XLIII CILAMCE
21-25, November, 2022
FOZ DO IGUAÇU - PR - BRAZIL

Approximating the operator of the wave equation using deep learning

Ziad Aldirany[1], Régis Cottereau[2], Marc Laforest[1], Serge Prudhomme[1]

[1]*Département de mathématiques et de génie industriel, Polytechnique Montréal*
Montréal, H3T 1J4, Québec, Canada
ziad.aldirany@polymtl.ca, marc.laforest@polymtl.ca, serge.prudhomme@polymtl.ca
[2]*Aix-Marseille Université, CNRS, Centrale Marseille, Laboratoire de Mécanique et d'Acoustique UMR 7031*
Marseille, France
cottereau@lma.cnrs-mrs.fr

Abstract. Deep operator networks (DeepONets) have demonstrated the capability of approximating nonlinear operators for initial- and boundary-value problems. One attractive feature of DeepONets is their versatility since they do not rely on prior knowledge about the solution structure of a problem and can thus be directly applied to a large class of problems. However, convergence in identifying the parameters of the networks may sometimes be slow. In order to improve on DeepONets for approximating the wave equation, we introduce the Green operator networks (GreenONets), which use the representation of the exact solution to the homogeneous wave equation in term of the Green's function. A comparison between the GreenONets and the DeepONets is shown on a series of numerical experiments for homogeneous and heterogeneous medias.

Keywords: Deep learning, Wave equation, Deep operator networks, Physics-informed neural networks

1 Introduction

In the last few years, a large amount of work, such as Jin et al. [1], Bihlo and Popovych [2], Pettit and Wilson [3], Moseley et al. [4], has been devoted to using deep learning methods for the solution of PDE-based problems, such as in fluid dynamics, acoustics, meteorology, etc. These works have been motivated by the ability of deep neural networks to approximate a large class of functions in high dimension over complicated domains.

Prominent deep learning methods for the solution of partial differential equations rely on either learning the solution function of the problem, as presented by Raissi et al. [5], Weinan and Yu [6], Sirignano and Spiliopoulos [7], or learning the operators that describe the physical problem, as introduced in Lu et al. [8], Li et al. [9, 10], Wang et al. [11]. Using these methods, the network is trained based on the physics of the problem, in the sense that it verifies the partial differential equations (along with boundary and initial conditions), and not solely on data. In the first approach, the solution is approximated with a neural network by minimizing the residual of the PDE, e.g. physics informed neural networks introduced by Raissi et al. [5]. The most common approach is to calculate the residual from the strong form of the PDE, which is evaluated using automatic differentiation. The second approach learns the differential operator for a given family of parameters, e.g. deep operator networks (DeepONets) as presented in Lu et al. [8], thus allowing one to subsequently approximate the solution to a physical problem for a specific parameter in the vicinity of the trained parameters. The training of this neural network is usually very expensive, but needs to be done only once. Obtaining the solution for a new parameter requires only one forward pass in the online phase, which is usually cost-effective. This makes the operator approximation method attractive when the physical problem should be solved for a family of parameters. For example, in seismology, uncertainties of the earth properties require thousands of simulations to obtain a solution that describes well the recorded data.

Solution of the wave propagation problem, which has been of great interest in several areas such as seismology, electromagnetism, or fluid dynamics, remains challenging using classical methods (such as finite elements, spectral methods, etc.). In this work, we will consider in particular the one-dimensional wave equation with homogeneous Dirichlet boundary conditions. Let Ω be an open domain in $\mathbb{R}$, with boundary $\partial\Omega$, and assume $c(x)$ is a known function. Given $u_0(x)$, $u_1(x)$, and $T > 0$, we want to find $u(x,t)$, for all $x \in \Omega$ and $t \in (0,T)$ such that

$$\partial_{tt}u(x,t) - c(x)^2\partial_{xx}u(x,t) = 0, \quad \forall(x,t) \in \Omega \times (0,T), \tag{1}$$

subjected to the initial and boundary conditions

$$u(x,0) = u_0(x), \quad \partial_t u(x,0) = u_1(x), \quad \forall x \in \Omega, \tag{2}$$
$$u(x,t) = 0, \quad \forall (x,t) \in \partial\Omega \times (0,T). \tag{3}$$

