DOI: 10.61782/fa.2023.0930

# A SENSITIVITY ANALYSIS ON THE EFFECT OF HYPERPARAMETERS IN DEEP NEURAL OPERATORS APPLIED TO SOUND PROPAGATION

**Nikolas Borrel-Jensen**[1*] **Allan P. Engsig-Karup**[2]
**Cheol-Ho Jeong**[1]
[1] Technical University of Denmark, Dep. of Electrical and Photonics Engineering, DK
[2] Technical University of Denmark, Dep. of Applied Mathematics and Computer Science, DK

## ABSTRACT

Deep neural operators have seen much attention in the scientific machine learning community over the last couple of years due to their capability of efficiently learning the nonlinear operators mapping from input function spaces to output function spaces showing good generalization properties. This work will show how to set up a performant DeepONet architecture in acoustics for predicting 2-D sound fields with parameterized moving sources for real-time applications. A sensitivity analysis is carried out with a focus on the choice of network architectures, activation functions, Fourier feature expansions, and data fidelity to gain insight into how to tune these models. Specifically, a default feed-forward neural network (FNN), a modified FNN, and a convolutional neural network will be compared. This work will de-mystify the DeepONet and provide helpful knowledge from an acoustical point of view.

**Keywords:** *neural operators, sensitivity analysis, virtual acoustics, DeepONet*

## 1. INTRODUCTION

Deep learning has seen rapid development over the last 20 years, with a many-fold of applications such as image classification and computer vision, speech recognition, language translation, autonomous driving, bioinformatics, chatbots, and more. More recently, scientific machine-learning (SciML) methods have been proposed in the context of deep learning. Physics-informed neural networks were introduced in 2017 [1] inspired by [2] and have already seen many applications [3, 4]. Typical for most of these techniques is that they are based on the assumption of function approximations of neural networks [5]. In 2019, the DeepONet was introduced [6], extending a theorem on the universal operator approximation for a single-layer neural network [7] to hold for deep neural networks. Moreover, the original architecture was improved to exhibit small generalization errors. Unlike function regression, operator regression aims to learn the mapping from one function space (inputs) to another function space (output), where the learned operator can be evaluated at arbitrary (continuous) locations. Another work on operator learning is the Fourier Neural Operator (FNO) introduced in 2021 [8], based on parameterizing an integral kernel directly in the Fourier space.

In this work, we will apply the DeepONet to approximate the wave equation operator for frequency-independent boundary conditions and parameterized source positions similar to [4] in 2-D for virtual acoustics applications such as computer games, AR/VR, and metaverses. The impulse response (IR) has traditionally been calculated using numerical methods. However, when handling moving sources, this approach gets challenging both from a computational and storage point of view since the IRs have to be re-calculated offline for each source position and the IRs stored for each source/receiver pair. This gets intractable when approaching the full frequency range.

It is vital to apply various techniques and fine-tune the hyperparameters for the DeepONet to exhibit fast

*[*] **Corresponding author**: nibor@dtu.dk.*

convergence with low generalization errors. This work aims to perform a sensitivity w.r.t. the network architecture, data resolution, batch size, activation functions, and Fourier feature expansion techniques to better understand the workings of the DeepONet when applied in an acoustical context.

## 2. METHODS

### 2.1 Deep operator network (DeepONet)

DeepONet [6] is a general deep learning framework for approximating continuous operators contrary to continuous functions. The underlying theory stems from the universal operator approximation theorem [7], stating that a neural network (NN) with a single hidden layer of infinite width can approximate any nonlinear continuous functional or operator. Let $G$ be the operator we want to learn using NNs, defined as $G : u \mapsto G(u)$, where $u$ is the input function to $G$ and $G(u)$ is the output function. For any point, $y$ in the domain of $G(u)$, $G(u)(y) \in \mathbb{R}$ is producing a real number. Translating this into a NN setting, the network takes two inputs, $u$ and $y$, and outputs $G(u)(y)$. The input function is discretized by evaluating $u$ at a finite number of points $\{x_i\}$ called 'sensors.' The approximation theorem by Chen and Chen considers shallow networks and only guarantees small approximation errors but does not consider generalization and optimization errors. In [6], the authors extended the original theorem by proposing deep neural networks instead of shallow networks and proved that the network is also universal approximators for operators. The proposed deep operator network, DeepONet, achieves small total errors, including approximation, optimization, and generalization errors. The DeepONet architecture consists of two subnetworks, the 'branch net' for the input functions and the 'trunk net' for the locations to evaluate the output function $G(u)$. The trunk network takes $y$ as input and outputs $[T_1, T_2, \ldots, T_p] \in \mathbb{R}^p$; the branch network takes $[u(x_1), u(x_2), \ldots, u(x_m)]^T$ at fixed sensors $\{x_1, x_2, \ldots, x_m\}$ and outputs a scalar $B_k \in \mathbb{R}$ for $k = 1, 2, \ldots, p$. By merging the trunk and branch in terms of their inner product, we get

