Physics-Informed Machine Learning for Solving Initial-Boundary Problems

A. Schou, A. Borup, M. Carøe, L. Nicolaisen, V. Hansen
s194353, s194326, s194317, s194330, s194318

Introduction

Sound waves propagate in space and time according to the wave equation. To calculate realistic approximations to sound waves, one can solve the PDE problem using classical numerical methods such as FEM-solvers. However, to obtain solutions in real time, neural networks are an emerging popular alternative. To approximate one particular solution, a regular single neural network can be used. Another approach is to use DeepONets, which are Neural Networks that can approximate the operator mapping the initial condition to the solution of the wave equation. DeepONets can be made "physics informed" by incorporating the PDE and boundary conditions in the loss function. The derivatives of the inputs can be computed using automatic differentiation just as it is done for the weights of the neural network. Alternatively, one can train the DeepOnet on already solved problems and just use a regular MSE between the output and the solution.
We will investigate the following:

- A Physics informed neural network (PINN),
- A Physics informed DeepONet (PI-DeepONet),
- A data driven DeepONet trained with data from simulation.

Problem formulation

The PDE problem we consider is to solve is the homogeneous wave equation on the domain $\Omega = [-1,1] \times [0,\infty)$ with initial condition $u(x,0) = \psi(x)$, and homogeneous Neumann boundary conditions.

$$\begin{cases} \frac{\partial^2}{\partial t^2} u(x,t) - c^2 \frac{\partial^2}{\partial x^2} u(x,t) = 0 & \text{in} \operatorname{int}\Omega \\ u(x,0) = \psi(x), & -1 \le x \le 1\,, \\ \frac{\partial}{\partial t} u(x,0) = 0, & -1 \le x \le 1\,, \\ \frac{\partial}{\partial x} u(-1,t) = \frac{\partial}{\partial x} u(1,t) = 0, & t \ge 0. \end{cases} \tag{1}$$

The initial condition consists of a Gaussian source, $\psi(x) = \exp\left[-\left(\frac{x-x_0}{\sigma}\right)^2\right]$ or combinations thereof. With the approximate solution $\hat{u}$, the loss function for the wave equation can be written as

$$\begin{cases} \mathcal{M}_\Omega(\Theta) = \frac{1}{N_f} \sum_{i=1}^{N_f} \left| \frac{\partial^2}{\partial t^2} \hat{u}(x_i,t_i) - c^2 \frac{\partial^2}{\partial x^2} \hat{u}(x_i,t_i) \right|^2 \\ \mathcal{M}_{\mathrm{IC}}(\Theta) = \frac{1}{N_i} \sum_{i=1}^{N_i} \left(\left| \hat{u}(x_i,0) - \psi(x_i) \right|^2 + \left| \frac{\partial}{\partial t} \hat{u}(x_i,0) \right|^2 \right) \\ \mathcal{M}_{\partial\Omega}(\Theta) = \frac{1}{N_g} \sum_{i=1}^{N_g} \left| \frac{\partial}{\partial x} \hat{u}(x_i,t_i) \right|^2 \end{cases} \tag{2}$$

The total loss function is the weighted sum $\mathcal{M}_{total} = \beta_\Omega \mathcal{M}_\Omega + \beta_{IC} \mathcal{M}_{IC} + \beta_{\partial\Omega} \mathcal{M}_{\partial\Omega}$ with $\beta_\Omega, \beta_{IC}, \beta_{\partial\Omega} > 0$.