In this work, we propose a novel architecture, inspired by the deep operator networks, for approximating the differential operator in the wave equation. The so-called Green operator networks (GreenONets) is based on the representation of the exact solution to the homogeneous wave equation on unbounded domains in terms of the Green's function, for details we refer the readers to Duffy [12]. Using this architecture, we can train the network for a family of initial conditions taking into account the governing equation and the boundary and initial conditions, i.e. without any solution data. We will show on a series of numerical examples involving the homogeneous and heterogeneous wave equations, that the approximation of the operator using GreenONets exhibits better results in terms of accuracy and convergence.

2 Deep operator networks

We briefly review the deep operator networks first introduced by Lu et al. [8]. In this work, we will be learning the operator from the partial differential equation and the initial-boundary conditions. In other words, the physics-informed DeepONets, described in Wang et al. [11], will be presented.

A neural network maps an input into an output by a composition of linear and nonlinear functions, with adjustable weights, with the goal of minimizing the error between an output and a target function on a specific training set. Therefore the network is trained by adjusting its weights in order to better describe the target function on the training set, in the hope of generalizing the output function to a wider set of input. In this section, we will introduce the feedforward neural network (FNN), that will be used later in the DeepONets and GreenONets. Let us consider a FNN with d layers, each layer having a width N_i, and let N_0 denote the size of the input data. Denoting the activation function by σ, the neural network with input $\boldsymbol{x}$ and output u is defined as

$$\begin{aligned}
&\text{Input layer:} && \boldsymbol{x}_0 = \boldsymbol{x}, \\
&\text{Hidden layers:} && \boldsymbol{x}_i = \sigma(\boldsymbol{W}_i \boldsymbol{x}_{i-1} + \boldsymbol{b}_i), \quad i = 1, \cdots, d-1, \\
&\text{Output layer:} && u = \boldsymbol{W}_d \boldsymbol{x}_{d-1} + \boldsymbol{b}_d,
\end{aligned} \tag{4}$$

where $\boldsymbol{W}_i$ is the weights matrix of size $N_i \times N_{i-1}$ and $\boldsymbol{b}_i$ is the biases vector of size N_i. When convenient, we will combine the weights and biases into the single parameter Θ of the neural network. In this work, we shall consider the $\tanh$ activation function, but other activation functions could be used as well.

The deep operator networks, first introduced by Lu et al. [8], aim at approximating nonlinear operators for parametric partial differential equations. The method seeks to learn the operator for a family of parameters, such as those provided in the initial or boundary conditions, domain properties, source term, etc. More precisely, for given Banach spaces $\mathcal{U}$ and $\mathcal{S}$, we learn the operator $G : \mathcal{S} \to \mathcal{U}$ such that for any input parameter $\boldsymbol{s} \in \mathcal{S}$, $u \equiv G(\boldsymbol{s}) \in \mathcal{U}$ is the solution to a given initial boundary-value problem.

Let Ω be an open domain in $\mathbb{R}^d$ with boundary $\partial\Omega$. We write the partial differential equation in terms of its associated residual as

$$R\left(\boldsymbol{x}, G(\boldsymbol{s}), \frac{\partial^{|k|} G(\boldsymbol{s})}{\partial \boldsymbol{x}^k}; \boldsymbol{s}\right) = 0, \quad \forall \boldsymbol{x} \in \Omega, \tag{5}$$

with the boundary condition

$$B(\boldsymbol{x}, G(\boldsymbol{s}); \boldsymbol{s}) = 0, \quad \forall \boldsymbol{x} \in \partial\Omega. \tag{6}$$

