Summer 7-31-2023

# Neural Network Learning for PDEs with Oscillatory Solutions and Causal Operators

Lizuo Liu
lizuol@smu.edu

NEURAL NETWORK LEARNING FOR PDES WITH OSCILLATORY SOLUTIONS AND CAUSAL OPERATORS

Approved by:

---

Dr. Wei Cai
Professor

---

Dr. Thomas Hagstrom
Professor

---

Dr. Weihua Geng
Associate Professor

---

Dr. Haizhao Yang
Associate Professor

# NEURAL NETWORK LEARNING FOR PDES WITH OSCILLATORY SOLUTIONS AND CAUSAL OPERATORS

A Dissertation Presented to the Graduate Faculty of the

Dedman College

Southern Methodist University

in

Partial Fulfillment of the Requirements

for the degree of

Doctor of Philosophy

with a

Major in Mathematics

by

Lizuo Liu

B.S., Mathematics, Shanghai Jiao Tong University
M.S., Mathematics, Southern Methodist University

July 31, 2023

## Acknowledgments

First of all, I would like to express my deepest gratitude to my advisor, Professor Wei Cai, for his guidance and support throughout my graduate studies. I am truly grateful for his trust in me and his belief in my potential. I consider myself incredibly fortunate to have been mentored by him, whose expertise and guidance have not only shaped me as a researcher but also as an individual, and I am truly thankful for the impact he has had on my academic and personal growth.

I would also like to extend my deepest gratitude and appreciation to Professor Xiaoguang Li for his invaluable collaboration on the research paper published in the SIAM Journal of Scientific Computing. Over the course of one year, His contributions and guidance have been instrumental in shaping not only the success of our work but also my personal growth as a researcher.

I would also like to thank Professor Bo Wang. His expertise has played a crucial role in refining my coding skills and in shaping my understanding of the field of machine learning.

I am also deeply grateful to Dr. Kamaljyoti Nath from Brown University for his collaboration on the research paper relating to the Causality DeepONet. His patient assistance, constructive feedback, and practical insights have significantly enhanced the clarity and coherence of our research work. I am truly appreciative of Dr. Kamaljyoti Nath's contribution to my growth as a writer and researcher.

I would also like to thank my thesis committee, Professor Thomas Hagstrom, Professor Weihua Geng, and Professor Haizhao Yang. Their prompt feedbacks and suggestions are the reasons for the completions of this thesis.

Many thanks to the Mathematics Department at Southern Methodist University for providing me with a stimulating and supportive environment to conduct my research. Also thank my friends and colleagues for their support and encouragement.

Finally, I would like to thank my family for their unconditional love and support. I am truly grateful for their encouragement and for their belief in me.

Liu, Lizuo

B.S., Mathematics, Shanghai Jiao Tong University
M.S., Mathematics, Southern Methodist University

Neural network learning for PDEs with oscillatory solutions and causal operators

Advisor: Dr. Wei Cai
Doctor of Philosophy degree conferred on July 31, 2023
Dissertation completed on July 6, 2023

In this thesis, we focus on developing neural networks algorithms for scientific computing.

First, we proposed a phase shift deep neural network (PhaseDNN), which provides a uniform wideband convergence in approximating high frequency functions and solutions of wave equations. Several linearized learning schemes have been proposed for neural networks solving nonlinear Navier-Stokes equations. We also proposed a causality deep neural network (Causality-DeepONet) to learn the causal response of a physical system. An extension of the Causality-DeepONet to time-dependent PDE systems is also proposed.

The PhaseDNN makes use of the fact that common DNNs often achieve convergence in the low frequency range first, and constructs a series of moderately-sized DNNs trained for selected high frequency ranges. With the help of phase shifts in the frequency domain, each of the DNNs will be trained to approximate the function's specific high frequency range at the speed of learning for low frequency. As a result, the proposed PhaseDNN is able to convert high frequency learning to low frequency one, allowing a uniform learning to wideband functions.

To solve the stationary nonlinear Navier-Stokes(NS) equation with deep neural networks, we integrate linearization of the nonlinear convection term in the NS equation into the

training process of multi-scale deep neural network (DNN) approximations of the NS solution. Four forms of linearization are considered. We solve highly oscillating stationary flows in complex domains utilizing the proposed linearized learning with multiscale neural networks.

The theorem of universal approximations to nonlinear operators proposed by Chen et al. [11] is extended to operators with causalities, and the proposed Causality-DeepONet implements the physical causality in its framework. The proposed Causality-DeepONet considers causality (the state of the system at the current time is not affected by that of the future, but only by its current state and past history) and uses a convolution-type weight in its design. To demonstrate its effectiveness in handling the causal response of a physical system, the Causality-DeepONet is applied to learn the operator representing the response of a building due to earthquake ground accelerations.

Finally, we proposed a deep neural network approximation to the evolution operator for time dependent PDE systems over long time period by recursively using one single neural network propagator, in the form of POD-DeepONet with built-in causality feature, for a small-time interval.

TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

To my parents, who have always supported me in my endeavors.

# Preface

Methodologies based on machine learning have achieved revolutionary results in many fields involving large data, including image recognition, and natural language processing. This Ph.D. research focuses on four projects applying machine learning methods to computational mathematics and scientific computing. The first two projects are related to using neural networks to solve PDEs with oscillatory solutions. The next two projects are related to using neural networks to approximate operators with causality.

## 0.1. Developing PhaseDNN and multi-scale deep neural network (DNN) for oscillatory PDE solutions

In many scientific applications, acquiring enough amount of data to train a model to satisfactory performance is prohibitively costly. Thus, for modeling and computation, partial differential equations and the relevant initial or boundary conditions are now used instead as regularization for the machine learning models. On the other hand, researchers have discovered frequency bias or spectral bias of most DNNs where neural networks capture the low frequency components of solutions first and then higher frequency components during the training of the neural network. The first two research projects address these issues.

### 0.1.1. PhaseDNN for learning high frequency wave solutions

We proposed a special neural network to learn highly oscillating functions which are hard to be learned by traditional fully connected neural networks. As an example, assume we use a normal fully connected neural network to learn the function $\cos(2\pi x)+\sin(200\pi x)$, then, the neural network could learn the low frequency part $\cos(2\pi x)$ quickly, but it will have difficul-

ties in reducing the error for the oscillating part $\sin(200\pi x)$ in its learning. An intuitive idea to accelerate the learning of the highly oscillating components is to shift the highly oscillating components $\hat{f}_j(k) \in \left[\omega_j - \frac{\Delta K}{2}, \omega_j + \frac{\Delta K}{2}\right]$ to a smooth domain $\hat{f}_j^{\text{shift}}(k) \in \left[-\frac{\Delta K}{2}, \frac{\Delta K}{2}\right]$ and train the neural network for $f_j^{\text{shift}}(x) = \mathcal{F}^{-1}\left(\hat{f}_j(k+\omega_j)\right)$ and shift the trained results back by multiplying $e^{i\omega_j x}$. The procedure mentioned is termed PhaseDNN. PhaseDNN learns not only the low frequency parts of the solution but also the oscillating parts uniformly, which makes it possible for solving equations like Helmholtz equation efficiently.

### 0.1.2. Linearized learning for oscillatory Navier-Stokes (NS) flows

The second research project is to use the neural network to learn the solution of nonlinear Navier-Stokes equation. Using the equation residual and the boundary or initial conditions as regularization in the training loss, neural network, viewed as adaptive basis, could be one of the promising alternatives of the basis function of finite element methods. Thus, we studied neural networks for learning the solutions of the stationary nonlinear Navier-Stokes equation. However, the traditional fully connected neural network can not learn the solution easily based on the residual and regularization. We discovered that the main difficulty comes from nonlinearity of the NS equation as the solution of the linear Stokes equation can be learned well with a multi-scale DNN [67].

Based on this observation, several linearized learning schemes have been proposed to mitigate the difficulties due to nonlinearity. The key idea is to first linearize the NS equation, learning is carried out based on the residual of the linearized equation and after some epochs of training, the linearized equation is updated using the newly learned solution. This procedure is carried on alternatively until the neural network solution for the nonlinear NS equation converges. To be specific, the stationary nonlinear Navier-Stokes equation is linearized by replacing one of the components in its nonlinear term by the learned solutions, even though the learned solutions are not accurate initially. However, they will converge at

the end of training. Furthermore, the pressure is regularized by a Poisson equation given that the flows are incompressible.

## 0.2. A Causality-DeepONet for learning operators with causality

The research above targets on solving one equation by one neural network. Once the boundary conditions, initial conditions or the coefficients in the equations are changed, the neural networks need to be retrained. Computing operators between physical quantities defined in function spaces have many applications in forward and inverse problems in scientific and engineering computations. For example, in wave scattering in inhomogeneous or random media, the mapping between the media physical properties, which can be modelled as a random field, and the wave field is a nonlinear operator, which represents some of the most challenging computational tasks in medical imaging, geophysical and seismic problems.

### 0.2.1. Learning operator mapping seismic excitations to responses of building

A specific example comes from earthquake safety studies of buildings and structures, the responses of structures to seismic ground accelerations give rise to a casual operator between spaces of highly oscillatory temporal signals. One natural approach is to use a multi-scale network [43] that could handle oscillating signals as part of DeepONet [47] to learn the response operator. However, the neural network still could not predict the testing case well even though training loss decay shows the convergence of the training of neural network. Then, we proposed that causality and the convolution are the key physical properties of the response operator, which was incorporated into the neural network framework, termed Causality-DeepONet.

### 0.2.2. DeepPropNet - Learning evolution PDE solution operator

A dynamic system that describes the mapping from seismic ground accelerations to building responses results from the discretization of dynamic elasticity equation, thus using the framework of Causality-DeepONet to learn the mappings between the solutions of a PDE and the corresponding right hand sides is a natural extension. In this work, the neural network was applied to learn the mapping from the right hand sides to the solutions of wave equations with variable coefficients. A simple memory consumption calculation shows that it is too memory-demanding in handling high frequency wave scattering problem if the spatial domain is discretized as in the building responses problem. Borrowing the idea of principal orthogonal decomposition, the spacial domain is represented by linear combinations of orthogonal basis, whose coefficients will be time-dependent.

Meanwhile, the global temporal dependence on the source terms of the waves hinders the attempts to learn mappings for problems with large terminal time. Therefore, we proposed a DeepPropNet where the time domain is decomposed to several small blocks and the predictions of the previous block will provide the initial conditions of the next immediate block. Once the DeepPropNet is trained, for time homogeneous problem that the velocity $c(x)$ only depends on $x$, we could use the DeepPropNet to provide the initial conditions for next time block. The procedure can be carried on recursively until the whole time period $[0, T]$ is covered, a global propagator network is thus obtained. Since it is like the initial propagator solver tracks the waves and propagates with the solutions along time direction and the propagator itself is a deep neural network, we named it DeepPropNet short for deep Propagator.

The rest of the thesis is organized as follows. In Chapter 1, a brief introduction to the deep neural networks, the corresponding universal approximation properties, and some works relating to the convergence with respect to training are presented. A review relating to the physics-informed neural network is also presented in this chapter. A theorem relating

to the convergence of physics-informed neural network is listed. A brief introduction to the DeepONet and the universal approximation property to operators are also presented. In Chapter 2, a detailed introduction to the phaseDNN is presented with numerical examples. In Chapter 3, the linearized learning scheme to accelerate the convergence of the neural network training is presented with several numerical experiments showing that the combination of linearized learning scheme and the multi-scale neural network can solve the stationary Navier-Stokes equation with oscillatory force terms. In Chapter 4, the Causality-DeepONet is presented with numerical examples. In Chapter 5, the DeepPropNet is presented with numerical examples. In Chapter 6, the conclusion and the future works are presented.

# Chapter 1

# Introduction

## 1.1. Deep neural network

Artificial neural network, as a mathematical analogy to the biological brain, has been studied for decades. Taking advantages of developments of hardware, software, and algorithms, neural network has been widely used in many fields, such as computer vision, natural language processing, and speech recognition. In this work, we specify the artificial neural network to be the feedforward neural network, which is the most common type of neural network.

A feedforward neural network contains one or more layers of neurons (nodes). The neurons are organized in layers, with each layer connected to the next layer. The first layer is called the input layer, and the last layer is called the output layer. The layers between the input layer and the output layer are called hidden layers. The neurons in the input layer, output layer, hidden layer are called input neurons, output neurons, and hidden neurons, respectively. The neurons in the same layer are not connected to each other, but neurons in the adjacent layers are fully connected to each other. The activation functions are applied elementwisely for neurons in hidden layers and the neurons in the output layer do not have activation function. The term *deep* in the deep neural network refers to that the neural network is a feedforward network with more than one hidden layer. The schematic of a deep neural network is shown in Figure 1.1.

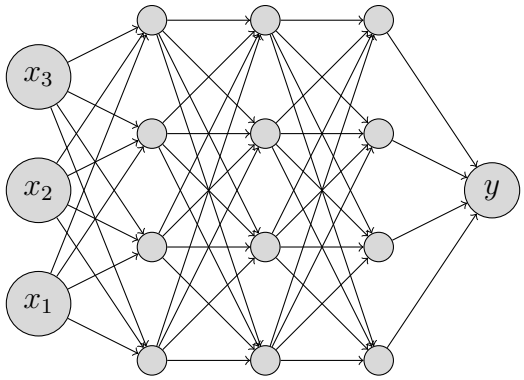



Figure 1.1: Schematic of a deep neural network. The input layer contains three input neurons. The three hidden layers contain four hidden neurons each. The output layer contains one output neuron. The neurons in the same layer share the same activation function. The neurons in the adjacent layers are fully connected to each other.

Similar to its biological counterpart, the artificial neuron model is connected to other neurons through weighted connections. The output of the neuron is determined by the weighted sum of the inputs, a specific bias and the activation function. The weights can be compared to the weights of synapses connecting the axon and the dendrites of the biological neurons. The bias is analogous to the axon hillock of the biological neuron. The activation function, typically a nonlinear function, introduces nonlinearity to the mathematical model. Popular choices of activation functions include sigmoid function, hyperbolic tangent function, and rectified linear unit (ReLU). The output of the neuron is akin to the action potentials travelling through axon of a biological neuron. The schematic of a neuron model is shown in Figure 1.2.

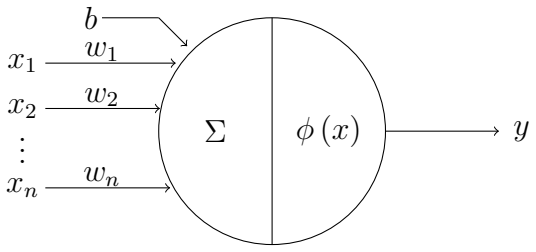



Figure 1.2: Mathematical model of a neuron. The output of the neuron is $y$ is determined by the weighted sum of the inputs $x_1, x_2, \ldots, x_n$ given weights $w_1, w_2, \ldots, w_n$, a specific bias $b$ and the activation function $\phi(x)$.

Thus, for a 2-layer neural network, or a shallow neural network, we should have the following form,

$$f_\theta(x) = \sum_{i=1}^{n} v_i \sigma\left(w_i^T x\right) \tag{1.1}$$

where $x \in \mathbb{R}^d$, $w_i \in \mathbb{R}^d, i = 1, 2, \ldots, n$, $v_i \in \mathbb{R}, i = 1, 2, \ldots, n$, and $\sigma(\cdot)$ is the activation function.

For a deep neural network, let $\boldsymbol{x} \in \mathbb{R}^d$ be the input, $\boldsymbol{W}^{(1)} \in \mathbb{R}^{n \times d}$ is the first weight matrix, $\boldsymbol{W}^{(h)} \in \mathbb{R}^{n \times n}$ is the weight at the $h$-th layer for $h = 2, 3, \ldots, H$, $\boldsymbol{b}^{(h)} \in \mathbb{R}^n$ is the bias at the $h$-th layer for $h = 2, 3, \ldots, H$, $\boldsymbol{v} \in \mathbb{R}^n$ is the weight vector for the output layer, and $\sigma(\cdot)$ is the activation function. The deep neural network can be expressed as,

$$\begin{aligned} \boldsymbol{x}^{(h)} &= \sigma\left(\boldsymbol{W}^{(h)} \boldsymbol{x}^{(h-1)} + \boldsymbol{b}^{(h)}\right), 1 \leq h \leq H, \\ f_\theta(\boldsymbol{x}) &= \boldsymbol{v}^T \boldsymbol{x}^{(H)}, \end{aligned} \tag{1.2}$$

where the activation function $\sigma(\cdot)$ is applied element-wise.

### 1.1.1. Training the neural network

The weights $\boldsymbol{v}, \boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}, \ldots, \boldsymbol{W}^{(H)}$ and bias $\boldsymbol{b}^{(1)}, \boldsymbol{b}^{(2)}, \ldots, \boldsymbol{b}^{(H)}$ are determined by minimizing the differences between the predictions or outputs of neural networks and the exact values. The process of determining the weights and bias forms training the neural network. The differences between predictions and exact values are expressed by the Euclidean distance, or different kinds of divergences, which are specific for different tasks. The optimization problem is typically solved by gradient descent or its variants due to the nonconvex essence of this optimization problem.

Algorithm 1 shows the gradient descent algorithm for training a neural network. Once the weights and biases of the neural network are initialized, the algorithm computes the

**Algorithm 1** Gradient descent for neural network training.

Initialize the weights and biases of the neural network
Set the learning rate $\alpha$, the maximum number of iterations $K$, the convergence threshold $\varepsilon$, the current iteration counter $k \leftarrow 0$, the initial loss $L_0 \leftarrow \infty$
**repeat**
    Compute the predicted output $\hat{y}$
    Compute the loss function $L(\hat{y}, y)$
        $\boldsymbol{W} \leftarrow \boldsymbol{W} - \alpha \frac{\partial L(\hat{y},y)}{\partial \boldsymbol{W}}$
        $\boldsymbol{b} \leftarrow \boldsymbol{b} - \alpha \frac{\partial L(\hat{y},y)}{\partial \boldsymbol{b}}$
    Compute the total loss over the training set:
        $L_k \leftarrow \frac{1}{N} \sum_{i=1}^{N} L(\hat{y}_i, y_i)$
    Increment the iteration counter $k \leftarrow k + 1$
**until** $k > K$ or $|L_k - L_{k-1}| < \varepsilon$

predicted output $\hat{y}$ of the neural network and evaluate the loss $L(\hat{y}, y)$. Then the gradients of loss with respect to the parameters are evaluated by auto differentiation. The weights and biases of the neural network are updated along the direction opposite to the gradients to minimize the loss. The algorithm terminates when the maximum number of iterations is reached or the loss function converges.

A specific case of gradient descent algorithm is stochastic gradient descent (SGD) [23]. Unlike the gradient descent algorithm, the SGD computes the gradients from a random selected subset of the data to introduce randomness and reduce the computational cost. From the implementation level, the SGD shuffles the training datasets and then apply the gradient descent algorithm. The SGD algorithm is shown in Algorithm 2.

### 1.1.2. Universal approximation theorems to functions

The existence of a feedforward neural network structure for approximating a continuous function is guaranteed by the universal approximation theorem to functions [13], which states that finite linear combinations of continuous discriminatory functions are dense in the continuous function space $C\left([0,1]^d\right)$. Specifically, we have the following theorem,

**Algorithm 2** Stochastic gradient descent for neural network training.

Initialize the weights and biases of the neural network
Set the learning rate $\alpha$, the maximum number of iterations $K$, the convergence threshold $\varepsilon$, the current iteration counter $k \leftarrow 0$, the initial loss $L_0 \leftarrow \infty$
**repeat**
    Shuffle the training set
    **for** each batch of training examples $(\boldsymbol{x}, y)$ **do**
        Compute the predicted output $\hat{y}$
        Compute the loss function $L(\hat{y}, y)$
        $\boldsymbol{W} \leftarrow \boldsymbol{W} - \alpha \frac{\partial L(\hat{y}, y)}{\partial \boldsymbol{W}}$
        $\boldsymbol{b} \leftarrow \boldsymbol{b} - \alpha \frac{\partial L(\hat{y}, y)}{\partial \boldsymbol{b}}$
    **end for**
    Compute the total loss over the training set:
    $L_k \leftarrow \frac{1}{N} \sum_{i=1}^{N} L(\hat{y}_i, y_i)$
    Increment the iteration counter $k \leftarrow k + 1$
**until** $k > K$ or $|L_k - L_{k-1}| < \varepsilon$

**Theorem 1** (Universal Approximation Theorem to Functions [13])**.** *Let $\sigma(x)$ be continuous and discriminatory, then for any $f(x) \in C\left([0,1]^d\right)$ and $\varepsilon > 0$, there exists a positive integer $n$, real constants $v_i, b_i \in \mathbb{R}$ and real vectors $w_i \in \mathbb{R}^d$, where $i = 1, 2, \ldots, n$, such that*

$$\left| f(x) - \sum_{i=1}^{n} v_i \sigma\left(w_i^T x + b_i\right) \right| < \varepsilon, \quad \forall x \in [0,1]^d.$$

**Definition 1.1.** A function $\sigma(x)$ is discriminatory if for a measure $\mu$

$$\int_{[0,1]^d} \sigma\left(w^T x + b\right) d\mu(x) = 0,$$

for all $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$ implies $\mu = 0$. □

For specific activation functions, the error estimations of approximations relate to the width and depth of neural networks. For example, if the activation function is ReLU, Lu et al. [46] stated that for any $n, H \in \mathbb{N}^+$, the neural network with ReLU as activation functions with width $O(n \ln n)$ and depth $O(H \ln H)$ could approximate function $f(x) \in C^s\left([0,1]^d\right)$

with a nearly optimal approximation error $O\left(\|f\|_{C^s\left([0,1]^d\right)} n^{-2s/d} H^{-2s/d}\right)$, where

$$\|f\|_{C^s\left([0,1]^d\right)} = \max\left\{\|\partial^\alpha f\|_{L^\infty\left([0,1]^d\right)} : \|\alpha\|_1 \leq s, \alpha \in \mathbb{N}^d\right\} \text{ for } \forall f \in C^s\left([0,1]^d\right).$$

In the expression above, $\|\alpha\|_1 = \sum|\alpha_i|$. In detail, there is a theorem for the approximation error estimation when applying neural networks with ReLU as activation function to approximate smooth functions.

**Theorem 2** (Deep Network Approximation for Smooth Functions [46])**.** *Given a smooth function $f \in C^s\left([0,1]^d\right)$ with $s \in \mathbb{N}^+$, for any $N, L \in \mathbb{N}^+$, there exists a function $\phi$ implemented by a ReLU fully connected neural network with width $C_1 (N+2) \log_2 (8N)$ and depth $C_2 (L+2) \log_2 (4L) + 2d$ such that*

$$\|f - \phi\|_{L^\infty\left([0,1]^d\right)} \leq \frac{C_3}{N^{2s/d} L^{2s/d}} \|f\|_{C^s\left([0,1]^d\right)},$$

*where $C_1 = 17 s^{d+1} 3^d d, C_2 = 18 s^2$, and $C_3 = 85 (s+1)^d 8^s$.*

### 1.1.3. Convergence of neural network approximations to functions

A long line of works have been focusing on the convergence of neural network approximations to functions. In this section, we provide a short review of the existing results. The problem discussed in this section is an empirical risk minimization problem with the quadratic loss function

$$\min_\theta L(\theta) = \frac{1}{N} \sum_{i=1}^{N} (f(\theta, x_i) - y_i)^2 \tag{1.3}$$

where $f(\theta, x_i)$ is the feedforward neural network with parameters $\theta$ and input $x_i$ defined as equation (1.2). $\theta$ are the collections of parameters $\{W_h, b_h\}_{h=1}^H$ of the neural network.

In the paper [17], Du et al. proved that for a deep neural network without bias, if the activation function is softplus: $\sigma(z) = \log(1 + \exp(z))$, then the gradient descent algorithm could achieve zero training loss.

