How important are specialized transforms in Neural Operators?

Ritam Majumdar [1] Shirish Karande [1] Lovekesh Vig [1]

Abstract

Transform-based Neural Operators like Fourier Neural Operators and Wavelet Neural Operators have received a lot of attention for their potential to provide fast solutions for systems of Partial Differential Equations. In this work, we investigate what could be the cost in performance, if all the transform layers are replaced by learnable linear layers. We observe that linear layers suffice to provide performance comparable to best-known transform-based layers and seem to do so at possibly a compute time advantage as well. We believe that this observation can have significant implications for future work on Neural Operators.

1. Introduction

The prediction and analysis of physical systems with the help of computational forward simulations have emerged as a crucial instrument in numerous industrial domains. These physical systems, constrained by an array of Partial Differential Equations (PDEs) along with initial and boundary values, call for precise and scalable simulation methodologies. Among the diverse range of techniques available, transform-based Neural Operators such as Fourier Neural Operator (FNO) (Li et al., 2021) and Wavelet Neural Operator (WNO) (Tripura & Chakraborty, 2023) have attracted considerable attention for their aptitude in delivering rapid, scale-free simulations.

One of the salient features of these transform-based operators is their ability to transform the input data into a domain where the data's inherent features become more apparent, thus making it easier to analyze and perform computations. Traditionally, the efficacy of such transformations is highly reliant on the intrinsic nature of the data, leading to an optimal choice of transform becoming a significant determinant of the overall performance of the system. However, despite the growing interest and success of these methods, a comprehensive investigation into the necessity of these sophisticated transforms, and their effect on system performance is lacking. Tremendous human effort is involved in the study of characteristics of the industrial data and making the appropriate choice of transform.

The universal approximation theorem (Nishijima, 2021) states that a neural network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets, given the activation function is a non-constant, bounded, and monotonically increasing continuous function. This includes specialized transforms like Fourier, Laplace, and wavelet transforms. It has been demonstrated, Neural Networks can successfully compute Discrete Fourier Transforms (Velik, 2008). As specific transforms lead to distinct transformation of the PDE-induced input data, the choice of transform becomes critical to the generalization capabilities of the neural operator. While there is significant effort involved in designing the correct transform for given input data and PDEs, most considered transforms are linear in nature, raising the question: Are these well-known pre-defined transforms essential, or could they be replaced by a parameterized linear layer which can learn an adaptive transform for the specific problem under consideration and deliver comparable performance? This study probes this pertinent question and explores the performance implications when replacing all transform layers in the network with basic learnable linear layers.

We hypothesize that learnable linear transformations suffice in terms of generalization and computational efficacy. The initial results of our exploration are surprising, pointing towards linear layers exhibiting performance parity with the best-known transform-based operators, seemingly with a computational advantage. Such a revelation prompts further questioning of the underlying significance of transform-based operators in the Neural Operator realm. This study serves as an exploration of this notion, providing empirical evidence that challenges the status quo of existing Neural Operator models. The primary objective is to delve into the trade-offs between using a pre-defined linear transform and using a parameterized linear transform which adapts to the

[1] Tata Consultancy Services Research, India. Correspondence to: Ritam Majumdar <ritam.majumdar@tcs.com>, Shirish Karande <shirish.karande@tcs.com>, Lovekesh Vig <lovekesh.vig@tcs.com>.

Accepted after peer-review at the 1st workshop on Synergy of Scientific and Machine Learning Modeling, SynS & ML ICML, Honolulu, Hawaii, USA. July, 2023. Copyright 2023 by the author(s).

PDE problem in-hand. We analyze the performance factor and the computational cost as criterion for our trade-offs. By doing this, we aim to shed light on the potential of learnable linear transformations and their capability to generalize over varying PDE systems of different complexity, without having to handcraft specialized transforms and save human effort.

2. Related Work

Traditional approaches to forward simulation of Partial Differential Equations (PDEs) primarily involve numerical methods such as Finite Difference, Finite Volume, and Finite Element Methods. These approaches discretize the problem domain into a grid or mesh and approximate the derivatives in the PDEs using this discretized representation. However, these methods are computationally demanding as they often require small time steps for stability, particularly in high-dimensional spaces. Furthermore, they face challenges in handling complex geometries, multiple scales, and non-linearities inherent in many physical systems. In the recent past, Neural network based techniques like Physics-Informed Neural Networks (PINNs), (Raissi et al., 2019), Deep Galerkin Methods (DGM) (Sirignano & Spiliopoulos, 2018), Deep BSDE solvers (Wang & Ni, 2022) have been proposed as surrogates for solutions of PDEs. DGM is a neural network-based approach for solving PDEs that formulates the solution as a continuous optimization problem, handling high-dimensional problems more effectively than traditional methods. Deep BSDE Solvers solve high-dimensional PDEs by representing them as backward stochastic differential equations. These methods however, don't generalize to changing initial-boundary conditions and requires retraining for every unique initial-boundary conditions and parameterized PDE coefficients.