For time-dependent problems, by considering the time as a component of $\boldsymbol{x}$, initial conditions can be specified within B as special boundary conditions. In order to approximate the operator G, we present the unstacked DeepONets architecture originally introduced in Lu et al. [8] and schematically shown in Fig. 1. We start by defining the input function $\boldsymbol{s}$ as a discrete vector $[\boldsymbol{s}(\boldsymbol{x}_i)]_{i=1,\ldots,m}$ evaluated at a collection of points $\{\boldsymbol{x}_i\}_{i=1}^m$, known as sensors. Then, as illustrated in Fig. 1 the operator is approximated as

$$\hat{G}(\boldsymbol{s})(\boldsymbol{x}) = \sum_{k=1}^{q} b_k\big(\boldsymbol{s}(\boldsymbol{x}_1), \ldots, \boldsymbol{s}(\boldsymbol{x}_m)\big) t_k(\boldsymbol{x}), \tag{7}$$

where $\{b_k\}_{k=1}^q$ is the output of the branch network that takes $\boldsymbol{s}$ as an input and $\{t_k\}_{k=1}^q$ is the output of the trunk network that takes $\boldsymbol{x}$ as an input. Here, we will consider a simple FNN for both the branch and trunk networks.

We consider here the physics-informed DeepONets presented in Wang et al. [11], where the network is trained by penalizing the residual R associated with the governing partial differential equation and the residual

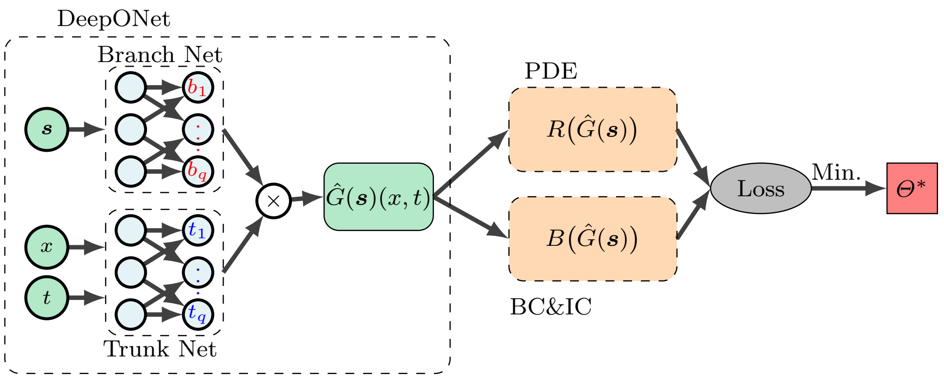

Figure 1. Illustration of the architecture of the unstacked DeepONets.

B associated with the initial and boundary conditions. In order to define the training set, on which we want to minimize our residuals, we start by defining a family of input functions $\{\boldsymbol{s}^{(i)}\}_{i=1}^{N}$. For each input function $\boldsymbol{s}^{(i)}$, a collection of points $\{\boldsymbol{x}_{b,j}^{(i)}\}_{j=1}^{P}$ is randomly sampled on the boundary $\partial\Omega$, where the initial-boundary conditions residual are penalized. Similarly, a set of points $\{\boldsymbol{x}_{r,j}^{(i)}\}_{j=1}^{Q}$ is randomly sampled in the domain Ω, where the residual of the PDE are be minimized. Finally, the loss function is defined as

$$L(\Theta) = w_r L_r(\Theta) + w_b L_b(\Theta), \tag{8}$$

where

$$L_r(\Theta) = \frac{1}{NQ}\sum_{i=1}^{N}\sum_{j=1}^{Q}\left|R\big(\boldsymbol{x}_{r,j}^{(i)}, \hat{G}(\boldsymbol{s}^{(i)})(\boldsymbol{x}_{r,j}^{(i)})\big)\right|^2,$$
$$L_b(\Theta) = \frac{1}{NP}\sum_{i=1}^{N}\sum_{j=1}^{P}\left|B\big(\boldsymbol{x}_{b,j}^{(i)}, \hat{G}(\boldsymbol{s}^{(i)})(\boldsymbol{x}_{b,j}^{(i)})\big)\right|^2,$$

and w_r and w_b are the weighting coefficients.