**Theorem 3** (Convergence Rate of Gradient Descent for Deep Fully Connected Neural Networks [17]). *Assume for all $i \in \{1, 2, \ldots, N\}$, $\|x_i\|_2 = 1$, $|y_i| = O(1)$ and the number of hidden nodes per layer $n$ satisfies*

$$n = \Omega\left(2^{O(H)} max\left\{\frac{N^4}{\lambda_{min}^4}, \frac{N}{\delta}, \frac{N^2 \log\left(\frac{HN}{\delta}\right)}{\lambda_{min}^2}\right\}\right),$$

*where $\lambda_{min}$ is the minimum eigenvalue of the gram matrix $\boldsymbol{K}^{(H)}$ for the deep neural network. If the step size is defined as*

$$\eta = O\left(\frac{\lambda_{min}}{N^2 2^{O(H)}}\right),$$

*then with probability at least $1 - \delta$ over the random initialization, the loss at iteration $k$ satisfies*

$$L(\theta(k)) \leq \left(1 - \frac{\eta \lambda_{min}}{2}\right)^k L(\theta(0)).$$

Theorem 3 shows that if the width $n$ is large enough, gradient descent will converge to the global minimum with zero loss in a linear rate if step size is set properly.

As for the activation function ReLU, Allen-Zhu et al. [2] proved that for an over-parameterized deep neural network, gradient descent will converge to the global minimum with zero loss, as long as the dataset is non-degenerate, i.e., the data points are distinct.

**Theorem 4** (Convergence Rate of Gradient Descent for DNN with ReLU Activations [2]). *Suppose the number of neurons $n$ satisfies $n \geq \tilde{\Omega}\left(poly\left(N, H, \delta_{min}^{-1}\right) \cdot d\right)$ where $\delta_{min}$ is the minimum (relative) distance between two training data points, and $poly(\cdot, \cdot, \cdot)$ means a polynomial. Starting from proper random initialization, with probability at least $1 - e^{-\Omega\left(\log^2 n\right)}$,*

*gradient descent with learning rate* $\eta = \Theta\left(\frac{d\delta_{min}}{poly_{(N,H)n}}\right)$ *finds a* $\theta$ *s.t.* $\sum_{i=1}^{N}\|f(\theta, \boldsymbol{x}_i) - y_i\|^2 \leq \varepsilon$ *in polynomial time*

$$T = \Theta\left(\frac{poly(N, H)}{\delta_{min}^2}\log\frac{1}{\varepsilon}\right).$$

### 1.2. Spectral bias

The training dynamics of deep neural network is fundamental. In this section, we will review the description of the training dynamics from the perspective of frequency domain. Xu et al. [70] and Rahaman et al. [59] proposed the frequency principle or spectral bias, respectively to describe one of the feature of the training dynamics of deep neural network. In a nutshell, the neural network captures the low-frequency components of the data before capturing the high-frequency components. For instance, when a deep neural network is trained to fit $f(x) = \sin(x) + \sin(10x)$, the neural network will fit the $\sin(x)$ part first and then fit the $\sin(10x)$ part. In the paper [70], Xu et al. verified the statement experimentally and theoretically.

Rigorously, the training dynamics of loss function with respect to frequency for a neural network of one hidden layer only with activation function $\sigma(x) = \tanh(x)$ is described by the following theorem.

**Theorem 5** (Frequency Principle [70])**.** *Suppose the target function has only two non-zero frequencies* $k_1, k_2$*, i.e.,* $|\hat{f}(k_1)| > 0, |\hat{f}(k_2)| > 0$*, and* $|k_2| > |k_1| > 0$ *and* $|\hat{f}(k)| = 0$ *otherwise. The loss function is defined as* $L(x) = L(k_1) + L(k_2)$*. Denote*

$$\mathcal{S} = \left\{\frac{\partial L(k_1)}{\partial t} \leq 0, \frac{\partial L(k_1)}{\partial t} \leq \frac{\partial L(k_2)}{\partial t}\right\},$$

*that is, $L(k_1)$ descends faster than $L(k_2)$. Then, for sufficiently small $\delta$, there exists constants $c, C > 0$ such that*

$$\frac{\mu\left(\{W : S \text{ holds}\} \bigcap B_\delta\right)}{\mu\left(B_\delta\right)} > 1 - C\exp\left(-\frac{c}{\delta}\right)$$

The study of the frequency principle has led to the development of several algorithms aimed at accelerating the training process of deep neural networks. Among these algorithms is PhaseDNN, which will be extensively discussed in the subsequent chapter.

### 1.3. Physics-informed neural network

Machine learning tools are typically reliant on data for training. However, in the context of analyzing complex physical, biological, or engineering systems, a scarcity of available data often poses a significant challenge. Consequently, we frequently encounter situations where we must draw conclusions and make decisions based on incomplete information. In this case, we need to incorporate the physical laws into the neural network. The physics-informed neural network [61] is a class of neural network that incorporates the physical laws into the loss of neural network. By doing so, it leverages the known principles governing the system to enhance the accuracy and reliability of predictions. In this section, we will delve into the physics-informed neural network.

Consider a partial differential equation (PDE) with general boundary conditions

$$\begin{cases} \mathcal{L}_t u(t, \boldsymbol{x}) + \mathcal{L}_{\boldsymbol{x}} u(t, \boldsymbol{x}) = f(t, \boldsymbol{x}), & \boldsymbol{x} \in \Omega, t > 0 \\ \mathcal{B} u(t, \boldsymbol{x}) = g(t, \boldsymbol{x}), & \boldsymbol{x} \in \partial\Omega \\ u(0, \boldsymbol{x}) = u_0(\boldsymbol{x}), & \boldsymbol{x} \in \Omega \end{cases} \tag{1.4}$$

where $\mathcal{L}_t$ is the differential operator with respect to $t$, $\mathcal{L}_{\boldsymbol{x}}$ could be either linear or nonlinear with respect to $\boldsymbol{x}$, $\mathcal{B}$ could be either Dirichlet boundary condition, Neumann boundary condition, or Robin boundary condition, or a combination thereof.

The loss function for the physics-informed neural network is defined as the combination of the mean squared error of the training data, the PDE residual, and the boundary condition residual, i.e.

$$L = \alpha L_{\text{data}} + \beta L_{\text{PDE}} + \gamma L_{\text{BC}} + \delta L_{\text{IC}}, \tag{1.5}$$

where $\alpha, \beta, \gamma, \delta$ are the penalties/weights for specific loss, $L_{\text{data}}$ is the empirical loss of the training data, which is defined as equation (1.3), and $L_{\text{PDE}}$ is defined as

$$L_{\text{PDE}} = \int_0^T \int_\Omega \left[\mathcal{L}_t u\left(t, \boldsymbol{x}\right) + \mathcal{L}_{\boldsymbol{x}} u\left(t, \boldsymbol{x}\right) - f\left(t, \boldsymbol{x}\right)\right]^2 \mathrm{d}\boldsymbol{x}\mathrm{d}t. \tag{1.6}$$

Similarly, $L_{\text{BC}}$ is defined as

$$L_{\text{BC}} = \int_0^T \int_{\partial\Omega} \left[\mathcal{B} u\left(t, \boldsymbol{x}\right) - g\left(t, \boldsymbol{x}\right)\right]^2 \mathrm{d}\boldsymbol{x}\mathrm{d}t. \tag{1.7}$$

Finally, $L_{\text{IC}}$ is defined as

$$L_{\text{IC}} = \int_\Omega \left[u\left(0, \boldsymbol{x}\right) - u_0\left(\boldsymbol{x}\right)\right]^2 \mathrm{d}\boldsymbol{x}. \tag{1.8}$$

The integrals in the PDE residual (1.6), the boundary condition residual (1.7) and the initial condition residual (1.8) are approximated by numerical integration methods, or the

Monte Carlo method,

$$\begin{aligned}
&\int_0^T \int_\Omega \left[\mathcal{L}_t u\left(t,\boldsymbol{x}\right)+\mathcal{L}_{\boldsymbol{x}} u\left(t,\boldsymbol{x}\right)-f\left(t,\boldsymbol{x}\right)\right]^2 \mathrm{d}\boldsymbol{x}\mathrm{d}t \approx \\
&\qquad\qquad \frac{1}{N}\sum_{i=1}^{N}\left[\mathcal{L}_t u\left(t_i,\boldsymbol{x}_i\right)+\mathcal{L}_{\boldsymbol{x}} u\left(t_i,\boldsymbol{x}_i\right)-f\left(t_i,\boldsymbol{x}_i\right)\right]^2, \\
&\int_0^T \int_{\partial\Omega} \left[\mathcal{B} u\left(t,\boldsymbol{x}\right)-g\left(t,\boldsymbol{x}\right)\right]^2 \mathrm{d}\boldsymbol{x}\mathrm{d}t \approx \frac{1}{M}\sum_{i=1}^{M}\left[\mathcal{B} u\left(t_i,\boldsymbol{x}_i\right)-g\left(t_i,\boldsymbol{x}_i\right)\right]^2, \\
&\int_\Omega \left[u\left(0,\boldsymbol{x}\right)-u_0\left(\boldsymbol{x}\right)\right]^2 \mathrm{d}\boldsymbol{x} \approx \frac{1}{J}\sum_{i=1}^{J}\left[u\left(0,\boldsymbol{x}_i\right)-u_0\left(\boldsymbol{x}_i\right)\right]^2.
\end{aligned} \tag{1.9}$$

The points $\{t_i,\boldsymbol{x}_i\}_{i=1}^N$, $\{t_i,\boldsymbol{x}_i\}_{i=1}^M$ and $\{\boldsymbol{x}_i\}_{i=1}^J$ are randomly sampled from $[0,T]\times\Omega$, $[0,T]\times\partial\Omega$, and $\{0\}\times\Omega$ respectively.

### 1.3.1. Convergence of physics-informed neural network approximations

The analysis of convergence of physics-informed neural network approximations is more involved and problem specific. We will review the results from the paper [50].

In the paper [50], the authors consider to incorporate the boundary conditions into the neural network structure to simplify the analysis. Consider 1D problem with Dirichlet boundary condition as an example, the neural network is defined as

$$\tilde{u}_\theta\left(x\right)=\left(x-a\right)^{p_a}\left(x-b\right)^{p_b}u_\theta\left(x\right)+\left(b_0-a_0\right)\left(x-a\right)/\left(b-a\right)+a_0,$$

given $u\left(a\right)=a_0, u\left(b\right)=b_0$, where $0<p_a,p_b\leq 1$. Thus the term $L_{BC}$ in loss function (1.5) vanishes. Problem with initial condition could be treated as a special case of problem with Dirichlet boundary condition. The empirical loss for PINNs is defined as

$$R_S(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left[\mathcal{L}_t\tilde{u}_\theta(t_i,\boldsymbol{x}_i) + \mathcal{L}_{\boldsymbol{x}}\tilde{u}_\theta(t_i,\boldsymbol{x}_i) - f(t_i,\boldsymbol{x}_i)\right]^2, \tag{1.10}$$

Assume the operator $\mathcal{L}_{\boldsymbol{x}}u = \sum_{\alpha,\beta=1}^{d} A_{\alpha\beta}(\boldsymbol{x})\, u_{x_\alpha x_\beta} + \sum_{\alpha=1}^{d} b_\alpha(\boldsymbol{x})\, u_{x_\alpha} + c(\boldsymbol{x})\, u$ and there exists a constant $C_M$ such that for all $\boldsymbol{x} \in \Omega = [0,1]^d$, $\alpha, \beta \in \{1,\ldots,d\}$,

$$|A_{\alpha\beta}(\boldsymbol{x})| \leq C_M, \quad |b_\alpha(\boldsymbol{x})| \leq C_M, \quad |c(\boldsymbol{x})| \leq C_M,$$

and further, $A_{\alpha\beta} = A_{\beta\alpha}$, then we have the following theorem.

**Theorem 6** (Training Convergence of PINNs [50])**.** *Let $u$ be the solution to the PDE (1.4) and $u_\theta$ be the solution to the PINN given loss function (1.10) that is minimized by the gradient descent method with a proper learning rate. The neural network considered is a two-layer neural network*

$$u_\theta(\tilde{\boldsymbol{x}}) = \sum_{i=1}^{n} v_i \sigma\left(w_i^T \tilde{\boldsymbol{x}}\right),$$

*where $\tilde{\boldsymbol{x}} = (t, \boldsymbol{x})$, $\sigma(x) = max\left\{\frac{1}{6}x^3, 0\right\}$ is the activation function. Define $\theta^0 = vec\{v_i^0, w_i^0\}_{i=1}^n$ be the parameters after initialization with $v_i^0 \sim \mathcal{N}(0,\gamma^2)$ and $w_i^0 \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_{d+1})$ for any $\gamma \in (0,1)$. Let $C_d := \mathbb{E}\|w\|_1^{12} < +\infty$, for any $\delta \in (0,1)$, given $N$ randomly sampled $\tilde{\boldsymbol{x}}_i \in [0,T]\times\Omega$, if the number of neurons $n$ satisfies*

$$n \geq max\left\{\frac{512N^4C_M^4C_d}{\lambda_{min}^2\delta}, \frac{200\sqrt{2}C_M d^3 N\log\left(4n(d+1)/\delta\sqrt{R_S(\theta^0)}\right)}{\lambda_S}, \right.$$
$$\left. \frac{2^{23}C_M^3 d^9 N^2 \left(\log\left(4n(d+1)/\delta\right)\right)^4 \sqrt{R_S(\theta^0)}}{\lambda_{min}^2}\right\},$$

*then, with probability at least $1-\delta$, the empirical loss $R_S(\theta(t))$ satisfies*

$$R_S(\theta(t)) \leq \exp\left(-\frac{n\lambda_{min}t}{N}\right) R_S(\theta^0)$$

*Recall the definition of $\lambda_{min}$ is the minimum eigenvalue of the Gram matrix of the two-layer neural network, like the definition in theorem 3.*

## 1.4. DeepONet

DeepONet [47] was proposed by Lu et al., generalizing the original work of Chen & Chen [11]. In the previous framework of physics-informed neural network, once the boundary conditions or initial boundary conditions are changed, the neural network may need to be retrained. DeepONet is proposed to solve this problem. It is expected to learn the mapping between the initial condition, coefficients or the boundary conditions to the solutions. Fourier neural operator [37] is another framework of neural operator that can be used to learn such mappings. For each layer of Fourier neural operator, the weights are viewed as tunable parameters in the Fourier space. The input of each layer is transformed into the Fourier space and then a matrix-vector multiplication is applied to the transformed inputs with the trainable parameter matrix. The output of that layer is transformed back to the original space. In this section, we will offer a brief introduction to the framework of DeepONet and the corresponding universal approximation theory to operators.

### 1.4.1. Universal approximation theory

The work of [11] gives a constructive procedure for approximating nonlinear operator $\mathcal{G}$ between continuous functions $\mathcal{G}(f)(x)$ in a compact subset $\mathcal{U} \subset C(\mathcal{X})$ with $\mathcal{X} \subseteq \mathbb{R}^d$ and continuous functions $f(x)$ in a compact subset $\mathcal{V} \subset C(\mathcal{F})$ with $\mathcal{F} \subseteq \mathbb{R}^d$

$$\mathcal{G} : f(x) \in \mathcal{V} \subset C(\mathcal{F}) \to \mathcal{G}(f)(x) \in \mathcal{U} \subset C(\mathcal{X}), \tag{1.11}$$

where $C(\mathcal{X})$ and $C(\mathcal{F})$ are the continuous function spaces over $\mathcal{X}$ and $\mathcal{F}$, respectively, and $\mathcal{X}$ and $\mathcal{F}$ are compact subsets of $\mathbb{R}^d$, the Euclidean space of dimension $d$. The universal approximations with respect to operators are based on the two following results:

- **Universal Approximation of Functions [11]:** Given any $\varepsilon_1 > 0$, there exists a positive integer $n$, $\{\boldsymbol{w}_k\}_{k=1}^n \in \mathbb{R}^d, \{b_k\}_{k=1}^n \in \mathbb{R}$, such that functions $\mathcal{G}(f)(x)$ selected from a compact subset $\mathcal{U}$ of $C(\mathcal{X})$ could be uniformly approximated by a one-hidden-layer neural network with any Tauber-Wiener (TW) activation function $\sigma_t$

$$\left| \mathcal{G}(f)(x) - \sum_{k=1}^{n} c_k\left(\mathcal{G}(f)\right) \sigma_t\left(\boldsymbol{w}_k \cdot x + b_k\right) \right| \leq \varepsilon_1, \quad \forall\, x \in \mathcal{X}, \tag{1.12}$$

where $c_k\left(\mathcal{G}(f)\right)$ is a linear continuous functional defined on $\mathcal{V}$ (a compact subset of $C(\mathcal{F})$), and all $\boldsymbol{w}_k, b_k$ are independent of $x$ and $f\left(x\right)$. A Tauber-Wiener activation function is defined as follows.

**Definition 1.2.** Assume $\mathbb{R}$ is the set of real numbers. $\sigma : \mathbb{R} \to \mathbb{R}$ is called a Tauber-Wiener (TW) function if all the linear combinations $g(x) = \sum_{i=1}^{I} c_i \sigma\left(w_i x + b_i\right)$ are dense in every $C\left[a, b\right]$, where $\{w_i\}_{i=1}^{I}, \{b_i\}_{i=1}^{I}, \{c_i\}_{i=1}^{I} \in \mathbb{R}$ are real constants. □

If the function is not polynomial, then it is a TW function. For example, $\tanh\left(x\right)$ and sigmoid $\sigma\left(x\right) = \dfrac{1}{1+e^{-x}}$ are TW functions.

- **Universal Approximation of Functionals [11]:** Given any $\varepsilon_2 > 0$, there exists a positive integer $M$, $m$ points $\{x_j\}_{j=1}^{m} \in \mathcal{F}$ with real constants $c_i^k, W_{ij}^k, B_i^k \in \mathbb{R}, i = 1, \ldots, M, j = 1, \ldots, m$, such that a continuous functional $c_k\left(\mathcal{G}(f)\right)$ defined on $\mathcal{V}$ could be approximated by a one-hidden-layer neural network with any TW activation function $\sigma_b$

$$\left| c_k\left(\mathcal{G}(f)\right) - \sum_{i=1}^{M} c_i^k \sigma_b\left(\sum_{j=1}^{m} W_{ij}^k f\left(x_j\right) + B_i^k\right) \right| \leq \varepsilon_2, \quad \forall f \in \mathcal{V}, \tag{1.13}$$

where the coefficients $c_i^k, W_{ij}^k, B_i^k$ and nodes $\{x_j\}_{j=1}^{m}$ and $m, M$ are all independent of $f\left(x\right)$.

Combining these two universal approximations, the authors of [11] proposed the universal approximation of nonlinear operator by neural networks when restricted to the compact

subset $\mathcal{V}$ of the continuous function space $C(\mathcal{F})$ defined on a compact domain $\mathcal{F}$ in $\mathbb{R}^d$. Namely, given any $\varepsilon > 0$, there exists positive integers $M, n$, $m$ points $\{x_j\}_{j=1}^m \in \mathcal{F} \subseteq \mathbb{R}^d$ with real constants $c_i^k, W_{ij}^k, B_i^k \in \mathbb{R}, i = 1, \ldots, M, j = 1, \ldots, m$, $\{\boldsymbol{w}_k\}_{k=1}^n \in \mathbb{R}^d, \{b_k\}_{k=1}^n \in \mathbb{R}$ that are all independent of continuous functions $f \in \mathcal{V} \subseteq C(\mathcal{F})$ and $x \in \mathcal{X} \subseteq \mathbb{R}^d$ such that

$$\left| \mathcal{G}(f)(x) - \sum_{k=1}^{n} \left\{ \sum_{i=1}^{M} c_i^k \sigma_b \left( \sum_{j=1}^{m} W_{ij}^k f(x_j) + B_i^k \right) \right\} \sigma_t (\boldsymbol{w}_k \cdot x + b_k) \right| \leq \varepsilon \tag{1.14}$$

### 1.4.2. DeepONet

Based on the universal approximation of nonlinear operator, Lu et al. [47] proposed the DeepONet by replacing the two one-hidden-layer neural networks in equation (1.14) with two deep neural networks. For a general operator $\mathcal{G}(f)(x)$, DeepONet has form

$$\mathcal{G}(f)(x) \sim \sum_{k=1}^{n} c_k \underbrace{\sigma_{B,k} \left( \{f(x_j)\}_{j=1}^m \right)}_{B_k} \underbrace{\sigma_{T,k}(x)}_{T_k}, \tag{1.15}$$

where $\sigma_B(\cdot)$ with a signal $\{f(x_j)\}_{j=1}^m$ as input is a deep neural network with $n$ outputs, named as the branch net, $\sigma_T(\cdot)$ with input $x$ is also a deep neural network with $n$ outputs which is called the trunk net. The schematics are shown in Figure 1.3. The DeepONet has been shown to be able to learn not only explicit mathematical operators like integration and fractional derivatives, but also PDE operators [8, 15, 16, 39, 47].

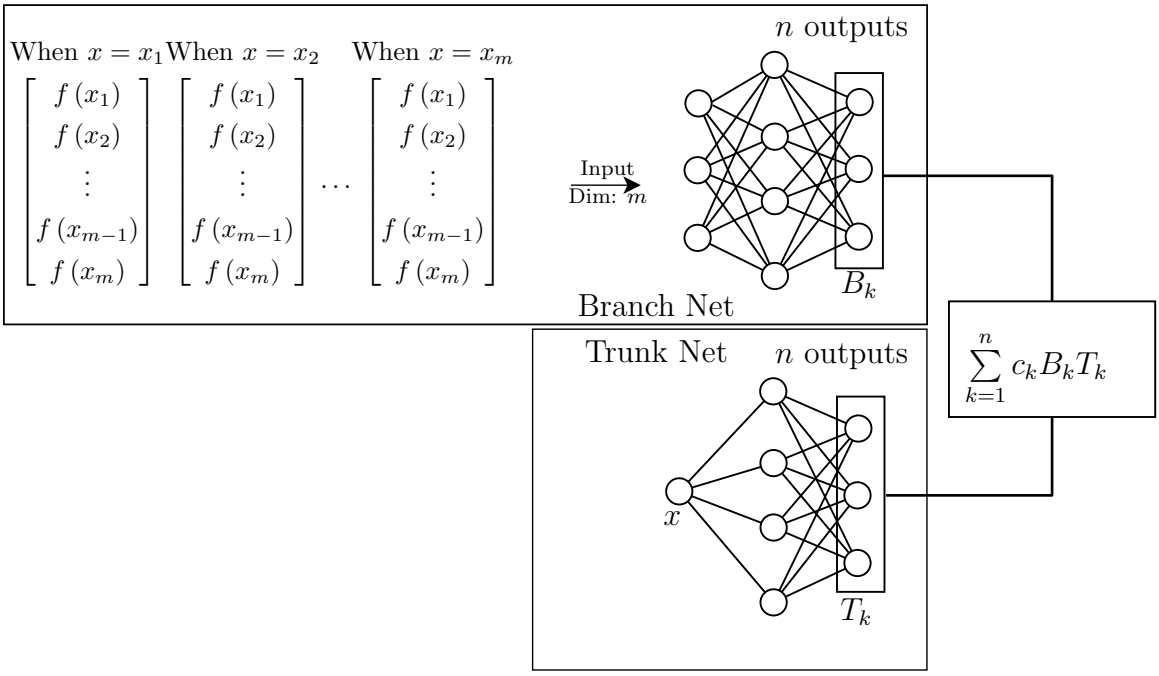



Figure 1.3: **Schematic Diagram of the DeepONet:** A schematic diagram of DeepONet showing branch and the trunk net along with the input data and output. The number of the input neurons of the branch net is equal to the number of sensor points in the input signals. The trunk net takes the input point $x$ at where the output function need to be evaluated. Thus, the first layer of trunk equal to the dimension of the problem.

# Chapter 2

## A Phase Shift Deep Neural Network (PhaseDNN) for High Frequency Approximation and Wave Problems

*The content in this chapter has been published in the following journal paper in collaboration with Xiaoguang Li and Wei Cai:*

Wei Cai, Xiaoguang Li, and Lizuo Liu, *A Phase Shift Deep Neural Network for High Frequency Approximation and Wave Problems*, SIAM J. Sci. Comput., 42(5), A3285-A3312 (2020) [10].