Operator-based methods circumvent around this issue by aiming to learn a mapping or an operator, that transforms a given set of input conditions (parameterized initial-boundary conditions) to the corresponding solutions. Given a new-instance of a PDE system, instead of developing solutions from scratch, a pre-trained operator can simply infer the solutions in real time, reducing computational cost. Some widely-used operators for learning systems of parameterized PDEs are DeepONets (Lu et al., 2021), Fourier Neural Operators (FNO) (Li et al., 2021), Wavelet Neural Operators (WNO) (Tripura & Chakraborty, 2023), Laplace Neural Operators (LNO) (Cao et al., 2023), just to name a few. DeepONets, rooted in the universal approximation theorem, consist of a branch network processing function parameters and a trunk network handling function variables. The final output, integration of the two network outputs, effectively approximate complex operators. Special linear-transforms like Fourier, Wavelet and Laplace have been incorporated into Neural transforms as tools for feature extraction and data compression to improve the representation-capacity of Neural Operators. FNO consists of a Fourier transform to convert the input data into Fourier space, followed by multiple layers of 1D convolutions and nonlinearities to learn the mapping, and then an inverse Fourier transform to convert the output back to the original space. WNO takes Wavelet transforms instead of Fourier transforms and is better equipped to handle non-periodic and irregular domains due to the superiority of wavelets in time-frequency localization. LNO uses the Laplace transform to decompose the input space and can handle non-periodic signals, take transient responses into account, and converge exponentially.

3. Methodology

We describe the architecture of our Neural Operator in Figure 1. The architecture resembles the original architectures of FNO and WNO, the only difference being, we replace the one-dimensional Fourier (Wavelet) and Inverse Fourier (Wavelet) transforms with learnable linear transforms M and N respectively. Our objective is to learn a operator G parameterized by θ mapping $G_\theta : A \rightarrow U$, wherein $A = A(D; R^{d_a})$ and $U = U(D; R^{d_u})$ are input and output Banach spaces and $D \subset R^d$ be a bounded, open set. $a(x) \in R^{d_a}$ and $u(x) \in R^{d_u}$ refer to the input and output signals sampled i.i.d from A and U respectively.

The neural operator, similar to (Li et al., 2021) is formulated as an iterative architecture $v_0 \rightarrow v_1 \rightarrow ... \rightarrow v_T$ where v_j with $j \in [0, T-1]$ is a sequence of functions each taking values in R^{d_v}. R^{d_v} is the dimension of the middle projection layers. As shown in Figure 1, the input $a \in A$ is first lifted to a higher dimensional representation

$$v_0(x) = P(a(x)) \tag{1}$$

by the local transformation $P : R^{d_a} \rightarrow R^{d_v}$ which is parameterized by a shallow neural network. Then we apply several iterations of updates $v_t \rightarrow v_{t+1}$ defined as the composition of a non-local integral operator K and a local, non-linear activation function σ. The projection Q transforms the projection back to the dimension of the output space $Q : R^{d_v} \rightarrow R^{d_u}$, while the output $u(x)$ is defined as

$$u(x) = Q(v_T(x)) \tag{2}$$

The update $v_t \rightarrow v_{t+1}$ is defined as:

$$v_{t+1}(x) = \sigma(W.v_t(x) + K(a;\phi)v_t(x)) \tag{3}$$

$$K(a;\phi)v_t(x) = \int_D \kappa(x, y, a(x), a(y); \phi)v_t(y)dy \tag{4}$$

$$\kappa(x, y, a(x), a(y); \phi)v_t(y) = N(R \cdot M(v_t(y))) \tag{5}$$

Here $\kappa(\phi) : R^{2(d+d_a)} \rightarrow R^{d_v X d_v}$ is a kernel integral operator represented by a neural network parameterized by