$$G(u)(y) \approx \sum_{k=1}^{p} \underbrace{B_k\left(u(x_1), u(x_2), \ldots, u(x_m)\right)}_{\text{branch}} \underbrace{T_k(y)}_{\text{trunk}} + b_0, \tag{1}$$

where $b_0$ is a trainable bias.

### 2.2 Network architectures

The DeepONet framework allows many network architectures, such as feed-forward neural networks (FNN), recurrent neural networks (RNN), convolutional neural networks (CNN), graph neural networks (GNN), and convolutional graph neural networks (CGNN). Since the inputs to the trunk net are the (continuous) locations on which the operator's output is evaluated, the dimensionality is typically low. In contrast, the input to the branch net is a function sampled at $m$ sensor locations and is typically high-dimensional. A common choice for the trunk network is to use an FNN, whereas FNNs and CNNs have been the most common architectures for the branch net. The latter choice is because of the high-dimensional nature of the input functions, making the CNNs particularly useful in many applications due to their ability to map higher-dimensional spaces to lower-dimensional spaces by extracting the essential features.

In this work, we compare the performance between using a CNN and an FNN network for the branch net of the DeepONet, while keeping the trunk net FNN architecture fixed.

#### *2.2.1 Feed-Forward Neural Network*

An FNN consists of an input layer $\mathbf{x}$, $n$ hidden layers, and an output layer and maps an input $\mathbf{x}$ to an output $\mathbf{y}$ as

$$\mathbf{y} = (f_0 \circ f_1 \circ \ldots \circ f_n)(\mathbf{x}), \tag{2a}$$

$$f_i(\mathbf{x}) = \sigma_i(\mathbf{W}^i\mathbf{x} + \mathbf{b}^i), \tag{2b}$$

where $\sigma_i(\mathbf{x})$ is a non-linear activation function, except for the last layer, where we are applying the identity mapping $\sigma_n(\mathbf{x}) = \mathbf{x}$. The weights $\mathbf{W}^i$ and biases $\mathbf{b}^i$ are the parameters to learn. A multilayer perceptron (MLP) is a special case of an FNN, where every layer is fully connected, and the number of nodes in each layer is the same. In this work, we have used the gradient descent optimizer ADAM.

A modification to the MLP (mod-MLP) was proposed in [9] and has been shown to outperform the conventional FNNs also in conjunction with DeepONet [10]. The key extension is the introduction of two encoder networks encoding the input variables to a higher-dimensional feature space. The networks consisting of a single layer are shared between all layers, and a pointwise multiplication operation is performed to update the hidden layers. Let the two transformer networks be denoted $u(\mathbf{x})$ and $v(\mathbf{x})$ and defined as a simple perceptron

$$u(\mathbf{x}) = \sigma(\mathbf{W}_u\mathbf{x} + \mathbf{b}_u), \tag{3a}$$

$$v(\mathbf{x}) = \sigma(\mathbf{W}_v\mathbf{x} + \mathbf{b}_v), \tag{3b}$$

then the mod-MLP is defined as

$$\begin{aligned}\mathbf{y} = {} & ((1 - f_0) \odot u + f_0 \odot v) \circ \\ & ((1 - f_1) \odot u + f_1 \odot v) \circ \\ & \vdots \\ & ((1 - f_n) \odot u + f_n \odot v)(\mathbf{x}),\end{aligned} \tag{4}$$

where $\odot$ denotes elementwise multiplication, and $\mathbf{W}_{\{u,v\}}$ and $\mathbf{b}_{\{u,v\}}$ are the weights and biases for the two transformer networks.