### 2.1. Introduction

Deep neural networks (DNNs) have shown greater potential in approximating high dimensional functions, compared with traditional approximations based on Lagrangian interpolation or spectral methods. Recently, it has been found [49, 69, 70] that some common NNs, including fully connected and convolution neural network (CNN) with tanh and ReLU activation functions, demonstrate a frequency dependent convergence behavior. Namely, the DNNs during the training are able to approximate the low frequency components of the targeted functions first before higher frequency components. Spectral bias is defined as the F-Principle of DNNs [69]. The stalling of DNN convergence in the later stage of training could be mostly related to learning the high frequency components of the data. The F-principle behavior of DNNs is the opposite to that of the traditional multigrid method (MGM) [5] in approximating the solutions of PDEs where the convergence occurs first in the higher frequency end of the spectrum, as a result of the smoothing operator employed in the MGM. The MGM takes advantage of this fast high frequency error reduction in the smoothing iteration cycles and restricts the original solution on a fine grid to a coarser grid, then

continuing the smoothing iteration on the coarse grid to reduce the higher end frequency spectrum in the context of the coarse grid. This downward restriction can be continued until errors over all frequency are reduced by a small number of iterations on each level of the coarse grids.

There are many scientific computing problems which involve high frequency solutions in complex domains, such as high frequency wave equations in inhomogeneous media, arising from electromagnetic wave propagation in turbid media, rough surface scattering, seismic waves, and geophysical problems. Finding efficient solutions, especially in random environments, poses great computational challenges due to the highly oscillatory natures of the solutions. To compute the high frequency waves in a deterministic medium, high order methods such as spectral methods for the differential equations or wideband fast multipole methods for the integral equations are often used. In this chapter, we will develop meshless DNNs based numerical methods to handle high frequency functions and solutions of high frequency wave equation in inhomogeneous media in complex domains. To improve the capability of usual DNNs for learning highly oscillatory functions in the physical spatial variables, we propose a phase shift DNN with wideband learning capabilities in error reductions in the approximation for all frequencies of the targeted function by taking advantage of the faster convergence in the low frequencies of the DNN during its training. To learn a function of specific frequency range, we employ a phase shift in the $k$-space to translate its frequency to the range $|k| < K_0$, then the phase shifted function with a low frequency content can be learned by common DNNs with a small number of training epochs. The resulting series of DNNs with phase shifts will make a phase shift deep neural network (PhaseDNN).

To achieve uniform wideband approximation of a general function, we can implement the PhaseDNN in a parallel manner where original data is decomposed into data of specific frequency range, which after a proper phase shift, is learned quickly. This approach can be implemented in a parallel manner, however, frequency extraction of the original training data

have to be done using convolutions with a frequency selection kernel numerically, which could become very expensive or not accurate for scattered training data. Alternatively, we can implement the PhaseDNN in a non-parallel manner where data from all range of frequencies are learned together with phase shifts included in the makeup of the PhaseDNN, resulting in a coupled PhaseDNN. Although the coupled PhaseDNN lacks parallelism, it avoids the costly convolution used in the parallel PhaseDNN to extract the frequency component from the original training data. This feature will be shown to be important when higher dimensional data are involved in the training. Thanks to this property, the coupled PhaseDNN will be used to solve high frequency wave problems where we seek solutions in a space of PhaseDNNs by minimizing the residuals of the differential equation in a least square approach. The idea of using frequency shifts to speed up the convergence of the DNN is similar to one used in the wave-ray MGM [44] for high frequency problems where phase factors set at some frequency lattices together with smooth slowly variant amplitude functions are used in the wave-ray MGM framework. Also, the coupled PhaseDNN can be viewed as a Fourier-expansion-like neural network, please refer to the wavelet-like networks [6, 21].

The rest of the chapter will be organized as follows. In Section 2.2, we will review the fast low frequency convergence property of neural network and present the parallel version phase shift deep neural network - PhaseDNN. Based on the properties of the PhaseDNN, a coupled PhaseDNN is introduced in Section 2.3 to reduce the cost of learning in training the DNN for approximations. Then, the coupled PhaseDNN is used to find the solutions of wave problems in inhomogeneous media using either differential equation or integral equation formulations. Section 2.4 contains various numerical results of the PhaseDNN for approximations and solutions of wave problems.

## 2.2. A Parallel phase shift DNN (PhaseDNN) for high frequency approximation

A deep neural network (DNN) is a sequential alternative composition of linear functions and nonlinear activation functions. Given $d_1, d_2 \geq 1$, let $\Theta(\boldsymbol{x}) : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}$ is a linear function with the form $\Theta(\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$, where $\boldsymbol{W} = (w_{ij}) \in \mathbb{R}^{d_2 \times d_1}$, $\boldsymbol{b} \in \mathbb{R}^{d_2}$ are called weights and biases, respectively. The nonlinear activation function $\sigma(u) : \mathbb{R} \to \mathbb{R}$. By applying $\sigma(u)$ componentwisely, we can extend the activation function to $\sigma(u) : \mathbb{R}^d \to \mathbb{R}^d$. A DNN with $H+1$ layers can be expressed in a compact form as

$$\begin{aligned} T(\boldsymbol{x}) &= T^H(\boldsymbol{x}), \\ T^h(\boldsymbol{x}) &= [\Theta^h \circ \sigma](T^{h-1}(\boldsymbol{x})), \quad h = 1, 2, \dots H, \end{aligned} \tag{2.1}$$

with $T^0(\boldsymbol{x}) = \Theta^0(\boldsymbol{x})$, or equivalently, it explicitly:

$$T(\boldsymbol{x}) = \Theta^H \circ \sigma \circ \Theta^{H-1} \circ \sigma \cdots \circ \Theta^1 \circ \sigma \circ \Theta^0(\boldsymbol{x}). \tag{2.2}$$

Here, $\Theta^h(\boldsymbol{x}) = \boldsymbol{W}^{(h)}\boldsymbol{x} + \boldsymbol{b}^{(h)} : \mathbb{R}^{d_h} \to \mathbb{R}^{d_{h+1}}$ are linear functions. This DNN is also said to have $H$ hidden layers and its $h$-th layer has $d_h$ neurons.

In approximating a function $f(x)$ by a DNN through training, we minimize the least square loss function

$$L(\boldsymbol{W}^{(0)}, \boldsymbol{b}^{(1)}, \boldsymbol{W}^{(1)}, \boldsymbol{b}^{(1)}, \dots, \boldsymbol{W}^{(H)}, \boldsymbol{b}^{(H)}) = \|f(\boldsymbol{x}) - T(\boldsymbol{x})\|_2^2 = \int_{-\infty}^{+\infty} |f(\boldsymbol{x}) - T(\boldsymbol{x})|^2 \, \mathrm{d}x. \tag{2.3}$$

For simplicity, we denote all the parameters in DNN by a parameter vector $\theta$, i.e.

$$\theta = (\boldsymbol{W}_{11}^{(0)}, \dots, \boldsymbol{W}_{d_0 d_1}^{(0)}, \boldsymbol{b}_1^{(0)} \dots \boldsymbol{b}_{d_1}^{(0)}, \boldsymbol{W}_{11}^{(1)}, \dots, \boldsymbol{W}_{d_1 d_2}^{(1)}, \boldsymbol{b}_1^{(1)} \dots \boldsymbol{b}_{d_2}^{(1)} \dots) \in \mathbb{R}^p.$$

Here, $p = (d_0+1)\times d_1 + (d_1+1)\times d_2 + (d_2+1)\times d_3 + \ldots (d_H+1)$ is the total number of the parameters. Numerically, with $N$ training data $\{x_1, x_2, \ldots, x_N\}$, the numerical loss function is defined as

$$L_N(\theta) = \sum_{i=1}^{N} |f(x_i) - T(x_i, \theta)|^2. \tag{2.4}$$

We can study the loss function in the frequency space and first, define the Fourier transform and its inverse of a function $f(\boldsymbol{x})$ by

$$\mathcal{F}[f](k) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} f(x) e^{-ikx}\, \mathrm{d}x, \qquad \mathcal{F}^{-1}[\hat{f}](x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} \hat{f}(k) e^{ikx}\, \mathrm{d}k. \tag{2.5}$$

Assuming the Fourier transform of $f(x)$ and $T(x,\theta)$ exist, by Parseval's equality, we have

$$L(\theta) = \int_{-\infty}^{+\infty} |f(x) - T(x)|^2\, \mathrm{d}x = \int_{-\infty}^{+\infty} |\hat{f}(x) - \hat{T}(x)|^2\, \mathrm{d}k. \tag{2.6}$$

Let $L(k,\theta) = |\hat{f}(x) - \hat{T}(x)|^2$ denote the $k$-frequency component of $L(\theta)$. According to [70], we have the following theorem.

**Theorem 7.** *Considering a DNN of one hidden layer with* tanh *activation function. For any frequencies $k_1$ and $k_2$ such that $|\hat{f}(k_1)| > 0$, $|\hat{f}(k_2)| > 0$, and $|k_2| > |k_1| > 0$, there exist positive constants $c$ and $C$ such that for sufficiently small $\delta$, we have*

$$\frac{\mu\left(\left\{\boldsymbol{W}^{(0)} : |\frac{\partial L(k_1)}{\partial \theta_j}| > |\frac{\partial L(k_2)}{\partial \theta_j}|, for\, all\, j = 1, 2, \ldots p\right\} \cap B_\delta\right)}{\mu(B_\delta)} > 1 - C\exp(-c/\delta), \tag{2.7}$$

*where $B_\delta$ is a ball with radius $\delta$ centered at the origin of the $\boldsymbol{W}^{(0)}$-parameter space and $\mu(\cdot)$ is the Lebesgue measure.*

Theorem 7 states that when gradient decent method is applied to the loss function $L(\theta)$, for most of the $\boldsymbol{W}^{(0)}$-parameter space, the low frequency component of loss function con-

verges faster than the high frequency component. It is equivalent to say that the low frequency part of the DNN converges to that of the target function $T(x)$ faster. Although the result is only proved for a DNN with one hidden layer, these phenomena has also been observed in higher dimensional experiments [59, 70]. Similar results have been proved for ReLU network [59]. Therefore, to speed up the learning of higher frequency contents of a target function $f(x)$, we can employ a phase shift technique [9] to translate higher frequency spectrum $\hat{f}(k)$ to a frequency range of $[-K_0, K_0]$ for some small frequency $K_0$. Such a shift in frequency is a simple phase factor multiplication on the training data in the physical space.

### 2.2.1. Frequency selection kernel $\phi_j^\vee(x)$

For a given frequency increment $\Delta k$, say, $\Delta k = 2K_0$, let us assume that for some integer $M > 0$,

$$\operatorname{supp} \hat{f}(k) \subset [-M\Delta k, M\Delta k].$$

We first construct a mesh for the interval $[-M\Delta k, M\Delta k]$ by

$$\omega_j = j\Delta k, j = -M, \cdots, M, \tag{2.8}$$

Then, we introduce a POU (partition of unit) $\{\phi_j(k)\}_{j=-M}^{M}$ for the interval $[-M\Delta k, M\Delta k]$ associated with the mesh as

$$1 = \sum_{j=-M}^{M} \phi_j(k), \ \ k \in [-M\Delta k, M\Delta k]. \tag{2.9}$$

The simplest choice of $\phi_j(k)$ is $\phi_j(k) = \phi(\frac{k-\omega_j}{\Delta k})$, and $\phi(k)$ is just the characteristic function of $[-\frac{1}{2}, \frac{1}{2}]$, i.e., $\phi(k) = \chi_{[-\frac{1}{2},\frac{1}{2}]}(k)$. The inverse Fourier transform $\mathcal{F}^{-1}$ of $\phi(k)$, indicated by $\vee$, is $\phi^\vee(x) = \frac{1}{\sqrt{2\pi}} \frac{\sin\frac{x}{2}}{\frac{x}{2}}$.

With the POU in (2.9), we can decompose the target function $f(x)$ in the Fourier space as follows,

$$\hat{f}(k) = \sum_{j=-M}^{M} \phi_j(k)\hat{f}(k) \triangleq \sum_{j=-M}^{M} \hat{f}_j(k), \tag{2.10}$$

which will give a corresponding decomposition in $x$-space as

$$f(x) = \sum_{j=-M}^{M} f_j(x), \tag{2.11}$$

where

$$f_j(x) = \mathcal{F}^{-1}[\hat{f}_j](x).$$

The decomposition (2.11) involves $2M+1$ functions $f_j(x)$, whose frequency spectrum is limited to $[\omega_j - \frac{\Delta k}{2}, \omega_j + \frac{\Delta k}{2}]$. Therefore, a simple phase shift could translate its spectrum to $[-\Delta k/2, \Delta k/2]$, and it could be learned quickly by a relatively small DNN $T_j(x)$ with a few training epochs.

Specifically, as the support of $\hat{f}_j(k)$ is $[\omega_j - \frac{\Delta k}{2}, \omega_j + \frac{\Delta k}{2}]$, then $\hat{f}_j(k+\omega_j)$ is supported in $[-\Delta k/2, \Delta k/2]$, and its inverse Fourier transform $\mathcal{F}^{-1}\left[\hat{f}_j(k+\omega_j)\right]$, denoted as

$$f_j^{\text{shift}}(x) = \mathcal{F}^{-1}\left[\hat{f}_j(k+\omega_j)\right](x) \tag{2.12}$$

can be learned quickly by a DNN $T_j(x,\theta)$ by minimizing a loss function

$$L_j(\theta) = \int_{-\infty}^{\infty} |f_j^{\text{shift}}(x) - T_j(x,\theta)|^2 \, \mathrm{d}x \tag{2.13}$$

in an $n_0$-epochs of training.

Moreover, we know that

$$f_j^{\text{shift}}(x_i) = e^{-i\omega_j x_i} f_j(x_i), 1 \le i \le N, \tag{2.14}$$

which provides the training data for $f_j^{\text{shift}}(x)$. Equation (2.14) shows that once $f_j^{\text{shift}}(x)$ is learned, $f_j(x)$ is also learned by removing the phase factor.

$$f_j(x) \approx e^{i\omega_j x} T_j(x, \theta^{(n_0)}). \tag{2.15}$$

Now with all $f_j(x)$ for $-M \leq j \leq M$ learned after $n_0$ steps of training each, we have an approximation to $f(x)$ over all frequency range $[-M\Delta k, M\Delta k]$ as follows

$$f(x) \approx \sum_{j=-M}^{M} e^{i\omega_j x} T_j(x, \theta^{(n_0)}), \tag{2.16}$$

where $\theta^{(n_0)}$ is the value of parameters after $n_0$ steps of training.

### 2.2.2. Training Data for parallel phase shift DNN (PhaseDNN) algorithm

In practice, we only know the value of $f(x)$ at some locations, which will be used to train the PhaseDNN, namely, our goal is to learn the function $f(x)$ using a training data set

$$\{x_i, f_i = f(x_i)\}_{i=1}^{N}. \tag{2.17}$$

In order to apply the decomposition (2.11) to $f(x)$, when carrying out the sub-training problem (2.13), we need to compute the training data for $f_j^{\text{shift}}(x)$ based on original training data (2.17). This procedure can be done in $x$-space through the following convolution

$$\begin{aligned} f_j^{\text{shift}}(x_i) = e^{-i\omega_j x_i} f_j(x_i) = e^{-i\omega_j x_i} \phi_j^{\vee} * f(x_i) = \int_{-\infty}^{\infty} \phi_j^{\vee}(x_i - s) f(s) ds \\ \approx \frac{2\delta}{N_s} \sum_{x_s \in (x_i - \delta, x_i + \delta)} e^{-i\omega_j x_i} \phi_j^{\vee}(x_i - x_s) f(x_s), \end{aligned} \tag{2.18}$$

where $\delta$ is chosen such that the kernel function $|\phi^{\vee}(k)|$ is small enough outside $(-\delta, \delta)$.

Note that the generation of training data for $f_j^{\text{shift}}(x_i)$ and the subsequent training of each DNN $T_j(x,\theta)$ to approximate $f_j^{\text{shift}}(x)$ can be done in parallel. For this reason, this approach will be termed as a parallel PhaseDNN, which will consist of the following steps: (1) select the phase frequency $\omega_j$, (2) for each $j$, construct the training data $f_j^{\text{shift}}(x_i)$, (3) train all DNN $T_j(x,\theta)$, and (4) combing all individual DNN $T_j(x,\theta)$ with a corresponding shift backward to get an approximation for the original function $f(x)$.

## 2.3. A coupled PhaseDNN

### 2.3.1. Approximating functions

In the previous section, we use the frequency selection kernel $\phi_j^\vee(x)$ to decompose the training data into different frequency components, each of them after being phase-shifted can be represented by a small DNN. This method can be implemented in parallel. This strategy requires the use of convolution in (2.18) to construct the training data for each small DNN. In principle, the computation cost of this part can be reduced to $O(N\log(N))$ by using FFT, here $N$ is the number of samples provided that the distribution of the data is close to uniform and cover the whole domain where the approximation is sought. However, for randomly distributed and scattered samples with many regions without data, the FFT technique will not be applicable or efficient, and computing the convolution directly via a matrix multiplication requires a storage and computation of $O(N^2)$. As a result, this convolution process strongly restricts the performance of PhaseDNN for higher dimensions and larger data set.

To avoid this problem, based on the construction of the parallel PhaseDNN (2.16), we would like to consider a coupled weighted phase-shifted DNNs as an ansatz for a coupled PhaseDNN,

$$T(x) = \sum_{m=1}^{M} e^{i\omega_m x} T_m(x), \tag{2.19}$$

to approximate $f(x), x \in \mathbb{R}^d$, where $T_m(x)$ are relatively small complex valued DNNs, i.e., $T_m(x) = T_m^{(real)}(x) + iT_m^{(imag)}(x)$. $T_m^{(real)}(x)$ and $T_m^{(imag)}(x)$ are two independent DNNs. $\{\omega_m\}_{m=1}^M$ are frequencies we are particularly interested in from the target function.

We will minimize the following least square loss function

$$L(\theta) = \int_{-\infty}^{+\infty} |f(x) - T(x)|^2 \, \mathrm{d}x, \tag{2.20}$$

or numerically,

$$L_N(\theta) = \sum_{i=1}^{N} |f(x_i) - T(x_i)|^2 = \sum_{i=1}^{N} \left| f(x_i) - \sum_{m=1}^{M} e^{i\omega_m x_i} T_m(x_i) \right|^2. \tag{2.21}$$

**Remark 1.** *This method is similar to an expansion with Fourier modes of selected frequency with variable coefficients defined by DNNs. When $f(x)$ is a real function, it is equivalent to use real 'Fourier' series rather than complex 'Fourier' series. Namely, we will consider the following sine and cosine expansions*

$$T(x) = \sum_{m=1}^{M} A_m \cos(\omega_m x) + B_m \sin(\omega_m x) \tag{2.22}$$

*to approximate $f(x)$, where $A_m, B_m$ are DNNs while $\omega = 0$ will always be included.*

It can be shown that under the condition that the weights of input layer for each $T_m$ is small, the coupled PhaseDNN is equivalent to the parallel PhaseDNN [10]. In practical applications, the condition that the weights of input layer for each $T_m$ is small holds at the beginning of training, since we always use small random values to initialize the network. As a matter of fact, to encourage this condition in the training process, we can add a weight

regularization in the loss function, namely,

$$L_N^R(\theta) = \sum_{i=1}^{N} |f(x_i) - T(x_i)|^2 + \beta \sum_{m,l} \|\boldsymbol{W}^{m,l}\|_F^2, \tag{2.23}$$

where $x_i$ are training data, $\boldsymbol{W}^{m,l}$ is the weight matrix of the $l$-th layer of sub DNN $T_m$, $\beta$ is a regularization parameter. This weight regularization can also restrain some training disasters like gradient blowing up, etc [23].

Comparing with the approach of phase selecting kernel of previous section, the main advantage of the coupled PhaseDNN is that there is no need for computing convolutions, without the additional quadrature errors, to generate training data for the training of a selected frequency range. This allows us to deal with a large data set and higher dimensional problems. However, the coupled PhaseDNN cannot be parallelized and we must choose the frequencies $\omega_m$ before training, then build DNN $T(x)$ using these $\omega_m$ frequencies. If coupled PhaseDNN does not contain enough frequencies, we can modify the coupled phaseDNN with additional frequencies to improve the result.

### 2.3.2. Solving differential equations through least square residual minimization

The coupled PhaseDNN (2.19) will be taken as an ansatz for finding the solution of differential equations (DEs) by minimizing the least squares of the DE's residual, similar to the least square finite element (LSFE) method [4, 30] and the physics-informed neural network (PINN) [61].

The coupled PhaseDNN will approximate the solution of the following high frequency Helmholtz equation

$$\mathcal{L}[u] \triangleq u'' + (\lambda^2 + c\omega(x))u = f(x), \tag{2.24}$$

where $\lambda > 0$, $c\omega(x)$ can be viewed as a perturbation modeling the inhomogeneity of the otherwise homogeneous media.

The PhaseDNN solution, in the form of (2.19) or (2.22), for (2.24) with different boundary conditions can be sought by minimizing the following loss function,

$$L_N(\theta) = L_{ode}(\theta) + \rho L_{bc}(\theta), \tag{2.25}$$

where

$$L_{ode}(\theta) = \sum_{i=1}^{N} |\mathcal{L}[T](\cdot, \theta)(x_i) - f(x_i)|^2, \tag{2.26}$$

$\{x_i\}_{i=1}^N \in [-1, 1]$ are pre-selected locations to evaluate the residual of the DE by the DNN, and $L_{bc}$ is the boundary condition regularization term, $\rho$ is the regularization parameter.

We consider two typical kinds of boundary value problems. One is Dirichlet boundary condition for an **interior Helmholtz problem**,

$$\begin{cases} u'' + (\lambda^2 + c\omega(x))u = f(x), \\ u(a) = u_1, u(b) = u_2. \end{cases} \tag{2.27}$$

For this case, the $L_{bc}$ term is chosen naturally as

$$L_{bc} = (T(a, \theta) - u_1)^2 + (T(b, \theta) - u_2)^2. \tag{2.28}$$

**Remark 2.** *For 1D Dirichlet boundary value problems, we can also deal with the boundary condition by a slight modification of the coupled PhaseDNN by replacing the coupled PhaseDNN $T(x, \theta)$ in (2.26) with*

$$\tilde{T}(x, \theta) = T(x, \theta) + P(\theta)x + Q(\theta), \tag{2.29}$$

*where* $P(\theta) = (u_2 - u_1 + T(a,\theta) - T(b,\theta))/(b-a)$, $Q(\theta) = (u_1(b - T(a,\theta)) - u_2(a - T(b,\theta)))/(b-a)$. *With this modification,* $\tilde{T}(x,\theta)$ *will be a coupled PhaseDNN who satisfies the Dirichlet boundary condition naturally, and we can use* $L_N(\theta) = L_{ode}(\theta)$ *as the loss function.*

*In the following numerical experiments, we use both* (2.25) *and* (2.29) *to deal with the Dirichlet boundary conditions. These two methods perform similarly in the numerical tests. However, the latter approach will be difficult to apply to problems in complex 3-D domains.*

The second type is an outgoing radiation condition for an **exterior Helmholtz problem** for the wave scattering of a finite inhomogeneity described by a compact supported function $\omega(x)$,

$$\begin{cases} u'' + (\lambda^2 + c\omega(x))u = f(x) \\ u' \pm \lambda u \to 0, (x \to \mp\infty). \end{cases} \tag{2.30}$$

For the exterior problem, we assume both the perturbation $\omega(x)$ and resource function $f(x)$ are compactly supported in $[-1,1]$, and we are only interested in the solution in $[-1,1]$. To solve the differential equation on the unbounded domain, we need to truncate the domain to a finite one with an absorbing boundary condition, which in this case is the same as the radiation condition. So, we will consider the following Robin problem of the Helmholtz equation,

$$\begin{cases} u'' + (\lambda^2 + c\omega(x))u = f(x) \\ u'(-a) + \lambda u(-a) = 0, \quad u'(a) - \lambda u(a) = 0, \end{cases} \tag{2.31}$$

where a constant $a \geq 2$ is chosen. It can be shown that with $\omega(x)$ and $f(x)$ supported in $[-1,1]$, boundary value problems (2.30) and (2.31) have the same solution in $[-1,1]$. The $L_{bc}$ is chosen as

$$L_{bc} = |T'(-a,\theta) + i\lambda T(-a,\theta)|^2 + |T'(-a,\theta) - i\lambda T(-a,\theta)|^2.$$

Note that the solution is complex valued, $T(x,\theta)$ here should use form (2.19) and each $T_m$ in (2.19) should also be complex valued.