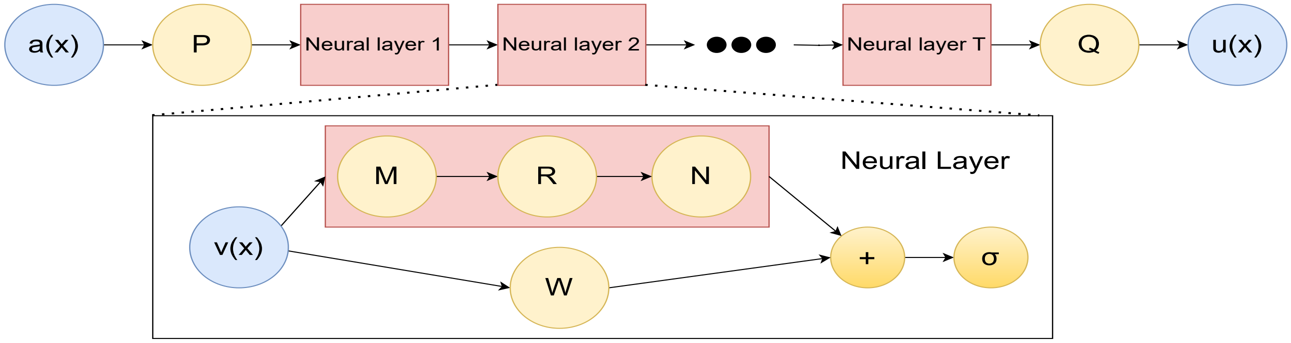

Figure 1. Model architecture of Neural Operator

ϕ. $W : R^{d_v} \rightarrow R^{d_v}$ is a linear transformation, and σ is a non-linear activation function.

An n-dimensional Fourier transform can be broken down into n 1D-Fourier Transforms along each dimension (Tolimieri et al., 2012). Given an n-dimensional tensor of shape $R^{d_1} \times R^{d_2} \times \cdots \times R^{d_n}$, the n-dimensional Fourier transform M of the tensor is defined as:

$$M(R^{d_1} \times \cdots \times R^{d_n}) = F(R^{d_1}) \times \cdots \times F(R^{d_n}) \quad (6)$$

where F refers to the one-dimensional Fourier transform. The resulting tensor is of the shape $C^{k_1} \times C^{k_2} \times \cdots \times C^{k_n}$ where $k_1, k_2, \ldots, k_n$ are the chosen number of Fourier modes in the low-frequency regime, and C refers to the complex domain. We replace the one-dimensional Fourier transform F with a learnable linear layer L_f and L_b where L_f stands for linear transform in forward space and L_b refers to linear transform in the inverse space. We define our n-dimensional linear transformation M as follows:

$$M(R^{d_1} \times \cdots \times R^{d_n}) = L_{f_1}(R^{d_1}) \times \cdots \times L_{f_n}(R^{d_n}) \quad (7)$$

where L_{f_i} where $i \in [1, n]$ represents learnable linear transforms (parameterized matrices of shape $R^{d_i} \times R^{k_i}$) along each dimension. In figure 1, R refers to the tensor multiplication in the transformed space. In the Fourier space, R takes in $C^{dv} \times C^{k_1} \times \cdots \times C^{k_n}$ dimensional tensor as input, takes an element wise tensor multiplication with a parameterized tensor of shape $C^{dv^2} \times C^{k_1} \times \cdots \times C^{k_n}$ to output a $C^{dv} \times C^{k_1} \times \cdots \times C^{k_n}$ dimensional tensor. In our case, we stay in the real domain, and our input and output tensors are of the shape $R^{dv} \times R^{k_1} \times \cdots \times R^{k_n}$, while the parameterized tensor is of the shape $R^{dv^2} \times R^{k_1} \times \cdots \times R^{k_n}$.

The n-dimensional inverse transform N is defined as

$$N(C^{k_1} \times \cdots \times C^{k_n}) = F^{-1}(C^{k_1}) \times \cdots \times F^{-1}(C^{k_n}) \quad (8)$$

where F^{-1} refers to the one-dimensional inverse Fourier transform. The resulting tensor is of the shape $R^{d_1} \times R^{d_2} \times \cdots \times R^{d_n}$, the original dimensions of the tensor. In our case, we replace F^{-1} with L_b (parameterized matrices of shape $R^{k_i} \times R^{d_i}$) and define the inverse transform N to be

$$N(R^{k_1} \times \cdots \times R^{k_n}) = L_{b_1}(R^{k_1}) \times \cdots \times L_{b_n}(R^{k_n}) \quad (9)$$

The M-R-N operation is repeated iteratively for T blocks, before being projected to the output space $u(x)$ using projection Q defined in Eqn 2. The code and datasets for the paper are available at https://github.com/Ritam-M/LearnableTransformsNO

4. PDE information

In order to make a fair comparison, we consider the same class of examples used in the original papers of (Li et al., 2021; Tripura & Chakraborty, 2023).