### 2.2.2 *Convolutional Neural Network*

Convolutional neural networks (CNN) are special networks for processing data with a grid-like topology [11] and use convolution instead of matrix multiplication in one or more layers. In traditional MLPs, each output unit interacts with each input unit through weight parameters describing the interaction. In contrast, CNNs typically have sparse interactions by applying a convolution kernel (much) smaller than the input dimension. Moreover, the convolutional kernel is used for every input position, meaning the parameters are shared. Aside from reducing the storage requirement, it also causes the layer to have equivalence to translation.

Although the networks in our work are not particularly deep, we will use the ResNet architecture [12]. Several ResNet blocks assemble the ResNet. A ResNet block consists of two stacked CNNs with skip connections and batch normalization; one or more ResNet blocks comprise a group (all with the same output shape), and one or more groups connect the final ResNet. Downsampling takes place in the first block of each group by increasing the strides, and the output channel dimension is increased, forcing the CNN to capture essential features in separate channels. We will use the notation ResNet-$\{$gr1,gr2,gr3,...$\}$, where the element counts inside the square brackets denote the number of groups, and the values denote the number of blocks inside the group indexed by its position. The hidden channel layers are denoted $\{$ch1,ch2,ch3,ch4,...$\}$, where each element index corresponds to the ResNet group with the same index. E.g., when we refer to a ResNet-$\{$2,2,2,2$\}$ with hidden channel layers $\{$8,16,32,64$\}$, it denotes a ResNet having 4 groups of 2 blocks each with 8, 16, 32, and 64 channels for each of the groups, respectively.

## 2.3 Activation functions

We will compare the convergence using the Rectified Linear Unit (`relu`) $\hat{x} \mapsto \hat{x}^+$, the hyperbolic tangent (`tanh`) $\hat{x} \mapsto (e^{\hat{x}} - e^{-\hat{x}})/(e^{\hat{x}} + e^{-\hat{x}})$ and sinusoidal (`sine`) $\hat{x} \mapsto \sin(x)$. The latter is a less common choice but has shown superior convergence for wave propagation problems [4]. Weight initialization is a significant step before training the model and depends on the activation function used. Glorot initialization [13] is used for `relu` and `tanh`, whereas the initialization advised in [14] is used for the `sine` activations. Data normalization is done in the spatial dimensions ($[-1, 1]$ for `sine` and `tanh` and $[0, 1]$ for `relu`), where the temporal dimension is normalized with the spatial normalization factor to ensure equal resolutions in all dimensions.

## 2.4 Loss functions

The mean-squared error (MSE) will be used in all experiments and is the default loss for function regression under the inference framework of maximum likelihood when the target variable is assumed Gaussian

$$\mathrm{MSE} = ||\hat{\mathbf{y}} - \mathbf{y}||^2. \tag{5}$$

The MSE will punish large values more and is a good choice when outliers are less pronounced, which is satisfied for our synthetic training data.

## 2.5 Data resolution

Deep learning methods typically require large data sets for proper training, which also applies when training the DeepONet. Since the training data for our simulations are created synthetically from high-fidelity spectral-element method (SEM) simulations [15], we can, in principle, create any training data sample set utilizing a simulator. However, it is crucial to find a lower bound since generating data quickly gets intractable when enlarging the domain and/or simulating a broader frequency range – especially when going higher dimensions due to the curse of dimensionality. More extensive training data sets also impact the training time and hardware requirement. Also, training a deep neural network on an unnecessary fine-grained data set is not always advantageous, as we will see.

We can estimate a theoretical lower bound of the sampling rate of the discretized wave-propagation data from the Nyquist Theorem, stating that a given band-limited continuous-time signal can be perfectly reconstructed from its discrete-time signal by ensuring that the continuous signal is sampled using at least two points per wavelength. The number of spatial points for one dimension is calculated as

$$N_{\text{samples}} = \left\lceil \frac{L}{c/(f_{\max} \times \text{ppw})} \right\rceil, \tag{6}$$

where ppw is the number of points per wavelength, and $L$ is the dimension length. Though the sufficient Nyquist sampling rate can be used in theory, the sampling rate for the DeepONet to generalize well and the optimizer to find meaningful optima might require oversampling. The data and the corresponding resolutions for the DeepONet can be divided into several parts with eventually different resolution requirements.

