeepONet Prediction of Linear Instability Waves in High-Speed Boundary Layers*, preprint, https://arxiv.org/abs/2105.08697, 2021.

[8] W. E and B. Yu, *The deep Ritz method: A deep learning-based numerical algorithm for solving variational problems*, Commun. Math. Stat., 6 (2018), pp. 1–12.

[9] M. Fabian, P. Habala, P. Hájek, V. Montesinos, and V. Zizler, *Banach Space Theory: The Basis for Linear and Nonlinear Analysis*, Springer, New York, 2011.

[10] S. Goswami, M. Yin, Y. Yu, and G. E. Karniadakis, *A physics-informed variational DeepONet for predicting crack path in quasi-brittle materials*, Comput. Methods Appl. Mech. Engrg., 391 (2022), 114587.

[11] H. Johan, *Tensor rank is NP-complete*, J. Algorithms, 4 (1990), pp. 644–654.

[12] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, *Physics-informed machine learning*, Nature Rev. Phys., 3 (2021), pp. 422–440.

[13] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, in Proceedings of the 3rd International Conference on Learning Representations (ICLR), San Diego, CA, Conference Track Proceedings, 2015.

[14] G. Kissas, J. Seidman, L. F. Guilhoto, V. M. Preciado, G. J. Pappas, and P. Perdikaris, *Learning Operators with Coupled Attention*, preprint, https://arxiv.org/abs/2201.01032, 2022.

[15] G. Kissas, Y. Yang, E. Hwuang, W. R. Witschey, J. A. Detre, and P. Perdikaris, *Machine learning in cardiovascular flows modeling: Predicting arterial blood pressure from non-invasive 4D flow MRI data using physics-informed neural networks*, Comput. Methods Appl. Mech. Engrg., 358 (2020), 112623.

[16] T. G. Kolda and B. W. Bader, *Tensor decompositions and applications*, SIAM Rev., 51 (2009), pp. 455–500, https://doi.org/10.1137/07070111X.

[17] N. Kovachki, S. Lanthaler, and S. Mishra, *On universal approximation and error bounds for Fourier neural operators*, J. Mach. Learn. Res., 22 (2021), 290.

[18] N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, and A. Anandkumar, *Neural Operator: Learning Maps between Function Spaces*, preprint, https://arXiv.org/abs/2108.08481, 2021.

[19] J. B. Kruskal, *Rank, decomposition, and uniqueness for 3-way and n-way arrays*, in Multiway Data Analysis, Papers from the International Meeting on the Analysis of Multiway Data Matrices held in Rome, 1988, R. Coppi and S. Bolasco, eds., North-Holland, Amsterdam, 1989, pp. 7–18.

[20] S. Lanthaler, S. Mishra, and G. E. Karniadakis, *Error Estimates for DeepONets: A Deep Learning Framework in Infinite Dimensions*, preprint, https://arxiv.org/abs/2102.09618, 2021.

[21] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, *Fourier Neural Operator for Parametric Partial Differential Equations*, preprint, https://arxiv.org/abs/2010.08895, 2020.

[22] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, *Neural Operator: Graph Kernel Network for Partial Differential Equations*, preprint, https://arxiv.org/abs/2003.03485, 2020.

[23] C. Lin, Z. Li, L. Lu, S. Cai, M. Maxey, and G. E. Karniadakis, *Operator learning for predicting multiscale bubble growth dynamics*, J. Chem. Phys., 154 (2021), 104118.

[24] C. Lin, M. Maxey, Z. Li, and G. E. Karniadakis, *A seamless multiscale operator neural network for inferring bubble dynamics*, J. Fluid Mech., 929 (2021), A18.

[25] G. Lin, C. Moya, and Z. Zhang, *Accelerated Replica Exchange Stochastic Gradient Langevin Diffusion Enhanced Bayesian DeepONet for Solving Noisy Parametric PDEs*, preprint, https://arxiv.org/abs/2111.02484, 2021.

[26] L. Liu and W. Cai, *Multiscale DeepONet for Nonlinear Operators in Oscillatory Function Spaces for Building Seismic Wave Responses*, preprint, https://arxiv.org/abs/2111.04860, 2021.

[27] L. Lu, P. Jin, and G. E. Karniadakis, *DeepONet: Learning Nonlinear Operators for Identifying Differential Equations Based on the Universal Approximation Theorem of Operators*, preprint, https://arxiv.org/abs/1910.03193, 2019.

[28] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis, *Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators*, Nature Mach. Intell., 3 (2021), pp. 218–229.