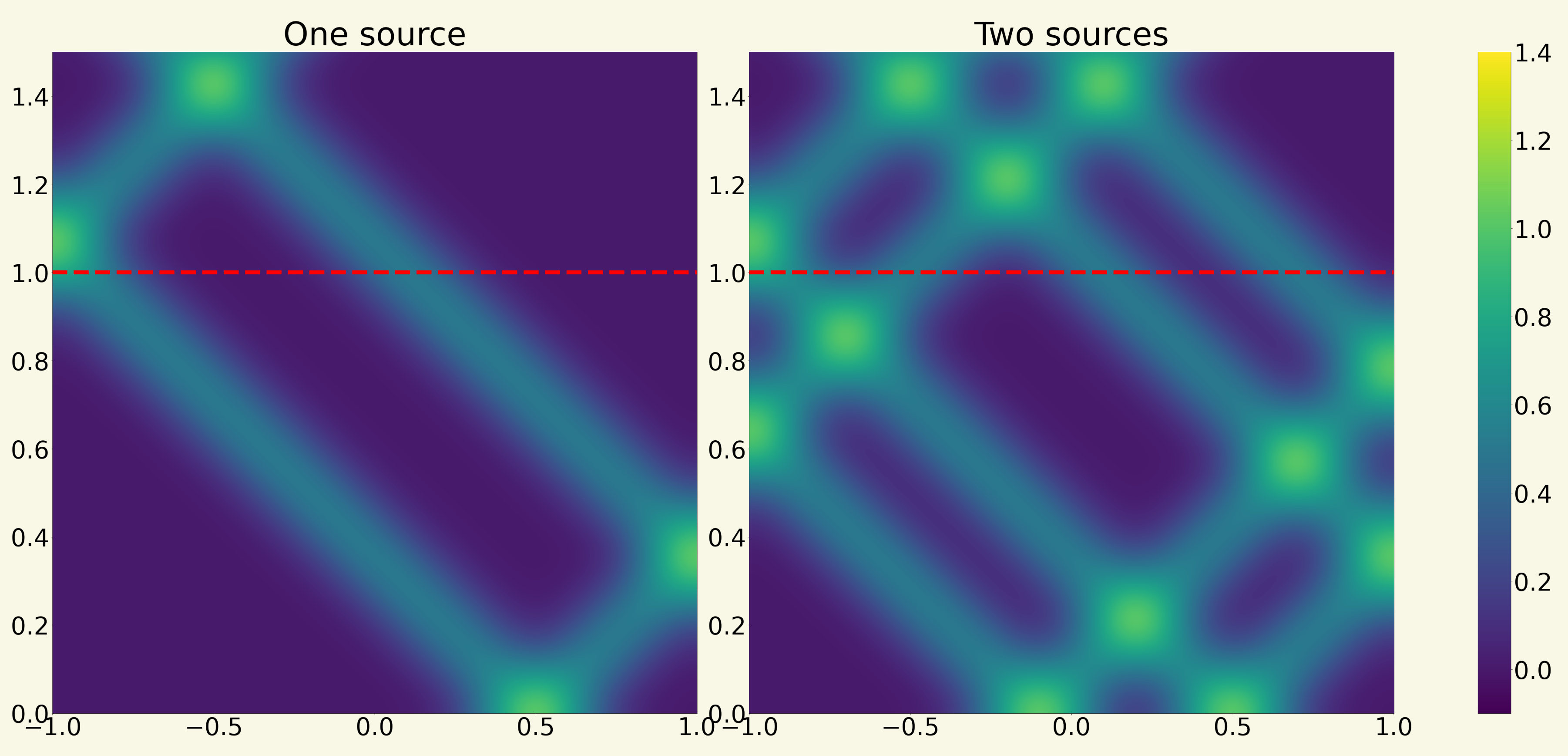

Figure: True solution with $c = 1.4$ and $\sigma = 0.2$. For the one-source plot, $x_0 = 0.5$. For the two-source plot, $x_0 \in \{-0.1, 0.5\}$. In all cases, the horizontal axis is spacial x, and the vertical axis is temporal t. Anything above the red line indicates extrapolated data.

PINNs

We start by solving the PDE by training a PINN, which is a simple neural network using the loss function $\mathcal{M}_{total}$, using Gaussian initial conditions. Including physics comes with a few caveats, it requires a smooth activation function, so we use Tanh instead of Relu, which makes our networks more prone to problems such as vanishing gradients.
The network trains on uniformly sampled points in $\operatorname{int}\Omega$ and on $\partial\Omega$ for points (x,t) with $t \le 1$. We use a PINN with 2 layers each with 32 neurons. When the network is evaluated outside of its training domain, i.e. (x,t) with $1 < t \le 1.5$ the error increases substantially, but since it has been trained using physics, there is still some structure. The biggest drawback with the PINN is that for each initial condition we have to re-train the network. Thus we propose using the DeepONet architecture.