3 Green operator networks

The architecture presented in the DeepONets is a general architecture that works well for different initial boundary-value problems with different input parameters. Our objective here is to develop an approach that improves upon the efficiency of the DeepONets. We will focus on the solution of the wave equation in one dimension domains for homogeneous and heterogeneous materials, as presented in the Introduction. We thus propose the Green operator networks (GreenONets), that approximate the Green's function of the operator, to solve aforementioned problem.

The exact solution of the wave equation in unbounded domains with homogeneous properties, i.e. $c(x) = c$, can be obtained with Green's function g. The exact solution is then given by

$$u(x,t) = -\frac{1}{c^2}\int_{\mathbb{R}} \partial_\tau g(x,t,\xi,\tau)|_{\tau=0} u_0(\xi) d\xi + \frac{1}{c^2}\int_{\mathbb{R}} g(x,t,\xi,0) u_1(\xi) d\xi. \tag{9}$$

In our case, the Green's function is derived as

$$g(x,t,\xi,\tau) = -\frac{c}{2}\mathcal{H}[c(t-\tau) - |x-\xi|], \tag{10}$$

where $\mathcal{H}$ is the Heaviside function. Similar solutions for the homogeneous wave equation can be found in Duffy [12] or Kausel [13] for higher dimensions and bounded domains.

In this work, we are interested in learning the operator of the wave equation for different initial conditions, i.e. the input function is defined as $\boldsymbol{s} = u_0$. Therefore, instead of using a general architecture as the DeepONets, we introduce the Green operator networks, shown in Fig. 2, as a discrete approximation of the first integral in eq. (9). The GreenONet is defined as

$$\hat{G}(\boldsymbol{s})(\boldsymbol{x}) = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{s}(\boldsymbol{x}_i) H(\boldsymbol{x}, \boldsymbol{x}_i), \tag{11}$$

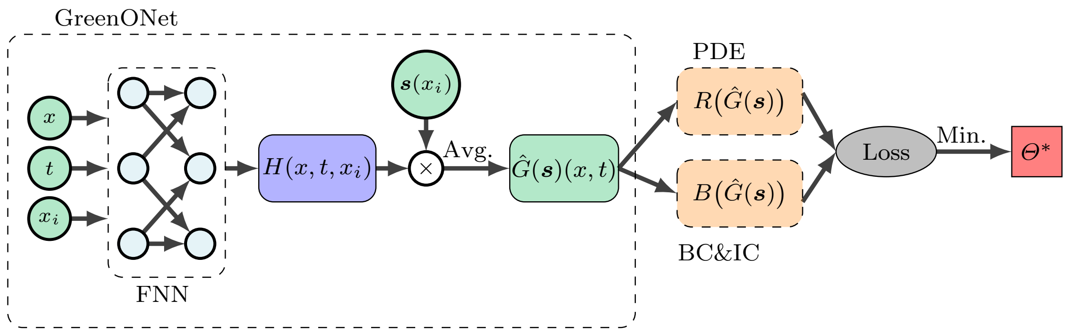

Figure 2. Illustration of the architecture of the GreenONets. A FNN takes as inputs the coordinates and sensor points and outputs an approximated Green's function H for each sensor point. Then, the operator is computed by averaging the product of the Green's function and the input function over the sensor points. Here, $\boldsymbol{x} = (x, t)$. The network is then trained to minimize the loss function that consists of the residuals associated with the partial differential equation and the initial and boundary conditions.

where H is a simple FNN. We note that the formulation of GreenONets depends explicitly on the sensor points $\{\boldsymbol{x}_i\}_{i=1}^m$. Moreover, the sensor points can be varied without deteriorating the solution, which makes it possible to add new sensor points throughout the training. However, in this work we will only focus on fixed sensor points.

Although the exact solution presented in eq. (9) is only valid for a homogeneous material and an unbounded domain, the following numerical results show that GreenONets yield better results when compared to DeepONets for bounded domains with homogeneous or heterogeneous properties.

