International Journal of Heat and Mass Transfer 203 (2023) 123809

Contents lists available at ScienceDirect

International Journal of Heat and Mass Transfer

journal homepage: www.elsevier.com/locate/hmt

Data-driven and physics-informed deep learning operators for solution of heat conduction equation with parametric heat source

Seid Koric[a,b,*], Diab W. Abueidda[a]

[a] National Center for Supercomputing Applications, University of Illinois, Urbana, IL 61801, USA
[b] Mechanical Science and Engineering, University of Illinois, Urbana, IL 61801, USA

基于DeepONet，以数据驱动和物理信息融合两种方式分别对以随机参数源项为输入的二维热传导方程进行了求解。与数据驱动的DeepONet相比，物理信息融合的DeepONet训练时间消耗更大，且对源项的维度诅咒现象更敏感，导致其预测精度的下降，然而在离线(数据生成)阶段后者所花费的时间明显减少，并且能够在没有生成或观察到任何解目标的情况下学习参数热传导方程的解算子。

ARTICLE INFO

Article history:
Received 11 October 2022
Revised 12 December 2022
Accepted 23 December 2022
Available online 31 December 2022

Keywords:
DeepONet
Heat (Poisson's) equation
Multi-dimensional parameter
Deep learning

ABSTRACT

Deep neural networks as universal approximators of partial differential equations (PDEs) have attracted attention in numerous scientific and technical circles with the introduction of Physics-informed Neural Networks (PINNs). However, in most existing approaches, PINN can only provide solutions for defined input parameters, such as source terms, loads, boundaries, and initial conditions. Any modification in such parameters necessitates retraining or transfer learning. Classical numerical techniques are no exception, as each new input parameter value necessitates a new independent simulation. Unlike PINNs, which approximate solution functions, DeepONet approximates linear and nonlinear PDE solution operators by using parametric functions (infinite-dimensional objects) as inputs and mapping them to different PDE solution function output spaces. We devise, apply, and compare data-driven and physics-informed DeepONet models to solve the heat conduction (Poisson's) equation, one of the most common PDEs in science and engineering, using the variable and spatially multi-dimensional source term as its parameter. We provide novel computational insights into the DeepONet learning process of PDE solution with spatially multi-dimensional parametric input functions. We also show that, after being adequately trained, the proposed frameworks can reliably and almost instantly predict the parametric solution while being orders of magnitude faster than classical numerical solvers and without any additional training.

1. Introduction

Machine learning (ML) has grown significantly lately, resulting in a wide range of applications in areas such as medical diagnosis, speech and image recognition, document classification, bioinformatics, and autonomous driving, to name a few. Deep Learning, a subset of machine learning inspired by the biological structure and function of the brain, has sparked a lot of interest in the fields of computational engineering and physics. Numerous data-driven surrogate deep learning models have been trained to learn and quickly inference models of heat conduction [1], temperature in data centers [2], topologically optimized materials and structures [3], nonlinear material responses such as plasticity, thermo-plasticity, thermo-viscoplasticity [4–6], and a variety of other applications. The complexity of the problem dictates the size of the dataset needed to train a deep learning model; for example, complicated problems demand large datasets to develop surrogate models capable of correctly predicting the solution. As a result, to produce the data needed to train such deep-learning-based surrogate models, a traditional discretization strategy, such as the finite element method (FEM), is typically required, and the whole process is usually time-consuming.

Deep neural networks have been also presented as a way to solve PDEs governing many physical phenomena with conservation and constitutive rules [7–11]. These physics-informed neural networks (PINNs) are designed to handle problems with little or no training data. The PINN method is augmented with physical principles regulating the situation at hand to compensate for the absence of data. Physics principles and data are also used combined to provide solutions to problems that are incomplete or ill-posed and cannot be solved using traditional numerical approaches [12]. Although trained PINNs are orders of magnitude quicker than traditional numerical PDE solvers, they have limited flexibility in terms of parameters such as source terms, material properties, loads, and boundary and initial conditions. Any change in such parameters necessitates retraining or transfer learning. The traditional numerical approaches are similar, and each new input parameter value necessitates a new simulation.

To solve this challenge, Lu et al. [13] recently developed DeepONet, a novel operator learning architecture motivated by the

* Corresponding author at: National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, 1205 W. Clark St, Urbana, IL 61801, USA.
E-mail address: koric@illinois.com (S. Koric).

https://doi.org/10.1016/j.ijheatmasstransfer.2022.123809
0017-9310/

universal approximation theorem for operators [14]. DeepONet effectively mapped between unseen parametric functions and solution spaces for a few linear and nonlinear PDEs in that seminal work, in addition to learning explicit operators such as integrals. This provided an effective new technique to solve parametric and stochastic PDEs. Wang et al. [15] have enhanced the DeepONet formulation by the information from governing PDE in so-called physics-informed DeepONet and reported increased prediction accuracy and data handling efficiency but under the higher computational cost for training. Both of these works have solved mostly spatially one-dimensional PDEs. We extended and compared these DeepONet formulations to solve the heat conduction equation with spatially 2D parametric source term, one of the most solved PDEs using traditional numerical approaches in science and engineering research.