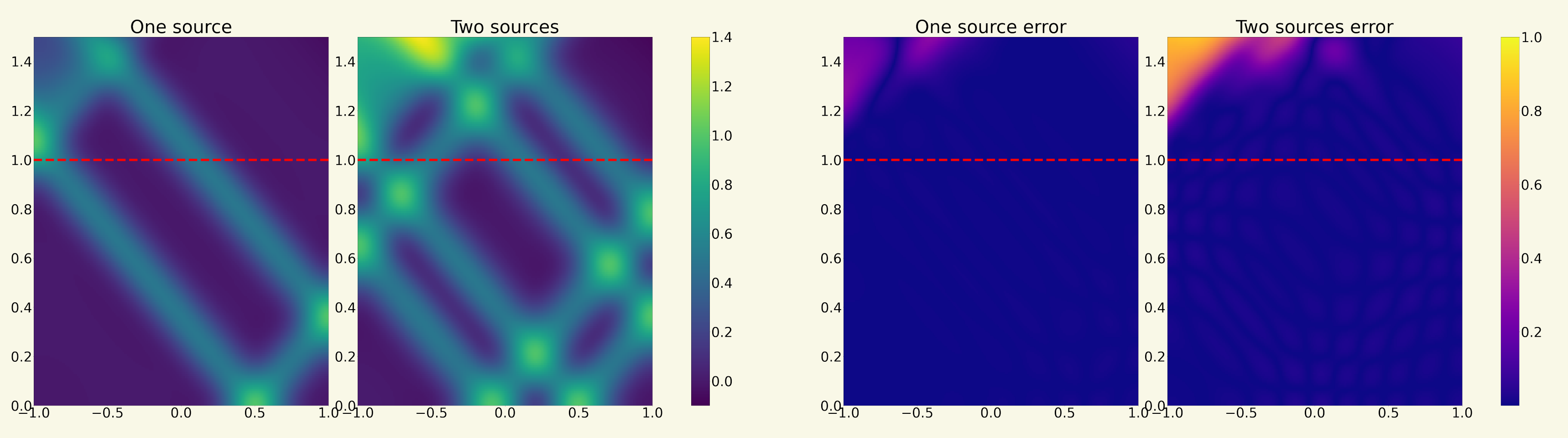

Figure: PINN with the same initial conditions as the true solution above. For one source the maximum error is 0.016 and the training time is 25 minutes. For two sources it is 0.025 and 45 minutes.

DeepONets

The DeepONet construction G ideally takes as input the initial condition $\psi(x)$ evaluated at the points $x^{(1)}, \ldots, x^{(m)} \in [-1,1]$ and some probes $(x_i, t_i) \in \Omega$, and outputs the function values $u(x_i, t_i)$. Thus we have

$$G(\psi)(x,t) = u(x,t). \tag{3}$$

In other words, the DeepONet approximates the map

$$\psi \mapsto G(\psi) = u. \tag{4}$$

The network architecture is shown in the figure below. A "branch" network B takes as input the m points $\psi\left(x^{(i)}\right)$ and output the numbers β_k, with $k = 1, \ldots, p$. A "trunk" network takes a point $(x,t) \in \Omega$ as input and output the numbers τ_k, with $k = 1, \ldots, p$. The dot-product is then calculated

$$G(\psi)(x,t) = \sum_{k=1}^{p} \beta_k \tau_k. \tag{5}$$

The weights for the branch and trunk networks are found by minimizing the loss-function. In the physics-informed case, the loss function is given by (2). In the data-driven case, the loss function is the MSE from a "ground truth" calculated using a classical numerical method.

