HYPERDEEPONET: LEARNING OPERATOR WITH COMPLEX TARGET FUNCTION SPACE USING THE LIMITED RESOURCES VIA HYPERNETWORK

Anonymous authors
Paper under double-blind review

ABSTRACT

Fast and accurate predictions for complex physical dynamics are a big challenge across various applications. Real-time prediction on resource-constrained hardware is even more crucial in the real-world problems. The deep operator network (DeepONet) has recently been proposed as a framework for learning nonlinear mappings between function spaces. However, the DeepONet requires many parameters and has a high computational cost when learning operators, particularly those with complex (discontinuous or non-smooth) target functions. In this study, we propose HyperDeepONet, which uses the expressive power of the hypernetwork to enable learning of a complex operator with smaller set of parameters. The DeepONet and its variant models can be thought of as a method of injecting the input function information into the target function. From this perspective, these models can be viewed as a special case of HyperDeepONet. We analyze the complexity of DeepONet and conclude that HyperDeepONet needs relatively lower complexity to obtain the desired accuracy for operator learning. HyperDeepONet was successfully applied to various operator learning problems using low computational resources compared to other benchmarks.

1 INTRODUCTION

Learning operators that map between infinite-dimensional function spaces, called operator learning, is a challenging problem and has been used in many applications, such as climate prediction (Kurth et al., 2022) and fluid dynamics (Guo et al., 2016). The computational efficiency to learn the mapping is still an important factor in real-world problems. The target function of the operator can be discontinuous or sharp for complicated dynamic systems. In this case, balancing model complexity and cost for computational time is a core problem for the real-time prediction on resource-constrained hardware (Choudhary et al., 2020; Murshed et al., 2021).

Many machine learning methods and deep learning-based architectures have been successfully developed to learn a nonlinear mapping from an infinite-dimensional Banach space to another. They focus on learning the solution operator of some PDEs, e.g., the initial or boundary condition of PDE to the corresponding solution. Li et al. (2020c) proposed an iterative neural operator scheme to learn the operator mapping with the graph neural network (Li et al., 2020b) and with the Fourier transform (Li et al., 2020a).

At the same time, Lu et al. (2019; 2021) proposed a deep operator network (DeepONet) architecture based on the universal operator approximation theorem of Chen & Chen (1995). The DeepONet consists of two networks: branch net taking sensor values of the input function, and trunk net taking a querying location of the output function domain. Each output of two networks is regarded as the p-coefficients and p-basis of the target function. The two p-outputs are combined as a linear combination (inner-product) to approximate the underlying operator. The DeepONet has been greatly applied to various problems, such as bubble growth dynamics (Lin et al., 2021), hypersonic flows (Mao et al., 2021), and fluid flow (Cai et al., 2021) with analysis of error estimates (Lanthaler et al., 2022).

While variant models of DeepONet are developed to improve the vanilla DeepONet, e.g. Physics-Informed DeepONet (Wang et al., 2021) and Variable-input DeepONet (Prasthofer et al., 2022), it

still has difficulty approximating the operator for complex target function with the limited computational resources. Lanthaler et al. (2022) and Kovachki et al. (2021b) pointed out the limitation of the linear approximation using the DeepONet. Some operators are known to have a slow spectral decay rate of the Kolmogorov n-width, which defines the error of the best possible linear approximation using n-dimensional space. This means that a very large n is needed to learn complex operators accurately, which implies that the DeepONet requires a large number of basis p and a large number of network parameters for them.

Hadorn (2022) investigated the behavior of DeepONet, which makes it challenging to produce the sharp features in the target function when the number of basis p is small. They proposed a Shift-DeepONet by adding two neural networks to shift and scale the input function. Venturi & Casey (2022) also analyzed the limitation of DeepONet via singular value decomposition (SVD) and proposed a flexible DeepONet (flexDeepONet) adding a pre-net and one additional output in the branch net. Recently, to overcome the limitation of the linear approximation, Seidman et al. (2022) proposed a nonlinear manifold decoder (NOMAD) framework by using a neural network that takes the output of the branch net as input with the querying location. The number of basis cannot directly correspond to the total number of parameters in the model, even though these methods reduce the number of basis functions. They still need many parameters for the trunk net to learn the complex operators, especially with complex (discontinuous or non-smooth) target functions.

In this study, we propose a new architecture, HyperDeepONet, to learn operators with a complex target function space using limited resources. The HyperDeepONet uses a hypernetwork, proposed by Ha et al. (2016), which produces parameters for the target network that is utilized for actual tasks. Wang et al. (2022) pointed out that the final inner product in DeepONet may be inefficient if the information of the input function fails to propagate through a branch net. The hypernetwork in HyperDeepONet facilitates transmission the information of the input function to all network parameters of the target network. Furthermore, the expressivity of the hypernetwork reduces the neural network complexity by sharing the parameters of the hypernetwork (Galanti & Wolf, 2020). Our main contributions are as follows.

- We propose a novel HyperDeepONet using hypernetwork to overcome the limitations of DeepONet and learn the operators with a complex target function space. The DeepONet and its variant models are analyzed with a focus on expressing the target function as a neural network (Figure 2). These models can be treated as simplified versions of our general HyperDeepONet model (Figure 3).
- We analyze the complexity of DeepONet (Theorem 2) and show that the complexity of the HyperDeepONet is smaller than that of the DeepONet. We have identified that the DeepONet should employ a large number of basis to obtain the desired accuracy, so it requires numerous parameters. We present a lower bound of the number of parameters in target network for variants of DeepONet, including nonlinear reconstructors.
- The experiments shows that the HyperDeepONet facilitates learning a operator with small number of parameters in target network when the target function space is complex with the discontinuity and sharpness, which the DeepONet and its variants suffer from. The HyperDeepONet learns the operator more accurately even when the total number of parameters in the overall model is same.

2 Related work

Many machine learning methods and deep learning-based architectures have been successfully developed to solve PDEs with several advantages. One research direction is to use the neural network directly to represent the solution of PDE (E & Yu, 2018; Sirignano & Spiliopoulos, 2018). The physics informed neural network (PINN), introduced by Raissi et al. (2019), minimized the residual of PDEs by using automatic differentiation instead of numerical approximations.

There is another approach to solving PDEs, called operator learning. Operator learning aims to learn nonlinear mapping from an infinite-dimensional Banach space to another. Many studies utilize the convolutional neural network to parameterize the solution operator of PDEs in various applications (Guo et al., 2016; Bhatnagar et al., 2019; Khoo et al., 2021; Zhu et al., 2019; Hwang et al., 2021). The neural operator (Kovachki et al., 2021b) is proposed to approximate the nonlinear operator inspired

by Green's function. Li et al. (2020a) extend the neural operator structure to the Fourier Neural Operator (FNO) to approximate the integral operator effectively using the fast Fourier transform (FFT). Kovachki et al. (2021a) proved the universality of FNO and identified the size of the network.

The DeepONet (Lu et al., 2019; 2021) has also been proposed as another framework for operator learning. Lanthaler et al. (2022) provided the universal approximation property of DeepONet. Wang et al. (2021) proposed physics-informed DeepONet by adding a residual of PDE as a loss function, and De Ryck & Mishra (2022) demonstrated the generic bounds on the approximation error for it. Prasthofer et al. (2022) considered the case where the discretization grid of the input function in DeepONet changes by employing the coordinate encoder. Lu et al. (2022) compared the FNO with DeepONet in different benchmarks to demonstrate the relative performance. FNO can only infer the output function of an operator in the same grid as the input function as it needs to discretize the output function to use FFT, whereas the DeepONet can predict at any location.

Ha et al. (2016) first proposed hypernetwork, a network that creates a weight of the primary network. Because the hypernetwork can achieve weight sharing and model compression, it requires a relatively small number of parameters even as the dataset grows. Galanti & Wolf (2020) proved that a hypernetwork provides higher expressivity with low complexity target networks. Sitzmann et al. (2020) and Klocek et al. (2019) employed this approach to restore images with insufficient pixel observations or resolutions. de Avila Belbute-Peres et al. (2021) investigated the relationship between the coefficients of PDEs and the corresponding solution. They combined the hypernetwork with the PINN's residual loss. For time-dependent PDE, Pan et al. (2022) designated the time t as the input of the hypernetwork so that the target network indicates the solution at t. Because the output dimension of the hypernetwork can be large, Von Oswald et al. (2019) devised a chunk embedding method that partitions the parameters of the target network.

3 Operator Learning

3.1 Problem Setting

The goal of operator learning is to learn a mapping from infinite-dimensional function space to the others using a finite pair of functions. Let $\mathcal{G} : \mathcal{U} \to \mathcal{S}$ be a non-linear operator, where $\mathcal{U}$ and $\mathcal{S}$ are compact subsets of infinite-dimensional function spaces $\mathcal{U} \subset C(\mathcal{X}; \mathbb{R}^{d_u})$ and $\mathcal{S} \subset C(\mathcal{Y}; \mathbb{R}^{d_s})$ with compact domains $\mathcal{X} \subset \mathbb{R}^{d_x}$ and $\mathcal{Y} \subset \mathbb{R}^{d_y}$. For simplicity, we focus on the case $d_u = d_s = 1$, and all the results could be extended to a more general case for arbitrary d_u and d_s. Suppose we have observations $\{u_i, \mathcal{G}(u_i)\}_{i=1}^N$ where $u_i \in \mathcal{U}$ and $\mathcal{G}(u_i) \in \mathcal{S}$. We aim to find an approximation $\mathcal{G}_\theta : \mathcal{U} \to \mathcal{S}$ with parameter θ using the N observations so that $\mathcal{G}_\theta \approx \mathcal{G}$. As explained in Lanthaler et al. (2022), the approximator $\mathcal{G}_\theta$ can be decomposed into the three components (Figure 1) as

$$\mathcal{G}_\theta := \mathcal{R} \circ \mathcal{A} \circ \mathcal{E}. \tag{1}$$

First, the encoder $\mathcal{E}$ takes an input function u from $\mathcal{U}$ to generate the finite-dimensional encoded data in $\mathbb{R}^m$. Then, the approximator $\mathcal{A}$ is an operator approximator from the encoded data in finite dimension space $\mathbb{R}^m$ to the other finite dimensional space $\mathbb{R}^p$. Finally, the reconstructor $\mathcal{R}$ reconstructs the output function $s(y) = \mathcal{G}(u)(y)$ using the approximated data in $\mathbb{R}^p$.