[29] L. Lu, X. Meng, S. Cai, Z. Mao, S. Goswami, Z. Zhang, and G. E. Karniadakis, *A Comprehensive and Fair Comparison of Two Neural Operators (with Practical Extensions) Based on FAIR Data*, preprint, https://arxiv.org/abs/2111.05512, 2021.

[30] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, *DeepXDE: A deep learning library for solving differential equations*, SIAM Rev., 63 (2021), pp. 208–228, https://doi.org/10.1137/19M1274067.

[31] L. Lu, R. Pestourie, W. Yao, Z. Wang, F. Verdugo, and S. G. Johnson, *Physics-informed neural networks with hard constraints for inverse design*, SIAM J. Sci. Comput., 43 (2021), pp. B1105–B1132, https://doi.org/10.1137/21M1397908.

[32] Z. Mao, L. Lu, O. Marxen, T. A. Zaki, and G. E. Karniadakis, *DeepM&Mnet for hypersonics: Predicting the coupled flow and finite-rate chemistry behind a normal shock using neural-network approximation of operators*, J. Comput. Phys., 447 (2021), 110698.

[33] C. Marcati and C. Schwab, *Exponential Convergence of Deep Operator Networks for Elliptic*

Partial Differential Equations, preprint, https://arxiv.org/abs/2112.08125, 2021.

[34] N. H. Nelsen and A. M. Stuart, *The random feature model for input-output maps between Banach spaces*, SIAM J. Sci. Comput., 43 (2021), pp. A3212–A3243, https://doi.org/10.1137/20M133957X.

[35] J. D. Osorio, Z. Wang, G. Karniadakis, S. Cai, C. Chryssostomidis, M. Panwar, and R. Hovsapian, *Forecasting solar-thermal systems performance under transient operation using a data-driven machine learning approach based on the deep operator network architecture*, Energy Conversion and Management, 252 (2022), 115063.

[36] G. Pang, L. Lu, and G. E. Karniadakis, *fPINNs: Fractional physics-informed neural networks*, SIAM J. Sci. Comput., 41 (2019), pp. A2603–A2626, https://doi.org/10.1137/18M1229845.

[37] R. G. Patel, N. A. Trask, M. A. Wood, and E. C. Cyr, *A physics-informed operator regression framework for extracting data-driven continuum models*, Comput. Methods Appl. Mech. Engrg., 373 (2021), 113500.

[38] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*, J. Comput. Phys., 378 (2019), pp. 686–707.

[39] M. Raissi, A. Yazdani, and G. E. Karniadakis, *Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations*, Science, 367 (2020), pp. 1026–1030.

[40] R. A. Ryan, *Introduction to Tensor Products of Banach Spaces*, Springer Monogr. Math., Springer-Verlag London, Ltd., London, 2002.

[41] J. Sirignano and K. Spiliopoulos, *DGM: A deep learning algorithm for solving partial differential equations*, J. Comput. Phys., 375 (2018), pp. 1339–1364.

[42] N. Trask, R. G. Patel, B. J. Gross, and P. J. Atzberger, *GMLS-Nets: A Framework for Learning from Unstructured Data*, preprint, https://arxiv.org/abs/1909.05371, 2019.

[43] S. Wang, H. Wang, and P. Perdikaris, *Learning the solution operator of parametric partial differential equations with physics-informed DeepONets*, Sci. Adv., 7 (2021), eabi8605.

[44] A. Yazdani, L. Lu, M. Raissi, and G. E. Karniadakis, *Systems biology informed deep learning for inferring parameters and hidden dynamics*, PLoS Comput. Biol., 16 (2020), e1007575.

[45] M. Yin, E. Ban, B. V. Rego, E. Zhang, C. Cavinato, J. D. Humphrey, and G. Em Karniadakis, *Simulating progressive intramural damage leading to aortic dissection using DeepONet: An operator–regression neural network*, J. R. Soc. Interface, 19 (2022), 20210670.

[46] H. You, Y. Yu, M. D'Elia, T. Gao, and S. Silling, *Nonlocal Kernel Network (NKN): A Stable and Resolution-Independent Deep Neural Network*, preprint, https://arxiv.org/abs/2201.02217, 2022.

[47] J. Yu, L. Lu, X. Meng, and G. E. Karniadakis, *Gradient-Enhanced Physics-Informed Neural Networks for Forward and Inverse PDE Problems*, preprint, https://arxiv.org/abs/2111.02801, 2021.

[48] D. Zhang, L. Lu, L. Guo, and G. E. Karniadakis, *Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems*, J. Comput. Phys., 397 (2019), 108850.