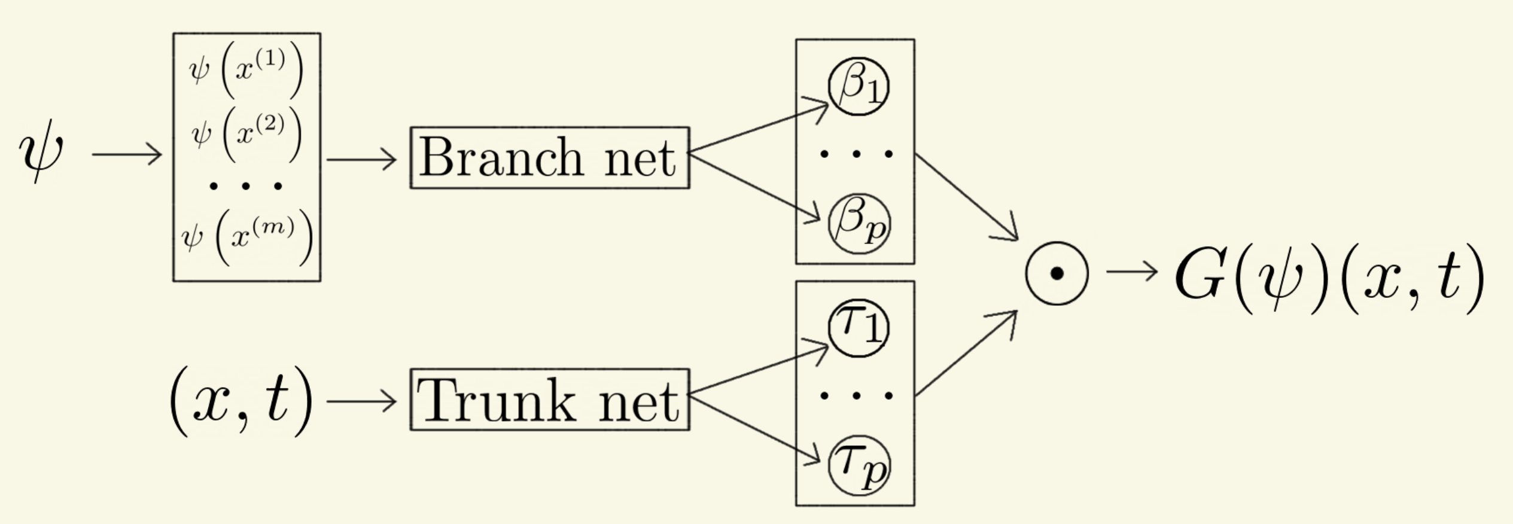

Figure: DeepONet structure, where $\odot$ is the dot product

To train the DeepOnet, we need a lot of different initial conditions. We sample functions from a Gaussian Random Field

$$\psi \sim \mathcal{G}\left(0, \exp\left(-\|x_1 - x_2\|^2 / (2l^2)\right)\right), \tag{6}$$

which means that the covariance between two points x_1 and x_2 depends on the distance between them, thus we get relatively smooth functions, depending on the length scale l, we use $l = 0.2$. We sample 3000 and 5000 functions for the data driven and physics informed DeepOnet respectively. In the data driven case we solve the problem using a Legendre Collocation method with Runge-Kutta time stepping.

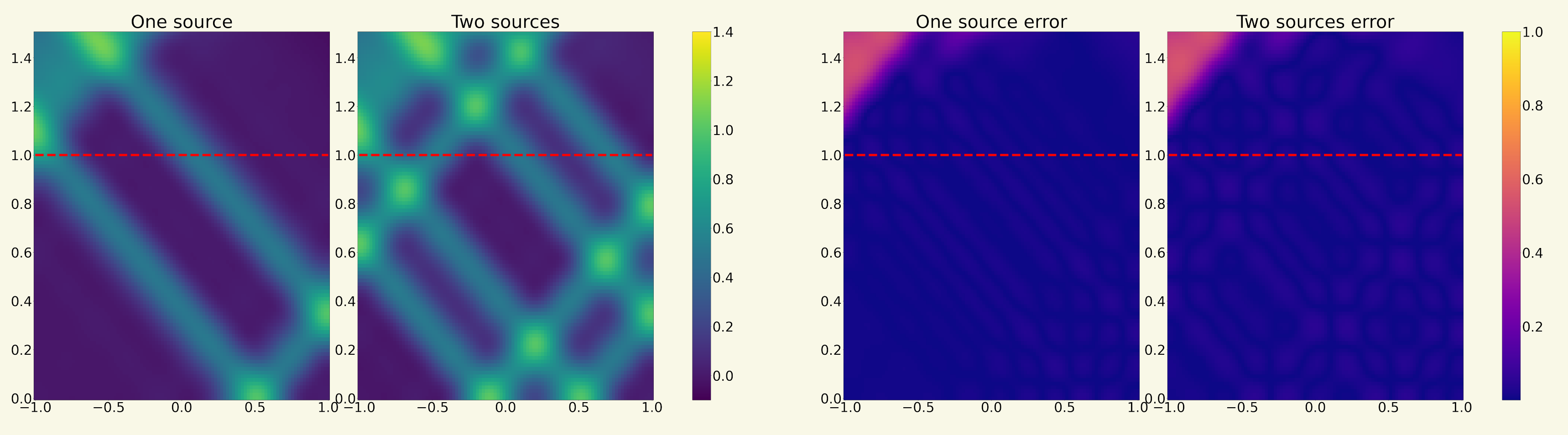

Figure: PI-DeepONet. Both the branch and trunk network has 3 layers with 64 neurons each and 64 outputs. Max error in the training domain is 0.041 for one source and 0.052 for two sources. Training time is about 70 minutes.