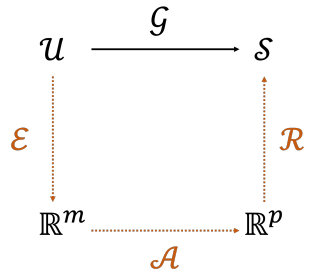

Figure 1: Diagram for the three components for operator learning.

3.2 DeepONet and its Limitation

DeepONet can be analyzed using the above three decompositions. Given a set of sensor points $x_j \in \mathcal{X}$, they use an encoder as the pointwise projection $\mathcal{E}(u) = (u(x_1), u(x_2), ..., u(x_m))$ of the continuous function, the so-called sensor values. For the approximator $\mathcal{A} : \mathbb{R}^m \to \mathbb{R}^p$, they use the fully connected neural network. They referred to the composition of these two maps as branch net

$$\beta : \mathcal{U} \to \mathbb{R}^p, \beta(u) := \mathcal{A} \circ \mathcal{E}(u) \tag{2}$$

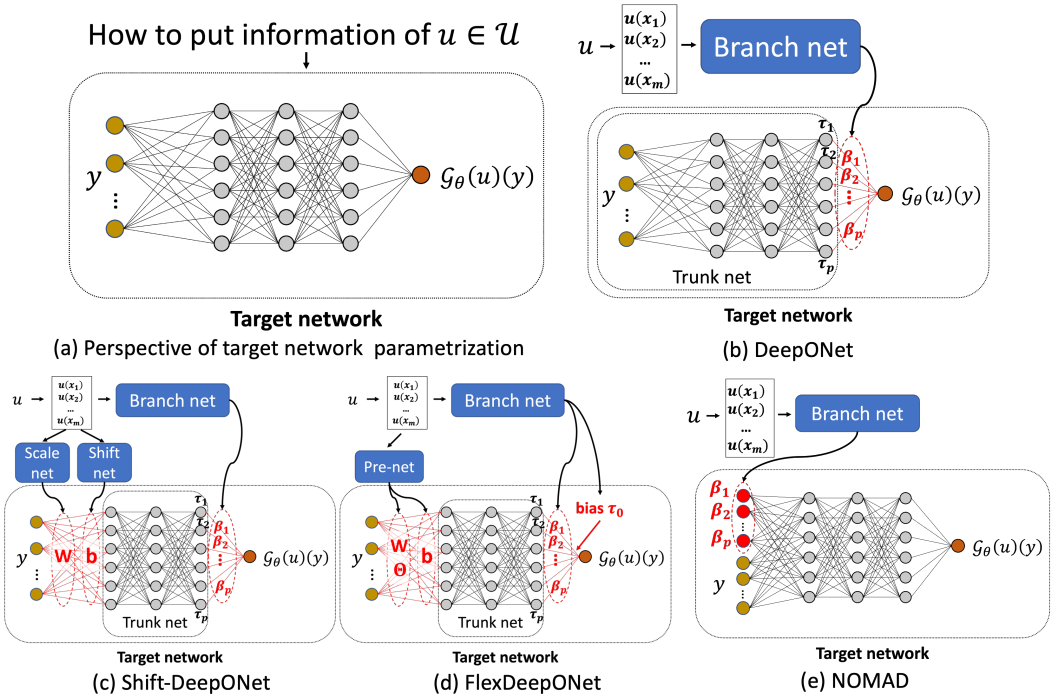

Figure 2: (a) The perspective target network parametrization for operator learning. (b-e)DeepONet and its variant models for operator learning.

for any $u \in \mathcal{U}$. The role of branch net can be interpreted as learning the coefficient of the target function $\mathcal{G}(u)(y)$. They use one additional neural network, called trunk net τ as shown below.

$$\tau : \mathcal{Y} \to \mathbb{R}^{p+1}, \tau(y) := \{\tau_k(y)\}_{k=0}^{p} \tag{3}$$

for any $y \in \mathcal{Y}$. The role of trunk net can be interpreted as learning an affine space V that can efficiently approximate output function space $C(\mathcal{Y}; \mathbb{R}^{d_s})$. The functions $\tau_1(y)$, ...,$\tau_p(y)$ become the p-basis of vector space associated with V and $\tau_0(y)$ becomes a point of V. By using the trunk net τ, the τ-induced reconstructor $\mathcal{R}$ is defined as

$$\mathcal{R}_\tau : \mathbb{R}^p \to C(\mathcal{Y}; \mathbb{R}^{d_s}), \mathcal{R}_\tau(\beta)(y) := \tau_0(y) + \sum_{k=1}^{p} \beta_k \tau_k(y) \tag{4}$$

where $\beta = (\beta_1, \beta_2, ..., \beta_p) \in \mathbb{R}^p$. In DeepONet, $\tau_0(y)$ is restricted to be a constant $\tau_0 \in \mathbb{R}$ that is contained in a reconstructor $\mathcal{R}$. The architecture of DeepONet is described in Figure 2 (b).

Here, the τ-induced reconstructor $\mathcal{R}_\tau$ is the linear approximation of the output function space. Because the linear approximation $\mathcal{R}$ cannot consider the elements in its orthogonal complement, a priori limitation on the best error of DeepONet is explained in Lanthaler et al. (2022) as

$$\left(\int_{\mathcal{U}} \int_{\mathcal{Y}} |\mathcal{G}(u)(y) - \mathcal{R}_\tau \circ \mathcal{A} \circ \mathcal{E}(u)(y)|^2 dy d\mu(u) \right)^{\frac{1}{2}} \geq \sqrt{\sum_{k>p} \lambda_k}, \tag{5}$$

where $\lambda_1 \geq \lambda_2 \geq ...$ are the eigenvalues of the covariance operator $\Gamma_{\mathcal{G}_{\#\mu}}$ of the push-forward measure $\mathcal{G}_{\#\mu}$. Several studies point out that the slow decay rate of the lower bound leads to inaccurate approximation operator learning using DeepONet (Kovachki et al., 2021b; Hadorn, 2022; Lanthaler et al., 2022). For example, the solution operator of the advection PDEs (Seidman et al., 2022; Venturi & Casey, 2022) and of the Burgers' equation (Hadorn, 2022) are difficult to approximate when we are using the DeepONet with the small number of basis p.

3.3 VARIANT MODELS OF DEEPONET

Several variants of DeepONet have been developed to overcome its limitation. All these models can be viewed from the perspective of parametrizing the target function as a neural network. When we think of the target network that receives y as an input and generates an output $\mathcal{G}_\theta(u)(y)$, the DeepONet and its variant model can be distinguished by how information from the input function u is injected into this target network $\mathcal{G}_\theta(u)$, as described in Figure 2 (a). From this perspective, the trunk net in the DeepONet can be considered the target network except for the final output as shown in Figure 2 (b). The output of the branch net gives the weight between the last hidden layer and the final output.

Hadorn (2022) proposed Shift-DeepONet. The main idea is that a scale net and a shift net are used to shift and scale the input query position y. Therefore, it can be considered that the information of input function u generates the weights and bias between the input layer and the first hidden layer, as explained in Figure 2 (c).

Venturi & Casey (2022) proposed Flex-DeepONet, explained in Figure 2 (d). They used the additional network, pre-net, to give the bias between the input layer and the first hidden layer. Additionally, the output of the branch net also admits the additional output τ_0 to provide more information on input function u at the last inner product layer.

NOMAD is recently developed by Seidman et al. (2022) to overcome the limitation of DeepONet which learns a linear output manifold using the inner product between branch net and trunk net. They devise a non-linear output manifold using a neural network that takes the output of branch net $\{\beta_i\}_{i=1}^p$ and the query location y. As explained in Figure 2 (e), the target network receives information about function u as an additional input, similar to other conventional neural embedding methods (Park et al., 2019; Chen & Zhang, 2019; Mescheder et al., 2019).

All of these methods provide information on the input function u to only a part of the target network. It makes sense to use a hypernetwork to share sensor information with all target network parameters. We propose a general model HyperDeepONet (Figure 3), which contains the vanilla DeepONet, Flex-DeepONet, and shift-DeepONet, as a special case of the HyperDeepONet, as explained in the following section.

4 PROPOSED MODEL : HYPERDEEPONET

4.1 ARCHITECTURE OF HYPERDEEPONET

The HyperDeepONet structure is described in Figure 3. The encoder $\mathcal{E}$ and the approximator $\mathcal{A}$ are used, similar to the vanilla DeepONet. The proposed structure replaces the branch net with the hypernetwork, and allows the output of the hypernetwork to generate not only the coefficient of basis but also all other parameters of the target network. More precisely, we define the hypernetwork h as

$$h_\theta : \mathcal{U} \to \mathbb{R}^p, h_\theta(u) := \mathcal{A} \circ \mathcal{E}(u) \tag{6}$$

for any $u \in \mathcal{U}$. Then, $h(u) = \Theta \in \mathbb{R}^p$ is a network parameter of the target network, which is used in reconstructor for the HyperDeepONet. We define the reconstructor $\mathcal{R}$ as

$$\mathcal{R} : \mathbb{R}^p \to C(\mathcal{Y}; \mathbb{R}^{d_s}), \mathcal{R}(\Theta)(y) := \text{NN}(y; \Theta) \tag{7}$$

where $\Theta = [W, b] \in \mathbb{R}^p$, and NN denotes the target network which is a fully connected neural network.

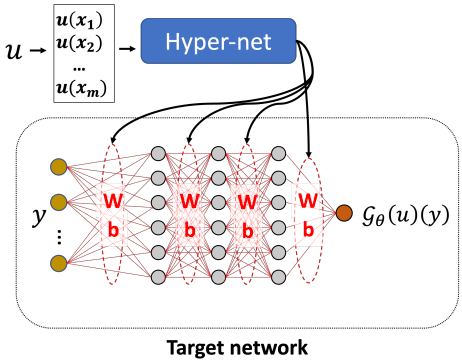

Figure 3: The proposed HyperDeepONet

Here, we can think that the hypernetwork determines p weights $\{\alpha_i\}_{i=1}^p$ between the final hidden layer and the output layer instead of DeepONet's last inner-product calculation. It implies that the structure of HyperDeepONet contains the entire structure of DeepONet. As shown in Figure 2 (c) and (d), Shift-DeepONet and Flex-DeepONet are also can be viewed as special cases of the

HyperDeepONet, where the output of the hypernetwork determines the weights or biases of some layers of the target network. The outputs of the hypernetwork determine the biases for the first hidden layer in the target network for NOMAD in Figure 2 (e).