2. Formulations

2.1. Heat conduction (Poisson's) equation

The Poisson's equation governs physical phenomena such as heat conduction with a moving heat source (laser head) in additive manufacturing, potential flow and pressure solvers in Computational Fluid Dynamics (CFD), electrostatics, gravity in astronomy, and molecular dynamics, to name a few. The Poisson's equation is an inhomogeneous elliptic PDE, with the inhomogeneous part *u(x,y)* representing the source of the field. In many engineering and scientific applications, a Poisson's equation with the same boundary conditions but different source terms are frequently solved, which may consume a significant fraction of the entire application solution time, even with state-of-the-art numerical methods and computer technology. Moreover, many iterative computationally processes used in thermal parametric studies, design, sensitivity analysis, uncertainty quantification, and optimization of classical and advanced manufacturing processes require a vast number of forward functional evaluations by sampling the parametric space and calculating temperature solutions to obtain converging statistics. These evaluations are traditionally performed by classical numerical methods such as finite elements. For high-fidelity thermo-mechanical models, these simulations are often prohibitively computationally expensive in the design and optimization loops, particularly with traditional sampling methods such as Monte Carlo or Latin Hypercube. Instead of numerical forward evolutions, the Deep Operator Neural Network, which in addition to the solution, can also learn its parameters, is a natural choice to be used as a surrogate model for instant functional evaluation. In the rest of this work, we will investigate how the data-driven and physics-informed DeepONet formulations can learn the solution of a parametric heat conduction equation. We also validate the DeepONet predictions and compare both approaches' computational performance and accuracy on the latest high-performance computing resources.

We employ the heat conduction (Poisson's) equation, Eq. (1), on a unit square domain with zero Dirichlet boundary conditions as a basic reference equation to solve with DeepONet throughout this paper.

$$\begin{aligned} &k_x \tfrac{\partial^2 s}{\partial x^2} + k_y \tfrac{\partial^2 s}{\partial y^2} + u(x,y) = 0 (x,y) \in [0,1]x[0,1] \\ &BC: s(x,0) = s(x,1) = s(0,y) = s(1,y) = 0 \end{aligned} \tag{1}$$

Where *s(x,y)* is an unknown function of two independent variables, *x* and *y*, *u(x,y)* is a source term function, and $k_x = k_y = 0.01$ is the diffusion coefficient. It is important to note that the data-driven and physics-informed DeepONet methodologies developed in this work for parametric heat conduction are not limited to 2D conditions and regular rectangular geometries. With further code modification beyond the scope of the current work, they can be implemented to various loading, boundary conditions, material properties, and even irregular 2 and 3D geometries.

2.2. Data-driven DeepONet

DeepONet was proposed in [13] as a method for learning nonlinear operators by mapping input functions into matching output functions. We illustrate how DeepONet can be used to tackle the challenge of learning the parametric PDE solution operator for the heat conduction Eq. (1), with the source term u being a parametric function that can take on a wide range of values. In an infinite functional U space, $u \in U$ represents the source term parameters (i.e., input functions) and $s \in S$ refers to the PDE's unknown solutions in the functional space S. We assume that there is a single solution $s = s(u)$ in S to the Eq. (1) for every u in U, which is also subject to the boundary conditions BC. As a result, the mapping solution operator $G: U \rightarrow S$ may be defined as:

$$G(u) = s(u) \tag{2}$$

Instead of only a collection of points $\bar{y}$ on a domain, the DeepONet considers both u and $\bar{y}$, and predicts $G(u)$ by combining them, as shown in Fig. 1.

Because it can accept a source term u function as an input variable, this network is significantly more capable than other PINN networks. Every $G(u)$ computation at a point $\bar{y}$ generates a new solution point, which can be written as $G(u)(\bar{y})$. Please keep in mind that, in this case, $\bar{y}$ represents coordinate points in the 2D domain (a unit square) where the network predicts the solution of the parametric heat conduction equation.

Every input function u is specified on m discrete points in the domain known as input sensors, and the output solution can be assessed on P output sensor locations. The unstacked DeepONet is employed, which is made up of two independent fully connected neural networks termed branch and trunk. According to Lu et al. [13], the unstacked DeepONet achieves better outcomes than the stacked DeepONet in balancing training between u and $\bar{y}$ inputs while consuming fewer computing resources. The branch and trunk networks employ the hyperbolic tangent activation function (Tanh) to model intricate functional linkages between inputs and outputs. The branch network receives u at each of its m locations and outputs q intermediate outputs b_k. The trunk network receives $\bar{y}$ as an input and outputs q intermediate outputs to t_k. In Eq. (3), a dot product is used to combine the intermediate outputs, resulting in a DeepONet solution operator prediction $\hat{G}(u)(\bar{y})$.

$$\hat{G}(u)(\bar{y}) = \sum_{i=1}^{q} b_i t_i \tag{3}$$

Eq. (4) gives the loss function L for a data-driven DeepONet based on mean squared error:

$$L = \frac{1}{N * P} \sum_{i=1}^{N} \sum_{j=1}^{P} \left(\hat{G}(u_i)(\bar{y}_{u_i}^j) - s(u_i)(\bar{y}_{u_i}^j) \right)^2 \tag{4}$$

N is a number of sample functions u. In a training step, the network predicts $\hat{G}(u)(\bar{y})$ for each sample function u_i, which is assessed at P output sensor locations $\bar{y}^j$ and compared to the associated target solutions $s(u)(\bar{y})$ calculated by a classical second-order finite difference solution in an offline (data-generation) stage. The gradient of the loss function with respect to the weights in both networks is then calculated as a part of a backpropagation process, and the Adam optimizer minimizes the loss value by modifying the weights. DeepONet learns the solution operator of the heat conduction equation after a sufficient number of feedforward and backpropagation iterations and can inference a discrete point

solution for an unknown source term parametric function almost instantly.