Firstly, we consider the resolution of the sample functions representing the initial conditions used as input to the branch net. Remember that the sensors (the sampling points for each sample function) must be located at equal locations across all the sample functions. Therefore, we cannot exploit any sensor location distribution favoring important samples (e.g., non-zero Gaussian sensors) because the source should be allowed to move freely inside the domain, hence a uniform sensor distribution is the best option. We will investigate the sensitivity from the total number of sensors $m$ in (1) calculated using (6).

Secondly, the spatial/temporal coordinate inputs to the trunk net are considered. Contrary to the sample functions, there are no restrictions on choosing the coordinate distributions for different samples, which enables us to learn the continuous operators of the solution. Various sampling techniques have been explored chiefly for physics-informed neural networks [16]. However, when a training data set with a given distribution has to be manipulated, weighting and re-distributing important points to where the physics is most important (e.g., when the gradient is relatively large or when the field is non-zero) becomes more involved and could require substantial computational efforts for realistic time-dependent problems in complex domains. In this work, we will keep the data distribution determined by the Gauss-Lobotto nodes from the fourth-order Lagrange polynomials on a non-uniform grid used in our SEM solver [15] and instead randomly sample data points corresponding to different grid resolutions for building up the training and validation data sets.

Thirdly, we will investigate the impact of the relative resolution between the temporal and spatial dimensions. From numerical theory, the relation between the temporal resolution $\Delta t$ and the spatial resolution $\Delta x$ is given by $\Delta t = \lambda \frac{\Delta x}{c}$ ensuring that the distance the wave has traveled after one time step $\Delta t \times c$ is no longer than the spatial resolution $\Delta x$ required to resolve the wave propagation. In numerical theory, $0 < \lambda \leq 1$ is the so-called Courant-Friedrichs-Lewy stability condition (CFL) that dictates numerical stability for explicit time-stepping schemes and should be set such that the chosen method is numerically stable. There is no such restriction in deep neural networks as training these relies on global optimization across the parameter domain, and the CFL can be set to the maximum value. In fact, the temporal and spatial resolutions can be set independently as long as the frequencies are well resolved according to the Nyquist Theorem. Hence, the temporal and spatial resolutions can be chosen freely as

$$\Delta x = \frac{c}{f_{\max} \times \text{ppw}_x} \tag{7a}$$

$$\Delta t = \frac{1}{f_{\max} \times \text{ppw}_t}, \tag{7b}$$

with ppw $\geq 2$. For the gradient descent to find meaningful optima, the speed of sound $c = 1$ m/s is used to ensure equal resolutions for all dimensions.

Lastly, the initial source sample density can be determined using (6). The Nyquist Theorem would tell us the minimum distance between source positions to reconstruct the signal as the source moves freely. However, a finer resolution might be needed for the network not to overfit, as we will investigate in the ‘Experiments’ section.

### 2.6 Fourier feature expansions

It is well-known that deep neural networks first learn the lower frequency modes of the data and suffer from learning the higher frequency modes. This phenomenon is known as spectral bias [17, 18]. This problem can, to some extent, be overcome by passing the temporal and spatial coordinates through a Fourier feature mapping that enables a deep FNN to learn the high-frequency modes of the data. In this work, we will experiment with the ‘Positional encoding mapping’ and the ‘Gaussian mapping’ from [19]. A third ‘ES mapping’ has been constructed from an analytical solution. In the following, $m$ is the number of feature expansion terms, and $d$ is the dimension of the input data:

**Positional encoding mapping:**

$$\gamma(\mathbf{x}) = \left[\ldots, \cos\left(2\pi f_j \mathbf{x}\right), \sin\left(2\pi f_j \mathbf{x}\right), \ldots\right]^T, \quad \text{for } j = 0, \ldots, m-1. \tag{8}$$

The frequencies $f_j$ can be log-linear spaced along each dimension or arbitrarily relative to the fundamental frequency $f_0 \in \mathbb{R}^+$.