4.2 Comparison on complexity of DeepONet and HyperDeepONet

In this section, we would like to clarify the complexity of the DeepONet required for the approximation $\mathcal{A}$ and reconstruction $\mathcal{R}$ based on the theory in Galanti & Wolf (2020). Furthermore, using the results on the upper bound for the complexity of hypernetwork Galanti & Wolf (2020), we will show that the HyperDeepONet entails relatively lower complexity than the DeepONet.

4.2.1 Notations and definitions

Suppose that the sensor values $\mathcal{E}(u) = (u(x_1), u(x_2), ..., u(x_m))$ are given. For simplicity, we consider the case where the domain $\mathcal{Y}$ of the target function and space of the m sensor values are $[-1,1]^{d_y}$ and $[-1,1]^m$, respectively. For the composition $\mathcal{R}\circ\mathcal{A} : \mathbb{R}^m \to C(\mathcal{Y};\mathbb{R})$, we are interested in approximating the mapping $\mathcal{O} : \mathbb{R}^{m+d_y} \to \mathbb{R}$, which is defined as follows:

$$\mathcal{O}(\mathcal{E}(u), y) := (\mathcal{R}\circ\mathcal{A}(\mathcal{E}(u)))(y), \quad \text{for } y \in \mathbb{R}^{d_y}, \mathcal{E}(u) \in \mathbb{R}^m.$$

The supremum norm $\|h\|_\infty$ refers to the value of $\max_{y\in\mathcal{Y}} \|h(y)\|$. Now, we introduce the Sobolev space $\mathcal{W}_{r,n} \subset C^r([-1,1]^n;\mathbb{R})$, which is defined as follows. For $r, n \in \mathbb{N}$,

$$\mathcal{W}_{r,n} := \left\{ h : [-1,1]^n \to \mathbb{R} \quad \middle| \|h\|_r^s := \|h\|_\infty + \sum_{1\le|\mathbf{k}|\le r} \|D^{\mathbf{k}}h\|_\infty \le 1 \right\},$$

where $D^{\mathbf{k}}h$ denotes the partial derivative of h with respect to multi-index $\mathbf{k} \in \{\mathbb{N}\cup\{0\}\}^{d_y}$. We assume that the mapping $\mathcal{O}$ lies in the Sobolev space $\mathcal{W}_{r,m+d_y}$.

For the nonlinear activation σ, the following class of neural network $\mathcal{F}$ represents the fully-connected neural network with depth k and corresponding width $(h_1, h_2, \cdots, h_{k+1})$, where $W^i \in \mathbb{R}^{h_i} \times \mathbb{R}^{h_{i+1}}$ and $b_i \in \mathbb{R}^{h_{i+1}}$ denote the weights and bias of the i-th layer respectively.

$$\mathcal{F} := \left\{ f : [-1,1]^n \to \mathbb{R} | f(y;[\mathbf{W},\mathbf{b}]) = W^k \cdot \sigma(W^{k-1}\cdots\sigma(W^1\cdot y + b^1) + b^{k-1}) + b^k \right\}$$

There are activation functions that facilitate a close approximation for the Sobolev space, with small complexity. We will refer to those activation functions as universal activation functions. The formal definition can be found below, where the distance between the class of neural network $\mathcal{F}$ and the Sobolev space $\mathcal{W}_{r,n}$ is defined by $d(\mathcal{F};\mathcal{W}_{r,n}) := \sup_{f\in\mathcal{F}} \inf_{\psi\in\mathcal{W}_{r,n}} \|f-\psi\|_\infty$.

Definition 1. *(Galanti & Wolf, 2020) (Universal activation). The activation function σ is called universal activation if there exists a class of neural network $\mathcal{F}$ with activation function σ such that the number of parameters of $\mathcal{F}$ is $O(\epsilon^{-n/r})$ with $d(\mathcal{F};\mathcal{W}_{r,n}) \le \epsilon$ for all $r, n \in \mathbb{N}$.*

Most well-known activation functions are universal activations which are infinitely differentiable and non-polynomial in any interval (Mhaskar, 1996). Furthermore, Hanin & Sellke (2017) state that the ReLU activation is also universal activation.

The result of providing a lower bound on the number of parameters also exists. We first introduce an assumption before introducing the theorem. For $r = 0$, Galanti & Wolf (2020) remark that the assumption is valid for 2-layered neural networks with respect to the L^2 norm when an activation function σ is either hyperbolic tangent or sigmoid function.

Assumption 1. *Suppose that $\mathcal{F}$ and $\mathcal{W}_{r,n}$ represent the class of neural network and the target function space to approximate, respectively. Let $\mathcal{F}'$ be a neural network class that represents a structure in which one neuron is added rather than $\mathcal{F}$. Then, the followings holds for all $\psi \in \mathcal{W}_{r,n}$ not contained in $\mathcal{F}$.*

$$\inf_{f\in\mathcal{F}} \|f-\psi\|_\infty > \inf_{f\in\mathcal{F}'} \|f-\psi\|_\infty.$$

The following theorem holds under the aformentioned assumption, which states that the universal activation indeed provides a sharp bound on the number of parameters. Note that a real-valued function $g \in L^1(\mathbb{R})$ is called bounded variation if its total variation $\sup_{\phi\in C_c^1(\mathbb{R}),\|\phi\|_\infty\le 1} \int_{\mathbb{R}} g(x)\phi'(x)dx$ is finite.

Theorem 1. *(Galanti & Wolf, 2020). Suppose that $\mathcal{F}$ is a class of neural networks with a piecewise $C^1(\mathbb{R})$ activation function $\sigma : \mathbb{R} \to \mathbb{R}$ of which derivative σ' is bounded variation. If any non-constant $\psi \in \mathcal{W}_{r,n}$ does not belong to $\mathcal{F}$, then $d(\mathcal{F}; W_{r,n}) \leq \epsilon$ implies the number of parameters in $\mathcal{F}$ should be $\Omega(\epsilon^{-n/r})$.*

4.2.2 LOWER BOUND FOR THE COMPLEXITY OF THE DEEPONET

Now, we present the minimum number of parameters in DeepONet. The following theorem gives a criterion on the least number of parameters in trunk net to get the desired error when using the DeepONet. It states that the number of required parameters increases when the target functions are irregular, corresponding to a small r. $\mathcal{F}_{DeepONet}(\mathcal{B}, \mathcal{T})$ denotes the class of function in DeepONet, induced by the class of branch net $\mathcal{B}$ and the class of trunk net $\mathcal{T}$.

Theorem 2. *(Complexity of DeepONet) Let $\sigma : \mathbb{R} \to \mathbb{R}$ be a universal activation function in $C^r(\mathbb{R})$ such that σ and σ' are bounded. Suppose that the class of branch net $\mathcal{B}$ has a bounded Sobolev norm (i.e. $\|\beta\|_r^s \leq l_1, \forall \beta \in \mathcal{B}$). If any non-constant $\psi \in \mathcal{W}_{r,n}$ does not belong to any class of neural network, then the number of parameters in the class of trunk net $\mathcal{T}$ is $\Omega(\epsilon^{-d_y/r})$ when $d(\mathcal{F}_{DeepONet}(\mathcal{B}, \mathcal{T}); \mathcal{W}_{r,d_y+m}) \leq \epsilon$.*

The fundamental approach to proving the theorem is substituting a neural network for the inner product between the branch net and trunk net. The neural network with small complexity could approximate the inner product with good performance since the inner product is an infinitely differentiable function. The DeepONet showed defects as it could be replaced with a neural network with $\mathcal{E}(u) \in [-1, 1]^m$ and $y \in [-1, 1]^{d_y}$ as input.

A large number of basis p in DeepONet increases the number of parameters of the trunk net which can be thought of as a target network in HyperDeepONet. Models such as Shift-DeepONet and flexDeepONet could achieve the desired accuracy with a small number of basis, but there was a trade-off in which the first hidden layer of the target network required numerous units. There was no restriction on the dimension of the last hidden layer in the target network for NOMAD which uses a fully nonlinear reconstruction. However, the first hidden layer of the target network had to be wide enough, increasing the number of parameters. Details can be found in the appendix.

For the proposed HyperDeepONet, the sensor values $\mathcal{E}(u)$ determines the weight and bias of all other layers as well as the weight of the last layer of the target network. Due to the nonlinear activation functions between linear matrix multiplication, it is difficult to replace HyperDeepONet with a single neural network that receives $[\mathcal{E}(u), y] \in \mathbb{R}^{d_y+m}$ as input. Galanti & Wolf (2020) state that there exists a HyperDeepONet approximation such that the number of parameters in target network is $O(\epsilon^{-d_y/r})$. It implies that the proposed method reduces the complexity compared to all the variants of DeepONet.

5 EXPERIMENTS

In this section, we verify the effectiveness of the proposed model HyperDeepONet to learn the operators with the complex target function space. To be more specific, we focus on operator learning problems in which the space of output function space is complex. One data point of with a triplet $(u_i, y, \mathcal{G}(u)(y))$ so that the input function u_i makes multiple data points for different values of y. Note that we use the 1000 train input-output function pairs and 200 test pairs for all experiments.

We first consider the **identity operator** $\mathcal{G} : u_i \mapsto u_i$ for the toy example. The Chebyshev polynomial is used as the input (=output) for the identity operator problem. The Chebyshev polynomials of the first kind T_l of degree 20 can be written as $u_i \in \{\sum_{l=0}^{19} c_l T_l(x) | c_l \in [-1/4, 1/4]\}$ with random sampling c_l from uniform distribution $U[-1/4, 1/4]$.

The **differentiation operator** $\mathcal{G} : u_i \mapsto \frac{d}{dx} u_i$ is considered for the second problem. Previous works handled the anti-derivative operator which makes the output function smoother by averaging (Lu et al., 2019; 2022). Here, we choose the differentiation operator instead of the anti-derivative operator to focus on operator learning when the operator's output function space is complex. We first sample the output function $\mathcal{G}(u)$ from the above Chebyshev polynomial of degree 20. The input function is generated using the numerical method that integrates the output function.

Model	DeepONet	Shift	Flex	NOMAD	Hyper(ours)
Identity	0.578±0.003	0.777±0.018	0.678±0.062	0.5783±0.020	**0.036±0.005**
Differentiation	0.559±0.001	0.624±0.015	0.562±0.016	0.558±0.003	**0.127±0.043**

Table 1: The mean relative L^2 test error with standard deviation for the identity operator and the differentiation operator. The DeepOnet, its variants, and the HyperDeepONet use the target network $d_y \to 20 \to 20 \to 10 \to 1$ with *tanh* activation function. Five training trials are performed independently.