2.3. Physics-informed DeepONet

The original, or purely data-driven, DeepONet architecture from Fig. 1 requires a large number of outputs, also known as labels or targets, $s(u)(\bar{y})$, which are used to calculate the loss function in Eq. (4) and thus properly train the network. As stated before, the generation of such data often requires repeated evaluation with classical numerical methods such as higher-order finite difference, finite volume, or finite element methods. This can be particularly time and computationally expensive with the governing PDEs defined on large multi-dimensional domains, even on high-performance computing platforms. It is even more difficult to obtain a sufficient number of labels from the experimental approach. We have devised a DeepONet model that can be trained without any generated or observable data at all, given only knowledge of the heat conduction PDE and its corresponding BCs. The so-called physics-informed DeepONet model architecture is given in Fig. 2. The major difference is that there are two contributions to the loss function. One is the operator loss similar to the data-driven DeepONet loss in Eq. (4) but applied only to the points on the boundary conditions where the targets (solutions) are already defined, zero-valued in the case of our Dirichlet BCs. The other loss is the physics loss calculated at the Q collocation points in the domain's interior where the estimated solution operator G is differentiated with respect to input coordinates by means of automatic differentiation. For each collocation point, residual in Eq. (5) is calculated, which satisfy the governing heat conduction PDE and thus provides a physics-based regularization contribution to the overall loss that constrains the space of admissible deep learning solutions.

$$L_{phys} = \frac{1}{N*Q}\sum_{i=1}^{N}\sum_{j=1}^{Q} \times \left| k_x \frac{\partial^2 G(u^i)(x_j^i, y_j^i)}{\partial (x_j^i)^2} + k_y \frac{\partial^2 G(u^i)(x_j^i, y_j^i)}{\partial (y_j^i)^2} + u^i(x_j^i, y_j^i) \right|^2 \quad (5)$$

N is again the number of sample functions u. Since u is originally defined on m points, whose coordinates are not necessarily coincided with the collocation points, the 2D interpolation is used to provide discrete u values at the collocation point coordinates.

2.4. Data sampling and computing environment

We use a 2D correlated and scale-invariant Gaussian random field to generate random input functions $u(x,y)$. We utilized a python implementation described in [16], in which the correlations are explained by a scale-free spectrum $P(k) \sim 1/|k|^{\alpha/2}$ (with $\alpha = 4$). The smoothness of the sampled function is determined by the length-scale coefficient, and a larger value means a smoother u. Because the grid where the finite difference solution for s targets is calculated is generally not the same as the P grid for output sensors, and similarly, the grid where u is generated is generally different than the input sensor grid m, bilinear 2D interpolation is used to provide discrete values for s and u across the different grids. Bilinear 2D interpolation is used to provide discrete

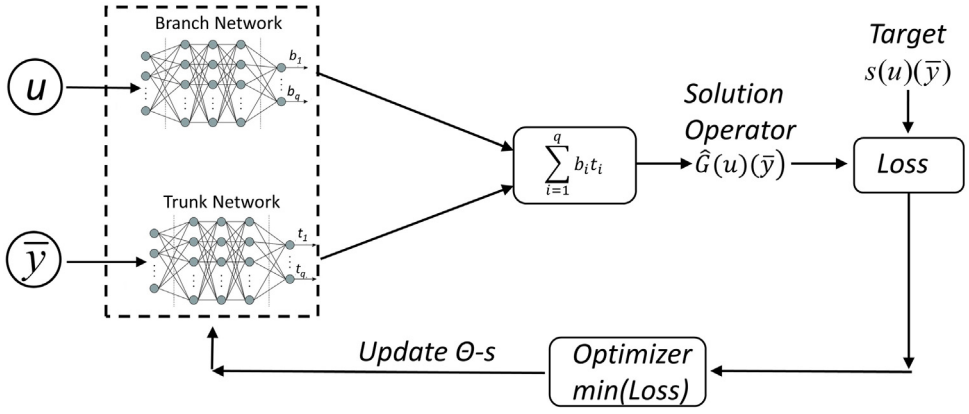

Fig. 1. Data-driven DeepONet.

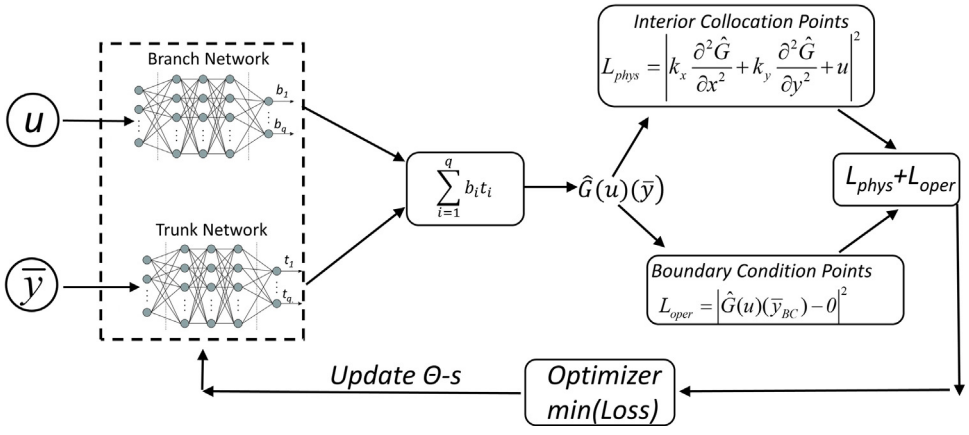

Fig. 2. Physics-informed deep operator network.

values for s and u across the different grids since the grid where the finite difference solution for s targets is calculated is generally not the same as the P grid for output sensors, and similarly, the grid where u is generated is generally different than the input sensor grid m. In this novel work, we have provided an extension of the data-driven and physics-informed DeepONet formulations to a 2D heat conduction domain under the variable spatial distribution of the heat source. The code is written in JAX [17], a relatively new Python-based toolkit built for high-performance machine learning research and created by Google. Many sophisticated capabilities of JAX include advanced automated differentiation (grad), just-in-time compilation (JIT), and cross-device compute replication (pmap). JIT was utilized to offload some of the computationally heavy kernels to the GPU, and pmap was very helpful in speeding up target creation. The Adam optimizer, also available in JAX as a high-level function, employed automatic differentiation. Computing was done on a computing node of the HPC cluster Delta [18], housed at the National Center for Supercomputing Applications (NCSA), and has four A100 Nvidia GPU cards.