**Gaussian mapping:**

$$\gamma(\mathbf{x}) = [\cos(2\pi \mathbf{B}\mathbf{x}), \sin(2\pi \mathbf{B}\mathbf{x})]^T, \tag{9}$$

where each entry in $\mathbf{B} \in \mathbb{R}^{m \times d}$ is sampled from $\mathcal{N}(0, f_0^2)$ and $f_0 \in \mathbb{R}^+$ is a fundamental frequency empirically chosen.

**ES mapping:**

$$\gamma(x, y, t) = \left[\ldots, \cos(\Omega t) \cos\left(\frac{\pi n_j}{L_x} x\right) \cos\left(\frac{\pi n_j}{L_y} y\right) \times \cos\left(\frac{\pi n_j}{L_y} y\right), \ldots\right]^T, \text{ for } j = 0, \ldots, m-1, \tag{10}$$

where $n_i \in \mathbb{Z}$ are the wave modes empirically chosen and $\Omega^2 = c^2\pi^2[(d_x/L_x)^2 + (d_y/L_y)^2 + (d_z/L_z)^2]$. The equation is an exact solution to the 3-D wave equation in a rectangular domain of size $L_x \times L_y \times L_z$ with perfectly reflecting boundaries. Sine terms can also be included in the expansion.

## 3. EXPERIMENTS

We will perform a sensitivity analysis on the convergence and accuracy by varying the DeepONet setup. Evaluating all permutations of all choices will be too exhaustive. Instead, we will determine a base model setup and investigate the sensitivity of various choices covering the impact of 1) activation functions, 2) Fourier feature expansions, 3) mod-MLP networks, 4) batch size, 5) the number of layers and neurons, 6) data resolution, and 7) using a CNN compared to mod-MLP for the branch net.

The base model uses the default MLP with 4 layers of width 1024 for both the branch and trunk net. The ADAM optimizer with learning rate 1e-3 and exponential decay is used together with the MSE loss. The batch size is 64 for the branch net and 100 for the trunk net with 2 `ppw` for the branch net input functions (initial condition), 2 `ppw` for the trunk net temporal dimension, 6 `ppw` for the trunk net spatial dimensions, and the source position density is sampled using 6 `ppw`.

To evaluate the learned model's generalization properties, we ensure that the grid points are (mostly) non-overlapping in the training and validation data sets. The training and validation data have been generated using our SEM solver. The training data is generated using 6 `ppw` for the spatial resolution and 6 `ppw` for the source density corresponding to 1363 source positions. The validation data is generated using a spatial resolution of 5 `ppw` with 5 and 33 source positions.