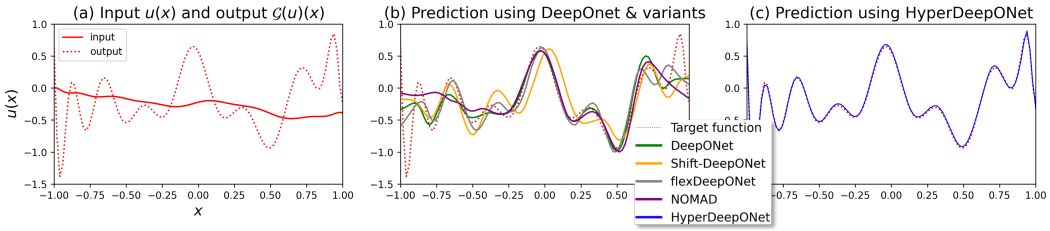

Figure 4: One test data example of differentiation operator problem.

Lastly, the solution operators of PDEs are considered. We deal with two problems which has the complex target function in previous works (Lu et al., 2022; Hadorn, 2022). For these two solution operator problems, we use the dataset generated in Lu et al. (2022). We consider the **solution operator of Burgers' equation** which maps the initial condition $w_0(x) = w(0, x)$ to the solution $w(1, x)$ at $t = 1$, i.e., $\mathcal{G} : w_0(x) \mapsto w(1, x)$. The solution of the Burgers' equation has a discontinuity in a short time, although the initial input function is smooth. The input function $w_0(x)$ is sampled from a Gaussian random field. The **solution operator of the advection equation** is also considered a mapping from the rectangle shape initial input function $w_0(x) = w(0, x)$ to the solution $w(0.5, x)$ at $t = 0.5$, i.e., $\mathcal{G} : w_0(x) \mapsto w(0.5, x)$. Detail explanation is provided in Appendix C.

Expressivity of target network We compare the expressivity of the small target network using different models. We focus on the identity operator and the differentiation operator in this experiment. All models employ the small target network $d_y \to 20 \to 20 \to 10 \to 1$ with the hyperbolic tangent activation function. The branch net and the additional networks (scale net, shift net, pre-net, and hypernetwork) in five models also use the same network size as the target network.

Table 1 shows the result on two operator learning problems. The DeepONet and other models have high error to learn complex operator when the small target network is used. On the other hands, the HyperDeepONet has lower error than the other models. This is consistent with the theorem in the previous section that HyperDeepONet can achieve improved approximations than the DeepONet when the complexity of the target network is the same. Figure 4 shows a prediction on differentiation operator which has highly complex target function. The same trends are observed when the number of layers in the branch net and the hypernetwork vary (Figure 7, 8), and the activation function or the number of sensors changes (Appendix C).

Same number of learnable parameters The previous experiments compare the models using the same target network structure. It seems that the larger number of parameters in the proposed

		Branch (Hyper)	Target	#Param	Rel error
Burgers	DeepONet	m-128-128-128-128	d_y-128-128-128-128-1	115K	0.0391±0.0040
	Hyper(ours)	m-66-66-66-66-66-N_θ	d_y-20-20-20-20-1	114K	**0.0196±0.0044**
Advection	DeepONet	m-256-256	d_y-256-256-256-256-1	274K	**0.0046±0.0017**
	Hyper(ours)	m-70-70-70-70-70-N_θ	d_y-33-33-33-33-1	268K	0.0048±0.0009

Table 2: The mean relative L^2 test error with standard deviation for two solution operator learning problem. N_θ and #Param denote the number of parameters in target network and the number of learnable parameters, respectively. Five training trials are performed independently.

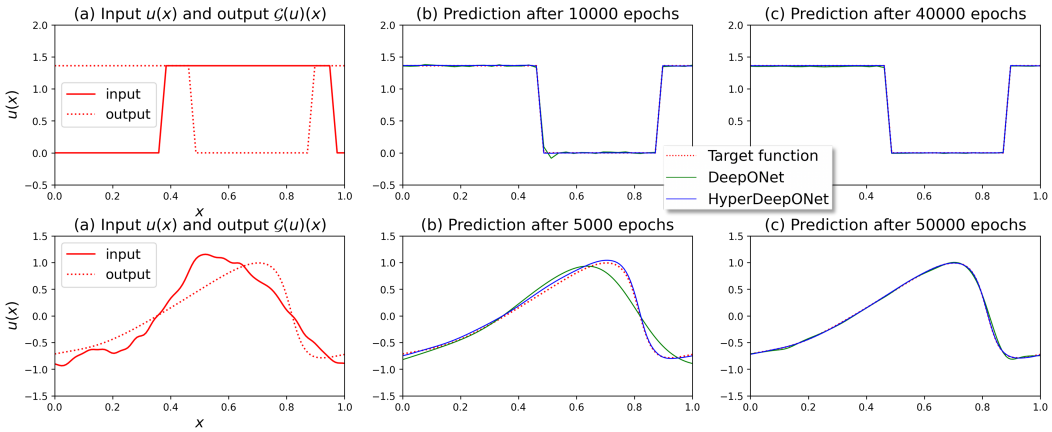

Figure 5: One test data example of prediction on the advection equation (First row) and Burgers' equation (Second row) using the DeepONet and the HyperDeepONet.

HyperDeepONet makes the operator approximation more accurate. In this section, the comparison between the DeepONet and the HyperDeepONet is considered when using the same number of learnable paramters. We focus on the solution operator of the advection equation and Burgers' equation in this experiment.

For the two solution operator learning problems, we use the same hyperparameters proposed in Lu et al. (2022) for DeepONet. We use the smaller target network with the larger hypernetwork for the HyperDeepONet to compare the DeepONet. The number of learnable parameters for the hypernetwork is similar but smaller than the DeepONet. Note that the vanilla DeepONet is used without the output normalization or the boundary condition enforcing techniques explained in Lu et al. (2022) to focus on the primary limitation of the DeepONet. More Details is in Appendix C. Table 2 shows that the HyperDeepONet achieves a similar or better performance than the DeepONet when the two models use the same number of learnable parameters. For advection equation problems, the hyperdeeponet has a slightly higher error, but this error is close to perfect operator prediction. It shows that not only the complexity of target network but also the number of learnable parameters can be reduced to obtain the desired accuracy using the HyperDeepONet. It is observed that the HyperDeepONet learns the complex target functions faster than the DeepONet (Figure 5).

6 Conclusion and discussion

In this work, the HyperDeepONet is developed to overcome the limitation of expressivity that DeepONet suffers from. The method of incorporating an additional network and a nonlinear reconstructor could not thoroughly solve this limitations. The hypernetwork, which involves multiple weights simultaneously, was a desired complexity-reducing structure based on theory and experiments.

We only focused on the case when the hypernetwork and the target network are constructed as fully connected neural networks. In the future, the structure of the two networks can be replaced with CNN or ResNet, as the structure of the branch net and trunk net of DeepONet can be changed to another network (Lu et al., 2022). Additionally, it seems interesting to research a simplified modulation network proposed by Mehta et al. (2021) still has the same expressivity as HyperDeepONet.

The output of the hypernetwork would be high-dimensional when the size of the target network is very large. There are several studies to reduce the number of outputs in the hypernetwork (Ha et al., 2016; Pawlowski et al., 2017). The chunked HyperDeepONet, developed by Von Oswald et al. (2019), can be used with a trade-off between accuracy and memory when a model with reduced parameters is required for the large target network. Some techniques from implicit neural representation can improve the expressivity of target network (Sitzmann et al., 2020). Using a sine function as an activation function with preprocessing will promote the expressivity of the target network. We also leave the research on the class of activation functions satisfying the assumption except for hyperbolic tangent or sigmoid functions as future work.

REFERENCES

Saakaar Bhatnagar, Yaser Afshar, Shaowu Pan, Karthik Duraisamy, and Shailendra Kaushik. Prediction of aerodynamic flow fields using convolutional neural networks. *Comput. Mech.*, 64 (2):525–545, 2019. ISSN 0178-7675. doi: 10.1007/s00466-019-01740-0. URL https://doi.org/10.1007/s00466-019-01740-0.

Shengze Cai, Zhicheng Wang, Lu Lu, Tamer A Zaki, and George Em Karniadakis. Deepm&mnet: Inferring the electroconvection multiphysics fields based on operator approximation by neural networks. *Journal of Computational Physics*, 436:110296, 2021.

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

Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5939–5948, 2019.

Tejalal Choudhary, Vipul Mishra, Anurag Goswami, and Jagannathan Sarangapani. A comprehensive survey on model compression and acceleration. *Artificial Intelligence Review*, 53(7): 5113–5155, 2020.

Filipe de Avila Belbute-Peres, Yi-fan Chen, and Fei Sha. Hyperpinn: Learning parameterized differential equations with physics-informed hypernetworks. In *The Symbiosis of Deep Learning and Differential Equations*, 2021.

Tim De Ryck and Siddhartha Mishra. Generic bounds on the approximation error for physics-informed (and) operator learning. *arXiv preprint arXiv:2205.11393*, 2022.

Weinan E and Bing Yu. The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems. *Commun. Math. Stat.*, 6(1):1–12, 2018. ISSN 2194-6701. doi: 10.1007/s40304-018-0127-z. URL https://doi.org/10.1007/s40304-018-0127-z.

Tomer Galanti and Lior Wolf. On the modularity of hypernetworks. *Advances in Neural Information Processing Systems*, 33:10409–10419, 2020.

Xiaoxiao Guo, Wei Li, and Francesco Iorio. Convolutional neural networks for steady flow approximation. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 481–490, 2016.

David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

Patrik Simon Hadorn. Shift-deeponet: Extending deep operator networks for discontinuous output functions. ETH Zurich, Seminar for Applied Mathematics, 2022.

Boris Hanin and Mark Sellke. Approximating continuous functions by relu nets of minimal width. *arXiv preprint arXiv:1710.11278*, 2017.

Rakhoon Hwang, Jae Yong Lee, Jin Young Shin, and Hyung Ju Hwang. Solving pde-constrained control problems using operator learning. *arXiv preprint arXiv:2111.04941*, 2021.

Yuehaw Khoo, Jianfeng Lu, and Lexing Ying. Solving parametric PDE problems with artificial neural networks. *European J. Appl. Math.*, 32(3):421–435, 2021. ISSN 0956-7925. doi: 10.1017/S0956792520000182. URL https://doi.org/10.1017/S0956792520000182.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Sylwester Klocek, Łukasz Maziarka, Maciej Wołczyk, Jacek Tabor, Jakub Nowak, and Marek Śmieja. Hypernetwork functional image representation. In *International Conference on Artificial Neural Networks*, pp. 496–510. Springer, 2019.