4 Numerical results

In this section, we approximate the operator of the one-dimensional wave equation for homogeneous and heterogeneous materials in the case of a family of initial conditions, in order to show the effectiveness of GreenONets when compared to DeepONets. We consider $\Omega = (-1, 1)$ and $T = 2$ and we set $u_1(x) = 0$ in all cases. The loss function is defined by

$$L(\Theta) = w_r L_r(\Theta) + w_{ic} L_{ic}(\Theta) + w_{bc} L_{bc}(\Theta), \tag{12}$$

where

$$L_r(\Theta) = \frac{1}{NQ} \sum_{i=1}^{N} \sum_{j=1}^{Q} \left| \partial_{tt} \hat{G}(\boldsymbol{s}^i)\big(x_{r,j}^{(i)}, t_{r,j}^{(i)}\big) - c\big(x_{r,j}^{(i)}\big)^2 \partial_{xx} \hat{G}(\boldsymbol{s}^i)\big(x_{r,j}^{(i)}, t_{r,j}^{(i)}\big)\Big) \right|^2,$$

$$L_{ic}(\Theta) = \frac{1}{NP} \sum_{i=1}^{N} \sum_{j=1}^{P} \left| \hat{G}(\boldsymbol{s}^i)\big(x_{ic,j}^{(i)}, 0)\big) - u_0\big(x_{ic,j}^{(i)}\big) \right|^2 + \left| \partial_t \hat{G}(\boldsymbol{s}^i)\big(x_{ic,j}^{(i)}, 0)\big) - u_1\big(x_{ic,j}^{(i)}\big) \right|^2,$$

$$L_{bc}(\Theta) = \frac{1}{NP} \sum_{i=1}^{N} \sum_{j=1}^{P} \left| \hat{G}(\boldsymbol{s}^i)\big(x_{bc,j}^{(i)}, t_{bc,j}^{(i)})\big) \right|^2.$$

Here, w_r, w_{ic}, and w_{bc} are the weighting coefficients. The initial conditions $\boldsymbol{s}^{(i)}$ are randomly sampled from a Gaussian random field (GRF), as presented by Lu et al. [8], with a defined length scale l. For each $\boldsymbol{s}^{(i)}$, we randomly define $\{x_{ic,j}^{(i)}\}_{j=1}^{P}$ at $t = 0$, $\{\big(x_{bc,j}^{(i)}, t_{bc,j}^{(i)}\big)\}_{j=1}^{P}$ on the boundary of the domain at different times, and $\{\big(x_{r,j}^{(i)}, t_{r,j}^{(i)}\big)\}_{j=1}^{Q}$ on $(-1, 1) \times (0, T)$. In the following experiments, the FNNs in the DeepONets and GreenONets are defined with $d = 6$ layers and $N_i = 50$ for all hidden layers. The loss function is minimized using the ADAM optimizer, introduced by Kingma and Ba [14], using the default parameters, while considering different learning rates for each experiment.

4.1 Homogeneous case with a length scale of 0.5

We start with approximating the operator for a homogeneous material using DeepONets and GreenONets. The sensor points are chosen uniformly with $m = 21$ while the input functions $\boldsymbol{s}$ are generated using a GRF with

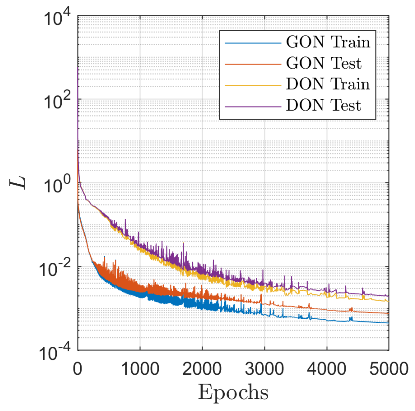

Figure 3. The evolution of the loss function on the training and testing sets during the training with GreenONets and DeepONets for the example of Section 4.1.

a length scale $l = 0.5$. We take the initial learning rate as 10^{-3} and let it decrease with a rate of 0.9995 at each epoch. In this example, we choose $N = 1000$ and $P = Q = 10$. The weights in eq. (12) are set to $w_r = 0.1$ and $w_{ic} = w_{bc} = 10$. The training is done for 5000 epochs with 16 mini-batches.