Burgers' Equation We consider the 1D Burgers' equation, which is a non-linear PDE with various applications, including modeling the flow of a viscous fluid. The 1D Burgers' equation takes the form:

$$\begin{aligned} \partial_t u(x,t) + \partial_x \left(u^2(x,t)/2\right) &= \nu \partial_{xx} u(x,t), \quad && t \in (0,1] \\ u(x,0) &= u_0(x), && x \in (0,1) \end{aligned}$$

where u_0 is the initial condition and ν is the viscosity coefficient. We aim to learn the operator mapping the initial condition to the solution.

Wave advection equation: The wave advection equation is a hyperbolic PDE and primarily describes the solution of a scalar under some known velocity field. The advection equation with periodic boundary condition is given by, $\partial_t u(x,t) + v\partial_x u(x,t) = 0$ and $u(x-\pi,t) = u(x+\pi,t)$. $v \in \mathbb{R} > 0$ represents the speed of the flow. The initial condition is given by, $u(x,0) = h_{\{c-\frac{\omega}{2}, c+\frac{\omega}{2}\}} + \sqrt{\max\left(h^2 - (a(x-c))^2, 0\right)}$. where the variables ω and h represents the width and height of the square wave, respectively and the wave is centered at $x = c$. For $v = 1$, the solution to the advection equation is given as, $u(x,t) = u_0(x-t)$. The objective is to learn the operator, that learns the mapping $u_0(x) \mapsto u(x,t)$, to a later time t.

2D Darcy Flow 2D Darcy Flow is a linear second-order elliptic PDE. We consider a steady-state flow in a unit box, given by:

$$\begin{aligned} -\nabla \cdot (a(x)\nabla u(x)) &= f(x), \quad x \in (0,1)^2 \\ u(x) &= 0, \qquad x \in \partial(0,1)^2 \end{aligned}$$

with a Dirichlet boundary where a is the diffusion coefficient and $f = 1$ is the forcing function. The objective is to learn the operator mapping the diffusion coefficient to the solution.

2D Darcy flow equation with a notch in triangular domain: This example is a special case of the 2D-Darcy problem in the triangular domain with a notch in the flow. The boundary conditions for the triangular domain are generated using the following Gaussian process (GP), $u(x) \sim \mathrm{GP}\left(0, \mathcal{K}\left(x, x'\right)\right)$, with the kernel $\mathcal{K}\left(x, x'\right) = \exp\left(-\left(x-x'\right)^2 / 2l^2\right); l = 0.2; x, x' \in [0,1]$. The permeability and the forcing function $f(x,y)$ is 0.1 and -1 respectively. The objective is to learn the operator mapping the boundary conditions to the pressure field, given by, $u(x,y)|_{\partial\omega} \mapsto u(x,y)$.

Navier-Stokes Equation The two dimensional temporally varying Navier-Stokes equation for a viscous, incompressible fluid on the unit torus is given by:

$$\begin{aligned} \partial_t w(x,t) + u(x,t) \cdot \nabla w(x,t) &= \nu \Delta w(x,t) + f(x), \\ \nabla \cdot u(x,t) &= 0, \quad x \in (0,1)^2 \\ w(x,0) &= w_0(x), \quad t \in (0,T] \end{aligned}$$

where $w = \nabla \times u$ is the vorticity, w_0 is the initial vorticity, $\nu \in \mathbb{R}_+$is the viscosity coefficient, and $f(x) = 0.1\left(\sin\left(2\pi\left(x_1+x_2\right)\right) + \cos\left(2\pi\left(x_1+x_2\right)\right)\right)$ is the forcing function. The objective is to learn the operator mapping the vorticity up to time 10 to the vorticity up to some later time.

Kolmogorov Flow The high-frequency Kolmogorov flow is governed by the 2D Navier-Stokes equation, defined in the earlier section. The forcing $f(x) = \sin\left(nx_2\right)\hat{x}_1$, where x_1 is the unit vector and a larger domain size [0, 2π]. The objective is to learn the evolution operator mapping the next time-step from the previous time-step.