Nikola Kovachki, Samuel Lanthaler, and Siddhartha Mishra. On universal approximation and error bounds for fourier neural operators. *Journal of Machine Learning Research*, 22:Art–No, 2021a.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces. *arXiv preprint arXiv:2108.08481*, 2021b.

Thorsten Kurth, Shashank Subramanian, Peter Harrington, Jaideep Pathak, Morteza Mardani, David Hall, Andrea Miele, Karthik Kashinath, and Animashree Anandkumar. Fourcastnet: Accelerating global high-resolution weather forecasting using adaptive fourier neural operators. *arXiv preprint arXiv:2208.05419*, 2022.

Samuel Lanthaler, Siddhartha Mishra, and George E Karniadakis. Error estimates for deeponets: A deep learning framework in infinite dimensions. *Transactions of Mathematics and Its Applications*, 6(1):tnac001, 2022.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020a.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Multipole graph neural operator for parametric partial differential equations. *arXiv preprint arXiv:2006.09535*, 2020b.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020c.

Chensen Lin, Zhen Li, Lu Lu, Shengze Cai, Martin Maxey, and George Em Karniadakis. Operator learning for predicting multiscale bubble growth dynamics. *The Journal of Chemical Physics*, 154(10):104118, 2021.

Lu Lu, Pengzhan Jin, and George Em Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021.

Lu Lu, Xuhui Meng, Shengze Cai, Zhiping Mao, Somdatta Goswami, Zhongqiang Zhang, and George Em Karniadakis. A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data. *Computer Methods in Applied Mechanics and Engineering*, 393:114778, 2022.

Zhiping Mao, Lu Lu, Olaf Marxen, Tamer A. Zaki, and George Em 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:Paper No. 110698, 24, 2021. ISSN 0021-9991. doi: 10.1016/j.jcp.2021.110698. URL `https://doi.org/10.1016/j.jcp.2021.110698`.

Ishit Mehta, Michaël Gharbi, Connelly Barnes, Eli Shechtman, Ravi Ramamoorthi, and Manmohan Chandraker. Modulated periodic activations for generalizable local functional representations. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 14214–14223, 2021.

Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4460–4470, 2019.

Hrushikesh N Mhaskar. Neural networks for optimal approximation of smooth and analytic functions. *Neural computation*, 8(1):164–177, 1996.

MG Sarwar Murshed, Christopher Murphy, Daqing Hou, Nazar Khan, Ganesh Ananthanarayanan, and Faraz Hussain. Machine learning at the network edge: A survey. *ACM Computing Surveys (CSUR)*, 54(8):1–37, 2021.

Shaowu Pan, Steven L Brunton, and J Nathan Kutz. Neural implicit flow: a mesh-agnostic dimensionality reduction paradigm of spatio-temporal data. *arXiv preprint arXiv:2204.03216*, 2022.

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 165–174, 2019.

Nick Pawlowski, Andrew Brock, Matthew CH Lee, Martin Rajchl, and Ben Glocker. Implicit weight uncertainty in neural networks. *arXiv preprint arXiv:1711.01297*, 2017.

Michael Prasthofer, Tim De Ryck, and Siddhartha Mishra. Variable-input deep operator networks. *arXiv preprint arXiv:2205.11404*, 2022.

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.*, 378:686–707, 2019. ISSN 0021-9991. doi: 10.1016/j.jcp.2018.10.045. URL https://doi.org/10.1016/j.jcp.2018.10.045.

Jacob H Seidman, Georgios Kissas, Paris Perdikaris, and George J Pappas. Nomad: Nonlinear manifold decoders for operator learning. *arXiv preprint arXiv:2206.03551*, 2022.

Justin Sirignano and Konstantinos Spiliopoulos. DGM: a deep learning algorithm for solving partial differential equations. *J. Comput. Phys.*, 375:1339–1364, 2018. ISSN 0021-9991. doi: 10.1016/j.jcp.2018.08.029. URL https://doi.org/10.1016/j.jcp.2018.08.029.

Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33:7462–7473, 2020.

Simone Venturi and Tiernan Casey. Svd perspectives for augmenting deeponet flexibility and interpretability. *arXiv preprint arXiv:2204.12670*, 2022.

Johannes Von Oswald, Christian Henning, João Sacramento, and Benjamin F Grewe. Continual learning with hypernetworks. *arXiv preprint arXiv:1906.00695*, 2019.

Sifan Wang, Hanwen Wang, and Paris Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed deeponets. *Science advances*, 7(40):eabi8605, 2021.

Sifan Wang, Hanwen Wang, and Paris Perdikaris. Improved architectures and training algorithms for deep operator networks. *Journal of Scientific Computing*, 92(2):1–42, 2022.

Yinhao Zhu, Nicholas Zabaras, Phaedon-Stelios Koutsourelakis, and Paris Perdikaris. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *J. Comput. Phys.*, 394:56–81, 2019. ISSN 0021-9991. doi: 10.1016/j.jcp.2019.05.024. URL https://doi.org/10.1016/j.jcp.2019.05.024.

A NOTATIONS

We list the main notations in Table 3 which is not concretely described in this paper.

$\mathcal{U}$	the domain of input function
$\mathcal{Y}$	the domain of output function
d_u	the dimension of $\mathcal{U}$
d_y	the dimension of $\mathcal{Y}$
d_x	the dimension of codomain of input function
d_s	the dimension of codomain of output function
$\{x_1, \cdots, x_m\}$	sensor points
m	the number of sensor points
$\mathbb{R}^d$	Euclidean space of dimension d
$C^r(\mathbb{R})$	a set of functions that has continuous r-th derivative.
$C(\mathcal{Y}; \mathbb{R}^{d_s})$	a set of continuous function from $\mathcal{Y}$ to $\mathbb{R}^{d_s}$
$C^r([-1,1]^n; \mathbb{R})$	a set of functions from $[-1,1]^n$ to $\mathbb{R}$ of which r-th partial derivatives are continuous.
$n = O(\epsilon)$	There exists a constant C such that $n \leq C \cdot \epsilon, \forall \epsilon > 0$
$n = \Omega(\epsilon)$	There exists a constant C such that $n \geq C \cot \epsilon, \forall \epsilon > 0$
$n = o(\epsilon)$	n/ϵ converges to 0 as ϵ goes to 0.

Table 3: Notation

B On complexity of DeepONet and its variants

B.1 Proof of Theorem 2 on DeepONet complexity

The following lemma implies that the class of neural networks is sufficiently efficient to approximate the inner product in the linear decoder process.

Lemma 1. *For a number of basis $p \in \mathbb{N}$, let us define the inner product function $\pi : [-1,1]^{2p} \to \mathbb{R}$ as $\pi(a_1, \cdots, a_p, b_1, \cdots, b_p) = \sum_{i=1}^{p} a_i \cdot b_i (= \langle (a_1, \cdots, a_p), (b_1, \cdots, b_p) \rangle)$. Then for all postivie $t \in \mathbb{R}$, there exists a class of neural network $\mathcal{F}$ with universal activation $\sigma : \mathbb{R} \to \mathbb{R}$ such that the number of parameters of $\mathcal{F}$ is $O(p^{1+1/t} \cdot \epsilon^{-1/t})$ with $\inf_{f \in \mathcal{F}} \|f - \pi\|_\infty \leq \epsilon$.*

Proof. Fix a real number $t > 0$. First, we would like to approximate the product function $\pi^* : [-1,1]^2 \to \mathbb{R}$ which is defined as

$$\pi^*(a, b) = a \cdot b. \tag{8}$$

Note that $D^{\mathbf{k}} \pi^* = 0$ for all multi-index $\mathbf{k} \in \{\mathbb{N} \bigcup \{0\}\}^2$ such that $|\mathbf{k}| \geq 2$. For a multi-index $\mathbf{k}$ with $\|\mathbf{k}\| = 1$, $D^{\mathbf{k}} \pi^*$ contains only one term which is either a or b. In this case, we can simply observe that $\sum_{|\mathbf{k}|=1} \|D^{\mathbf{k}} f\|_\infty \leq 2 \cdot 1 = 2$ by the construction of the domain $[-1,1]^2$ for π. And finally,

$$\|\pi^*\|_\infty \leq \|a \cdot b\|_\infty \leq \|a\|_\infty \|b\|_\infty \leq 1 \cdot 1 = 1,$$

so that a function $(\pi^*/3)$ should be contained in $\mathcal{W}_{r,2}$ for any $r \in \mathbb{N}$. In particular, $f/3$ lies in $\mathcal{W}_{2t,2}$ so that there exists a neural network approximation $\{_{nn}$ in some class of neural network $\mathcal{F}^*$ with an universal activation function σ such that the number of parameters of $\mathcal{F}^*$ is $O((\epsilon/3p)^{-2/2t}) = O(p^{1/t} \cdot \epsilon^{-1/t})$, and

$$\|\pi^* - f_{nn}\|_\infty \leq \epsilon/3p,$$

by definition 1. Then the neural network $3 \cdot f_{nn}$ approximates the function f by an error ϵ/p which can be constructed by adjusting the weight value between the last two layer in f_{nn}.

Let us decompose the $2p$ dimensional inner product function π into p product functions $\{\pi_i\}_{i=1}^{p}$ which are defined as

$$\pi_i : \mathbb{R}^{2p} \to \mathbb{R}, \quad \pi_i(a_1, \cdots, a_p, b_1, \cdots, b_p) := \pi^*(a_i, b_i) = a_i \cdot b_i,$$

$\forall i \in \{1, \cdots, p\}$. Then each function π_i could be approximated within an ϵ/p-error by neural network NN_i which has $O(p^{1/t} \cdot \epsilon^{-1/t})$ parameters by the above discussion. Finally, by adding the

last weight $[1, 1, \cdots, 1] \in \mathbb{R}^{1\times p}$ which has input as the outputs of p neural networks $\{NN_i\}_{i=1}^p$, we can construct the neural network approximation NN of π such that the number of parameters is $O(p + p \cdot p^{1/t} \cdot \epsilon^{-1/t}) = O(p^{1+1/t} \cdot \epsilon^{-1/t})$. Class of neural network $\mathcal{F}$, which represents the structure of NN, satisfies the desired property. □

Consider the target function corresponding to observation $\mathcal{E}(u)$, $\mathcal{R} \circ \mathcal{A}(\mathcal{E}(u)) : [-1, 1]^m \to \mathbb{R}$. Let us define the map $\mathcal{O} : [-1, 1]^{d_y+m} \to \mathbb{R}$ by $\mathcal{O}(\mathcal{E}(u), y) := \mathcal{R} \circ \mathcal{A}(\mathcal{E}(u))(y)$ where $y \in [-1, 1]^{d_y}$. If we assume that $\mathcal{O}$ lies in $\mathcal{W}_{r,d_y+m}$ and ϕ cannot be represented as neural network of which input is $[\mathcal{E}(u), y]$. Then, we can obtain the following lemma which presents the lower bound on the number of parameters in DeepONet structure.