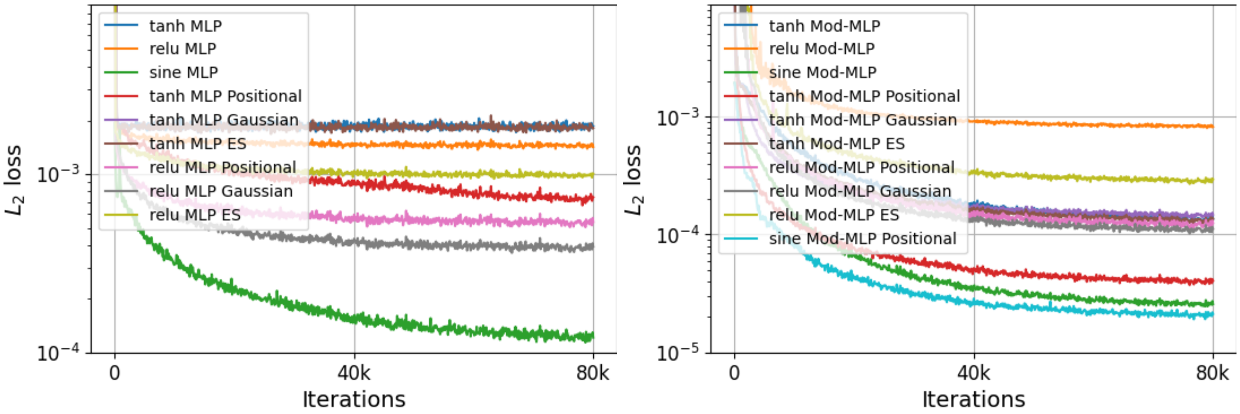
### 3.1 Activation function

As a first investigation, we will compare the performance using the `tanh`, the `relu`, and the `sine` activation functions for the default MLP architecture with and without Fourier feature expansion. Applying the Fourier feature expansion when the `sine` activation functions are used would be redundant and also show degraded performance (not shown). The $L_2$ training loss is depicted in Figure 1a (the validation loss is considered in later experiments). We notice that `relu` and `tanh` activation functions are not performing well without Fourier feature expansions. Applying the Fourier feature expansions improves the learning significantly, with the best results obtained for the `relu` activation function with Gaussian expansions. However, using the `sine` activation function outperforms the other choices by a large margin.

In Figure 1b, the same experiments are performed but for the mod-MLP network. We see dramatic improvements for all activations, especially for the `sine` activation choice showing almost an order of magnitude lower $L_2$ errors. We notice a slight improvement for the `sine` activation when combining the mod-MLP with the positional encodings. It is also interesting to note that the `tanh` activation performs well when positional encodings are applied using the mod-MLP architecture. In the following experiments, we will use the `sine` activation function with positional encodings.

### 3.2 Batch size

Here, we will investigate the impact of the batch size on the network optimizer to find good optima. The validation data was generated for 5 source positions. In Table 1, the $L_2$ errors are shown when varying the batch sizes of the branch net function samples from 16 to 96 and the spatial/temporal coordinates for the outputs from 100 to 800. We note that the batch size for the branch net should have size 64 or 96, with corresponding batch sizes of 200 or more for the trunk net. The batch size impacts the computational effort, and a good compromise could be to use a batch size of 64/200 for the branch and trunk net, re-



(a) Default MLP architecture.

(b) Modified MLP architecture.

**Figure 1**: $L_2$ validation loss for combinations of activation functions and Fourier feature expansion types. The mod-MLP with `sine` activations dramatically outperforms the other combinations.

**Table 1**: Batch size experiments using `sine` activation functions. The $L_2$ training/validation errors are given for each batch size combination, where the total batch size is a multiple of the branch net (BN) sample size and the temporal/spatial batch size for the trunk net (TN). The region with the lowest $L_2$ errors is highlighted.

Batch sizes branch/trunk net

| | BN | Batch size | | | |
|---|---|---|---|---|---|
| | TN | 16 | 32 | 64 | 96 |
| | | 1e-6× | 1e-6× | 1e-6× | 1e-6× |
| Batch size | 100 | 7.1/7.2 | 3.9/5.2 | 3.0/3.6 | 2.5/3.4 |
| | 200 | 3.9/5.1 | 2.7/3.3 | **2.2/2.9** | **2.4/2.6** |
| | 400 | 3.1/4.3 | 2.0/3.1 | **2.0/2.9** | **2.1/2.9** |
| | 800 | 2.3/3.7 | 1.9/3.1 | **1.9/2.8** | **2.0/2.9** |

**Table 2**: $L_2$ errors for varying the number of layers (L) and neurons (N) using the sine activation function.

Layers/neurons architectures

| L \ N | | 512 | 1024 | 2048 |
|---|---|---|---|---|
| 2 | t | 1.9e-5 | 2.8e-6 | **1.6e-6** |
| | v | 1.9e-5 | 4.4e-6 | **3.2e-6** |
| 3 | t | 6.8e-6 | **2.0e-6** | **1.5e-6** |
| | v | 7.6e-6 | **3.0e-6** | **3.0e-6** |
| 4 | t | 4.6e-6 | **2.0e-6** | **1.8e-6** |
| | v | 5.4e-6 | **2.9e-6** | **2.4e-6** |
| 5 | t | 4.2e-6 | **2.0e-6** | **1.6e-6** |
| | v | 4.8e-6 | **2.9e-6** | **2.4e-6** |

spectively, which we have chosen in the following experiments.