3. Results

3.1. Data-driven DeepONet

The data-driven DeepONet network was trained during 80,000 epochs with 1,000-8,000 training u samples (m=121). For each u data sample, target solutions were supplied for 200-600 output sensor points (P=200-600), whose coordinates were picked at random in the 2D domain. During the offline data-generation stage, the second-order finite-difference (FED) solution on the 121x121 grid is used to derive the target solutions on those points. We tested both simple explicit Jacobi iterative and implicit FED solver schemes, Özişik et al [19]. A vector-style mapping of computing across devices (pmap) in JAX is used to significantly accelerate data sample generation. It took the Jacobi FED iterative solver 4–6 min to generate all training data samples with pmap, while using highly optimized python libraries, the implicit solver needed only 30-35 sec.

Fig. 3 depicts three randomly picked test u samples, which the network has never seen before, as well as the matching classical numerical heat conduction equation solutions (targets) and data-driven DeepONet forecasts. The network was trained using 5,000 u samples and 400 output sensors (P=400) in this scenario. For both the branch and trunk networks, we start with the nominal setup by Wang et al. [15], which had five hidden layers, each with 50 neurons, trained on 80,000 epochs. Even though the peaks and valleys in the parametric source distributions u differ significantly between the test samples, data-driven DeepONet correctly predicted their 2D diffusive nature solution in the interior governed by the heat conduction equation and on the zero-valued Dirichlet boundaries. Visual assessment of the numerous additional test samples corroborated the data-driven DeepONet predictions' quantitative correctness.

The mean of the relative L^2 prediction error over N_P number of examples in the test data-set, each defined on $\bar{y}$ with P training point coordinates in the domain (output sensors), provided a more qualitative test error analysis of a trained data-driven DeepONet in Eq. (6).

$$\bar{L}^2 = \frac{1}{N_P}\sum_{i=1}^{N_P}\frac{\left\|\hat{G}(u_i)(\bar{y}) - s(u_i)(\bar{y})\right\|_2}{\left\|\hat{G}(u_i)(\bar{y})\right\|_2} \tag{6}$$

For a variable number of input parametric functions #u (or N) and output sensors P utilized for training, the test error is shown in Fig. 4 for Np=100. Because the output sensors are sparsely spreading at random in the 2-dimensional domain where the predicted solution is compared to targets, P has less impact on the testing error than the size of the training data set #u. Nevertheless, the error was as low as 3% for #u=8000 and P=600. The diffusion-reaction PDE across its space-time grid has a smaller L^2 test error, according to the supplement of [15]. Whilst its u parametric function is defined on a one-dimensional spatial space (line), our u input function, on the other hand, is defined on a two-dimensional spatial domain. To have the same closeness of the discrete points defining u in [14], it would need a square of m (number of input sensors), which would surpass the device memory of our present GPU hardware.

If the branch and trunk network sizes are varied, i.e., the number of neurons per hidden layer (network width) in Fig. 5 and the number of hidden layers (network depth) in Fig. 6, a similar pattern emerges. Increasing the network's width or depth, in particular, tends to enhance prediction accuracy. Surprisingly, as network sizes get larger by increasing their width and depth, the computational training time rises very little (2 min maximum), owing to JAX's excellent handling of deep learning training kernels on GPUs. Finally, if we had naively utilized a single fully connected feedforward neural network instead of DeepONet, we would have needed 14,641 outputs to provide predictions on the 121x121 grid used for DeepONet data generation and prediction validations. This would call for a significantly larger neural network. It is debatable whether it would be feasible to correctly train such a network even on the most recent and powerful high-performance computing platforms.

Table 1
Solution times for classical non-optimized and highly optimized Poisson's solvers and DeepONet inference.

FD Iterative (Jacobi)	FD Implicit	DeepONet Inference
2.1 sec.	0.05 sec.	9 x 10^{-4} sec.

Table 1 shows the computational cost of inferencing using a trained data-driven DeepONet model, as well as the solution times of a heat conduction PDE using second-order finite difference iterative (Jacobi) and highly optimized implicit solution schemes on GPU with the numpy and scipy implementations in JAX [17]. Because JAX employs asynchronous dispatch to disguise python overheads, we properly waited for the JAX calculation to finish before providing accurate measurements. Particularly, once training data is generated, and a data-driven DeepONet model is adequately trained (which combined takes about 22 min on A100 GPU), the DeepONet can predict the solution of a heat conduction PDE in a fraction of a second on modern GPU, which is two to three orders of magnitude faster than traditional PDE solvers. The inferencing involves a single forward pass consisting of dense matrix-vector multiplication kernels, which are exceedingly well optimized not only on GPUs but also on CPUs, which allows inferencing on low-end computers, too, with the trained parameters transferred from high-end computers with GPUs. This is also comparable to traditional PINNs, but with the significant difference that trained DeepONets can produce solutions relatively instantaneously when a new and unknown spatially distributed parametric input is supplied, whereas PINNs must be retrained. Since DeepONets can generally be trained to solve any other PDE with its parameters, the surrogate DeepONet-trained models might possibly replace the traditional and often computationally expensive PDE solver kernels, requiring just a forward pass of the network (inferencing) for each new input such as source term, material properties, boundary conditions, loads, and other parameters. This can considerably speed up high-fidelity scientific and engineering applications governed by parametric PDEs, particularly those that solve large problems [20,21] or repeatedly a large number of PDEs with parameters.