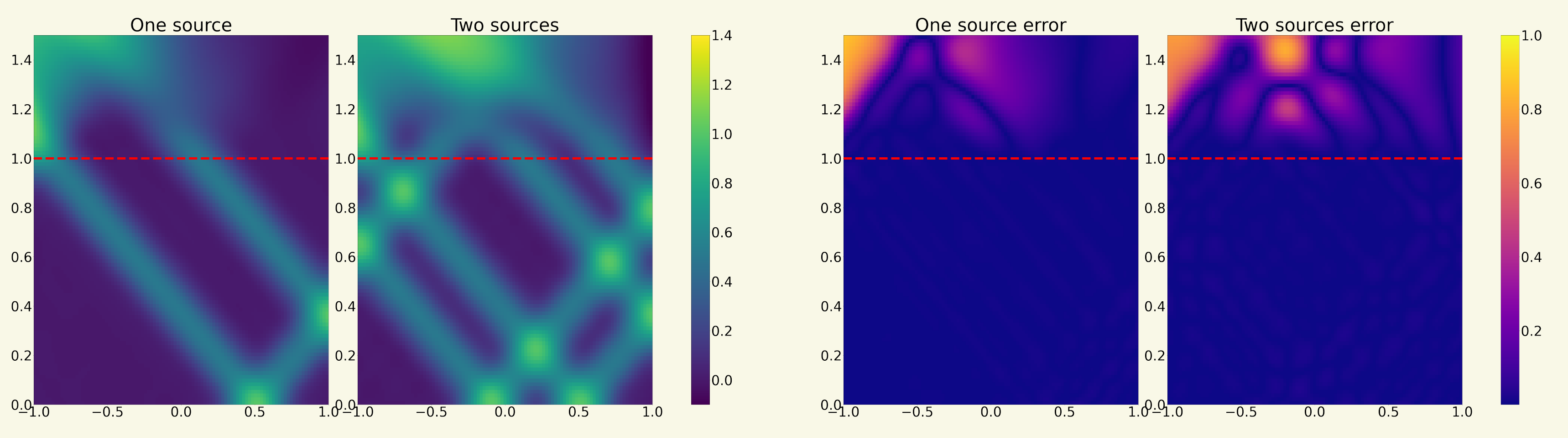

Figure: Data driven DeepONet. Both the branch and trunk network has 3 layers with 128 neurons each and 128 outputs. Max error is 0.026 and 0.033. Data generation and training took 15 minutes each.

Discussion and Conclusion

Considering the errors of the PINNs versus the DeepONets on the trained domain $t \in [0,1]$, we see that the PINNs in general obtain a lower error than the DeepONets do. However, it is necessary to train a new network for each initial condition, whereas the DeepONets approximates both the one source and the two source initial condition. Thus, it makes sense that the PINN that is trained to solve a single problem is better at solving this problem than a network that is trained to generalize to any initial condition.
Considering extrapolation of the networks, the PI-DeepONet is significantly better than the Data driven DeepONet. This is also as expected, as the Data driven DeepOnet has no information about the PDE. For the two source initial condition, it is also noticeable that the PI-DeepONet is better at extrapolating data than the PINN. Extrapolation is a known issue for PINNs, which the DeepONet structure seems to be better at.
Thus, when considering a fixed initial condition, then PINN serves as a nice tool for getting good results. However, if the initial condition for problem is unknown, and/or extrapolation is desired, the DeepONet structure is proposed as a superior solution.

References

[1] N. Borrel-Jensen et al. *DeepONet: Learning the sound propagation in 1D with parametrized sources using deep learning for approximating the wave equation with operators*, 2022
[2] Lu Lu et al. *DeepONet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators*, 2020
[3] S. Goswami et al. *Physics-Informed Deep Neural Operator Networks*, 2022