### 3.3 Number layers and neurons

Table 2 shows the $L_2$ errors when varying the network depth from 3 to 5 and the network width from 512 to 2048. We observe that wider networks are more critical for achieving good results than deeper ones. Using `sine` activation function can be seen as a Fourier series expansion [20], which could be the reason for the better performance using wider networks. Applying wider networks is more computationally expensive; therefore, choosing a compromise with more layers with fewer neurons can be necessary. We will use the wider layers with 2048 neurons and 2 layers, although slightly better results are obtained with 4 layers.

### 3.4 Data resolution

The validation data for the following experiments includes 33 source positions instead of the five source position for the previous two experiments to get a more truthful picture of generalization when varying the data resolutions. In Subtable 3a, results are shown for combinations of function resolutions for the branch net of 2 and 4 `ppw` with spatial data resolutions for the trunk net of 2, 4, and 6 `ppw`

**Table 3**: Data resolution for a 2/2048 layers/neurons MLP for the branch and trunk net and batch size 64/200.

Relative resolution branch/trunk net

| BN \ TN | | PPW 2 | PPW 4 | PPW 6 |
|---|---|---|---|---|
| PPW | 2 | 1.5e-6/1.1e-4 | **1.9/6.1e-6** | **1.9/5.5e-6** |
| | 4 | 1.6e-6/6.3e-5 | 1.9/6.0e-6 | 1.7/5.1e-6 |

(a) Data resolution combinations for the branch net (BN) and trunk net (TN).

Relative resolution src density/space/time

| PPW | | | | |
|---|---|---|---|---|
| $\Delta x_{\text{srcpos}}$ | $\Delta x$ | $\Delta t$ | train | val |
| 2 | 6 | 2 | 1.5e-6 | 1.7e-5 |
| 3 | 6 | 2 | 1.7e-6 | 7.5e-6 |
| 4 | 6 | 2 | 1.7e-6 | 6.1e-6 |
| 5 | 6 | 2 | **1.7e-6** | **5.0e-6** |
| 6 | 6 | 2 | **1.8e-6** | **5.3e-6** |
| 6 | 6 | 4 | **2.0e-6** | **4.8e-6** |

(b) Impact from the source density on the generalization properties.

while keeping the temporal resolution fixed at 2 ppw. We notice severe overfitting when only 2 ppw are used for the spatial resolution, with 6 ppw giving the best results. No significant impact is observed when varying the input function resolution of the branch net. In Subtable 3b, the influence from the sampling density of the source positions $\Delta x_{\text{srcpos}}$ is shown, and we notice the least overfitting when 5 or 6 ppw are used. We also note that using 4 ppw for the temporal resolution is not increasing the accuracy by much. This is true since we are overfitting to the uniform temporal steps when only using 2 ppw, which is not a problem since temporal interpolation is not of practical interest.

### 3.5 Convolutional neural network for the branch net

Finally, we will compare two ResNet architectures, the ResNet-{3,3,3,3} with {16,32,64,128} hidden channels for input functions of size 18 × 18 sampled with 2 ppw and the ResNet-{3,3,3,3,3} with hidden channels {16,32,64,128,256} for input functions of size 35 × 35 sampled with 4 ppw. The standard ReLU activation function is used for the CNN architecture. A single linear output MLP layer is used with sine activation functions. The convergence of the loss can be seen in Figure 2 plotted together with the mod-MLP for comparison and shows similar performance for all ResNet architectures on par with the mod-MLP architecture reference.

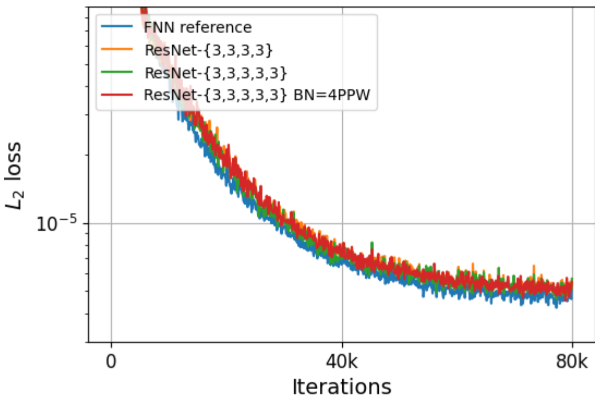


**Figure 2**: Validation loss for ResNet-{3,3,3} and ResNet-{3,3,3,3} for branch input function resolutions of 2 and 4 ppw.