### 2.3.3. Solving integral equations for exterior Helmholtz problems

For exterior scattering problem, a more convenient approach is by converting (2.30) into an integral equation via a Green's function.

When $c = 0$, the Green's function of problem (2.30) is simply

$$G(x,x') = \frac{1}{2i\lambda} e^{i\lambda|x-x'|}. \tag{2.32}$$

We can write the solution to (2.30) with $c > 0$ in terms of $G(x,x')$ by an integral equation

$$\begin{aligned} u(x) &= \int_{-\infty}^{\infty} f(x')G(x,x')\,\mathrm{d}x' - \int_{-\infty}^{\infty} c\omega(x')u(x')G(x,x')\,\mathrm{d}x' \\ &= \int_{-1}^{1} f(x')G(x,x')\,\mathrm{d}x' - \int_{-1}^{1} c\omega(x')u(x')G(x,x')\,\mathrm{d}x' \\ &\triangleq f_G(x) - \mathcal{K}[u]. \end{aligned} \tag{2.33}$$

The second equality holds because $f(x)$ and $\omega(x)$ are supported in $[-1,1]$. The term $f_G(x)$ can be calculated by a Gaussian quadrature before training.

In order to apply PhaseDNN to approximate the solution of the integral equation (2.33), we will first discretize the integral operator in a finite dimensional space by considering a finite element mesh $\{\xi_j\}_{j=1}^M$ for the interval $[-1,1]$ and a finite element nodal basis $\{\phi_j(x)\}_{j=1}^M$ with the Kronecker property, i.e.,

$$\phi_j(\xi_k) = \delta_{jk}. \tag{2.34}$$

For a function $u(x)$ expressed in term of the basis functions $\phi_j(x)$,

$$u(x) = \sum_{j=1}^{M} u_j \phi_j(x), \qquad u_j = u(\xi_j), \tag{2.35}$$

the application of integral operator $\mathcal{K}[u]$ gives

$$\mathcal{K}[u](x) = \sum_{j=1}^{M} u_j \int_{-1}^{1} G(x,\xi)\omega(\xi)\phi_j(\xi)d\xi \triangleq \sum_{j=1}^{M} u_j \psi_j(x), \tag{2.36}$$

where

$$\psi_j(x) = \int_{-1}^{1} G(x,\xi)\omega(\xi)\phi_j(\xi)d\xi. \tag{2.37}$$

Substituting (2.35) and (2.37) into (2.33) , we have

$$\sum_{j=1}^{M} u_j \phi_j(x) = f_G(x) - c\sum_{j=1}^{M} u_j \psi_j(x). \tag{2.38}$$

We will find a DNN $T(x,\theta)$ approximation for solution $u(x)$ by minimizing the loss function of residual of (2.38) at $N$-locations $\{x_i\}_{j=1}^{N}$ with $u_j$ replaced by $T(\xi_j,\theta)$,

$$L_N(\theta) = \|\boldsymbol{A}\boldsymbol{T}(\theta) + c\boldsymbol{B}\boldsymbol{T}(\theta) - \boldsymbol{f_G}\|^2, \tag{2.39}$$

where $\boldsymbol{T}(\theta) = [T(\xi_1,\theta), T(\xi_2,\theta), \ldots T(\xi_M,\theta)] \in R^M$, $\boldsymbol{f_G} = [(f_G)(x_1), \ldots (f_G)(x_N)] \in R^N$ and $\boldsymbol{A}_{ij} = \phi_j(x_i)$, $\boldsymbol{B}_{ij} = \psi_j(x_i)$, $1 \le i \le N, 1 \le j \le M$. The matrix $\boldsymbol{B}$ can also be calculated by a Gaussian quadrature before training.

The integral equation method also applies to other types of homogenous boundary conditions, provided that one can write down the Green's function for equation (2.24) with the corresponding boundary condition, the corresponding matrix $\boldsymbol{B}$ can be computed by Gaussian quadrature, similarly.

**Remark 3.** *The residual of the integral equation formulation can also be viewed as a preconditioned version of that of the differential equation. If we write* $\mathcal{L} = \mathcal{L}_1 + c\mathcal{L}_2$*, where* $\mathcal{L}_1[u] = u'' + \lambda^2 u$*,* $\mathcal{L}_2[u] = \omega(x)u(x)$*, the operator* $G * (\cdot)$ *can be regarded as the inverse operator of* $\mathcal{L}_1$*. Thus the equation* (2.33) *is just* $(I + c\mathcal{L}_1^{-1}\mathcal{L}_2)u = \mathcal{L}_1^{-1}f$*. When* $c$ *is small, this preconditioned residual is expected to give a better performance than the PhaseDNN with least square residual of the differential equation and our numerical results later will confirm this.*

In the next section, we will apply the parallel PhaseDNN and coupled PhaseDNN method to the approximation problem, and solving differential equations with the coupled PhaseDNN. All problems in this chapter are running on an NVIDIA Tesla V100 GPU, which is on the SMU Maneframe II, SMU's high performance computing (HPC) cluster. The platform we use both include Tensorflow and Pytorch. We use Tensorflow 1.13 to solve the differential form and use Pytorch to solve the integral equation. All the training processes are carried out by Adam algorithm [34]. We set all the parameters of Adam as default except the learning rate. We will specify the learning rate in the following numerical examples.

## 2.4. Numerical results

### 2.4.1. Approximation of functions with PhaseDNN

#### *2.4.1.1. Parallel PhaseDNN*

In this section, we will present numerical results to demonstrate the capability of PhaseDNN to learn the high frequency content of target functions. In practice, we could sweep over all frequency ranges with a prescribed frequency increment $\Delta k = 5$. For the test function for which we have some rough idea about the range of frequencies in the data, only a few frequency intervals are selected for the phase shift.

We choose a target function $f(x)$ in $[-\pi, \pi]$

$$f(x) = \begin{cases} 10(\sin x + \sin 3x), & \text{if } x \in [-\pi, 0], \\ 10(\sin 23x + \sin 137x + \sin 203x), & \text{if } x \in [0, \pi]. \end{cases} \tag{2.40}$$

Because the frequencies of this function are well separated, we need not to sweep all the frequencies in $[-\infty, +\infty]$. Instead, we select $\Delta k = 5$, and use the following functions

$$\begin{aligned} \phi_1(k) &= \chi_{[-205,-200]}(k) \quad \phi_2(k) = \chi_{[-140,-135]}(k) \\ \phi_3(k) &= \chi_{[-25,-20]}(k) \qquad \phi_4(k) = \chi_{[-5,0]}(k) \\ \phi_5(k) &= \chi_{[0,5]}(k) \qquad \phi_6(k) = \chi_{[20,25]}(k) \\ \phi_7(k) &= \chi_{[135,140]}(k) \qquad \phi_8(k) = \chi_{[200,205]}(k) \end{aligned}$$

to collect the frequency information in the corresponding frequency intervals and shift the center of the interval to the origin by a phase factor. For each $f_j(x) = \mathcal{F}^{-1}[\hat{f}\phi_j](x)$, we construct two DNNs to learn its real part and imaginary part, separately. Every DNN has 4 hidden layers and each layer has 40 neurons. Namely, the DNN has a structure 1-40-40-40-40-1. The training data is obtained by 10,000 samples from the uniform distribution on $[-\pi, \pi]$ and the testing data is 10000 evenly spaced points in $[-\pi, \pi]$. We train these DNNs with 1000 epochs by Adam optimizer with training rate 0.002. The batchsize is 2000 for each DNN. The result is shown in Figure 2.1 while the detail of the training result is shown in Figure 2.2. These figures clearly shows that phase DNN can capture the various high frequencies, from low frequency $\pm 1$, $\pm 3$ to high frequency $\pm 203$ quite well. The training times of PhaseDNN are collected in Table 2.1

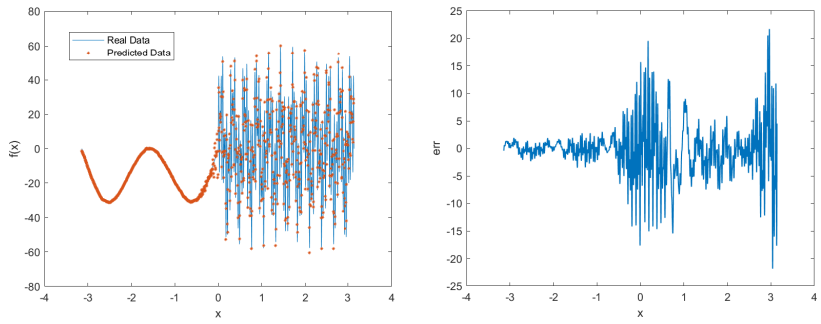



Figure 2.1: The fitting result for $f(x)$ using the parallel PhaseDNN. Left panel: the blue solid line is $f(x)$ and the data marked by red dots are the predicted value by PhaseDNN at testing data set. Right panel: the error plot.

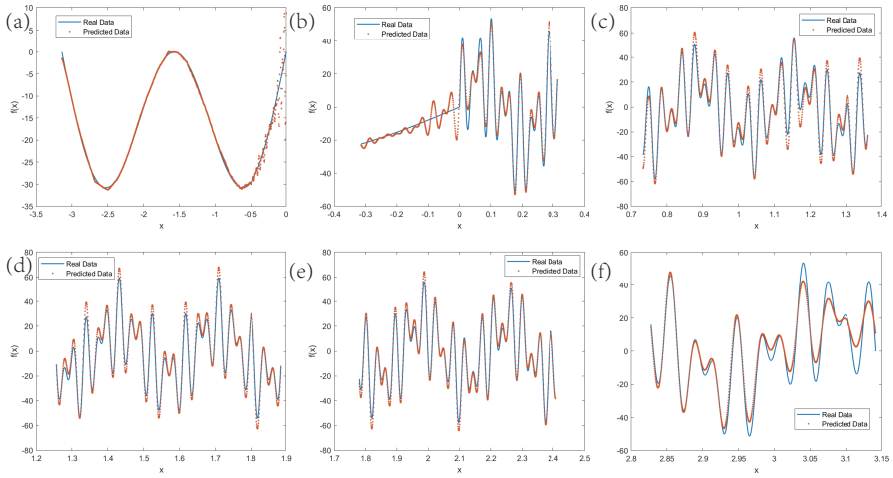



Figure 2.2: The detail results of training in different intervals. The subfigures (a)-(f) shows the results in interval $[-\pi, 0]$, $[-\pi/10, \pi/10]$, $[\pi/3 - \pi/10, \pi/3 + \pi/10]$, $[\pi/2 - \pi/10, \pi/2 + \pi/10]$, $[2\pi/3 - \pi/10, 2\pi/3 + \pi/10]$ and $[\pi - \pi/10, \pi]$ correspondingly. The blue solid line is $f(x)$ and the data marked by red dots are the values of PhaseDNN at testing data set.

It is shown that the convolution calculus for preparing data for $f_j^{\text{shift}}(x)$ costs about 40% of the total computing time. It is quite a large portion and inefficient. Because in different intervals, $f_j(x)$ can be trained in parallel, PhaseDNN is ideal to take advantage of parallel computing architectures. Although the total computing time is 210 seconds, in practice, the computation can be done in 27 seconds with parallelization. In comparison, a normal single

| Frequency Interval | Convolution time(s) | Training Time(s) | Total Time (s) |
|---|---|---|---|
| $[-205, -200]$ | 11.32 | 15.05 | 26.38 |
| $[-140, -135]$ | 11.32 | 15.18 | 26.51 |
| $[-25, -20]$ | 11.46 | 14.99 | 26.45 |
| $[-5, 0]$ | 10.70 | 14.97 | 25.67 |
| $[0, 5]$ | 11.22 | 14.98 | 26.21 |
| $[20, 25]$ | 11.26 | 15.00 | 26.27 |
| $[135, 140]$ | 11.32 | 15.13 | 26.45 |
| $[200, 205]$ | 11.32 | 15.03 | 26.36 |
| Total | 89.94 | 120.37 | 210.32 |

Table 2.1: The training time statistics. For each $j$, the training time is the sum of training time of real and imaginary part. Each DNN is trained by 1000 epochs with batchsize 2000.

fully connected 24-layer DNN with 640 neurons per hidden layer shows non-convergence in Figure 2.3(c) and (d) after over 5 hours of training.

#### *2.4.1.2. Coupled PhaseDNN*

**1-D Problem:** We will apply the coupled PhaseDNN method to the same test problem (2.40). The frequencies $\{\omega_m\}$ are selected to be $0, 5, 25, 135, 200$. For each $A_m$ and $B_m$, we also set it as a 1-40-40-40-40-1 DNN.

The training parameters are set as the same as before. Testing data is 10000 evenly spaced points in $[-\pi, \pi]$. The testing result is shown in Figure 2.3(a). The average $L^2$ relative training error and testing error are both $1.4 \times 10^{-3}$. The pointwise testing error is shown in Figure 2.3(b). It is clear that the error is concentrated in the neighborhood of 0, where the derivative of $f(x)$ is discontinuous. Out of this neighborhood, the relative maximum error is $8 \times 10^{-3}$.

To show the accuracy and efficiency of the coupled PhaseDNN, we try to learn $f(x)$ by a single fully connected DNN. The DNN is set to have 24 hidden layers and 640 neurons in each layer. The training is also carried out with 10000 random training samples, 2000 batchsize and learning rate 0.001. After 50000 epochs during 5 hours of training, the result with a total loss of 100 is shown in Figure 2.3(c) and (d) (blue line).

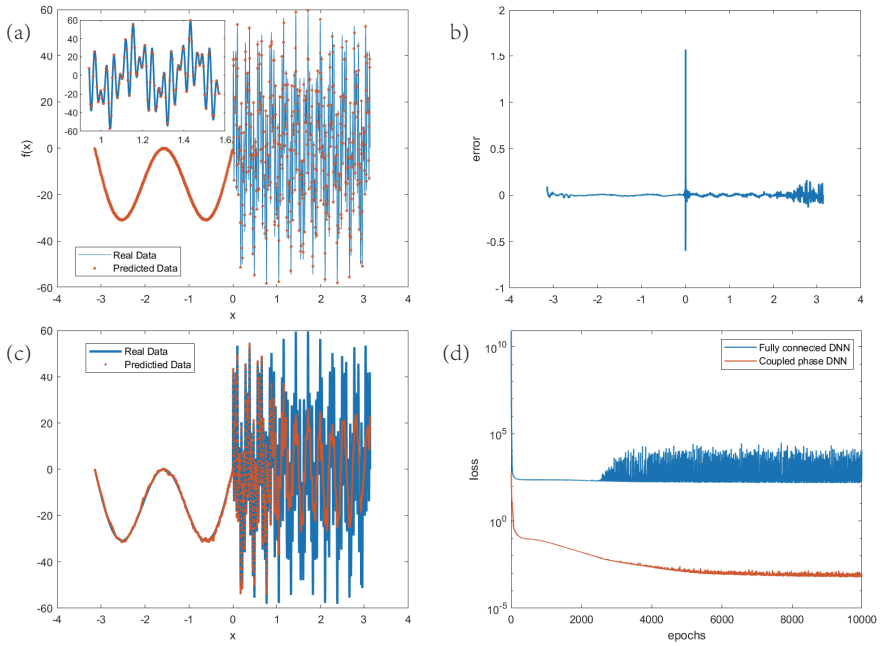



Figure 2.3: Fitting result for $f(x)$ using coupled PhaseDNN and a single fully connected DNN. (a) Fitting result of couple phase DNN after 10000 epochs of training. We select frequency $\{\omega_m\} = \{0, 5, 25, 135, 200\}$. The blue solid line is real data while the red dots are predicted value of couple PhaseDNN. The subplot at upper left is the local detail plot for interval $[0.9, 1.6]$. (b) The pointwise error of coupled PhaseDNN. (c) Fitting result of a fully connected DNN after 50000 epochs of training. The DNN has 24-layers and 640 neurons in each layer. (d) The convergence properties of coupled phase DNN and single fully connected DNN in log scale. The blue line is training error of fully connected DNN. The red line is training error of coupled phase DNN.

One can see that a single fully connected DNN cannot learn this highly oscillated function even with such a large network and a very long training time. The convergence behavior of a single DNN and coupled PhaseDNN are shown in Figure 2.3(d). It is shown that the training loss of coupled phaseDNN reduces quickly to $O(10^{-1})$ after 1000 epochs while the

loss of a single DNN stays $O(10^2)$ even after 10000 epochs. Coupled PhaseDNN is proven to be efficient in learning high frequency functions.

In fact, 10000 samples are too much for this example. It turns out that 1000 samples can lead to a good approximation with equation (2.40). Even with 500 samples, which is a much too small data set for the frequency 203, We can still get a 'reasonable' result. The testing results with 10000 evenly spaced points in $[-\pi, \pi]$ is shown in Figure 2.4.

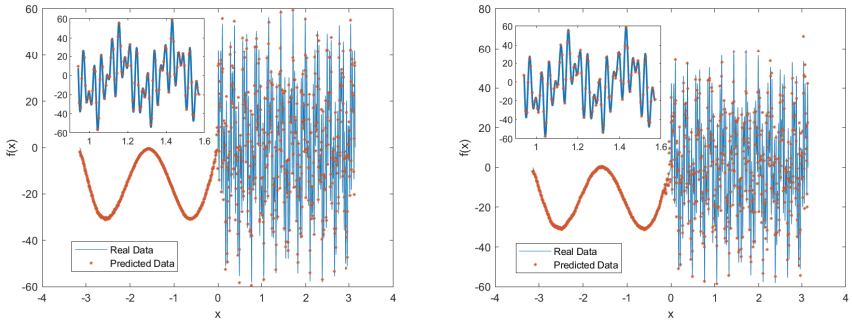



Figure 2.4: Fitting result for $f(x)$ using fewer data. We select frequency $w_n \in \{0, 5, 25, 135, 200\}$. Left panel: training with 1000 random samples. Right panel: training with 500 random samples. The subplots at upper left in each panel are local detail plots for interval $[0.9, 1.6]$.

- **Shift frequency adaptivity**

In general, there is no prior knowledge of the distribution of frequency content in the target function, it will be difficult to pre-fix the shift frequencies in the PhaseDNN. However, we could adopt an adaptive approach where the shift frequencies $\omega_m$ could be dynamically added in the construction of the PhaseDNN as follows.

We start with a coupled PhaseDNN $T(x) = \sum_{m=1}^{M} A_m \cos(\omega_m) + B_m \sin(\omega_m)$ with some pre-selected frequencies based on the best knowledge of the target function and assume it has a loss $L_M$. We can continue to train the DNN for another $n_0$ epochs, if the new loss $L'_M$ does not decrease sufficiently enough, say, $L'_M > \eta L_M$ for some constant $\eta$, we then

conclude that the coupled PhaseDNN as it is does not contain enough frequencies to learn the target function well. Therefore, to improve the coupled PhaseDNN, we can add a new frequency $\omega_{M+1}$ to $T(x)$ so $T(x) \leftarrow T(x) + A_{M+1}\cos(\omega_{M+1}) + B_{M+1}\sin(\omega_{M+1})$ and train the new coupled PhaseDNN for another $n_0$ epochs. If the loss decreases significantly, we can continue the training, otherwise we can add another new (higher) frequency again if the loss does not decay enough.

The additional $\omega_{M+1}$ should be bigger than all $\omega_m$, $m = 1, 2, \ldots M$ where $\omega_m$ can simply be $\omega_m = (m-1)\Delta K$. The decay parameter $\eta$ is chosen in $[0.8, 0.9]$. The adaptive phase shift frequency strategy is summarized as follows:

1. Set $T(x) = T_0(x)$ with an initial loss $L = L_0$, $m = 0$.
2. Train the coupled PhaseDNN for $n_0$ epochs, denote the current loss is $L'$
3. If $L' > \eta L$, $0 < \eta < 1$, then choose a new $\omega_{m+1} > \max_{1\le i\le m}\omega_i$, set $T(x) \leftarrow A_{m+1}\cos(\omega_{m+1}) + B_{m+1}\sin(\omega_{m+1})$.
4. $m \leftarrow m + 1$, back to step 2.
5. Repeat the process until the loss is small enough or $\omega_m$ is sufficiently large.

This strategy is tested on the problem in Section 2.4.1.2. We set all the training parameters the same as in Section 2.4.1.2, and choose $\omega_m = 20m, m = 1, 2, \ldots 12$. The decay parameter $\eta = 0.9$ while $n_0 = 500$. The loss curve of training is shown in Figure 2.5

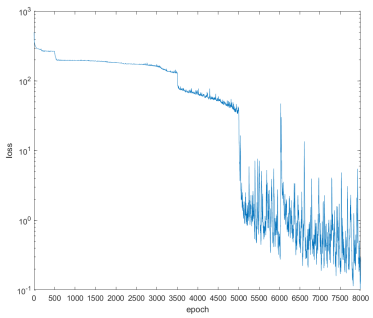



Figure 2.5: The loss curve for fitting $f(x)$ using adaptive phase range strategy. We use frequency $w_m = 20m$, $m = 1, 2, \ldots 12$ sequentially and $\eta = 0.9$. The steps on the curve corresponds the adding of frequency 20, 140 and 200.

One can see there are three abrupt drops on the loss curve, indicating the significant decrease of loss due to the adaptive procedure, i.e., these three drops at epoch 500, 3500 and 5000 correspond to adding frequency 20,140 and 200, respectively. We should note that adding more frequencies will lead to larger DNNs, thus more computing cost. Therefore, how to identify the most relevant frequencies so that the coupled PhaseDNN can be most efficient is an important issue demanding further investigations.

- **Discontinuous functions and frequency sweep**

Next we consider discontinuous functions and we replace the sin in equation (2.40) by square wave function with same frequency and learn it by (2.22) with $\omega_m \in \{0, 5, 25, 135, 200\}$ and $\omega_m = -1600 : 10 : 1600$. These two DNNs are trained with 10000 samples and 1000 epochs. The results are shown in Figure 2.6(a), (b). It can be seen that the coupled PhaseDNN has a larger error for discontinuous function, compared with the case for the smooth sin case. The sweeping strategy is preferred for discontinuous functions. From the plot of error's DFT in Figure 2.6(c) and (d), one may find that the sweeping strategy does learn the information in frequency domain $[-1600, 1600]$. For the frequency larger than 1600, neither strategy can learn it.

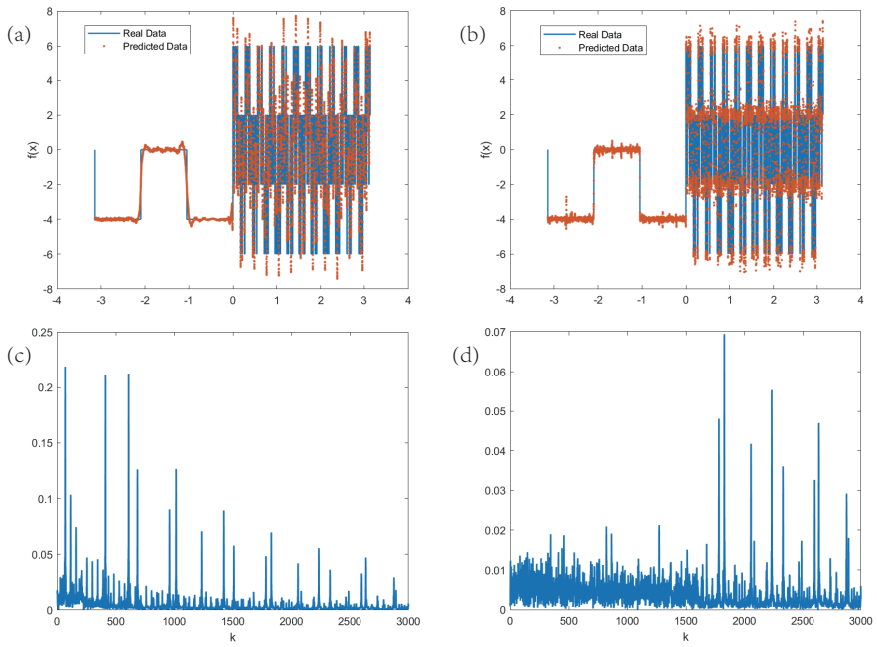



Figure 2.6: Fitting result for square wave function using selecting and sweeping methods. (a) Fitting result using selecting method with $\omega_m \in \{0, 5, 25, 135, 200\}$. (b) Fitting result using sweeping method. Frequency domain is [-1600,1600]. (c) DFT of error of selecting method. (d) DFT of error of sweeping method.