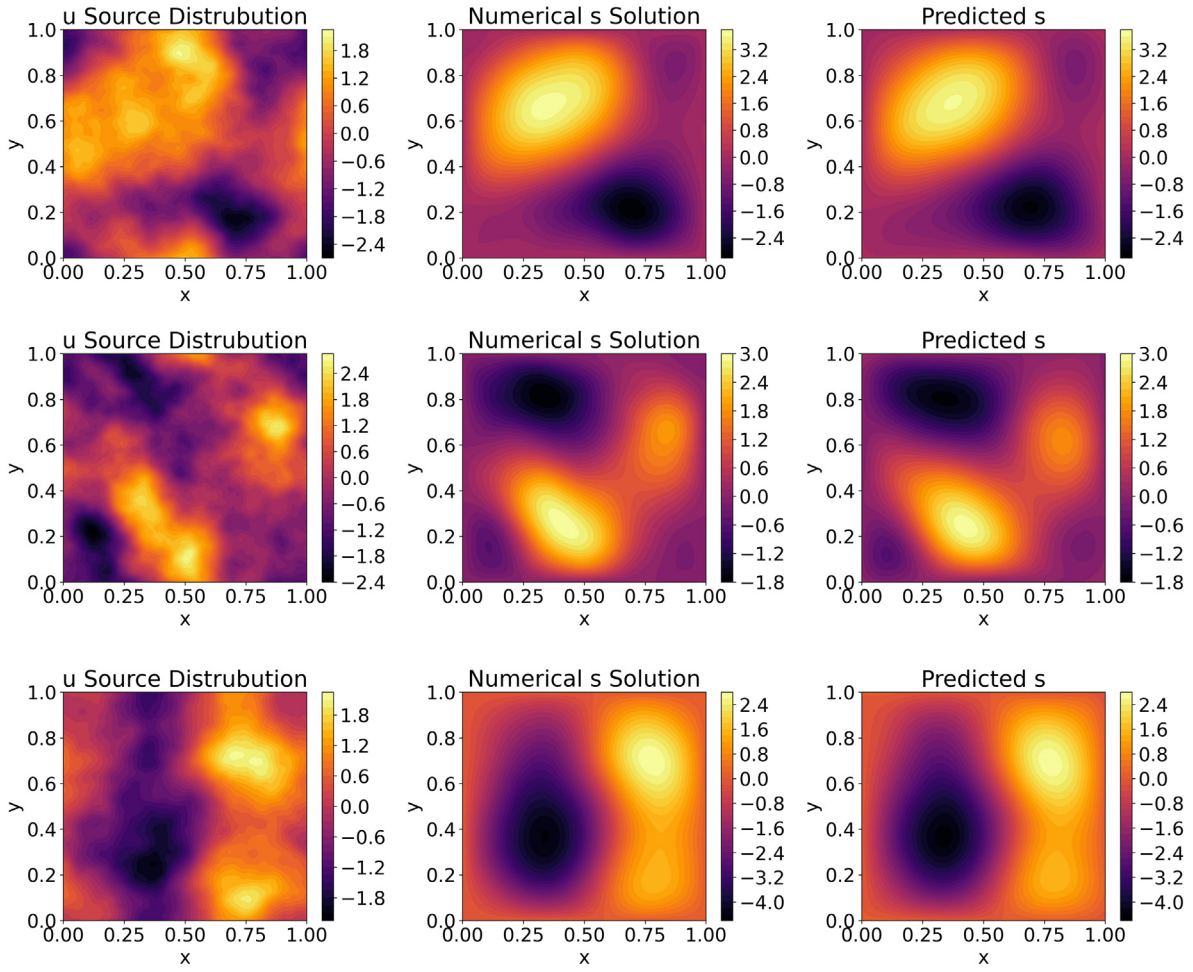

Fig. 3. Target (numerical) solution versus the prediction of a trained data-driven DeepONet for 3 random u data samples, represented in 3 rows, from the test dataset.

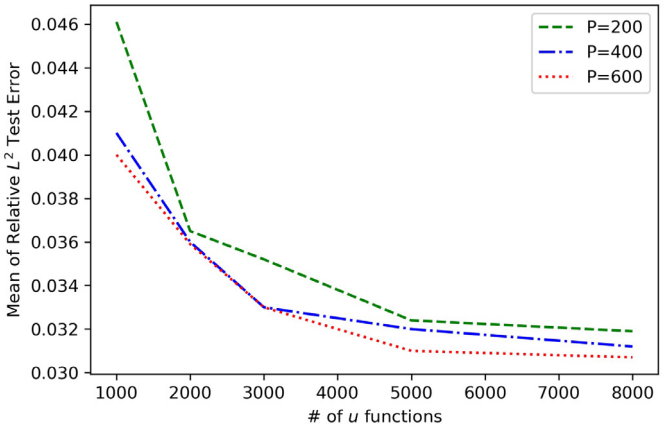

Fig. 4. Effect of training data set size #u and number of output sensors P on test error.

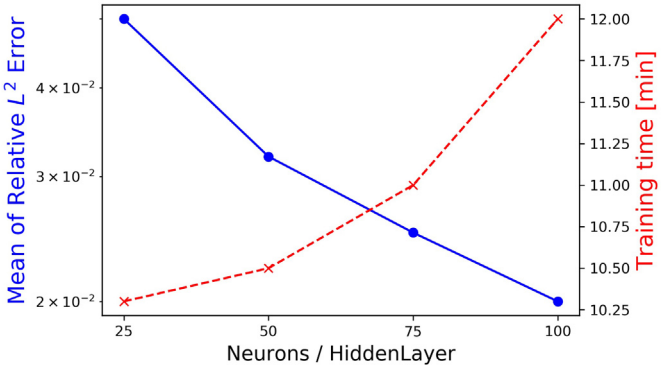

Fig. 5. Effect of number of neurons on test error and corresponding training time with 5-hidden-layer branch and trunk networks (#u=5000, P=400).