Lemma 2. *Let $\sigma : \mathbb{R} \to \mathbb{R}$ be a universal activation function in $C^r(\mathbb{R})$ such that σ' is a bounded variation. Suppose that the class of branch net $\mathcal{B}$ has a bounded Sobolev norm (i.e. $\|\beta\|_r^s \leq l_1, \forall \beta \in \mathcal{B}$). Additionally assume that supremum norm $\|\cdot\|_\infty$ of the class of trunk net $\mathcal{T}$ is bounded by l_2 and the number of parameters in $\mathcal{T}$ is $o(\epsilon^{-(d_y+m)/r})$. If any non-constant $\psi \in \mathcal{W}_{r,n}$ does not belong to any class of neural network, then the number of basis p in $\mathcal{T}$ is $\Omega(\epsilon^{-d_y/r})$ when $d(\mathcal{F}_{DeepONet}(\mathcal{B}, \mathcal{T}); \mathcal{W}_{r,d_y+m}) \leq \epsilon$.*

Proof. Suppose that there is no constant C satisfies the inequality $p \geq C \cdot (\epsilon^{-d_y/r})\ \forall \epsilon > 0$. In other words, there exists a sequence of DeepONet which has $\{p_n\}_{n=1}^\infty$ as latent dimension with a sequence of error $\{\epsilon_n\}_{n=1}^\infty$ in $\mathbb{R}$ such that $\epsilon_n \to 0$, and

$$p_n \leq \frac{1}{n} \cdot (\epsilon_n^{-d_y/r}) \quad (i.e. \quad p_n = o(\epsilon_n^{-d_y/r}) \quad \text{with respect to } n). \tag{9}$$

Let us denote corresponding sequence of branch net and trunk net by $\{\beta_n\}_{n=1}^\infty$ and $\{\tau_n\}_{n=1}^\infty$. Since the output of β_n is multi-dimensional vector, we should consider the norm of a vector. For consistency with other norms, we decide to apply L^∞ norm for the p_n-dimensional vector. By the assumption on the boundedness of B, the ith component of $\beta_n : \mathbb{R}^m \to \mathbb{R}$, $(\beta_n)_i$, has a Sobolev norm bounded by l_1. In other words, $\|(\beta_n)_i/l_1\|_r^s \leq 1$ and therefore, $(\beta_n)_i/l_1$ is contained in $W_{1,m}$. Since σ is a universal activation function, there exists a neural network approximation $(f_n)_i$ of $(\beta_n)_i$ such that the number of parameters is $O((\epsilon_n/l_1)^{-m/r}) = O(\epsilon_n^{-m/r})$, and

$$\|(f_n)_i - (b_n)_i\|_\infty \leq \epsilon_n/l_1.$$

Then, $f_n := (l_1 \cdot (f_n)_1, l_1 \cdot (f_n)_2, \cdots, l_1 \cdot (f_n)_p)$ becomes neural network approximation of β_n which has $O(p_n \cdot \epsilon_n^{-m/r})$ parameters within an error ϵ.

Let us denote the target function corresponding to observation $\mathcal{E}(u)$ by $\mathcal{R} \circ \mathcal{A}(\mathcal{E}(u)) : \mathbb{R}^{d_y} \to \mathbb{R}$. Then we can simply observe the following through the assumption on DeepONet approximation and above discussion.

$$\begin{aligned}\|\mathcal{R} \circ \mathcal{A}(\mathcal{E}(u))(x) - \langle f_n(\mathcal{E}(u)), \tau_n(x)\rangle\|_\infty \leq & \|\mathcal{R} \circ \mathcal{A}(\mathcal{E}(u))(x) - \langle \beta_n(\mathcal{E}(u)).\tau_n(x)\rangle\|_\infty + \\ & \|\langle \beta_n(\mathcal{E}(u)) - f_n(\mathcal{E}(u)), \tau_n(x)\rangle\|_\infty \\ \leq & \epsilon_n + \epsilon_n \cdot \|\tau_n(x)\|_\infty \\ \leq & \epsilon_n(1 + l_2), \quad \forall I \in [-1, 1]\end{aligned}$$

Now we would like to come up with the sequence of neural network which is an approximation of inner product between p_n dimensional vector in $[-1, 1]^{p_n}$. Note the following two inequalities $\|f_n(\mathcal{E}(u))\|_\infty \leq \|f_n(\mathcal{E}(u)) - \beta_n(\mathcal{E}(u))\|_\infty + \|\beta_n(\mathcal{E}(u))\|_\infty \leq \epsilon_n + \|\beta_n(\mathcal{E}(u))\|_r^s \leq \epsilon_n + l_1 \leq 2l_1$ and $\|\tau_n\|_\infty \leq l_2$ for large n. It implies that $f_n(\mathcal{E}(u))/2l_1$ and $\tau_n(x)/l_2$ lie in $[-1, 1]^{p_n}$. By lemma 1, there exists a class of neural network $\mathcal{H}_n$ such that the number of parameters is $O(p_n^{1+1/2d_yr} \cdot \epsilon_n^{-1/2d_yr})$ and,

$$\inf_{h \in \mathcal{H}_n} \|h - \pi_n\|_\infty \leq \epsilon_n$$

where π_n is the inner product corresponding to p_n dimensional vector. Choose a neural network $h_n : [-1, 1]^{2p_n} \to \mathbb{R}$ such that $\|h_n - \pi_n\|_\infty \leq 2\epsilon$. Then, by the triangular inequality,

$$\begin{aligned}&\|\mathcal{O}(\mathcal{E}(u))(x) - 2l_1 \cdot l_2 \cdot h_n(f_n\mathcal{E}(u)/2l_1, \tau_n(x)/l_2)\|_\infty \\ \leq & \|\mathcal{O}(\mathcal{E}(u))(x) - \langle f_n(\mathcal{E}(u)), \tau_n(x)\rangle\|_\infty + 2l_1 \cdot l_2 \cdot \|\langle f_n(\mathcal{E}(u))/2l_1, \tau_n(x)/l_2\rangle - \\ & h_n(f_n(\mathcal{E}(u))/2l_1, \tau_n(x)/l_2)\|_\infty \\ \leq & \epsilon_1(1 + l_2) + 2l_1 \cdot l_2 \cdot \epsilon_n = \epsilon_n(1 + l_2 + 2l_1 \cdot l_2).\end{aligned} \tag{10}$$

Finally, let us compute the number of parameters which is required to implement the function $2l_1 \cdot l_2 \cdot h_n(f_n(\mathcal{E}(u))/2l_1, \tau_n(x)/l_2)$. The only part that needs further consideration is scalar multiplication. Since we need one weight to multiply a constant with one real value, three scalar multiplications $h_n(f_n(\mathcal{E}(u))/2l_1, \tau_n(x)/l_2) \mapsto 2l_1 \cdot l_2 \cdot 2h_n(f_n(\mathcal{E}(u))/2l_1, \tau_n(x)/l_2)$, $f_n(\mathcal{E}(u)) \mapsto f_n(\mathcal{E}(u))/2l_1$ and $\tau_n(x) \mapsto \tau_n(x)/l_2$ require $1, p_n, p_n$ parameters respectively. Combining all the previous results with the size of trunk net, the total number of parameters is obtained in the form of

$$O(1 + 2 \cdot p_n + p_n^{1+1/2d_y r} \cdot \epsilon_n^{-1/2d_y r} + p_n \cdot \epsilon_n^{-m/r}) + o(\epsilon_n^{-(d_y+m)/r}) = o(\epsilon_n^{-(d_y+m)/r}),$$

since the initial assumption (9) on the number of latent dimension gives the following inequality.

$$\begin{aligned} p_n^{1+1/2d_y r} \cdot \epsilon_n^{-1/2d_y r} + p_n \cdot \epsilon_n^{-m/r} &\leq p_n(p_n^{1/2d_y r} \cdot \epsilon_n^{-1/2d_y r} + \epsilon_n^{-m/r}) \\ &\leq (1/n) \cdot \epsilon_n^{-d_y/r} \cdot (\epsilon_n^{-1/2r^2 - 1/2d_y r} + \epsilon_n^{-m/r}) \\ &\leq (1/n) \cdot \epsilon_n^{-d_y/r} \cdot 2 \cdot \epsilon_n^{-m/r} \\ &= (2/n) \cdot \epsilon_n^{-(d_y+m)/r}. \end{aligned}$$

On the one hand, the sequence of function $\{2l_1 \cdot l_2 \cdot h_n(f_n(\mathcal{E}(u))/2l_1, \tau_n(x)/l_2)\}_{n=1}^{\infty}$ is an sequence of approximation for $\mathcal{O}(\mathcal{E}(u))(x)$ within a corresponding sequence of error $\{\epsilon_n(1 + l_2 + 2l_1 \cdot l_2)\}_{n=1}^{\infty}$. Let us denote the sequence of class of neural networks corresponding to the sequence of function $\{2l_1 \cdot l_2 \cdot h_n(f_n(\mathcal{E}(u))/2l_1, \tau_n(x)/l_2)\}_{n=1}^{\infty}$ by $\{\mathcal{F}_n\}_{n=1}^{\infty}$. By the assumption, Theorem 1 implies the number of parameters in $\{\mathcal{F}_n\}_{n=1}^{\infty}$ is $\Omega((\epsilon_n \cdot (1 + l_2 + 2l_1 \cdot l_2))^{-(d_y+m)/r}) = \Omega(\epsilon_n^{-(d_y+m)/r})$. Therefore, the initial assumption 9 would result in a contradiction so that the desired property is valid. □

Through the above results, we can obtain the following results.