Figure 3 compares the loss function during the training and test sets for the GreenONets and DeepONets. We observe that with GreeONets the loss functions decrease faster and after 5000 epochs we have smaller losses when compared to the DeepONets loss functions. In order to verify our operators, we will compute the solution at $T = 2$ for the initial conditions $u_0(x) = (1 - x^2)^k$, with $k = 2$ and $k = 10$, as shown in Fig. 4 (left). We observe in Fig. 4 (middle), that the pointwise error at $T = 2$ for $k = 2$ is slightly larger when using DeepONets. However, as shown in Fig. 4 (right), for $k = 10$ the DeepONets solution exhibits a maximum pointswise error of 0.14 while the maximum pointwise error for the GreenONets solution is 0.04. Therefore, we observe that the GreenONets solutions generalize better for higher frequencies.

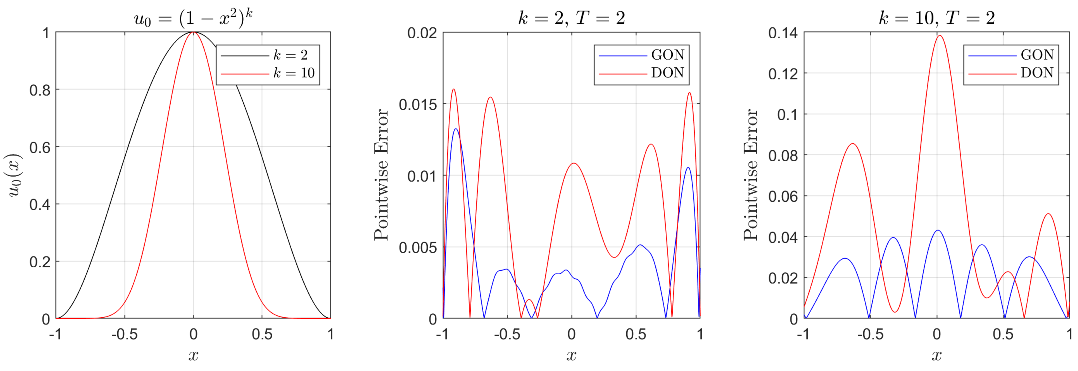

Figure 4. Example of Section 4.1: (left) Initial conditions with which we test the networks. (middle) Pointwise error at $T = 2$ for $k = 2$ using GreenONets and DeepONets. (right) Pointwise error at $T = 2$ for $k = 10$ using GreenONets and DeepONets.

4.2 Homogeneous case with a length scale of 0.1

Here, we solve the same problem as in the last section but the input functions s are defined using a GRF with a length scale $l = 0.1$. In other words, we now compare the two methods for higher frequency solutions. The sensor points are defined uniformly with $m = 60$. We initialize the learning rate to 5×10^{-4} and let it decrease with a rate of 0.999 each epoch. In this example, we take $N = 3000$, $Q = 30$, and $P = 3$. The loss functions weights are $w_r = 0.2$ and $w_{ic} = w_{bc} = 100$. We train both networks for 2000 epochs with 128 mini-batches.

We observe in Fig. 5 (left) that with similar hyper-parameters the loss functions of the GreenONets attain 8×10^{-2} in 2000 epochs while those of the DeepONets plateau earlier. In Fig. 5 (middle), we exhibit the pointwise errors of the GreenONets solutions for $u_0 = (1 - x^2)^k$, with $k = 50$ and $k = 200$. The maximum pointwise error is around 0.01 for $k = 50$ and around 0.1 for $k = 200$. We do not present the pointwise error of the DeepONets solutions since the loss functions did not converge. To better characterize the error for $k = 200$, we compare the