5. Training and Hyperparameter Details

In order to have a fair comparison and benchmark in an unbiased manner, we consider the original PDE examples and datasets used in the original papers of (Li et al., 2021; Tripura & Chakraborty, 2023), same set of hyperparameters (Same number of layers, same spatial and temporal grid dimensions of PDE examples, same transform dimensions, optimizers, schedulers, batch sizes of examples, etc). We construct our neural operator by stacking four integral operator layers with ReLU as the activation function. We use an Adam optimizer for 500 epochs with an initial learning rate of $1e^{-3}$ that is halved every 100 epochs. All experiments were conducted on Nvidia P100 GPU with 16 GB GPU Memory and 1.32 GHz GPU Memory clock using Pytorch framework. In Burger's, Wave-Advection, 2D-Darcy Rectangular and Triangular domains and Navier-Stokes, we consider 500 training examples and 200 test-examples, while in Darcy-notch we consider 1900 and 100 train and test examples respectively. Our dimensions of linear transforms are same as the number of modes in Fourier transform used in (Li et al., 2021). For one-dimensional examples, our final transform-dimension is 64, while for two-dimensional and three-dimensional examples, our transform-dimension is 32. Across all examples, we use % relative L2-error as our metric.

6. Results and Discussion

In this section, we discuss the results of our experiments. We divide our analysis into two sections, Table 1 consists of the lower-dimensional PDEs, while Table 2 consists of higher-dimensional PDEs. In all examples in Table 1, the experiments are trained and evaluated on the same resolution. In Table 2, the property column refers to viscosity in Navier-Stokes and Reynold's number (Re) in Kolmogorov flow. The complexity of the problem is increased by the decrease in viscosity in Navier-Stokes and higher Re in Kolmogorov flow, which leads to turbulent behavior.

PDE	Resolution	Ours	FNO	WNO
Burger's	1024	0.18	**0.17**	0.20
	2048	**0.18**	0.19	0.25
	4096	**0.19**	**0.19**	0.24
Advection	40×40	0.61	45.14	0.59
Darcy-flow	106×106	**0.27**	**0.27**	0.34
	211×211	**0.26**	0.27	0.35
	421×421	**0.28**	**0.28**	0.35
Darcy-Notch	100×100	**0.69**	-	0.74

Table 1. Test % relative L2-error for lower dimensional PDEs

From Table 1, we observe our Neural Operator with a learnable linear transform performs competitively against the best-performing architecture between FNO and WNO. FNO struggles on the examples of Wave-Advection and triangular domain Darcy flow with a notch, due to the inability of Fourier transform to handle irregular geometry. Our neural linear transform is capable of learning the transform suitable for irregular geometries, indicated by the superior performance over FNOs and comparable performance with WNOs. Numerically, learnable linear-transform based Neural operators are within 0.01 to 0.02 points (outperforms Darcy-triangular-notch by 0.05 points) against the best-performing transform between FNO and WNO across all examples and resolutions.

PDE	Property	Ours	FNO	WNO
Navier Stokes	$1e^{-3}$	3.03	**3.02**	4.54
		3.24	**3.22**	3.74
	$1e^{-4}$	**7.07**	7.16	9.39
		7.12	7.33	9.16
	$1e^{-5}$	15.27	**14.85**	18.57
		17.19	17.32	24.15
Kolmogorov	100	4.02	**3.91**	5.53
	400	9.66	**9.64**	10.56
	500	13.36	**13.35**	15.54

Table 2. Test % relative L2-error for higher dimensional PDEs

Table 2 represents the results on higher-dimensional PDEs, the more difficult examples in our study. In Navier-Stokes, the first row for each viscosity refers to modeling the two-dimensional spatial component using a 2D-Neural Operator and the temporal component using an LSTM , while the second row directly models the spatial and temporal components together using a 3D-Neural Operator. The resolution is fixed to 64×64 for both training and testing. We observe, across all examples, taking a learnable linear transform performs comparably against pre-defined Fourier and Wavelet transforms across all examples. Thus, parameterized transforms have the ability to scale to the complexity of the PDE and can adapt to learn a transform best suited to the PDE problem under consideration. In the Kolmogorov flow examples, learnable transforms based NOs are within 0.1 points against FNOs, while in the Navier-Stokes examples, learnable transforms are within 0.42 points against FNOs, while being the best performing architecture among 3 out of 6 scenarios.