Theorem 3. *Let $\sigma : \mathbb{R} \to \mathbb{R}$ be a universal activation function in $C^r(\mathbb{R})$ such that σ and σ' are bounded. Suppose that the class of branch net $\mathcal{B}$ has a bounded Sobolev norm (i.e. $\|\beta\|_r^s \leq l_1, \forall \beta \in \mathcal{B}$). If any non-constant $\psi \in \mathcal{W}_{r,n}$ does not belong to any class of neural network, then the number of parameters in the class of trunk net $\mathcal{T}$ is $\Omega(\epsilon^{-d_y/r})$ when $d(\mathcal{F}_{DeepONet}(\mathcal{B}, \mathcal{T}); \mathcal{W}_{r,d_y+m}) \leq \epsilon$.*

Proof. Let us denote the number of parameters in $\mathcal{T}$ by $N_{\mathcal{T}}$. Suppose that there is no constant C satisfies the inequality $N_{\mathcal{T}} \geq C \cdot \epsilon^{-d_y/r}, \forall \epsilon > 0$. That is, there exists a sequence of DeepONet with the corresponding sequence of trunk net $\{\tau_n\}_{n=1}^{\infty}$ and sequence of error $\{\epsilon_n\}_{n=1}^{\infty}$ such that $\epsilon_n \to 0$ and,

$$N_{\tau_n} < (1/n) \cdot \epsilon^{-d_y/r} (i.e. N_{\tau_n} = o(\epsilon_n^{-d_y/r}) \text{ and, } N_{\tau_n} = o(\epsilon_n^{-(d_y+m)/r}) \text{ with respect to } n).$$

Since σ is bounded so that τ_n is a bounded function with respect to the supremum norm. Therefore, if we apply the lemma 2 with respect to n, the number of basis p_n should be $\Omega(\epsilon_n^{-d_y/r})$. Since p_n is also the number of output dimensions for trunk net τ_n, the number of parameters in τ_n should be larger than $p_n = \Omega(\epsilon_n^{-d_y/r})$. This leads to a contradiction. □

We argue that it does not improve performance for the hypernetwork to output the weights on the first layer of a target network. The following lemma shows that that procedure can be replaced by a small neural network structure.

Lemma 3. *Consider a function $\Phi : \mathbb{R}^{d_y \cdot (w+1)} \to \mathbb{R}^w$ which is defined below.*

$$\Phi(x_1, \cdots, x_{d_y}, \cdots, x_{(d_y-1)\cdot w+1} \cdots, x_{d_y \cdot w}, y_1, \cdots, y_{d_y}) := (\sum_{i=1}^{d_y} x_i \cdot y_i, \cdots, \sum_{i=1}^{d_y} x_{(d_y-1)\cdot w+i} \cdot y_i).$$

Then for any fixed positive real number t, there exists a neural network $f : \mathbb{R}^{d_y \cdot (w+1)} \to \mathbb{R}^w$ with universal activation σ such that the number of parameters in f is $O(w \cdot d_y^{1+1/t} \cdot \epsilon^{-1/t})$ and $\|\Phi - f\|_\infty \leq \epsilon$.

Proof. Using the lemma 1, we can construct a sequence of neural network $\{f_i\}_{i=1}^{w}$ which is an ϵ-approximation of the inner product with $O(d_y^{1+1/t} \cdot \epsilon^{-1/t})$ parameters. If we combine all of the w approxmiations, we get the desired neural network. □

Now we denote the class of Pre-net in Shift-DeepONet and flexDeepONet by $\mathcal{P}$. The class of Shift-DeepONet and flexDeepONet will be written as $\mathcal{F}_{Shift-DeepONet}(\mathcal{P}, \mathcal{B}, \mathcal{T})$ and $\mathcal{F}_{flexDeepONet}(\mathcal{P}, \mathcal{B}, \mathcal{T})$ respectively. The structure of Shift-DeepONet can be summarized as follows. For $\rho : \mathbb{R}^m \to \mathbb{R}^{w\cdot(d_y+1)} = (\rho_1 : \mathbb{R}^m \to \mathbb{R}^{w\times d_y}, \rho_2 : \mathbb{R}^m \to \mathbb{R}^w)$, $\beta : \mathbb{R}^m \to \mathbb{R}^p$, and $\tau : \mathbb{R}^w \to \mathbb{R}$, the Shift-DeepONet $f_{Shift-DeepONet}(\rho, \beta, \tau)$ induced by pre-net ρ, branch net β, and trunk net τ is defined as

$$f_{Shift-DeepONet}(\rho, \beta, \tau)(\mathcal{E}(u), y) := \langle \beta(\mathcal{E}(u)), \tau(\Phi(\rho_1(\mathcal{E}(u)), y) + \rho_2(\mathcal{E}(u)))\rangle$$

where Φ is defined as in 3.

B.2 Proof of Theorems on variants of DeepONet.

For the neural network f, the upper bound of the Lipschitz constant for f could be obtained as $L^{k-1} \cdot \Pi_{i=1}^{k} \|W^i\|_1$, where L is the Lipschitz constant of σ and the norm $\|\cdot\|_1$ denotes the matrix norm induced by vector 1-norms. We can impose constraints on the upper bound of the weights, which consequently enforce affine transformation W^i to be bounded with respect to the L^1 norm. Therefore, the Lipschitz continuity of the entire neural network can be guaranteed.

Theorem 4. *Let $\sigma : \mathbb{R} \to \mathbb{R}$ be a universal activation function in $C^r(\mathbb{R})$ such that σ and σ' are bounded. Suppose that the class of branch net $\mathcal{B}$ and pre-net $\mathcal{P}$ has a bounded Sobolev norm (i.e. $\|\beta\|_r^s \leq l_1, \forall\beta \in \mathcal{B}$, and $\|\rho\|_r^s \leq l_3.\forall\rho \in \mathcal{P}$) and any neural network in the class of trunk net $\mathcal{T}$ is Lipschitz continuous with constant l_2. If any non-constant $\psi \in \mathcal{W}_{r,n}$ does not belong to any class of neural network, then the number of parameters in $\mathcal{T}$ is $\Omega(\epsilon^{-d_y/r})$ when $d(\mathcal{F}_{Shift-DeepONet}(\mathcal{P}, \mathcal{B}, \mathcal{T}); \mathcal{W}_{r,d_y+m}) \leq \epsilon$. Finally, the inner product $\langle \beta_n, \tau_n\rangle$ is replaced with a neural network as in the 10 of lemma 2.*

Proof. Denote the number of parameters in $\mathcal{T}$ by $N_{\mathcal{T}}$. Suppose that there exists a sequence of pre-net $\{\rho_n\}_{n=1}^{\infty}$, branch net $\{\beta_n\}_{n=1}^{\infty}$ and trunk net $\{\tau_n\}_{n=1}^{\infty}$ with the corresponding sequence of error $\{\epsilon_n\}_{n=1}^{\infty}$ such that $\epsilon_n \to 0$ and,

$$N_{\mathcal{T}} = o(\epsilon_n^{-d_y/r}), \quad \text{and} \quad \sup_{\psi\in\mathcal{W}_{r,d_y+m}} \|f_{Shift-DeepONet}(\rho_n, \beta_n, \tau_n) - \psi\|_\infty \leq \epsilon_n.$$

The proof can be divided into three parts. Firstly, we come up with a neural network approximation ρ_n^{NN} of ρ_n of which size is $O(w \cdot \epsilon_n^{-m/r})$ within an error ϵ_n. Next, construct a neural network approximation of Φ using the lemma 3. Finally, the inner product is replaced with a neural network as in the inequality 10.

Since all techniques such as triangular inequality are consistent with the previous discussion, we will briefly explain why Lipschitz continuity is required for the trunk network, and omit the details. Approximating the Pre-Net of Shift DeepOnet, which is not in DeepOnet, inevitably results in an error in the input of the trunk net. We are reluctant to allow this error to change the output of the trunk net significantly. In this situation, the Lipschitz continuity provides the desired result. □

For $d_y = 1$, the additional rotation is only multiplying by 1 or -1. Since the weight and bias of the first layer alone can cover the scalar multiplication, flexDeepONet has the same properties as Shift-DeepONet in the above theorem.

Theorem 5. *Consider the case $d_y = 1$. Let $\sigma : \mathbb{R} \to \mathbb{R}$ be a universal activation function in $C^r(\mathbb{R})$ such that σ and σ' are bounded. Suppose that the class of branch net $\mathcal{B}$ has a bounded Sobolev norm (i.e. $\|\beta\|_r^s \leq l_1, \forall\beta \in \mathcal{B}$) and, any neural network in the class of trunk net $\mathcal{T}$ is Lipschitz continuous with constant l_2. If any non-constant $\psi \in \mathcal{W}_{r,n}$ does not belong to any class of neural network, then the number of parameters in $\mathcal{T}$ is $\Omega(\epsilon^{-d_y/r})$ when $d(\mathcal{F}_{flexDeepONet}(\mathcal{B}, \mathcal{T}); \mathcal{W}_{r,d_y+m}) \leq \epsilon$.*

Proof. The main difference between flexDeepONet and Shift-DeepONet, which is not mentioned earlier, is that the branch net affects the bias of the output layer. However, adding the values of the two neurons can be implemented in a neural network by adding only one weights of value 1 for each neuron, so all of the previous discussion is all valid. □

In fact, NOMAD can be substituted with the embedding method handled by Galanti & Wolf (2020). Suppose that the branch net function of NOMAD is continuously differentiable. Let's also assume that the Lipschitz constant of branch net and trunk net is bounded. We would like to briefly cite the relevant theorem here.

Theorem 6. *(Galanti & Wolf (2020)) Suppose that σ is a universal activation in $C^1(\mathbb{R})$ such that σ' is a bounded variation on $\mathbb{R}$. Additionally, suppose that there is no class of neural network can represent any function in $\mathcal{W}_{1,d_y+m}$ other than a constant function. If the weight on the first layer of target network in NOMAD is bounded with respect to L^1-norm, then $d(\mathcal{N};\mathcal{W}_{1,d_y+m}) \leq \epsilon$ implies the number of parameters in $\mathcal{N}$ is $\Omega(\epsilon^{-min(d_y+m,2\cdot m_y)})$ where $\mathcal{N}$ denotes the class of function contained as a target network of NOMAD.*

Finally, we present a lower bound on the number of parameters of DeepONet, including all up to branch net. Keep in mind that the proof of this theorem can apply to other variants of DeepONet.