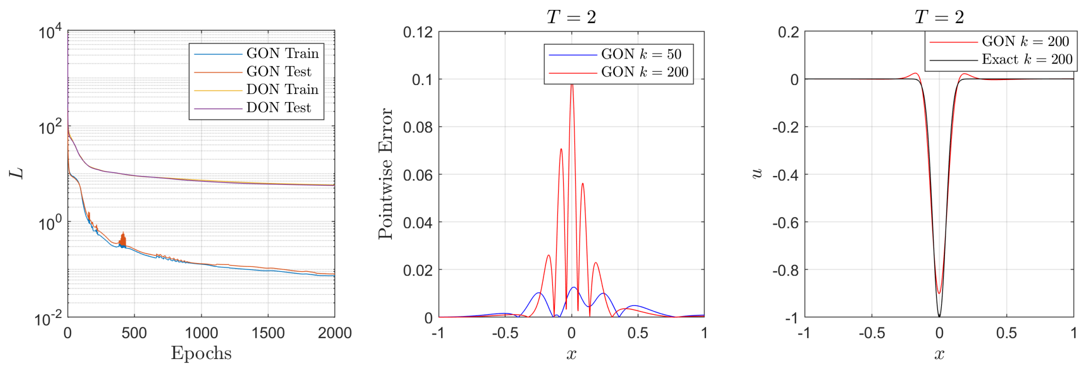

Figure 5. Example of Section 4.2: (left) Evolution of the loss function on the training and testing sets with GreenONets and DeepONets. (middle) Pointwise error at $T = 2$ for $k = 50$ and $k = 200$ using GreenONets. (right) Comparison of the solution obtained by GreenONets at $T = 2$ for $k = 200$ with the exact solution.

solution using GreenONets to the exact solution at $T = 2$ in Fig. 5 (right). We remark that the large errors are close to the propagating wave and did not spread in the rest of the solution.

4.3 Heterogeneous case with a length scale of 0.3

The Green's function is usually presented for the homogeneous wave equation. However, in this section, we use the GreenONets to approximate the operator of the heterogeneous wave equation. First, we define $c(x)^2 = 1 + \mathcal{H}(x - 0.5)$, where $\mathcal{H}$ is the Heaviside function. The input parameters are defined with a length scale $l = 0.3$. We define the sensor points uniformly with $m = 30$. The learning rate is 10^{-3} and has a decay rate of 0.9995 per epoch. We set $N = 2000$, $Q = 15$, and $P = 3$. The weights of the loss functions are $w_r = 1$ and $w_{ic} = w_{bc} = 100$. We divide our training set to 32 mini-batches and we train our networks for 2500 epochs.

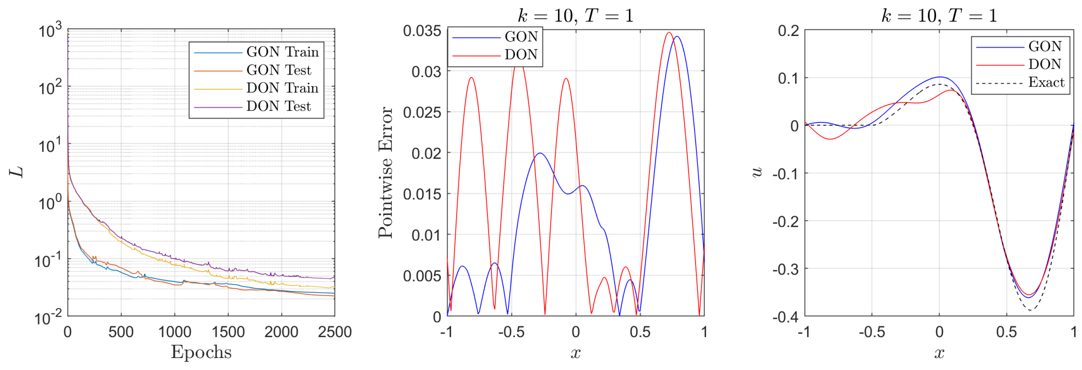

Figure 6. Example of Section 4.3: (left) Evolution of the loss function on the training and testing sets with GreenONets and DeepONets. (middle) Pointwise error at $T = 1$ for $k = 10$, using GreenONets and DeepONets. (right) Comparison of the solutions obtained by GreenONets and DeepONets at $T = 1$ for $k = 10$ to the exact solution.