3.2. Comparison of data-driven and physics-informed DeepONets

Solution predictions from Data-driven and Physics-informed networks are compared in Fig. 7 for three randomly chosen source distributions from the test datasets. The nominal neural networks are used consisting of 5 hidden layers with 50 neurons each, 5,000 u training samples trained with 80,000 epochs, and a variable number of input sensors m and collocation points Q. The number of output sensors P where data–driven DeepONet is evaluated matches the number of random points enforcing zero-valued Dirichlet boundary conditions in the physics-informed DeepONet. The test error analysis in Fig. 8 compares prediction errors across 100 test samples from the data-driven and physics-informed nominal size DeepONets. While the number of input sensors m is set to be equal to the number of collocation points Q in both physics-informed cases, in the first case, their spatial coordinates do not

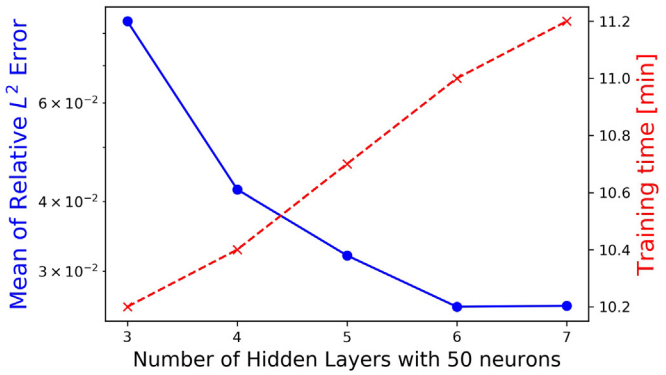

Fig. 6. Effect of number of hidden layers, with 50 neurons each, on test error and corresponding training time of the branch and trunk networks (#u=5000, P=400).

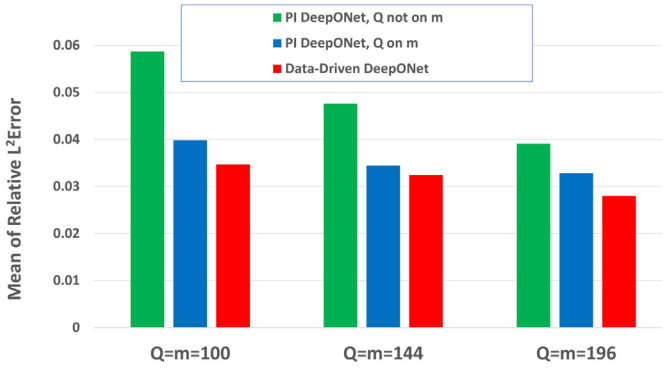

Fig. 8. Test Error comparison between data-driven and physics-informed DeepONets.

match (collocation points are randomly chosen, Q not on m), while in the second case, they are matched precisely (collocation points are exactly at input sensor point locations, Q on m).

The data-driven prediction is slightly more accurate than physics-informed prediction with the collocation points coinciding with the input sensor locations for u. The error is, however, larger for the physics-informed case with the randomly spaced collocation points. This is a consequence of the 2D bi-linear interpolation that interpolates u values between input sensor and collocation coordinates, as stated before. Wang et al. [14], however, reported in his work with the Diffusion-Reaction equation sampling one-dimensional u space that the physics-informed DeepONet provided more accurate test predictions than the data-driven DeepONet. Therefore, it seems that physics-informed DeepONet is more sensitive to the dimensionally cursed phenomena of the source term than the data-driven one, i.e., when the dimensionality of u sampling space increases, the surface of the sampling