**2-D problem:** We use equation (2.22) to learn 2D and 3D problems. For 2D test, the function $G(x, y) = g(x)g(y)$ is used, where $g(x)$ is defined by

$$g(x) = \begin{cases} \sin x + \sin 3x, & \text{if } x \in [-\pi, 0], \\ \sin 23x + \sin 137x, & \text{if } x \in [0, \pi]. \end{cases} \tag{2.41}$$

The function $g(x)$ is the $f(x)$ in (2.40) without the $\sin 203x$ component. In this test, we choose $\{w_m\} \in \{0, 5, 25, 135\} \times \{0, 5, 25, 135\}$. Training setting are $640 \times 640 = 409600$ samples and 80 epochs with batchsize 100. Testing data is $100 \times 100$. The result is shown in Figure 2.7 (left).

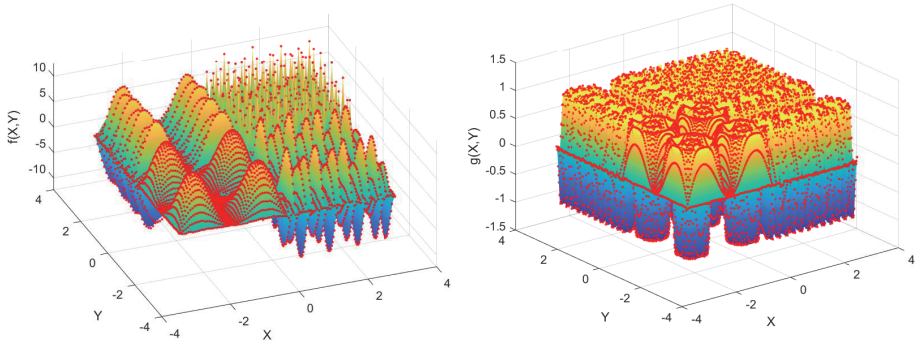



Figure 2.7: Fitting results for 2D problems using coupled PhaseDNN. Left panel: fitting result for $G(x, y)$. Right panel: fitting result for $\tilde{G}(x, y)$.

The result is good even for the highest frequency region. In this example, the highest frequency is 137. With more data, we can learn a function of even higher frequency.

Furthermore, we test another problem with $\tilde{G}(x, y) = \sin(\tilde{g}(x)\tilde{g}(y))$ with

$$\tilde{g}(x) = \begin{cases} \sin x + \sin 3x, & \text{if } x \in [-\pi, 0], \\ \sin 23x, & \text{if } x \in [0, \pi]. \end{cases} \tag{2.42}$$

The highest frequency of $\tilde{G}(x, y)$ is about 200. We use the similar training setups as the previous test and choose $\omega_m \in \{10 : 10 : 210\} \times \{10 : 10 : 210\}$. The fitting result is shown in Figure 2.7 (right). The $L^2$ fitting error is $5.2 \times 10^{-3}$.

**3-D Problem:** The test problem for 3D is $H(x, y, z) = h(x)h(y)h(z)$, where

$$h(x) = \begin{cases} \sin x + \sin 3x, & \text{if } x \in [-\pi, 0] \\ \sin 23x + \sin 32x, & \text{if } x \in [0, \pi]. \end{cases} \tag{2.43}$$

The selected frequency is $\omega_m \in \{0, 5, 25, 30\} \times \{0, 5, 25, 30\} \times \{0, 5, 25, 30\}$. Training uses $250 \times 250 \times 250 = 1.5625 \times 10^7$ random samples and 150 epochs with batchsize 15625. For

plotting, we choose hypersurface $z = 1$ and $z = \frac{1}{2}(x+y)$ as test data. Each $A_m, B_m$ is chosen to be 1-20-20-20-20-1 The relative $L^2$ error is $3 \times 10^{-2}$. Results are shown in Figure 2.8.

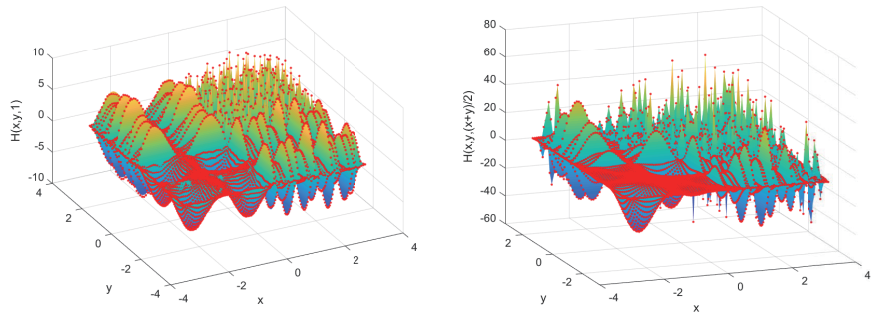



Figure 2.8: Fitting result for $H(x, y, z)$. Left panel: fitting result on hypersurface $z = 1$. Right panel: fitting result on hypersurface $z = \frac{1}{2}(x + y)$.

**The number of data**. Basically, the data set must be big enough so that it can reveal all the frequencies. That means we still need $O(N^d)$ data. For each direction, $N$ samples must reveal the highest frequency of this direction. This means, even though DNN has the advantage that the number of unknowns $p$ increases linearly with respect to the number of dimension, we still need an exponentially large data set. In our 3D test, we use 250 samples to reveal frequency 32 on average, the total number $1.56 \times 10^7$ is really a very big data set.

It is clear that the PhaseDNN approach cannot overcome curse of dimensionality. If we have no prior knowledge on the frequency distribution, coupled phaseDNN can be considered as building a mesh in Fourier space. Thus, in general, the number of $\omega_m$ will increase exponentially. In our 3D test problem, there are 172 different $\omega_m$, which corresponds to 343 sub DNNs. The whole coupled phaseDNN $T(x)$ has a width over 6000. It is a shallow but very wide DNN. The number of parameters is large. With a large number of data, the whole training takes about 8 hours.

### 2.4.2. Coupled PhaseDNN for solving PDEs with high frequency solutions

#### *2.4.2.1. Helmholtz equation with constant wave numbers*

We will solve the constant coefficient case for (2.27), namely, $c = 0$ with zero boundary condition $u(-1) = u(1) = 0$ and the following high oscillatory forcing term

$$f(x) = (\lambda^2 - \mu^2)\sin(\mu x). \tag{2.44}$$

We set $\omega_m \in \{0, \lambda, \mu\}$, each $A_m, B_m$ to be 1-40-40-40-40-1 DNN. The entire $T(x, \theta)$ is trained with 10000 evenly spaced samples, 100 epochs and batchsize 100. We choose four special cases: $\lambda = 3, \mu = 2$; $\lambda = 200, \mu = 2$; $\lambda = 2, \mu = 200$; and $\lambda = 300, \mu = 200$. The result is shown in Figure 2.9.

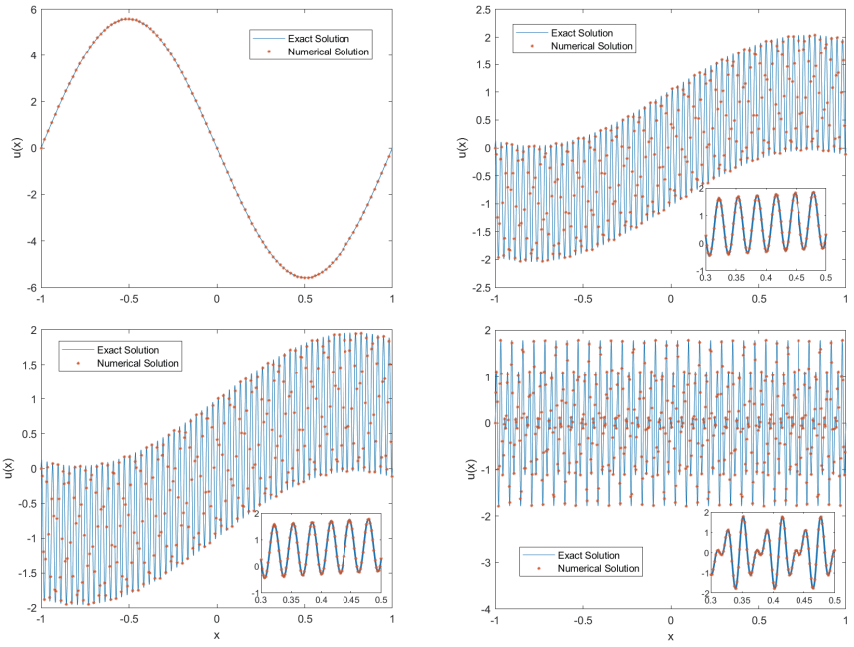



Figure 2.9: Numerical and exact solution of problem (2.27) with $c = 0$ and different $\lambda$ and $\mu$. (a): $\lambda = 3, \mu = 2$. (b): $\lambda = 200, \mu = 2$. (c): $\lambda = 2, \mu = 200$. (d): $\lambda = 300, \mu = 200$. The subplots in figure (b), (c) and (d) are local detail plots for interval $[0.3, 0.5]$.

The training takes about 5 minutes with a maximum error is $O(10^{-4})$. For comparison, a single fully connected DNN with similar scale as $T(x)$ cannot solve the equation at all when the frequency is high. The training result after 1500 epochs for $\lambda = 3, \mu = 2$ and $\lambda = 200, \mu = 2$ are shown in Figure 2.10, showing the non-convergence for high frequency solution using a common fully connected DNN (Figure 2.10 (right)).

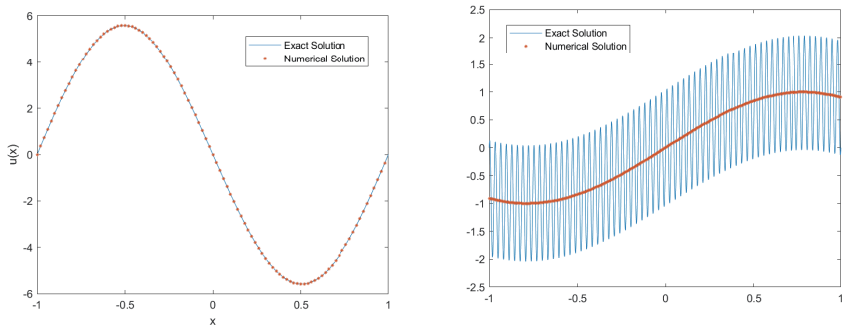



Figure 2.10: Non-convergence of usual fully connected DNN for high frequency case: numerical and exact solution of problem (2.27) with different $\lambda$ and $\mu$. Left panel: $\lambda = 3, \mu = 2$. Right panel: $\lambda = 200, \mu = 2$.

Next, we will consider the case of a more complicated solution beyond plane waves with an exact solution $u(x) = J_0(\mu x) + 0.2\cos(\lambda x)$, where $J_0(x)$ is the 0-order Bessel function. For this case, the forcing term in (2.27) is $f(x) = \mu^2 J_0''(\mu x) + \lambda^2 J_0(\mu x)$ with corresponding nonzero Dirichlet boundary conditions. We choose $\omega_j \in \{0, \lambda, \mu\}$ in a coupled PhaseDNN, which gives accurate numerical results for $\lambda = 200, \mu = 100$ as shown in Figure 2.11.

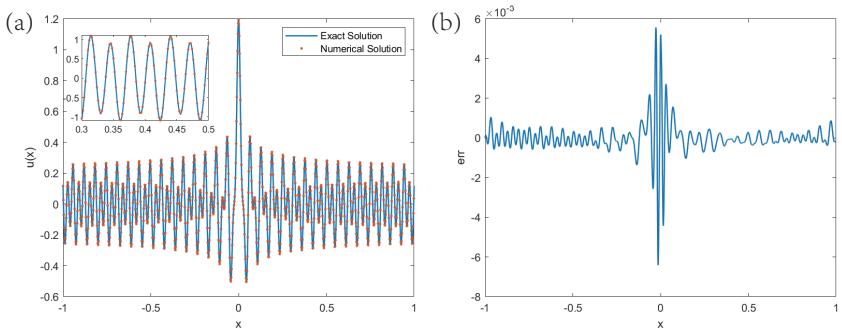



Figure 2.11: Numerical and exact solution of problem (2.27) with exact solution $J_0(\mu x) + 0.2\cos(\lambda x)$. We choose $\lambda = 200$, $\mu = 100$. (a): Numerical and exact solution of problem (2.27). Blue line:exact solution. Red dots: numerical solution obtained by coupled PhaseDNN. The subplot at upper left is the local detail plot for interval [0.3,0.5](b): Error of the numerical solution.

#### *2.4.2.2. Helmholtz equation with variable wave numbers*

Next we solve problem (2.27) with $u(-1) = u(1) = 0$ and $c > 0$, and a variable wave number

$$\omega(x) = \sin(mx^2), \tag{2.45}$$

where $m > 0$ is a constant. As there is no explicit exact solution to this equation, the numerical solution obtained by a finite difference method with a fine mesh will be used as the reference solution.

**Differential equation formulation.** We will first find the solution by solving the Helmholtz differential equation with a coupled PhasedDNN.

We consider the parameter $\lambda = 2$, $\mu = 200$, $c = 0.9\lambda^2 = 3.6$ and $m = 1$ in equation (2.27), which corresponds to a high frequency external wave source and a low wave number with small background media inhomogeneity. In the coupled PhaseDNN, we choose $w_m \in \{1, 2, 3, 4, 200\}$. Other training parameters are set to be similar as in the constant coefficient

case. The numerical result of coupled phaseDNN and reference solution is shown in Figure 2.12 and the absolute error is in the order of $O(10^{-3})$.

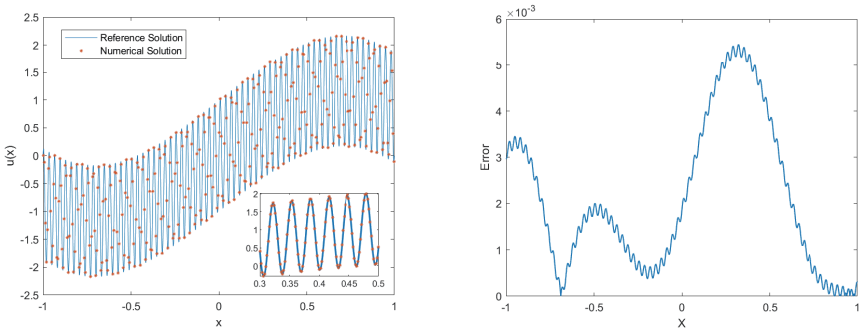



Figure 2.12: Variable coefficient Helmholtz equations (2.27): Numerical and exact solution using coupled phaseDNN. $\lambda = 2$, $\mu = 200$, $c = 3.6$ and $m = 1$. Left panel: The numerical solution and reference solution obtained by finite difference method. The subplot at lower right is the local detail plot for interval $[0.3, 0.5]$. Right panel: The absolute value of the difference between numerical solution and reference solution.

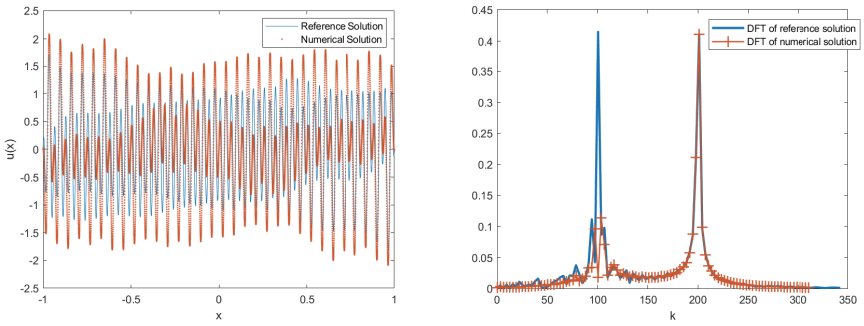



Figure 2.13: Numerical and reference solutions to problem (2.27) using coupled PhaseDNN. $\lambda = 100$, $\mu = 200$, $c = 0.1\lambda^2$ and $m = 100$. Left panel: The numerical solution obtained by least square based coupled PhaseDNN method and the reference solution. Right panel: the discrete Fourier transform of reference solution and numerical solution. Blue line: DFT of reference solution. Red line: DFT of PhaseDNN numerical solution.

Next, we choose $\lambda = 100$, $\mu = 200$, $c = 0.1\lambda^2 = 1000$ and $m = 100$, which corresponds to a high frequency external wave source and a high wave number with larger background media inhomogeneity. $\omega_m$ is picked to be $\{0, 90, 100, 110, 190, 200, 210\}$. The learning result is shown in Figure 2.13. One can see in the Fourier space for the solution (the right panel of Figure 2.13) that coupled PhaseDNN with least square residual of the differential equation

shows the unresolved error at the $\pm\lambda$ frequency while the $\mu$ frequency converges well. This error is due to the behavior of the Fourier symbol of the differential operator at the $\lambda$ frequency and turns out to be difficult to avoid. Theoretical analysis and remedy of these phenomena will be carried out in a future work.

**Discontinuous coefficient - waves in layered media** We apply the PhaseDNN to the wave propagation in a two layer media. Let $c = 1$ in (2.27) and,

$$\omega(x) = \begin{cases} -0.75\lambda^2, & \text{if } x < 0 \\ 0, & \text{if } x \geqslant 0. \end{cases}$$

This equation models a wave in a two layer media with an interface at $x = 0$ where transmission conditions are imposed. On the left of the boundary, the wave number is $\lambda/2$ while $\lambda$ on the right side. With Dirichlet boundary condition $u(-1) = u(1) = 0$, the numerical result is shown in Figure 2.14. The parameters here are $\lambda = 50, \mu = 200$. We choose $\omega_j \in \{0, \lambda/2, \lambda, \mu\}$ for the coupled PhaseDNN.

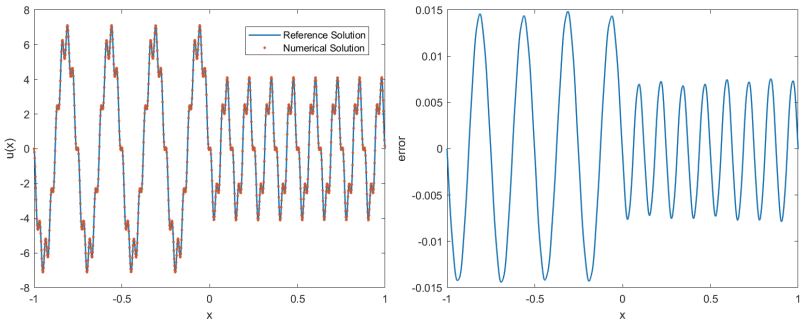



Figure 2.14: Discontinuous coefficient Helmholtz equations (2.27): Numerical and reference solutions to two layer media problem using coupled PhaseDNN. $\lambda = 50$, $\mu = 200$. Left panel: The numerical solution obtained by least square based coupled PhaseDNN method and the reference solution. Right panel: the difference between numerical solution and reference solution.

**Integral equation formulation.** Next, we will apply the integral equation approach (2.33) (2.39) to this problem. The Green's function for $u''+\lambda^2 u = \delta(x-x')$ with Dirichlet boundary condition $u(-1) = u(1) = 0$ is given by

$$G(x', x) = \begin{cases} \dfrac{(-\tan\lambda\cos\lambda x' + \sin\lambda x')(\tan\lambda\cos\lambda x + \sin\lambda x)}{2\lambda\tan\lambda}, s \le x, \\ \dfrac{(\tan\lambda\cos\lambda x' + \sin\lambda x')(-\tan\lambda\cos\lambda x + \sin\lambda x)}{2\lambda\tan\lambda}, s > x \end{cases} \tag{2.46}$$

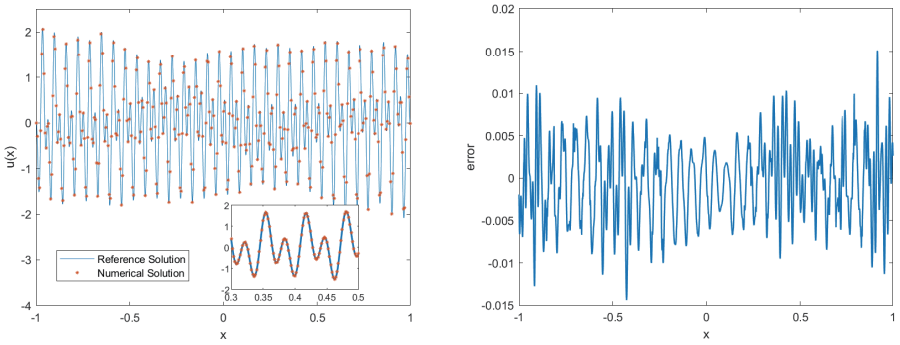



Figure 2.15: Numerical and reference solution of problem (2.27) using coupled PhaseDNN and integral equation method. $\lambda = 100$, $\mu = 200$, $c = 1000$ and $m = 100$. Left panel: The numerical solution with reference solution. The subplot is the local detail plot for interval $[0.3, 0.5]$. Right panel: The difference between numerical solution and reference solution.

With the same parameter setting, the numerical solution obtained by integral equation method is shown in Figure 2.15. The absolute error is in the order of $O(10^{-3})$. It is clear that coupled PhaseDNN with least square residual of the integral equation (2.33) (2.39) gives a much more accurate solution than that with the differential equation method and does not suffer from high wave number errors as in Figure 2.13.

*2.4.2.3. Solving elliptic equation*

We can also solve elliptic differential equation with high frequency external sources using the coupled PhaseDNN. We consider a test problem

$$\begin{cases} u'' - \lambda^2 u = -(\lambda^2 + \mu^2)\sin(\mu x), \\ u(-1) = u(1) = 0, \end{cases} \tag{2.47}$$

which has an exact solution as

$$u(x) = -\frac{\sin\mu}{\sinh\lambda}\sinh(\lambda x) + \sin(\mu x). \tag{2.48}$$

We choose $\lambda = 3$, $\mu = 250$ in equation (2.47). To solve this equation, we set a coupled PhaseDNN with $\omega_m \in \{0, \mu\}$. Each subnetwork is a fully connected DNN with 4 layers and 20 neurons in each layer. Accurate training result is shown in Figure 2.16.

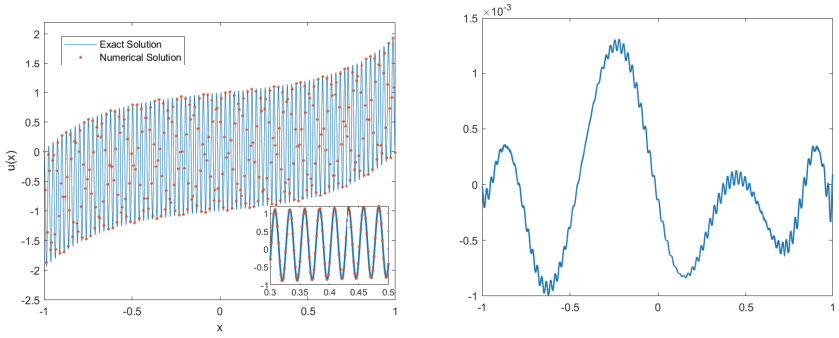



Figure 2.16: The numerical solution of equation (2.47) using coupled PhaseDNN. $\lambda = 3$, $\mu = 250$. Left panel: Numerical solution(in red) and exact solution(in blue) of equation (2.47). The subplot is the local detail plot for interval $[0.3, 0.5]$. Right panel: the error of numerical solution.