As shown in Fig. 6 (left), the loss functions on the training set and the testing set decrease faster using GreenONets, when compared to DeepONets, and the loss after 2500 is smaller. Figure 6 (middle) shows the pointwise error in the solution at $T = 1$ using GreenONets and DeepONets with an initial condition $u_0 = (1 - x^2)^{10}$. The maximum pointwise errors are similar for both networks with a value close to 0.035. In order to analyze the errors, we plot the solutions at $T = 1$ along with the exact solution in Fig. 6 (right). We remark that the errors in the solution of the GreenONet remain localized around the main pulses and are proportional to the wave amplitude. However, in the case of DeepONets, the pointwise errors have the tendency to spreak over the whole domain. Therefore, we may conclude that GreenONets tend to provide better approximations of the propagating waves without introducing large errors away from the main pulses.

5 Conclusions

We conclude that the Green operator networks show better results, when compared to the Deep operator networks, when approximating the wave operator for homogeneous and heterogeneous domains. We showed that the loss functions of the GreenONets always converge with fewer epochs. The numerical results also showed that the pointwise errors are generally smaller with GreenONets and that the solutions generalize better for higher frequencies. Finally, we observed that the errors were localized around the peak amplitudes with GreenONets while the errors with DeepONets were randomly distributed in the domain. For future work, we want to test the GreenONets for higher dimensions and for materials with various heterogeneous properties within the domain.

Acknowledgements. SP and ML are grateful for the support from the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grants [grant numbers RGPIN-2019-7154, PGPIN-2018-06592]. This research was also partially supported by an NSERC Collaborative Research and Development Grant [grant number RDCPJ 522310-17] with the Institut de Recherche en Électricité du Québec and Prompt.

Authorship statement. The authors hereby confirm that they are the sole liable persons responsible for the authorship of this work, and that all material that has been herein included as part of the present paper is either the property (and authorship) of the authors, or has the permission of the owners to be included here.

References

[1] X. Jin, S. Cai, H. Li, and G. E. Karniadakis. Nsfnets (navier-stokes flow nets): Physics-informed neural networks for the incompressible navier-stokes equations. *Journal of Computational Physics*, vol. 426, pp. 109951, 2021.
[2] A. Bihlo and R. O. Popovych. Physics-informed neural networks for the shallow-water equations on the sphere. *Journal of Computational Physics*, vol. 456, pp. 111024, 2022.
[3] C. L. Pettit and D. K. Wilson. A physics-informed neural network for sound propagation in the atmospheric boundary layer. In *Proceedings of Meetings on Acoustics 179ASA*, volume 42, pp. 022002. Acoustical Society of America, 2020.
[4] B. Moseley, A. Markham, and T. Nissen-Meyer. Solving the wave equation with physics-informed deep learning. arXiv preprint:2006.11894, 2020.
[5] 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. *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019.
[6] E. Weinan and B. Yu. The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, vol. 6, n. 1, pp. 1–12, 2018.
[7] J. Sirignano and K. Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, vol. 375, pp. 1339–1364, 2018.
[8] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, vol. 3, n. 3, pp. 218–229, 2021.
[9] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier neural operator for parametric partial differential equations. arXiv preprint:2010.08895, 2020a.
[10] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Neural operator: Graph kernel network for partial differential equations. arXiv preprint:2003.03485, 2020b.
[11] S. Wang, H. Wang, and P. Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed deeponets. *Science Advances*, vol. 7, n. 40, pp. eabi8605, 2021.
[12] D. G. Duffy. *Green's functions with applications*. Chapman and Hall/CRC, Boca Raton, FL, 2001.
[13] E. Kausel. *Fundamental Solutions in Elastodynamics - A Compendium*. Cambridge University Press, 2006.
[14] D. P. Kingma and J. Ba. ADAM: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.