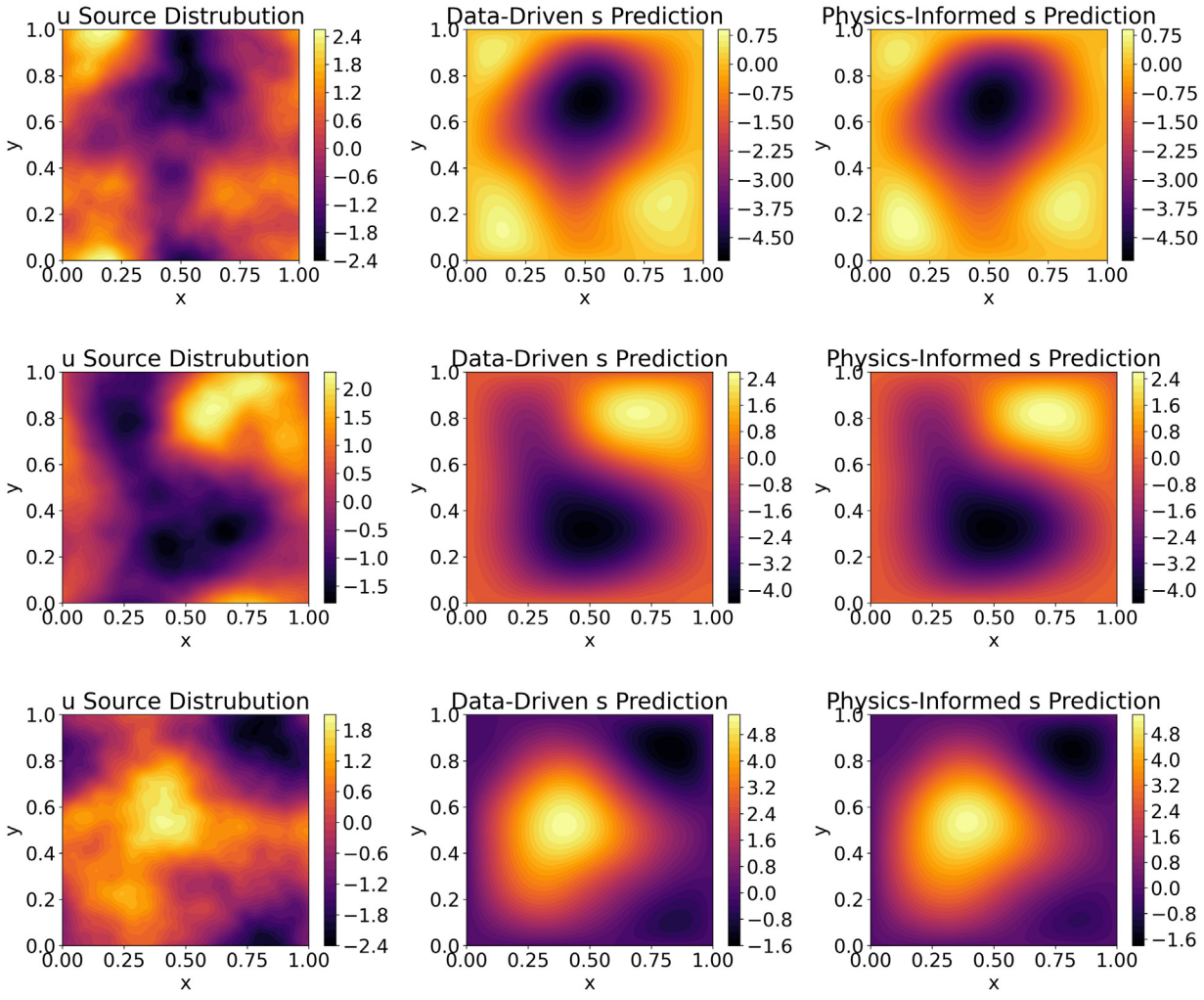

Fig. 7. Data-driven and Physics-informed DeepONet predictions for 3 random u data samples, represented in 3 rows, from the test dataset.

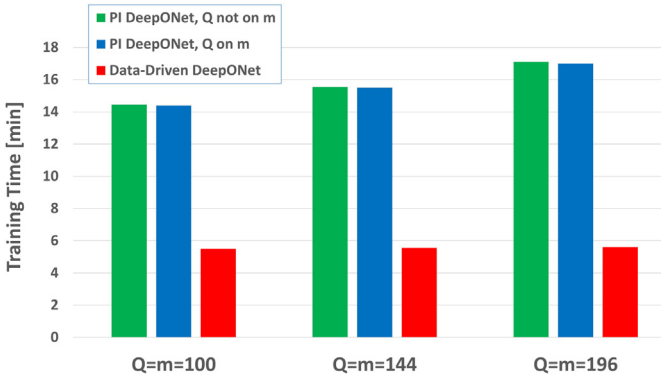

Fig. 9. Training time comparison between data-driven and physics-informed DeepONets.

space increases much faster so that the available sampling data become sparse. Similar results have been reported in other works for convolutional physics-informed neural networks compared to the data-driven equivalents, such as by Zhu et al. [22] and, lately, Fuhg et al. [23].

Finally, the comparison between training time for data-driven and physics-informed DeepONet networks on an A100 GPU node is given in Fig. 9. Similarly to what Wang et al. [14] have observed, training physics-informed DeepONets is computationally more expensive than the data-driven equivalent mainly due to a physics-informed network needing to compute the PDE residual by means of automatic differentiation producing a larger computational graph then the data-driven network. However, since the physics-informed network's offline (non-training) step just produces u samples, it takes only 1-2 min. for the Jacobi and 10-15 sec. for the implicit FED schemes. This is 2-3x quicker than in data-driven DeepONets, which must also compute matching solutions (targets) at collocation locations.

4. Conclusions

After presenting the data-driven and physics-informed DeepONet frameworks, which employ the dual branch-trunk neural network to learn parametric PDE solution operators, we concentrated on solving the heat conduction equation with spatially 2D parametric source term inputs in this paper. This is best to our knowledge one of the first uses of DeepONets to a spatially multi-dimensional application governed by an extremely important parametric PDE in science and engineering. We have found that data-driven DeepONets can reasonably quickly learn to solve the heat conduction equation and inference reasonably accurate results on contemporary GPUs when a new random multi-dimensional source term input is presented from the test dataset previously unseen by the network. We also looked at the impact of several factors on test error, such as training size, the number of output sensors, network depth and width, and their effect in spatially 2D dimensional input and output regions. Most importantly, we have found that the computational cost of running a previously trained DeepONet model to solve spatially multi-dimensional parametric PDEs can be magnitudes of order lower than for the classical numerical solvers. Finally, we have compared test accuracy and training time between data-driven and physics-informed DeepONets. We have found that the data-driven DeepONet provides a more accurate prediction for test samples with less training time. On the other hand, physics-informed DeepONet takes significantly less time in the offline (data generation) stage and is able to learn the solution operator of the parametric heat conduction equation without any generated or observed solution targets. It is worth mentioning here that other parameters, such as anisotropic material properties with k_x and k_y thermal conductivities, can be easily added as additional two parameters to u. Similarly, a spatial variation of boundary conditions can be represented by additional parameters in u representing discrete values on the boundaries. It is merely a function of adequately training a DeepONet with the value ranges of additional parameters, which is an increased computational task, particularly with the data-driven approach that also requires additional data generation. Nevertheless, this is readily feasible on modern computers with GPUs, particularly since the DeepONet training has to be usually done only once. Similar DeepONet-based surrogate deep learning models are expected to aid or even replace traditional numerical PDE solver kernels in the future, allowing for considerably faster high-fidelity simulations, optimizations, designs, and online controls in many scientific and engineering applications and processes governed by parametric PDEs.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Seid Koric: Conceptualization, Methodology, Supervision, Formal analysis, Writing – review & editing. **Diab W. Abueidda:** Investigation, Validation, Writing – review & editing.