*2.4.2.4. Coupled PhaseDNN for solving exterior wave scattering problem*

We consider problem (2.30) with $\lambda = 100$, $\mu = 200$, $c = 0.1\lambda^2$. The variable wave coefficient

$$\omega(x) = \chi_{[-1,1]}(x)\sin(1 - x^2), \tag{2.49}$$

and the forcing term

$$f(x) = \chi_{[-1,1]}(x)(\lambda^2 - \mu^2)(1 - x^2)\sin(\mu x). \tag{2.50}$$

We first solve the problem with radiation boundary condition (2.31), which is an exact absorbing boundary condition in this case, by the coupled PhaseDNN for the differential equation. The real part of the solution is shown in Figure 2.17. We set each subnetwork in (2.22) to be a 1-20-20-20-20-1 DNN. Training data set is 3000 evenly spaced points in $[-2, 2]$. The training runs 3000 epochs with batchsize 600.

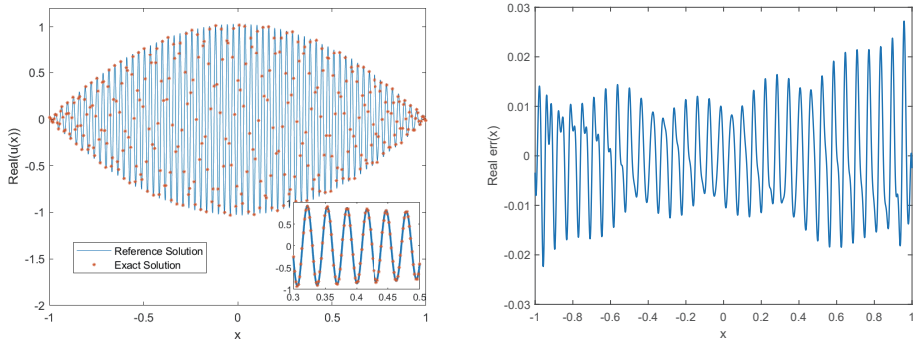



Figure 2.17: The result of exterior problem using coupled PhaseDNN for the differential equation (2.30) after 3000 epochs training. Left panel: The real part of numerical and reference solution to exterior problem. Blue line: reference solution. Red dots: numerical solution. The subplot is the local detail plot for interval $[0.3, 0.5]$. Right panel: the error of the real part of numerical solution.

Again, this problem will be solved by the coupled PhaseDNN with the integral equation (2.33) (2.39), and the result is given in Figure 2.18. The training parameters are set similarly

as for the differential equation method. Training runs 300 epochs. Better performance of the coupled PhaseDNN for the integral equation approach (2.33) (2.39), requiring much fewer training epochs, is shown, compared with the differential equation coupled PhaseDNN. Again , we can see that the PhaseDNN with integral equation formulation gives much better results of that with a differential equation.

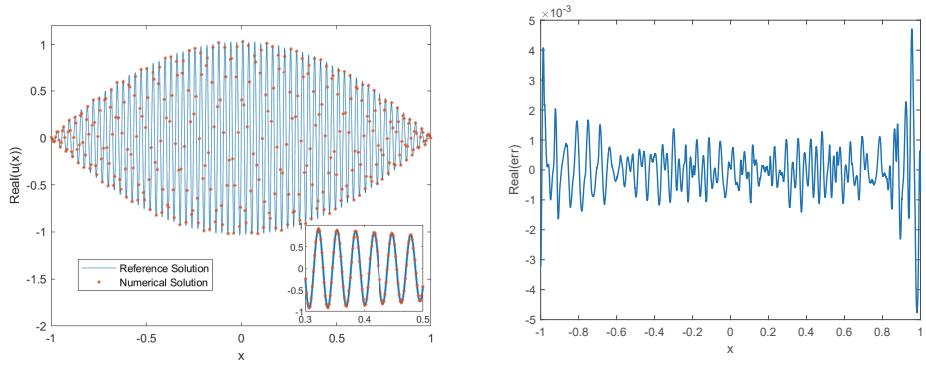



Figure 2.18: The result of exterior problem using integral equation method after 300 epochs training. Left panel: The real part of numerical and reference solution to exterior problem. Blue line: reference solution. Red dots: numerical solution. The subplot is the local detail plot for interval $[0.3, 0.5]$. Right panel: the error of the real part of numerical solution.

### 2.4.3. PhaseDNN as a meshless solver for 2D Helmholtz equation in a complex domain

Since the coupled PhaseDNNs based on least square loss do not require a mesh, this method is ideal for handling complicated domains without expensive meshing cost as in traditional finite element methods. We consider the following 2D Helmholtz equation in a domain $\Omega$ of an equilateral triangle with a circle removed inside. The bottom of the triangle is $\{(x, -\frac{\sqrt{3}}{3})|-1 \leqslant x \leqslant 1\}$ and the center of circle locates at the center of triangle and the radius is $1/\sqrt{12}$.

$$\begin{cases} \Delta u + \lambda^2 u = f(x,y), & \text{if } (x,y) \in \Omega \\ u(x,y) = g(x,y), & \text{if } (x,y) \in \partial\Omega. \end{cases} \tag{2.51}$$

The exact solution is chosen as $u(x,y) = \exp(\sin(\mu_1 x)\sin(\mu_2 y))$. $f(x,y), g(x,y)$ in (2.51) is chosen according to the differential equation. In the numerical test, the parameters are $\lambda = 100$, $\mu_1 = 30$, $\mu_2 = 50$. A coupled PhaseDNN with $\omega_j \in \{0, \lambda, \mu_1, \mu_2, 2\mu_1, 2\mu_2\} \times \{0, \lambda, \mu_1, \mu_2, 2\mu_1, 2\mu_2\}$ gives accurate numerical solution as shown in Figure 2.19

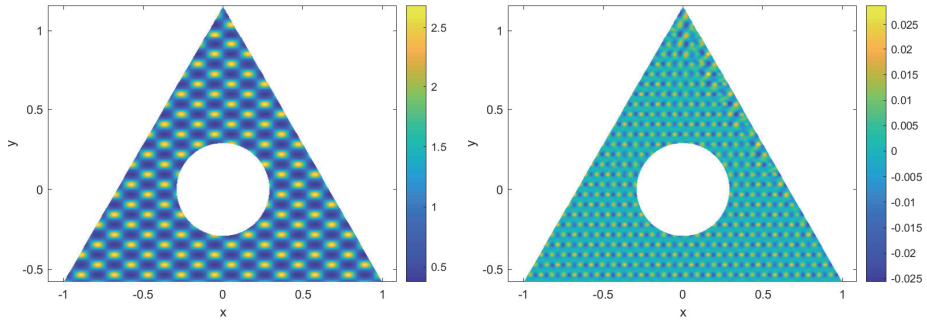



Figure 2.19: The numerical solution and error of 2D Helmholtz equation (2.51) in a complicated domain. $\lambda = 100$, $\mu_1 = 30$, $\mu_2 = 50$. Left panel: The color map of numerical solution. Right panel: the error of the numerical solution.

# Chapter 3

## Linearized Learning with Multiscale Deep Neural Networks for Stationary Navier-Stokes Equations with Oscillatory Solutions

*The content in this chapter has been published in the following journal paper in collaboration with Bo Wang and Wei Cai:*

Lizuo Liu, Bo Wang and Wei Cai, *Linearized Learning with Multiscale Deep Neural Networks for Stationary Navier-Stokes Equations with Oscillatory Solutions*, East Asian J. Appl. Math., 13, pp. 740-758 (2023) [42].

### 3.1. Introduction

Deep neural network (DNN) machine learning methods have been researched as alternative numerical methods for solving partial differential equations arising from many practical engineering problems. The deep learning framework for solving those kinds of problems uses the given partial differential equations as regularization in the loss function during training, where the auto-differentiation can be applied to the inputs of the neural network. Since auto differentiation with respect to the inputs of neural network are built-in, thus there is no need for any pre-generated meshes in the solution domain. Therefore, such a framework has the potential of being a flexible meshless method to solve governing equations from fluid and solid mechanics in complex geometries, as an alternative method to traditional finite element method. Moreover, these methods have shown much power in solving high dimensional parabolic PDEs [26, 60, 73].

Fluid mechanics, on the other hand, has also been one of the active research fields for the applications of neural network with physical information as regularization. In the work

of [7,62], the authors proposed a method that combines the Navier-Stokes (NS) equation with visualization data to predict the velocity field and pressure field, with synthetic data in [62] and real experimental imaging data in [7], respectively. In [19], a physical-informed neural network is used for solving the Reynolds-averaged Nevier-Stokes equations with Reynolds-stress components $\left(\overline{u^2}, \overline{uv}, \text{ and } \overline{v^2}\right)$ as extra outputs of the neural networks. Rao, Sun & Liu [63] proposed a mixed-variable scheme with Cauchy stress tensor to eliminate the intractability of the complex form of naive Navier-Stokes equation and its high-order derivatives (e.g., $\nabla^2$) and this scheme was applied to learn the steady flow and the transient flow passing a cylinder respectively. Furthermore, Oldenburg et al. [54] proposed the Geometry Aware Physics Informed Neural Network to handle the Navier-Stokes Equations with irregular geometry where they utilize the shape encoding network, i.e., an encoder, to reduce the geometry dimensions to a size-fixed latent vector $k$ and $k$ will be the input of two additional neural networks, one to handle the boundary constraints and one to handle physical information, i.e., the governing PDEs. In the meantime, the error estimations for neural networks to approximate the Navier-Stokes equations has been studied in [14].

Recent studies on DNNs have shown that they have a frequency dependence performance in learning solution of PDEs and fitting functions. Namely, the lower frequency components of the solution are learned first and quickly compared with the higher frequency components [70]. Several attempts have been made to remove such a frequency bias for the DNNs. The main idea is to convert the higher frequency content of the solution to a lower frequency range so the conventional DNNs can learn the solution in acceptable training epochs. One way to achieve this goal is to use phase shifts [10] while the other is to introduce a multiscale structure into the DNNs [43] in which sub-neural networks with different scales will target different ranges of the frequency in the solutions. The PhaseDNN has been shown to be very effective for high frequency wave propagation while the MscaleDNN [43] has been used to learn highly oscillatory Stokes flow solutions in complex domains [67] as well as high dimensional PDEs [73].

Most of the previous works are focusing on linear PDEs. The learning of the solution of linear PDEs via least squared residuals of the PDEs is in some sense equivalent to a fitting problem in the frequency domain in view of the Parseval's identity of Fourier transforms. So it is natural the performance improvements of multiscale DNN also holds for learning the solution of linear PDEs.

Additional difficulties arise when there are nonlinearities introduced in the PDEs. Based on the results from Jin et al. [31], it is found that it could take O $(10^4)$ epochs to solve a simple domain problem, thus ineffective and impractical especially when highly oscillating problems are to be considered. Also, the MscaleDNN applied directly to the nonlinear Navier-Stokes equation did not produce the same large improvement over conventional DNNs as in the case of the linear Stokes equations [67]. To handle such issues, we developed a linearized learning procedure for the Navier-Stokes equation by integrating linearizations of the Navier-Stokes equation in the loss function and dynamically updating the linearization as the learning is being carried out. Numerical results demonstrated the fast convergence of this approach in producing highly accurate approximations to oscillatory solutions of the Navier-Stokes equations.

The rest of the chapter will be organized as follows. Section 3.2.1 will review several iterative schemes for solving Navier-Stokes equations commonly used by the finite element methods that inspires the linearized learning scheme of this chapter. Section 3.3 will introduce the multiscale DNN structure for learning oscillatory solutions with wide range of frequencies, and then 4 linearized learning schemes for the Navier-Stokes equation will be proposed in Section 3.3.2. Numerical tests of the linearized learning schemes will be conducted for 2-D oscillatory flows in a domain containing one or multiple random cylinder(s) in Section 3.4.

## 3.2. Iterative method for stationary Navier–Stokes equations

### 3.2.1. Stationary Navier-Stokes equations

The problem considered in this chapter is the following stationary Navier-Stokes equations

$$\begin{cases} (\mathbf{u}\cdot\nabla)\mathbf{u} - \nu\Delta\mathbf{u} + \nabla p = \mathbf{f} & \text{in } \Omega, \\ \nabla\cdot\mathbf{u} = 0 & \text{in } \Omega, \\ \mathbf{u} = \mathbf{g} & \text{on } \partial\Omega, \end{cases} \tag{3.1}$$

where $\Omega$ is an open bounded domain in $\mathbb{R}^d, d = 2, 3$. $\mathbf{u}$ are the stationary flow velocity, $p$ is the pressure, $\nu$ is the kinematic viscosity. $\mathbf{f}$ is the external source term. In this chapter, we consider the incompressible flow, thus the constraints $\nabla\cdot\mathbf{u} = 0$ are considered. The boundary conditions are Dirichlet boundary conditions.

To solve the stationary Navier-Stokes equation in a least squared minimal residual approach, intuitively, the PDEs (3.1) are written in a system of first order equations as in [4] by introducing an extra velocity-gradient term, $\underline{\mathbf{U}}$, where $\underline{\mathbf{U}}_{ij} = (\partial u_i/\partial x_j), i, j = 1, \ldots, d$, for $d = 2, 3$ such that $\nabla\cdot\underline{\mathbf{U}} = \Delta\mathbf{u}$. In this chapter, we consider the velocity-pressure formulation for the Navier Stokes equations as our benchmark loss, following the results of [67],

$$-\nu\nabla\cdot\underline{\mathbf{U}} + \underline{\mathbf{U}}\cdot\mathbf{u} + \nabla p = \mathbf{f} \quad \text{in } \Omega, \tag{3.2a}$$

$$\underline{\mathbf{U}} - (\nabla\mathbf{u})^T = 0 \quad \text{in } \Omega, \tag{3.2b}$$

$$\nabla\cdot\mathbf{u} = 0 \quad \text{in } \Omega. \tag{3.2c}$$

Similarly, to obtain an equation for the pressure $p$, we take the divergence on both sides of equation (3.2a) and apply the equation (3.2c) to arrive at

$$\Delta p + 2(-u_x v_y + u_y v_x) = \nabla \cdot \mathbf{f}, \tag{3.3}$$

where the subscripts $\cdot_x, \cdot_y$ means the derivative with respect to $x, y$, respectively. Then, a loss function for the velocity gradient (Vg) and a velocity-pressure (VP) formulation of the NS equations can be defined as

$$\begin{aligned} L_{VgVP}\left(\theta_u, \theta_p, \theta_{\underline{\mathbf{U}}}\right) := & \left\|\nu \nabla \cdot \underline{\mathbf{U}} - \underline{\mathbf{U}} \cdot \mathbf{u} - \nabla p + \mathbf{f}\right\|_{\Omega}^2 \\ & + \alpha \left\|\mathbf{u} - \mathbf{g}\right\|_{\partial\Omega}^2 \\ & + \beta \left\|\Delta p + 2(-u_x v_y + u_y v_x) - \nabla \cdot \mathbf{f}\right\|_{\Omega}^2 \\ & + \gamma \left\|\nabla \cdot \mathbf{u}\right\|_{\Omega}^2 + \left\|\underline{\mathbf{U}} - (\nabla \mathbf{u})^T\right\|_{\Omega}^2 \end{aligned} \tag{3.4}$$

where $\alpha$ and $\beta$ are penalty terms to enforce the Poisson equation for the pressure $p$ and the Dirichlet boundary conditions of the velocity $\mathbf{u}$, respectively. $\|\cdot\|_{\Omega}$ is the $L^2$ norm on $\Omega$ and $\|\cdot\|_{\partial\Omega}$ is the $L^2$ norm on $\partial\Omega$. Later, we will show in Section 3.4.1 the training of the network based on the formulation (3.2), using the nonlinear first order system and the Poisson equation (3.3), converges slowly( even with the MscaleDNNs).

### 3.2.2. Iterative methods to solve stationary Navier-Stokes equations

Three iterative methods were introduced for solving the stationary Navier-Stokes equations in [27]. We will give a short review for those methods in this section.

Let $X, Y, M$ be the Hilbert spaces $X = H_0^1(\Omega)^d, Y = L^2(\Omega)^d, M = L_0^2(\Omega)$. The superscript $d$ at the end of $H_0^1(\Omega)^d$ and $L^2(\Omega)^d$ means the dimension. For the case we are interested in, $d = 2$. Then assume $\mathbf{u}_h^n \in X, p_h^n \in M$ are the solutions at $n$-th iteration of velocity $\mathbf{u}$ and pressure $p$, the three iterative schemes are given as follows.

- **Iterative method I**

$$a\left(\mathbf{u}_h^n, \mathbf{v}_h\right) - d\left(\mathbf{v}_h, p_h^n\right) + d\left(\mathbf{u}_h^n, q_h\right) + b\left(\mathbf{u}_h^{n-1}, \mathbf{u}_h^{n-1}, \mathbf{v}_h\right) = \left(\mathbf{f}, \mathbf{v}_h\right), \quad \forall \mathbf{v}_h \in X, \quad \forall q_h \in M, \quad n \geq 1 \tag{3.5}$$

- **Iterative method II**

$$a\left(\mathbf{u}_h^n, \mathbf{v}_h\right) - d\left(\mathbf{v}_h, p_h^n\right) + d\left(\mathbf{u}_h^n, q_h\right) + b\left(\mathbf{u}_h^n, \mathbf{u}_h^{n-1}, \mathbf{v}_h\right) + b\left(\mathbf{u}_h^{n-1}, \mathbf{u}_h^n, \mathbf{v}_h\right) = b\left(\mathbf{u}_h^{n-1}, \mathbf{u}_h^{n-1}, \mathbf{v}_h\right) + \left(\mathbf{f}, \mathbf{v}_h\right), \quad \forall \mathbf{v}_h \in X, \quad \forall q_h \in M, \quad n \geq 1 \tag{3.6}$$

- **Iterative method III**

$$a\left(\mathbf{u}_h^n, \mathbf{v}_h\right) - d\left(\mathbf{v}_h, p_h^n\right) + d\left(\mathbf{u}_h^n, q_h\right) + b\left(\mathbf{u}_h^{n-1}, \mathbf{u}_h^n, \mathbf{v}_h\right) = \left(\mathbf{f}, \mathbf{v}_h\right), \quad \forall \mathbf{v}_h \in X, \quad \forall q_h \in M, \quad n \geq 1 \tag{3.7}$$

where $a\left(\mathbf{u}, \mathbf{v}\right) = \nu\left(\nabla \mathbf{u}, \nabla \mathbf{v}\right), \mathbf{u}, \mathbf{v} \in X$, $d\left(\mathbf{v}, q\right) = \left(q, \mathrm{div}\mathbf{v}\right), \mathbf{v} \in X, q \in M$, $b\left(\mathbf{u}, \mathbf{v}, \mathbf{w}\right) = \left(\left(\mathbf{u} \cdot \nabla\right)\mathbf{v} + \frac{1}{2}\mathbf{u}\mathbf{v}, \mathbf{w}\right), \mathbf{u}, \mathbf{v}, \mathbf{w} \in X$. $(\cdot, \cdot)$ is the $L^2$-scalar inner product.

Assume $a_1(\mathbf{u}, \mathbf{v}, \mathbf{w}) = \left(\left(\mathbf{u} \cdot \nabla\right)\mathbf{v}, \mathbf{w}\right)$, then $b\left(\mathbf{u}, \mathbf{v}, \mathbf{w}\right) = \frac{1}{2}a_1\left(\mathbf{u}, \mathbf{v}, \mathbf{w}\right) - \frac{1}{2}a_1\left(\mathbf{u}, \mathbf{w}, \mathbf{v}\right)$. The bilinear term $a\left(\cdot, \cdot\right)$ is continuous and coercive on $X \times X$; the bilinear $d\left(\cdot, \cdot\right)$ satisfies that for all $q \in M$

$$\sup_{\mathbf{v} \in X} \frac{\left|d\left(\mathbf{v}, q\right)\right|}{\left\|\nabla \mathbf{v}\right\|_{2,X}} \geq \beta_0 \|q\|_{2,M},$$

where $\beta_0 > 0$, $\|\cdot\|_{2,X}$ and $\|\cdot\|_{2,M}$ are the corresponding $L^2$ norm in $X$ and $M$ respectively.

The trilinear form $a_1\left(\cdot, \cdot, \cdot\right)$ satisfies

$$|a_1(\mathbf{u},\mathbf{v},\mathbf{w})| \leq N\|\nabla\mathbf{u}\|_{2,X}\|\nabla\mathbf{v}\|_{2,X}\|\nabla\mathbf{w}\|_{2,X},$$

where $N > 0$.

Under stability conditions

$$\frac{4N\|\mathbf{f}\|_{-1}}{\nu^2} < 1, \quad \frac{25N\|\mathbf{f}\|_{-1}}{3\nu^2} < 1$$

where $\|\mathbf{f}\|_{-1} = \|\nabla\mathbf{f}\|_{2,X}$ and the uniqueness condition

$$\frac{N\|\mathbf{f}\|_{-1}}{\nu^2} < 1,$$

the following error estimates for the three schemes can be obtained.

Given a mesh size $h$ for a finite element method and the number of iterative steps $m$, for iterative methods I (3.5) and III (3.7), it has been shown [27] that

$$\begin{aligned}
&\nu\left\|\mathbf{u}-\mathbf{u}_h^n\right\|_{2,X} \leqslant C_1h^2 + C_2\nu\left\|\mathbf{u}_h^n-\mathbf{u}_h^{n-1}\right\|_{2,X},\\
&\nu\left\|\nabla\left(\mathbf{u}-\mathbf{u}_h^n\right)\right\|_{2,X} + \left\|p-p_h^n\right\|_{2,M} \leqslant C_3h + C_4\nu\left\|\mathbf{u}_h^n-\mathbf{u}_h^{n-1}\right\|_{2,X},
\end{aligned}$$

where $C_1,\ldots,C_4$ are constants.

For the iterative method II (3.6), we have

$$\begin{aligned}
&\nu\left\|\mathbf{u}-\mathbf{u}_h^n\right\|_{2,X} \leqslant C_5h^2 + C_6|\log h|^{1/2}\left\|\nabla\left(\mathbf{u}_h^n-\mathbf{u}_h^{n-1}\right)\right\|_{2,X}\left\|\mathbf{u}_h^n-\mathbf{u}_h^{n-1}\right\|_{2,X},\\
&\nu\left\|\nabla\left(\mathbf{u}-\mathbf{u}_h^n\right)\right\|_{2,X} + \left\|p-p_h^n\right\|_{2,X} \leqslant C_7h+\\
&\qquad C_8|\log h|^{1/2}\left\|\nabla\left(\mathbf{u}_h^n-\mathbf{u}_h^{n-1}\right)\right\|_{2,X}\left\|\mathbf{u}_h^n-\mathbf{u}_h^{n-1}\right\|_{2,X},
\end{aligned}$$

where $C_5, \ldots C_8$ are constants.

The strong forms of these three iterative methods, which will be used in defining the loss functions of linearized learning schemes, are given below.

- **Iterative method I**

$$-\nu\Delta\mathbf{u}^n + (\mathbf{u}^{n-1}\cdot\nabla)\mathbf{u}^{n-1} + \nabla p = \mathbf{f} \tag{3.8}$$

- **Iterative method II**

$$-\nu\Delta\mathbf{u}^n + \left[\left(\mathbf{u}^{n-1}\cdot\nabla\right)\mathbf{u}^n + \left(\mathbf{u}^n\cdot\nabla\right)\mathbf{u}^{n-1}\right] + \nabla p = \mathbf{f} + (\mathbf{u}^{n-1}\cdot\nabla)\mathbf{u}^{n-1} \tag{3.9}$$

- **Iterative method III**

$$-\nu\Delta\mathbf{u}^n + \left(\mathbf{u}^{n-1}\cdot\nabla\right)\mathbf{u}^n + \nabla p = \mathbf{f} \tag{3.10}$$

## 3.3. Linearized learning algorithm with multiscale deep neural network

### 3.3.1. Multiscale deep neural network (MscaleDNN)

In order to improve the capability of the DNN to represent functions with multiple scales, MscaleDNN was developed in [43], which consists of a series of parallel fully connected sub-neural networks, for solving partial differential equations. Each of the subnetworks will receive a scaled input with different scales. The final output of the MscaleDNN is a linear combination of the outputs of the parallel fully connected neural networks(refer to Figure 3.1). The individual subnetwork in the MscaleDNN with a scaled input is designed to approximate a segment of frequency content of the targeted function and the scaling is to convert a specific high frequency segment to a lower frequency domain, thus leading to frequency

uniform convergence of approximations for highly oscillating functions. Furthermore, due to the radial scaling used in the MscaleDNN as shown in [43], it could be very powerful once we consider to approximate solutions of high dimensional PDEs [73].