PDE	Ours	FNO	WNO
1D-Burgers	2.52	2.75	5.58
Advection	2.72	2.86	6.03
2D-Darcy	13.55	13.84	38.34
Navier-Stokes (2D+time)	67.41	130	221
Navier-Stokes (3D)	22.14	46.33	196
Kolmogorov	16.35	19.56	47.75

Table 3. Training times of operators (per epoch) in seconds

We refer to the training time per epoch for each of the operators in Table 3. We train all the operators for 500 epochs (the original hyperparameter considered in FNO and WNO papers) to keep the benchmarking consistent. Learnable linear transform is significantly faster to train than FNO on complex examples and WNO across all examples.

We investigate the reasons behind this, and make a comparsion of the parameter count of differentiating blocks of each operator in Table 4. All notations in Table 4 are consistent with those defined in the methodology section of the paper. M, N refers to the forward and inverse transforms, while R refers to the tensor multiplication. n refers to number of dimensions, while d_i, k_i refer to input and output dimensions before and after computing a forward transform, respectively. While FNO/WNO have no extra parameters while undergoing the forward, inverse transforms, the tensor multiplication R takes place in the complex space, contributing to an additional ${d_v}^2 \prod_{i=1}^{n} k_i$ parameters as against learnable linear-layer based transforms. In all scenarios, FNO/WNO architectures have an additional ${d_v}^2 \prod_{i=1}^{n} k_i - 2 \sum_{i=1}^{n} d_i k_i$ parameters per block. The training time of a neural network architecture is proportional to the number of parameters. Since linear-transform based Neural Operators end up having a lower number of parameters, their training time is faster as compared to FNO/WNO.

	M	R	N
FNO/WNO	0	$2{d_v}^2 \prod_{i=1}^{n} k_i$	0
Ours	$\sum_{i=1}^{n} d_i k_i$	${d_v}^2 \prod_{i=1}^{n} k_i$	$\sum_{i=1}^{n} d_i k_i$

Table 4. Parameter size comparison between Neural Operators

7. Conclusion

We replace specialized transforms like Fourier and Wavelet in Neural Operators with a learnable linear transform. The learnable linear transforms generalize well and perform competitively on a wide class of examples like low-viscous Navier Stokes and high Re Kolmogorov flow. Additionally, learnable linear transforms generalize on Wave-Advection and irregular domain Darcy flows which Fourier transforms fail to do and is computationally faster than Wavelet transform-based operators. Thus, a parameterized linear layer can replace specialized transforms and saves human effort for Neural Operator architecture design. In future, we seek to investigate the learnt transforms and their resemblance with known transforms. Additionally, theoretical studies on convergence rates and error-bounds for parameterized linear layer-based Neural Operators remain.

8. Broader impact

The findings of this work hold the potential to streamline computational simulations across various industries, from engineering and meteorology to physics, by utilizing learnable linear transformations instead of pre-defined transform-based Neural Operators. This may foster faster, more accessible and more generalizable simulations, expanding their applicability even to higher complexity entities.

References

Cao, Q., Goswami, S., and Karniadakis, G. E. Lno: Laplace neural operator for solving differential equations, 2023.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. Fourier neural operator for parametric partial differential equations, 2021.

Lu, L., Jin, P., Pang, G., Zhang, Z., and Karniadakis, G. E. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, mar 2021. doi: 10.1038/s42256-021-00302-5. URL https://doi.org/10.1038%2Fs42256-021-00302-5.

Nishijima, T. Universal approximation theorem for neural networks, 2021.

Raissi, M., Perdikaris, P., and Karniadakis, G. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.10.045. URL https://www.sciencedirect.com/science/article/pii/S0021999118307125.

Sirignano, J. and Spiliopoulos, K. DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, dec 2018. doi: 10.1016/j.jcp.2018.08.029. URL https://doi.org/10.1016%2Fj.jcp.2018.08.029.

Tolimieri, R., An, M., and Lu, C. *Mathematics of multidimensional Fourier transform algorithms*. Springer Science & Business Media, 2012.

Tripura, T. and Chakraborty, S. Wavelet neural operator for solving parametric partial differential equations in computational mechanics problems. *Computer Methods in Applied Mechanics and Engineering*, 404:115783, 2023. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2022.115783. URL https://www.sciencedirect.com/science/article/pii/S0045782522007393.

Velik, R. Discrete fourier transform computation using neural networks. In *2008 International Conference on Computational Intelligence and Security*, volume 1, pp. 120–123, 2008. doi: 10.1109/CIS.2008.36.

Wang, Y. and Ni, Y.-H. Deep bsde-ml learning and its application to model-free optimal control, 2022.