Data availability

Data will be made available on request.

Acknowledgements

The authors would like to thank the National Center for Supercomputing Applications (NCSA) at the University of Illinois, and particularly its Industry Program and the Center for Artificial Intelligence Innovation (CAII) for their support and hardware resources. This research is also a part of the Delta research computing project, which is supported by the National Science Foundation (award OCI 2005572) and the State of Illinois.

References

[1] J. Zhao, W. Zhao, Z. Ma, W. Yong, B. Dong, Finding models of heat conduction via machine learning, Int. J. Heat Mass Transf. 185 (2022) 122396, doi:10.1016/j.ijheatmasstransfer.2021.122396.
[2] J. Athavale, M. Yoda, Y. Joshi, Comparison of data driven modeling approaches for temperature prediction in data centers, Int. J. Heat Mass Transf. 135 (2019) 1039–1052, doi:10.1016/j.ijheatmasstransfer.2019.02.041.
[3] H.T. Kollmann, D.W. Abueidda, S. Koric, E. Guleryuz, N.A. Sobh, Deep learning for topology optimization of 2D metamaterials, Mater. Des. 196 (2020) 109098, doi:10.1016/j.matdes.2020.109098.
[4] M. Mozaffar, R. Bostanabad, W. Chen, K. Ehmann, J. Cao, M. Bessa, Deep learning predicts path-dependent plasticity, Proc. Natl. Acad. Sci. 116 (52) (2019) 26414–26420, doi:10.1073/pnas.1911815116.
[5] D.W. Abueidda, S. Koric, N.A Sobh, H. Sehitoglu, Deep learning for plasticity and thermo-viscoplasticity, Int. J. Plast. 136 (2021) 102852, doi:10.1016/j.ijplas.2020.102852.
[6] B.P. Savali, A. Mielke, T. Ricken, Data-driven stress prediction for thermoplastic materials, Proc. Appl. Math. Mech. (PAMM) 21 (1) (2021) e202100225, doi:10.1002/pamm.202100225.
[7] Y. Feng, W. Gao, D. Wu, F. Tin-Loi, Machine learning aided stochastic elasto-plastic analysis, Comput. Methods Appl. Mech. Eng. 357 (2019) 112576, doi:10.1016/j.cma.2019.112576.
[8] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707, doi:10.1016/j.jcp.2018.10.045.
[9] K.M. Kim, P. Hurley, J.P. Duarte, Physics-informed machine learning-aided framework for prediction of minimum film boiling temperature, Int. J. Heat Mass Transf. 191 (2022) 122839, doi:10.1016/j.ijheatmasstransfer.2022.122839.

[10] D.W. Abueidda, Q. Lu, S. Koric, Meshless physics-informed deep learning method for three-dimensional solid mechanics, Int. J. Numer. Methods Eng. 122 (23) (2021) 7182–7201, doi:10.1002/nme.6828.
[11] J.N. Fuhg, N. Bouklas, The mixed deep energy method for resolving concentration features in finite strain hyperelasticity, J. Comput. Phys. 451 (2022) 110839, doi:10.1016/j.jcp.2021.110839.
[12] S. Cai, Z. Wang, F. Fuest, Y.J. Jeon, C. Gray, G.E. Karniadakis, Flow over an espresso cup: inferring 3-D velocity and pressure fields from tomographic background oriented Schlieren via physics-informed neural networks, J. Fluid Mech. 915 (2021) A102, doi:10.1017/jfm.2021.135.
[13] L. Lu, P. Jin, G. Pang, Z. Zhang, G.E. Karniadakis, Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators, Nat. Mach. Intell. 3 (2021) 218–229, doi:10.1038/s42256-021-00302-5.
[14] T. Chen, H. Chen, Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems, IEEE Trans. Neural Netw. 6 (1995) 911–917, doi:10.1109/72.392253.
[15] S. Wang, H. Wang, P. Perdikaris, Learning the solution operator of parametric partial differential equations with physics-informed DeepONets, Sci. Adv. 7 (40) (2021) 1–9, doi:10.1126/sciadv.abi8605.
[16] Sciolla B.:, Generator of 2D gaussian random fields, https://github.com/bsciolla/gaussian-random-fields.
[17] Bradbury J., Frostig R., Hawkins P., Johnson M.J., Leary C., Maclaurin D., Necula G., Paszke, VanderPlas A.J., Wanderman-Milne S., Zhang Q.: JAX: composable transformations of Python+NumPy programs (2018).
[18] Delta HPC system at NCSA, https://www.ncsa.illinois.edu/research/project-highlights/delta/.
[19] M.N. Özişik, H.R.B. Orlande, M.J. Colaço, R.M. Cotta, Finite Difference Methods in Heat Transfer, 2nd ed., CRC Press, 2017, doi:10.1201/9781315121475.
[20] S. Koric, A. Gupta, Sparse matrix factorization in the implicit finite element method on petascale architecture, Comput. Methods Appl. Mech. Eng. 32 (2016) 281–292, doi:10.1016/j.cma.2016.01.011.
[21] M. Vázquez, G. Houzeaux, S. Koric, et al., Alya: Multiphysics engineering simulation toward exascale, J. Comput. Sci. 14 (2016) 15–27, doi:10.1016/j.jocs.2015.12.007.
[22] Y. Zhu, N. Zabaras, P.S. Koutsourelakis, P. Perdikaris, Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data, J. Comput. Phys. 394 (2019) 56–61, doi:10.1016/j.jcp.2019.05.024.
[23] J.N. Fuhg, A. Karmarkar, T. Kadeethum, H. Yoon, N. Bouklas, Deep convolutional ritz method: parametric PDE surrogates without labeled data, arXiv:2206.04675v1 [cs.CE], Jun 2022.