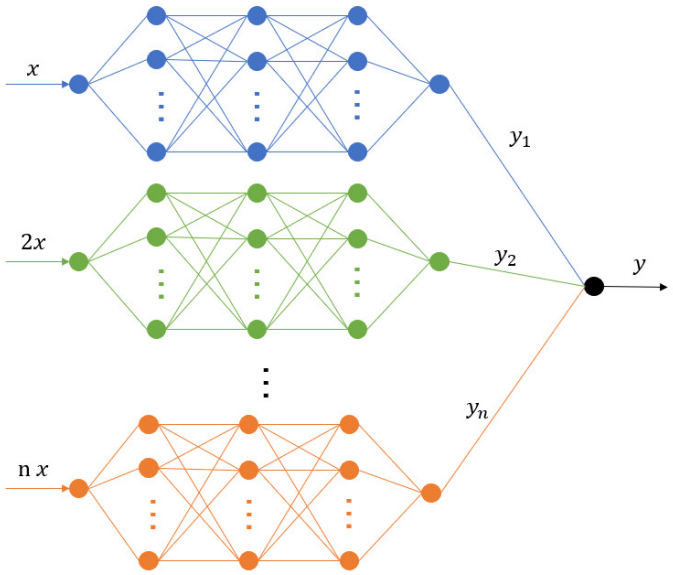



Figure 3.1: **Schematics of MscaleDNN:** The MscaleDNN shown has $n$ scales $1, 2, \ldots n$. The outputs of MscaleDNN $y$ are linear combinations of $y_1, y_2, \ldots y_n$.

Figure 3.1 shows the schematics of a typical MscaleDNN consisting of $n$ parallel subnetworks. Each subnetwork are with $L$ hidden layers and can be expressed as

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{W}^{[L-1]}\sigma \circ (\cdots(\boldsymbol{W}^{[1]}\sigma \circ (\boldsymbol{W}^{[0]}(\boldsymbol{x}) + \boldsymbol{b}^{[0]}) + \boldsymbol{b}^{[1]})\cdots) + \boldsymbol{b}^{[L-1]},$$

where $W^{[1]}, \cdots, W^{[L-1]}$ are trainable weights and $b^{[1]}, \cdots, b^{[L-1]}$ are bias to be optimized via the training, $\sigma(x)$ is the activation function. Mathematically, a MscaleDNN solution $f(\boldsymbol{x})$ is represented by the following weighted sum of subnetworks $f_{\theta^{n_i}}$ with network parameters denoted by $\theta^{n_i}$

$$f(\boldsymbol{x}) = \sum_{i=1}^{M} \omega_i f_{\theta^{n_i}}(\alpha_i \boldsymbol{x}),$$

where $\alpha_i$ is the chosen scale for the $i$-th subnetwork as shown in Figure 3.1. For more details on the design and discussion about the MscaleDNN, we refer to the original paper [43, 71].

In this chapter, the following plane wave activation function will be used due to its localized frequency property [65, 67],

$$\sigma(x) = \sin(x).$$

For the input scales, we consider the scale to be $2^{i-1}$ for the $i$-th subnetwork.

### 3.3.2. Linearization schemes for neural network training

To speed up the convergence of the training of the DNN solutions of the NS equations, in this section, we will propose an iterative training procedure for the stationary Navier-Stokes equation based on the iterative scheme introduced in Section 3.2.2 so that the residual of the training procedure are linearized. Note the term "linearized" is specific for the non-linear term $(\mathbf{u} \cdot \nabla)\,\mathbf{u}$. In a nutshell, by fixing either $\mathbf{u}$ or $\nabla\mathbf{u}$ in $(\mathbf{u} \cdot \nabla)\,\mathbf{u}$, the stationary Navier-Stokes equation turns to be a linear equation, thus we coin the term linearized learning.

Assume the learned velocities up to current epoch are $\mathbf{u}_\theta^* = (u^*, v^*)$ and the velocities to be learned at current epoch are denoted as $\mathbf{u}_\theta = (u, v)$ with $p_\theta$ as the pressure to be learned at current epoch. Thus 4 schemes with different linearization are listed below.

- **Scheme 1** (**GradFixed**): Gradients of velocities in the nonlinear term will be fixed during training.

$$\begin{aligned} -\nu\Delta\mathbf{u}_\theta + (\mathbf{u}_\theta \cdot \nabla)\mathbf{u}_\theta^* + \nabla p_\theta &= \mathbf{f}, \\ \Delta p_\theta + 2(-u_x v_y + u_y v_x) &= \nabla \cdot \mathbf{f}. \end{aligned} \tag{3.11}$$

- **Scheme 2** (**VFixed**): Velocities in the nonlinear term are fixed during training, inspired by iterative method III (3.10).

$$\begin{aligned} -\nu\Delta\mathbf{u}_\theta + (\mathbf{u}_\theta^* \cdot \nabla)\mathbf{u}_\theta + \nabla p_\theta &= \mathbf{f}, \\ \Delta p_\theta + 2(-u_x v_y + u_y v_x) &= \nabla \cdot \mathbf{f}. \end{aligned} \tag{3.12}$$

- **Scheme 3** (**VFixed1**): This is the strong form of the Iterative method II (3.9), which can be seen as a modification of Scheme 2 (VFixed) (3.12).

$$\begin{aligned} -\nu\Delta\mathbf{u}_\theta + (\mathbf{u}_\theta^* \cdot \nabla)\mathbf{u}_\theta + (\mathbf{u}_\theta \cdot \nabla)\mathbf{u}_\theta^* + \nabla p_\theta &= \mathbf{f} + (\mathbf{u}_\theta^* \cdot \nabla)\mathbf{u}_\theta^*, \\ \Delta p_\theta + 2(-u_x v_y + u_y v_x) &= \nabla \cdot \mathbf{f}. \end{aligned} \tag{3.13}$$

- **Scheme 4** (**Hybrid**): The Navier-Stokes equation in this scheme is represented as the average of Scheme 1(GradFixed) (3.11) and Scheme 2(VFixed) (3.12).

$$\begin{aligned} -\nu\Delta\mathbf{u}_\theta + \frac{1}{2}\left[(\mathbf{u}_\theta^* \cdot \nabla)\mathbf{u}_\theta + (\mathbf{u}_\theta \cdot \nabla)\mathbf{u}_\theta^*\right] + \nabla p_\theta &= \mathbf{f}, \\ \Delta p_\theta + 2(-u_x v_y + u_y v_x) &= \nabla \cdot \mathbf{f}. \end{aligned} \tag{3.14}$$

Based on equations (3.11)-(3.14), we can design four loss functions as follows.

- **Loss function for Scheme 1:**

$$\begin{aligned} \mathrm{L}_\nabla &= \mathrm{R}_\mathbf{u} + \alpha \mathrm{B}_\mathbf{u} + \beta \mathrm{R}_p + \gamma \mathrm{D}_\mathbf{u}, \\ \mathrm{R}_\mathbf{u} &= \| -\nu\Delta\mathbf{u}_\theta + (\mathbf{u}_\theta \cdot \nabla)\mathbf{u}_\theta^* + \nabla p_\theta - \mathbf{f} \|_{2,X}^2 \\ \mathrm{R}_p &= \left\| \Delta p + 2(-u_x v_y + u_y v_x) - \nabla \cdot \mathbf{f} \right\|_{2,M}^2, \\ \mathrm{B}_\mathbf{u} &= \int_{\partial\Omega} (\mathbf{u} - \mathbf{g})^2 \, dS, \\ \mathrm{D}_\mathbf{u} &= \int_\Omega (\nabla \cdot \mathbf{u})^2 \, dx. \end{aligned} \tag{3.15}$$

- **Loss function for Scheme 2:**

$$\begin{aligned}
&\mathrm{L}_{\mathbf{u}} = \mathrm{R}_{\mathbf{u}} + \alpha \mathrm{B}_{\mathbf{u}} + \beta \mathrm{R}_p + \gamma \mathrm{D}_{\mathbf{u}}, \\
&\mathrm{R}_{\mathbf{u}} = \| -\nu \Delta \mathbf{u}_\theta + (\mathbf{u}_\theta^* \cdot \nabla)\mathbf{u}_\theta + \nabla p_\theta - \mathbf{f} \|_{2,X}^2 \\
&\mathrm{R}_p = \left\| \Delta p + 2(-u_x v_y + u_y v_x) - \nabla \cdot \mathbf{f} \right\|_{2,M}^2, \\
&\mathrm{B}_{\mathbf{u}} = \int_{\partial\Omega} (\mathbf{u} - \mathbf{g})^2 \, dS, \\
&\mathrm{D}_{\mathbf{u}} = \int_{\Omega} (\nabla \cdot \mathbf{u})^2 \, dx.
\end{aligned} \tag{3.16}$$

- **Loss function for Scheme 3:**

$$\begin{aligned}
&\mathrm{L}_{\mathbf{u}1} = \mathrm{R}_{\mathbf{u}} + \alpha \mathrm{B}_{\mathbf{u}} + \beta \mathrm{R}_p + \gamma \mathrm{D}_{\mathbf{u}}, \\
&\mathrm{R}_{\mathbf{u}} = \| -\nu \Delta \mathbf{u}_\theta + (\mathbf{u}_\theta^* \cdot \nabla)\mathbf{u}_\theta + (\mathbf{u}_\theta \cdot \nabla)\mathbf{u}_\theta^* + \nabla p_\theta - \mathbf{f} - (\mathbf{u}_\theta^* \cdot \nabla)\mathbf{u}_\theta^* \|_{2,X}^2 \\
&\mathrm{R}_p = \left\| \Delta p + 2(-u_x v_y + u_y v_x) - \nabla \cdot \mathbf{f} \right\|_{2,M}^2, \\
&\mathrm{B}_{\mathbf{u}} = \int_{\partial\Omega} (\mathbf{u} - \mathbf{g})^2 \, dS \\
&\mathrm{D}_{\mathbf{u}} = \int_{\Omega} (\nabla \cdot \mathbf{u})^2 \, dx.
\end{aligned} \tag{3.17}$$

- **Loss function for Scheme 4:**

$$\begin{aligned}
&\mathrm{L}_{\mathrm{H}} = \mathrm{R}_{\mathbf{u}} + \alpha \mathrm{B}_{\mathbf{u}} + \beta \mathrm{R}_p + \gamma \mathrm{D}_{\mathbf{u}}, \\
&\mathrm{R}_{\mathbf{u}} = \left\| -\nu \Delta \mathbf{u}_\theta + \frac{1}{2}\left[ (\mathbf{u}_\theta^* \cdot \nabla)\mathbf{u}_\theta + (\mathbf{u}_\theta \cdot \nabla)\mathbf{u}_\theta^* \right] + \nabla p_\theta - \mathbf{f} \right\|_{2,X}^2 \\
&\mathrm{R}_p = \left\| \Delta p + 2(-u_x v_y + u_y v_x) - \nabla \cdot \mathbf{f} \right\|_{2,M}^2, \\
&\mathrm{B}_{\mathbf{u}} = \int_{\partial\Omega} (\mathbf{u} - \mathbf{g})^2 \, dS, \\
&\mathrm{D}_{\mathbf{u}} = \int_{\Omega} (\nabla \cdot \mathbf{u})^2 \, dx.
\end{aligned} \tag{3.18}$$

Note $\|\cdot\|_{2,X}$ are the $L^2$ norm of space $X$, $\|\cdot\|_{2,M}$ are the $L^2$ norm of space $M$, and $\alpha, \beta, \gamma$ are the penalty terms. The Poisson equation for pressure $p$ is also considered as a further regularization for training $p_\theta$.

#### 3.3.3. Linearized learning algorithms

Our linearized learning algorithm for the Navier-Stokes equation is implemented through the following steps illustrated in **Algorithm 3**. In the implementation, $\mathbf{u}_\theta^*$ and $\mathbf{u}_\theta$ are two different neural networks with same number of hidden layers and the same number of hidden neurons at each layer. Once the loss (3.15)-(3.18) decreases below a specific ratio, then the parameters of $\mathbf{u}_\theta$ are copied to $\mathbf{u}_\theta^*$, and the parameters of $\mathbf{u}_\theta^*$ will be frozen up to $c$ epochs until the loss decreases the specific ratio again. The algorithm will terminate once the training epoch is up to the maximum training epoch $N$. The optimizer considered is the popular Adam Optimizer [34].

### 3.4. Numerical results

#### 3.4.1. A benchmark: A non-oscillatory problem - effect of linearized learning

We first consider a non-oscillatory problem in a rectangle domain $\Omega = [0, 2] \times [0, 1]$ with one cylinder hole centered at $(0.7, 0.5)$ whose radius is 0.2 as shown in Figure 3.2, the analytical solutions of the incompressible Navier-Stokes equations is given as follows:

$$\begin{aligned} u &= 1 - e^{\lambda x}\cos\left(2m\pi x + 2n\pi y\right), \\ v &= \frac{\lambda}{2n\pi}e^{\lambda x}\sin\left(2m\pi x + 2n\pi y\right) + \frac{m}{n}e^{\lambda x}\cos\left(2m\pi x + 2n\pi y\right), \\ p &= \frac{1}{2}(1 - e^{2\lambda x}), \quad \lambda = \frac{\mathrm{Re}}{2} - \sqrt{\frac{\mathrm{Re}^2}{4} + 4\pi^2}, \quad \mathrm{Re} = \frac{1}{\nu}. \end{aligned} \tag{3.19}$$

**Algorithm 3 Linearized Learning Algorithm:** Once the loss $L$ is smaller than or equal to $\gamma\tau$, the parameters of $\mathbf{u}_\theta^i$ will be copied to $\mathbf{u}_\theta^*$ and note that the parameters of $\mathbf{u}_\theta^*$ will **never be updated by Adam algorithm**. It should be noted that the number of epochs before updating the fixed (linearized) term $c$ will be a hyperparameter to be adjusted carefully.

**procedure** LINEARIZEDLEARNING($\mathbf{u}_\theta^0, \mathbf{u}_\theta^*, p_\theta$)
  $\gamma \leftarrow 0.9$ ▷ The ratio to make sure the loss is strictly less than the threshold $\tau$ when updating the network $\mathbf{u}_\theta^*$
  $\tau \leftarrow 10^{12}$ ▷ The threshold
  **for** $i \leftarrow 0, \ldots, N$ **do**
    **for** $j \leftarrow 1, \ldots, c$ **do** ▷ $c$ is a variable to determine the epochs to train the new network $\mathbf{u}_\theta^i$
      $L \leftarrow \text{Loss}(\mathbf{u}_\theta^*, \mathbf{u}_\theta^i, p_\theta)$ ▷ The Loss is one of (3.15)-(3.18)
      Update $\mathbf{u}_\theta^i$ by Adam with $L$
      Update $p_\theta$ by Adam with $L$
    **end for**
    **if** $L \leq \gamma\tau$ **then**
      $\tau = L$
      $\mathbf{u}_\theta^* \leftarrow \mathbf{u}_\theta^i$
      $\mathbf{u}_\theta^{i+1} \leftarrow \mathbf{u}_\theta^i$
    **end if**
  **end for**
  **return** $\mathbf{u}_\theta^*, p_\theta$ ▷ The outputs
**end procedure**

In this non-oscillatory case, we consider the frequencies $m = 1, n = 2$ and the viscosity $\nu = 0.05$. The source term $\mathbf{f}$ is obtained by substituting the exact solution (3.19) into the Navier-Stokes equation (3.1). The boundary conditions are Dirichlet boundary conditions which are obtained by computing the analytical solutions (3.19) on boundaries, including the four edges of the rectangle and the circle inside. The penalty terms $\alpha, \beta, \gamma$ are set to be $10^4, 1, 1$ for this case. We will show the linearization could speed up the learning procedure for a neural network to learn the solution of the stationary Navier-Stokes equation in this section.

We compared the performance of the convergence of fully connected network (fcn) with different loss schemes, including the VgVP formulation (3.4) and three linearized schemes (3.15) or (3.16) or (3.18). The training data are generated by randomly sampling 160000 points inside $\Omega$ and 16000 points on $\partial\Omega$ during each epoch. In the learning process, The

number of batches for each epoch are set to be 50. We choose the fully connected neural network with 4 hidden layers, 100 hidden neurons each layer for $\mathbf{u}_\theta, \mathbf{u}_\theta^*, p_\theta$ for all cases. The hyperparameters are the same for the four cases. The losses during training for different cases by minimizing given loss function are compared in Figure 3.3. The results show that the three linearized learning neural networks converge in 300 epochs for all schemes while learning using the loss function (3.4) for the nonlinear Navier-Stokes equations fails to (top line in Figure 3.3 ). The comparisons of the x component of velocity and pressure along the line $y = 0.7$ of different linearized schemes after 300 epoch training are shown in Figure 3.4 and 3.5. For the current case, The Hybrid scheme (3.18) offers the best approximation, but the results of the GradFixed scheme (3.15) and the VFixed scheme (3.16) lose some accuracy, which corresponds to the loss Figure 3.3.

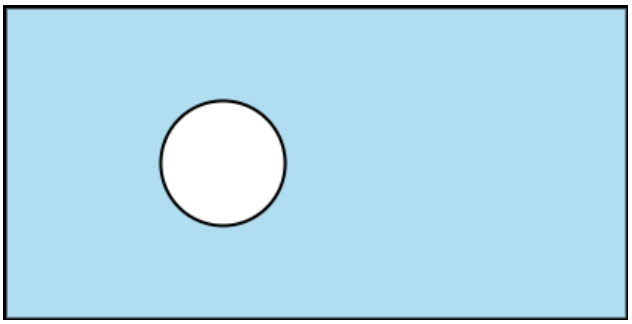


Figure 3.2: **A simple domain with one hole**: A rectangle domain $\Omega = [0, 2] \times [0, 1]$ with one cylinder hole centered at $(0.7, 0.5)$ whose radius is 0.2.

### 3.4.2. Performance: Oscillating flows learned by MscaleDNN with linearized learning

In a previous work [67], it has been shown that the MscaleDNN could improve the approximation performance dramatically when learning oscillatory solutions for the linear Stokes equations. Based on the previous experience, we consider MscaleDNN is the preferential framework for solving Navier-Stokes equation with oscillatory solutions, combined with the linearized learning scheme. The frequencies now are taken to be $m = 40, n = 35$, much

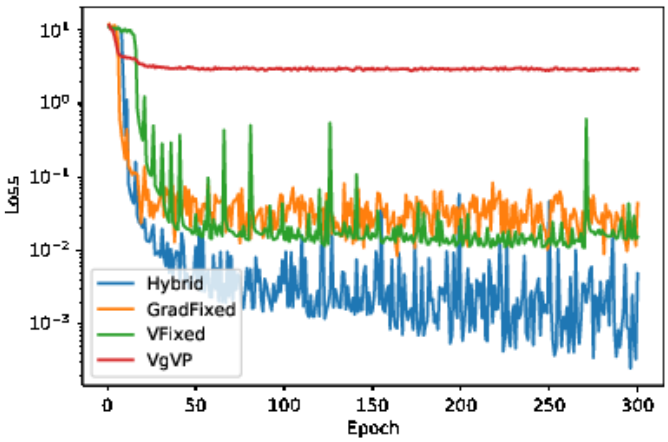
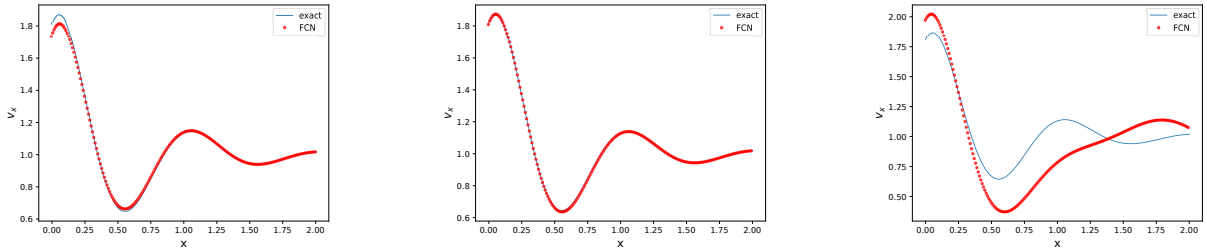
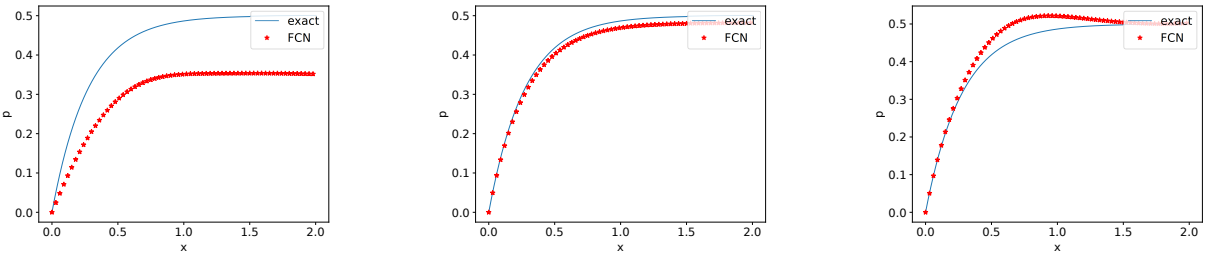



Figure 3.3: Losses (bottom 3 lines) of three linearized learning schemes (3.15) (3.16) or (3.18) and loss (top line) based on nonlinear Navier-Stokes equation (3.4). The results show that the neural networks with 3 linearized learning schemes (GradFixed(3.15), VFixed(3.16) and Hybrid(3.18)) converge fast compared with the neural networks using the VgVp loss function (3.4).




a) VFixed scheme b) Hybrid scheme c) GradFixed scheme

Figure 3.4: Linearized learning of fully connected network (FCN): The x components of velocity after 300 epoch training for benchmark problem along the line $y = 0.7$




a) VFixed scheme b) Hybrid scheme c) GradFixed scheme

Figure 3.5: Linearized learning of fully connected network (FCN):The pressures after 300 epoch training for benchmark problem along the line $y = 0.7$

higher than the benchmark problem. We also adjusted the learning rate during training by a decrease of 5% every 50 epochs for the oscillating flow case to accelerate the convergence.

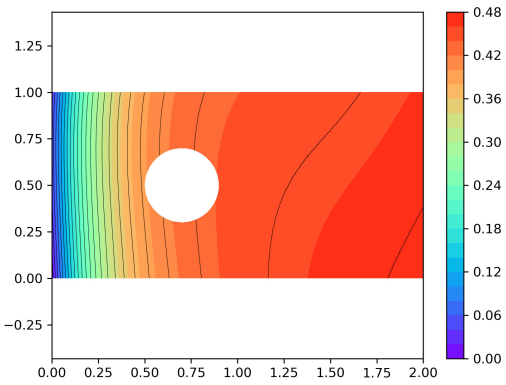
#### *3.4.2.1. A simple domain - effect of MscaleDNN*

In this section, we consider the same simple domain as in Figure 3.2 and utilize Scheme 1(GradFixed) (3.11) of the linearized learning algorithms where the previous velocity is used to linearize the convection term. The purpose of this section is to show that MscaleDNN combining with linearized learning scheme could offer extraordinary performance improvement for stationary Navier-Stokes equations with oscillating solutions.