Theorem 7. *(Total Complexity of DeepONet) Consider the case $d_y = 1$. Let $\sigma : \mathbb{R} \to \mathbb{R}$ be a universal activation function in $C^r(\mathbb{R})$ such that σ and σ' are bounded. Suppose that the class of branch net $\mathcal{B}$ has a bounded Sobolev norm (i.e. $\|\beta\|_r^s \leq l_1, \forall \beta \in \mathcal{B}$). If any non-constant $\psi \in \mathcal{W}_{r,n}$ does not belong to any class of neural network, then the number of parameters in DeepONet is $\Omega(\epsilon^{-(d_y+m)/R})$ for any $R > r$ when $d(\mathcal{F}_{DeepONet}(\mathcal{B},\mathcal{T});\mathcal{W}_{r,d_y+m}) \leq \epsilon$.*

Proof. For a positive $\epsilon < l_1$, suppose that there exists a class of branch net $\mathcal{B}$ and trunk net $\mathcal{T}$ such that $d(\mathcal{F}_{DeepONet}(\mathcal{B},\mathcal{T});\mathcal{W}_{r,d_y+m}) \leq \epsilon$. Let us denote the number of parameters in $\mathcal{F}_{DeepONet}$ by $N_{\mathcal{F}_{DeepONet}}$. Using the lemma 1 to replace DeepONet's inner products with neural networks as in 10, we can construct a class of neural network $\mathcal{F}$ such that the number of parameters $\mathcal{F}$ is $O(N_{\mathcal{F}_{DeepONet}} + p^{1+1/t} \cdot \epsilon^{-1/t})$ and,

$$d(\mathcal{F};\mathcal{W}_{r,d_y+m}) \leq (1 + 2l_1 \cdot l_2)\epsilon.$$

Suppose that $N_{\mathcal{F}_{DeepONet}} = o(\epsilon^{-(d_y+m)/r})$. Then, by theorem 1, $p^{1+1/t} \cdot \epsilon^{-1/t}$ should be $\Omega(\epsilon^{-(d_y+m)/r})$. Since t can be arbitrary large, the number of basis p should be $\Omega(\epsilon^{-(d_y+m)/R})$ for any $R > r$. □

C Experimental details and additional experiments

Activation: ReLU, $M = 30$		DeepONet	Shift	Flex	NOMAD	Hyper(Ours)
Size of target network	d_y-30-30-30-1	0.16797	0.02059	1.30852	1.04292	0.27209
	d_y-50-50-1	0.04822	0.05562	1.08760	1.11957	0.21391
Activation: ReLU, $M = 100$		DeepONet	Shift	Flex	NOMAD	Hyper(Ours)
Size of target network	d_y-30-30-30-1	0.02234	0.01743	1.08310	1.03741	0.19089
	d_y-50-50-1	0.07255	0.04645	1.47373	1.13217	0.14020
Activation: PReLU, $M = 30$		DeepONet	Shift	Flex	NOMAD	Hyper(Ours)
Size of target network	d_y-30-30-30-1	0.11354	0.02844	1.09395	1.03502	0.25651
	d_y-50-50-1	0.00873	0.04302	1.14073	1.06947	0.04054
Activation: PReLU, $M = 100$		DeepONet	Shift	Flex	NOMAD	Hyper(Ours)
Size of target network	d_y-30-30-30-1	0.01035	0.01083	1.05080	1.07791	0.16592
	d_y-50-50-1	0.07255	0.04645	1.47373	1.13217	0.14020

Table 4: Experiments of training identity operator under various conditions

In most experiments, we follow the hyperparameter setting in Lu et al. (2019; 2021; 2022). We use ADAM in Kingma & Ba (2014) as an optimizer with a learning rate of $1e-3$ and zero weight decay. In all experiments, an InverseTimeDecay scheduler was used, and the step size was fixed to 1. In the experiments of identity and differentiation operators, grid search was performed using the sets 0.0001, 0.0002, 0.0005, 0.001, 0.002, 0.005 for decay rates. The selected values of decay rate for each model could be found in Table 5.

Cases	**DeepONet**	**HyperDeepONet**	**Shift-DeepONet**	**flexDeepONet**	**NOMAD**
Decay rate (Identity)	0.0005	0.0001	0.0002	0.0005	0.0001
Decay rate (Differentiation)	0.0005	0.0001	0.0002	0.0005	0.0001
Decay rate (Advection)	0.0005	0.0005	-	-	-
Decay rate (Burgers')	0.0001	0.0001	-	-	-

Table 5: Setting of the Hyperparameter

C.1 IDENTITY

As in the text, we developed an experiment to learn the identity operator for the 20th-order Chebyshev polynomials (Figure 6). Note that the absolute value of the coefficients of all orders is less than or equal to $1/4$. We discretize the domain $[-1, 1]$ with spatial resolution 50. In the experiments provided in the text, we construct all of the neural networks with *tanh*. The experimental results under various conditions were included in a Table 4 by modifying the network structure, the activation function, and the number of sensor points. We use 1000 training pairs and 200 pairs to validate our experiments. The batch size during the training is determined to be 5000 which is one-tenth the size of the entire data set.

C.2 DIFFERENTIATION

In this experiment, we set functions whose derivatives are 20th-order Chebyshev polynomials as input functions. As above, all coefficients of the Chebyshev polynomial are between $-1/4$ and $1/4$. We use the 100 uniform grid on the domain [-1, 1]. The number of training samples and test samples are 1000 and 200 respectively. We use batch size as 10000 which is one-tenth the size($100000 = 100 \cdot 1000$) of the entire data set.

C.3 ADVECTION EQUATION

We consider the linear advection equation on the torus $\mathbb{T} := \mathbb{R}/\mathbb{Z}$ as follows:

$$\begin{cases} \frac{\partial w}{\partial t} + c \cdot \frac{\partial w}{\partial x} = 0. \\ w(x, 0) = w_0(x), \quad x \in \mathbb{T}, \end{cases} \tag{11}$$

where c is a constant which denotes the propagation speed of w. By the construction of the domain as $\mathbb{T}$ we implicitly assume the periodic boundary condition is assumed. In this paper, we consider the case when $c = 1$. Our goal is to learn the operator which maps $w_0(x)(= w(0, x))$ to $w(0.5, x)$. We use the same data as in Lu et al. (2022). We discretize the domain $[0, 1]$ with the spatial resolution 40. The number of training samples and test samples is 1000 and 200, respectively. We use the full batch for training so that the size of batch is $40 \cdot 1000 = 40000$.

C.4 BURGERS' EQUATION

We consider the 1D Burgers' equation which describes the movement of the viscous fluid

$$\begin{cases} \frac{\partial w}{\partial t} = -u \cdot \frac{\partial w}{\partial x} + \nu \frac{\partial^2 w}{\partial x^2}, & (x, t) \in (0, 1) \times (0, 1], \\ w(x, 0) = w_0(x), & x \in (0, 1), \end{cases} \tag{12}$$

where w_0 is the initial state and ν is the viscosity. Out goal is to learn the nonlinear **solution operator of the Burgers' equation** which is a mapping from the initial state $w_0(x)(= w(x, 0))$ to the solution $w(x, 1)$ at $t = 1$. We use the same data of Burgers' equation provided in Li et al. (2020a). The initial state $w_0(x)$ is generated from the Gaussian random field $\mathcal{N}(0, 5^4(-\Delta + 25I)^{-2})$

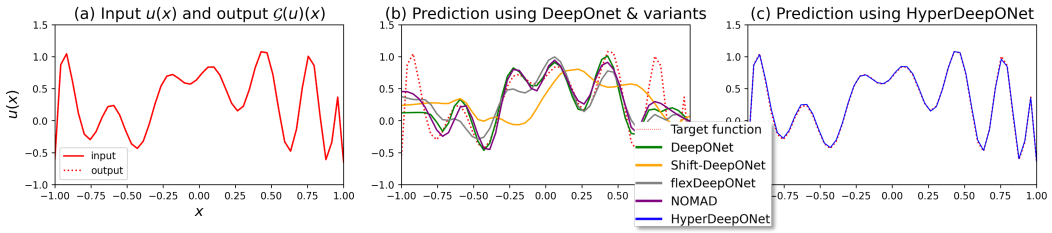

Figure 6: One test data example of differentiation operator problem.

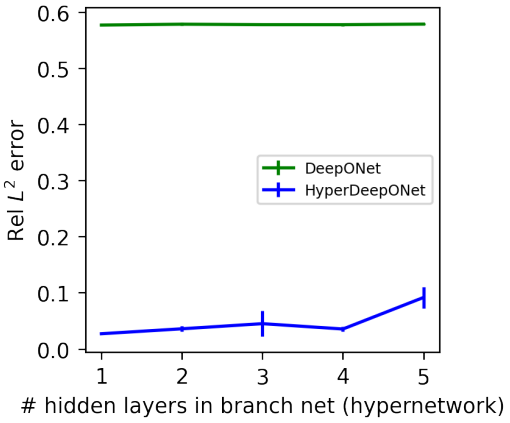

Figure 7: Varying the number of layers in branch net and hypernetwork in DeepONet and HyperDeepONet for identity operator problem.

with the periodic boundary conditions. The split step method and the fine forward Euler method were employed to generate a solution at $t = 1$. We set viscosity ν and the spatial resolution to 0.1 and $2^7 = 128$, respectively. The size of the training sample and test sample we used are 1000 and 200, respectively.

We take the ReLU activation function and InverseTimeDecay scheduler to conduct the experiment in the same setting as in Lu et al. (2022). For a fair comparison, all experiments on DeepONet retained the hyperparameter values used in that. We use the full batch so that the batch size is 1280000.

C.5 Additional Experiments

Figure 7 and Figure 8 compares the relative L^2 error of the test set for DeepONet and HyperDeepONet by varying the number of layers in the branch net and hypernetwork while maintaining the same small target network. It shows that the test error is not reduced despite the branch net becoming larger. The HyperDeepONet approximates operator with high accuracy in all the cases of a different number of layers. The HyperDeepONet has the same tendency when the structure of the small target network, number of sensors m, or the activation functions are changed.

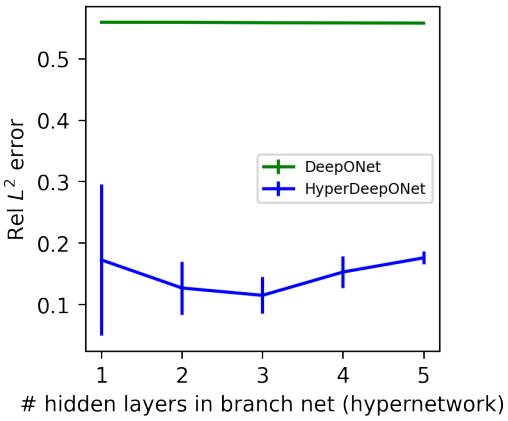

Figure 8: Varying the number of layers in branch net and hypernetwork in DeepONet and Hyper-DeepONet for differentiation operator problem.