## 4. CONCLUSION

We have systematically studied the DeepONet sensitivity for wave propagation problems, focusing on network architectures, data fidelity, and operator learning parameters. The most prominent choice for successfully training the model is to use the mod-MLP architecture in combination with sine activation functions. Moreover, the spatial data resolution greatly impacts the accuracy of the trained model, where 4-6 ppw for the spatial and 5-6 ppw for the source density resolutions are required for the network to generalize well for unseen source and receiver positions.

## 5. ACKNOWLEDGMENTS

Thanks to DTU Computing Center GPULAB for access to GPU clusters and especially to Sebastian Borchert for his assistance. Also, thanks to Finnur Pind for access to an SEM code for data generation. Thanks to George Em Karniadakis for an inspiring stay with the CRUNCH group.

## 6. REFERENCES

[1] 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 equa-

tions," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019.

[2] I. Lagaris, A. Likas, and D. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 987–1000, 1998.

[3] S. Cai, Z. Mao, Z. Wang, M. Yin, and G. E. Karniadakis, "Physics-informed neural networks (pinns) for fluid mechanics: a review," *Acta Mechanica Sinica*, vol. 37, no. 12, pp. 1727–1738, 2021. http://arxiv.org/pdf/2105.09506.

[4] N. Borrel-Jensen, A. Engsig-Karup, and C.-H. Jeong, "Physics-informed neural networks for one-dimensional sound field predictions with parameterized sources and impedance boundaries," *Jasa Express Letters*, vol. 1, no. 12, 2021.

[5] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[6] "Learning nonlinear operators via deeponet based on the universal approximation theorem of operators," *Nature Machine Intelligence*, vol. 3, pp. 218–229, 2021.

[7] T. Chen and H. Chen, "Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems," *IEEE Transactions on Neural Networks*, vol. 6, no. 4, pp. 911–917, 1995.

[8] Z. Li, N. B. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. M. Stuart, and A. Anandkumar, "Fourier neural operator for parametric partial differential equations," *CoRR*, vol. abs/2010.08895, 2020.

[9] S. Wang, Y. Teng, and P. Perdikaris, "Understanding and mitigating gradient flow pathologies in physics-informed neural networks," *SIAM Journal on Scientific Computing*, vol. 43, no. 5, pp. A3055–A3081, 2021.

[10] S. Wang, H. Wang, and P. Perdikaris, "Learning the solution operator of parametric partial differential equations with physics-informed deeponets," 2021.

[11] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[13] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. of the Thirteenth International Conference on Artificial Intelligence and Statistics* (Y. W. Teh and M. Titterington, eds.), vol. 9, pp. 249–256, PMLR, 13–15 May 2010.

[14] V. Sitzmann, J. N. P. Martel, A. W. Bergman, D. B. Lindell, and G. Wetzstein, "Implicit neural representations with periodic activation functions," 2020.

[15] F. Pind, A. P. Engsig-Karup, C.-H. Jeong, J. S. Hesthaven, M. S. Mejling, and J. Strømann-Andersen, "Time domain room acoustic simulations using the spectral element method," *The Journal of the Acoustical Society of America*, vol. 145, no. 6, pp. 3299–3310, 2019.

[16] C. Wu, M. Zhu, Q. Tan, Y. Kartha, and L. Lu, "A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks," *Computer Methods in Applied Mechanics and Engineering*, vol. 403, p. 115671, Jan. 2023.

[17] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. A. Hamprecht, Y. Bengio, and A. Courville, "On the spectral bias of neural networks," 6 2018.

[18] R. Basri, M. Galun, A. Geifman, D. Jacobs, Y. Kasten, and S. Kritchman, "Frequency bias in neural networks for input of non-uniform density," 3 2020.

[19] M. Tancik, P. P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. T. Barron, and R. Ng, "Fourier features let networks learn high frequency functions in low dimensional domains," 6 2020.

[20] N. Benbarka, T. Hofer, H. Ul-Moqeet Riaz, and A. Zell, "Seeing Implicit Neural Representations as Fourier Series," in *2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, (Waikoloa, HI, USA), pp. 2283–2292, IEEE, Jan. 2022.