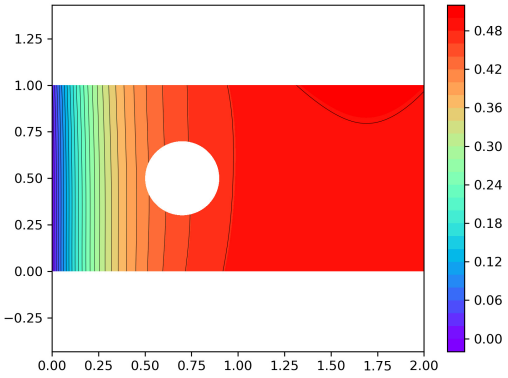
The multiscale deep neural networks are given 8 scales: $\{x, 2x, 4x, 8x, 32x, 64x, 128x\}$, whose subnetworks contain 4 hidden layers and 128 hidden neurons in each layer. As a comparison, we also trained a 4-layer fully connected neural network with 1024 hidden neurons combining GradFixed scheme in 1000 epochs. Figures 3.6 and 3.7 show the predictions of networks after 1000 epoch training. Figures 3.8 and 3.9 give more details along the line $y = 0.7$. The penalty terms $\alpha, \beta, \gamma$ are set to be $10^4, 1, 1$, respectively, as the same as in Section 3.4.1. Figure 3.10 shows the relative errors of these 2 different neural network structures along the line $y = 0.7$. We could conclude that the MscaleDNN improves the accuracy of both the pressure field and the velocity field compared with the fully connected neural network.

#### *3.4.2.2. A complex domain*

In this section, we consider an oscillating case in complex domain with more than one hole. The domain is shown in Figure 3.11. In this case, we use the similar settings for the multiscale deep neural networks, adjustments of learning rates, and sampling strategies like what we choose in the oscillatory case with the scale of the simple domain with one hole in Section

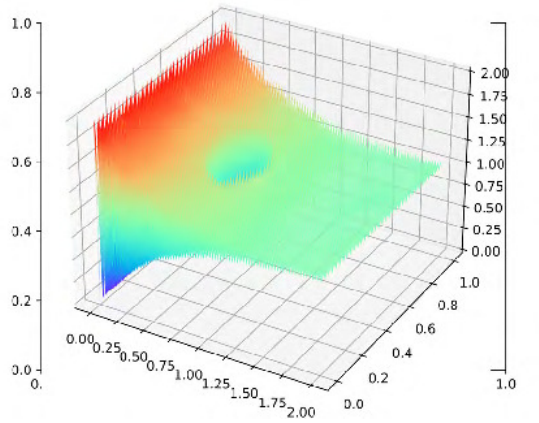


a) Contour of pressure of the oscillatory case after 1000 epoch training for linearized learning with fully-connected network (FCN)

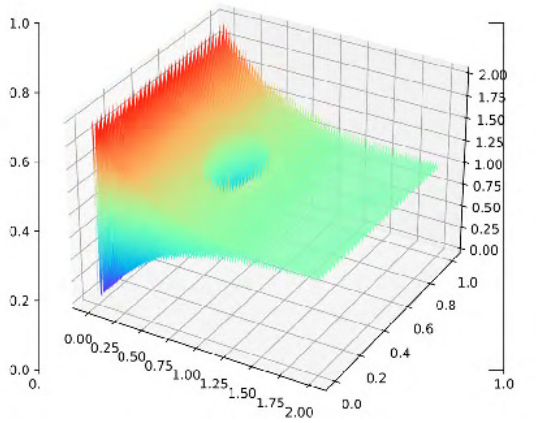


b) Contour of velocity of the first component after 1000 epoch training for linearized learning with MscaleDNN

Figure 3.6: Pressure of the oscillatory case for linearized learning with MscaleDNN and fully connected networks (FCN)



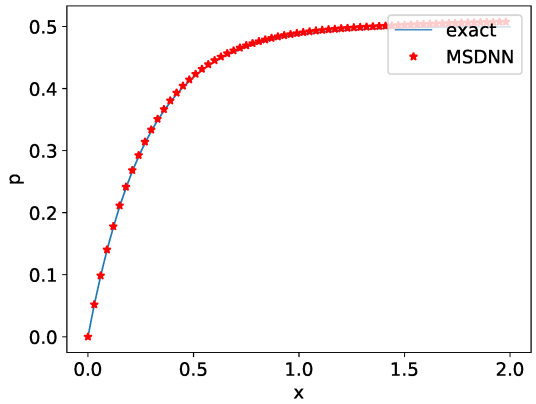
a) Contour of the first component of velocity of the oscillatory case after 1000 epoch training for linearized learning with fully-connected network (FCN)



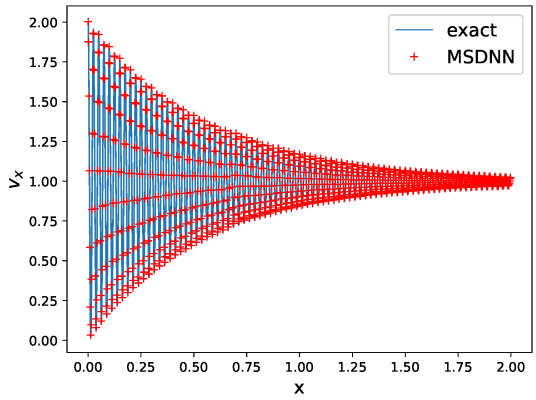
b) Contour of the first component of velocity of the oscillatory case after 1000 epoch training for linearized learning with MscaleDNN

Figure 3.7: The first component of velocity of the oscillatory case for linearized learning with MscaleDNN and fully connected networks (FCN)

3.4.2. The multiscale deep neural network has 8 scales: $\{x, 2x, 4x, 8x, 32x, 64x, 128x\}$, whose subnetworks contain 4 hidden layers and 128 hidden neurons in each layer. The frequencies selected for this case are the same as what in Section 3.4.2.1. The initial learning rate for this case is $4e-3$. The penalty terms $\alpha, \beta, \gamma$ are set to be $10^4, 1, 1$, respectively, as the same as in Section 3.4.1. Figure 3.12 shows contours of the first component of velocity for different schemes. Figure 3.14 and Figure 3.13 show the relative errors of these 4 different linearization

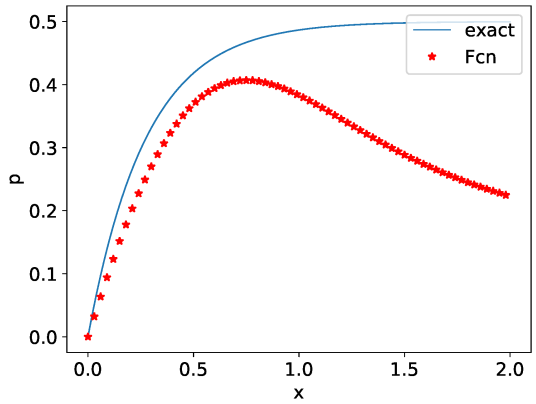



a) Pressure of the oscillatory case after 1000 epoch training alone line $y = 0.7$

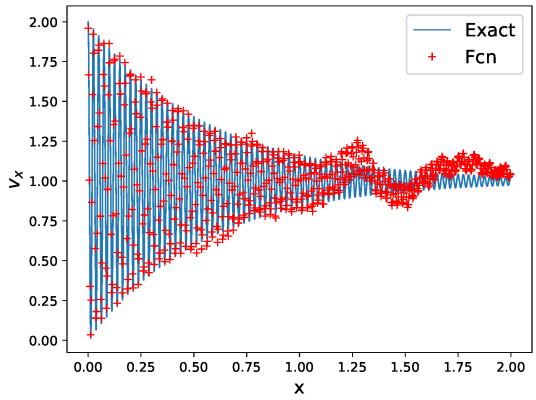



b) Velocity of the first component after 1000 epoch training along the line $y = 0.7$

Figure 3.8: The results of the oscillatory case using linearized learning with multi-scale neural networks




a) Pressure of the oscillatory case after 1000 epoch training alone line $y = 0.7$

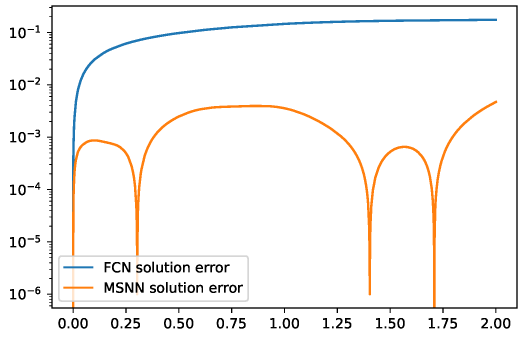



b) Velocity of the first component after 1000 epoch training along the line $y = 0.7$

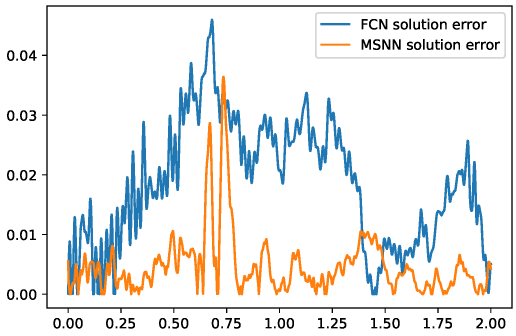
Figure 3.9: The results of the oscillatory case using linearized learning with fully connected networks (FCN)

schemes along the line $y = 0.7$. All schemes we propose converge more accurately to the exact solutions.

Figure 3.15 displays the behavior of the velocity field's divergence under varying penalty values $\gamma$ while the neural networks are trained by scheme (3.13). The results indicate that as $\gamma$ increases, the divergence of velocity decreases. The divergence-free property thus is enforced through the extra regularization with large penalty.




a) Error of two different models w.r.t. pressure of the oscillatory case after 1000 epoch training alone line $y = 0.7$




b) Error of two different models w.r.t. velocity of the first component after 1000 epoch training along the line $y = 0.7$

Figure 3.10: The errors of the oscillatory case for linearized learning with fully-connected network (FCN) and multiscale network (MSNN)

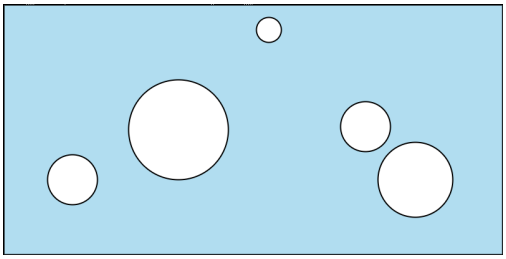
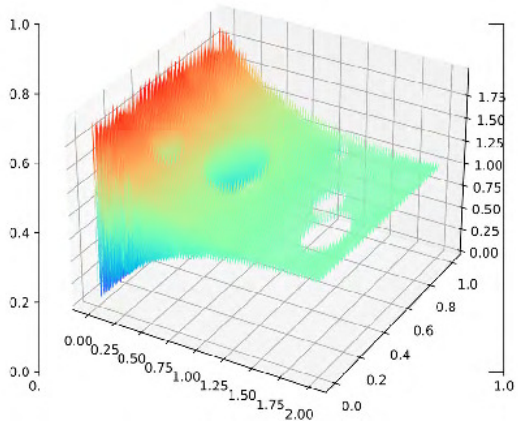


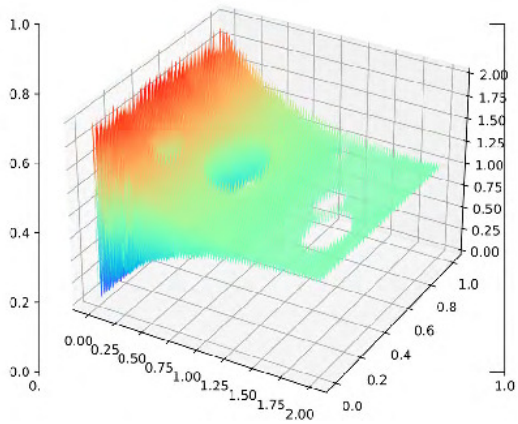
Figure 3.11: A more complex domain.

### *3.4.2.3. Small viscosity coefficient*

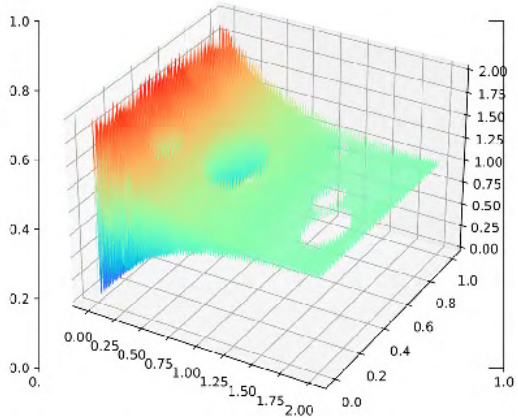
In this section, we consider the same oscillating case in the complex domain but with smaller viscosity coefficient, $\nu = 0.001$. The multiscale deep neural networks are the same as in the oscillatory case with 8 scales: $\{x, 2x, 4x, 8x, 32x, 64x, 128x\}$ and the corresponding subnetworks contain 4 hidden layers and 128 hidden neurons in each layer. The learning rates are multiplied by 0.1 at the $100\text{th}, 300\text{th}, 600\text{th}$ epoch. The batch size is 8092 in the domain and 512 on the boundary of domain, respectively. The penalties considered in the case are $10^4$ for $\alpha$, 1 for $\beta$ and $10^3$ for $\gamma$, respectively. The frequencies selected for this case are the same as what in Section 3.4.2.1. The initial learning rate is $4e-3$. The scheme is (3.13) as it gives the best results for the complex domain case in Section 3.4.2.2. Figure 3.16



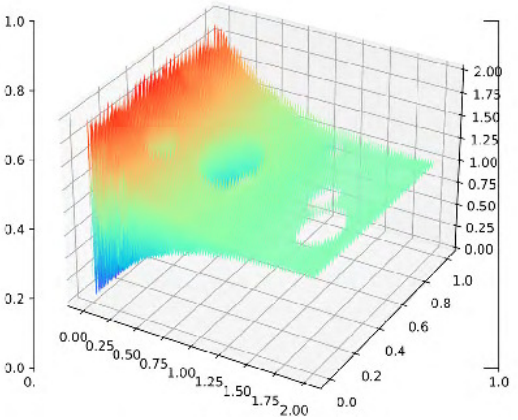
a) Contour of the first component of velocity of the oscillatory case after 1000 epoch training for scheme gradFixed (3.11)



b) Contour of the first component of velocity of the oscillatory case after 1000 epoch training for scheme vFixed (3.12)



c) Contour of the first component of velocity of the oscillatory case after 1000 epoch training for scheme vFixed1 (3.13)



d) Contour of the first component of velocity of the oscillatory case after 1000 epoch training for scheme Hybrid (3.14)

Figure 3.12: The results of the first component of velocity of the oscillatory case for four linearized learning schemes with MscaleDNN

and 3.17 show the results of linearized learning scheme (3.13). The divergence of velocity is even better comparing with the larger viscosity coefficient case.

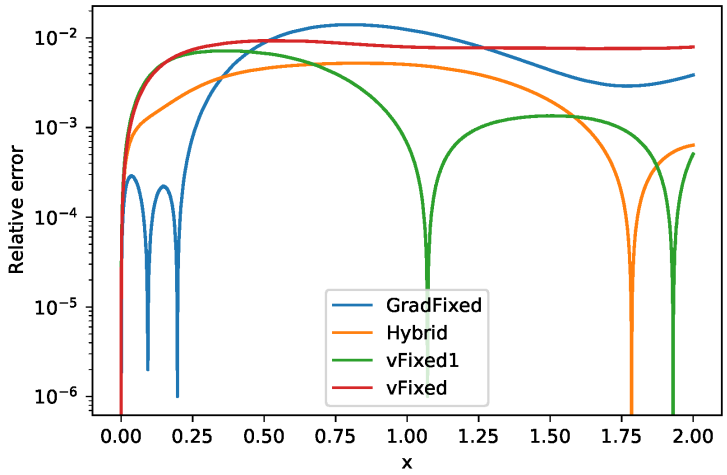
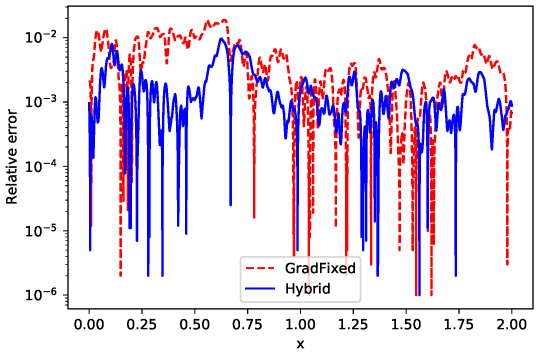
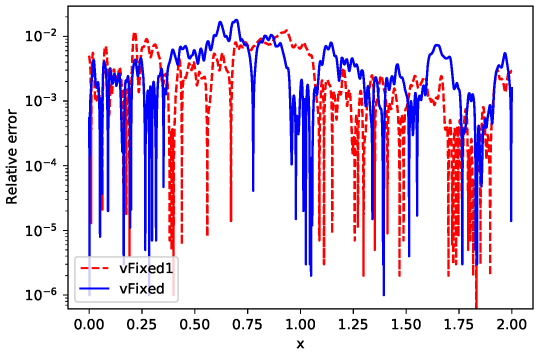



Figure 3.13: Relative errors at the line $y = 0.7$ of four linearized learning schemes with MscaleDNN for pressure of the complex domain case after 1000 epoch training

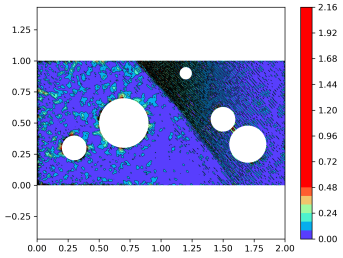



a) Relative errors alone line $y = 0.7$ of GradFixed linearized learning scheme (3.11) and Hybrid linearized learning scheme (3.14) for the first component of velocity of the complex domain case after 1000 epoch training

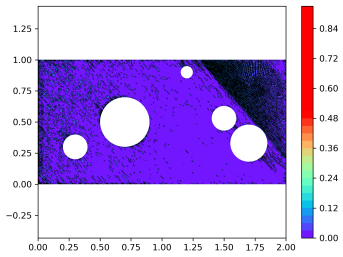



b) Relative errors along the line $y = 0.7$ of vFixed linearized learning scheme (3.12) and vFixed1 linearized learning scheme (3.13) for the first component of velocity of the complex domain case after 1000 epoch training

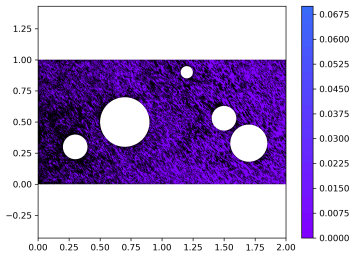
Figure 3.14: The relative errors of the complex domain case for four linearized learning schemes with MscaleDNN



a) $\nabla \cdot \mathbf{u}$ given $\gamma = 10^0$. The largest value is 2.16

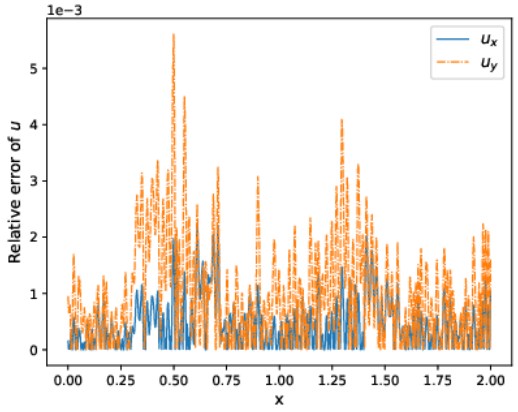


b) $\nabla \cdot \mathbf{u}$ given $\gamma = 10^3$. The largest value is 0.9

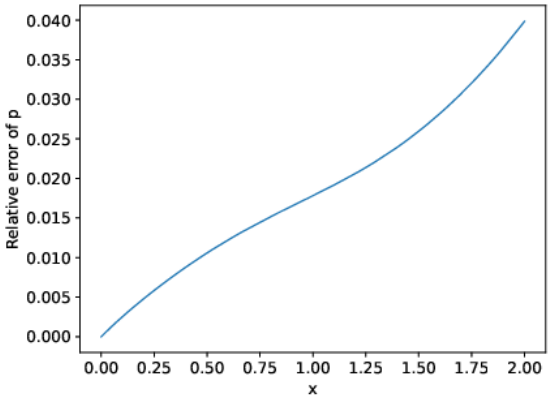


c) $\nabla \cdot \mathbf{u}$ given $\gamma = 10^6$. The largest value is 0.07

Figure 3.15: The divergence of velocity $\nabla \cdot \mathbf{u}$ where $\mathbf{u}$ is trained by linearized learning scheme (3.13) with 1000 epochs




a) Relative errors for velocities in both x direction and y direction alone line $y = 0.7$




b) Relative errors for pressure $p$ alone line $y = 0.7$

Figure 3.16: The relative errors of velocity $\mathbf{u}$ and pressure $p$ along the line $y = 0.7$ where neural networks are trained by linearized learning scheme (3.13) with 1000 epochs as $\nu = 0.001$

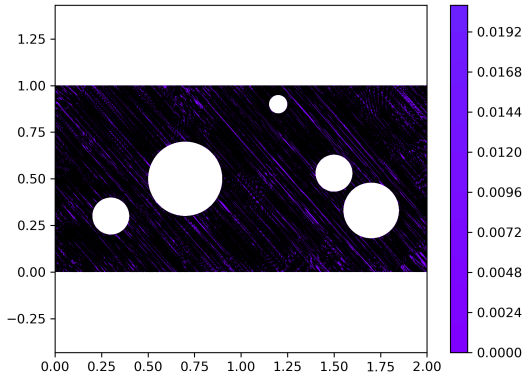


Figure 3.17: The divergence of velocity $\nabla \cdot \mathbf{u}$ where neural networks are trained by linearized learning scheme (3.13) with 1000 epochs as $\nu = 0.001$. The largest value is 0.023

# Chapter 4

## A Causality-DeepONet for Causal Responses of Linear Dynamical Systems

*The content in this chapter has been submitted as a journal paper in collaboration with Kamaljyoti Nath and Wei Cai:*

Lizuo Liu, Kamaljyoti Nath, and Wei Cai, *A Causality-DeepONet for Causal Responses of Linear Dynamical Systems*, arXiv preprint arXiv:2209.08397 (2022), under review in Commun. Comput. Phys. [41].

### 4.1. Introduction

Computing operators between physical quantities defined in function spaces have many applications in forward and inverse problems in scientific and engineering computations. For example, in wave scattering in inhomogeneous or random media, the mapping between the media physical properties, which can be modelled as a random field, and the wave field is a nonlinear operator, which represents some of the most challenging computational tasks in medical imaging, geophysical and seismic problems. A specific example comes from earthquake safety studies of buildings and structures, the response of structures to seismic ground accelerations gives rise to a causal operator between spaces of highly oscillatory temporal signals.

Structural dynamic analysis has always been one of the crucial problems in the civil engineering field. Traditionally, researchers in this field analyzing structural dynamic response focus on constructing proper mathematical models like ordinary or partial differential equations and utilizing grid-based numerical methods to solve them. The finite element method [76] is one of the popular methods considered for the solutions along with an appro-

priate time integration scheme like Newmark's-Beta method [12, 53]. Alternatively, system identification-based methods, as an attempt to construct a surrogate model by mapping the input signals to the output responses directly, have shown their superior capability in accelerating the computations. A comprehensive review of this approach was provided in [32, 66]. Meanwhile, recently learning time sequential response operator between input and output signals [35] has been studied using recurrent neural network (RNN) [20, 36], long short-term memory neural network (LSTM) [29], WaveNet [55], the one-step ResNet approximation [58] and the multi-step recurrent ResNet approximation [58]. The RNN and its variant LSTM are ubiquitous network structures for predicting time series in financial engineering, machine translation, and sentiment analysis and so on in the natural language processing field. In particular, LSTM has been shown to have the potential to predict building responses excited by seismic ground accelerations [33, 72]. The one-step ResNet approximation and the multi-step recurrent ResNet approximation, provide the approximation to the integral form of the dynamical system and have demonstrated effective equation recovery for linear and nonlinear dynamical systems [22, 58].

Deep neural networks (DNNs), as one of the most intuitive frameworks for model reductions with its superior ability to approximate general high dimensional functions [13], have been considered recently in learning mappings whose closed forms are not known. So far, DNNs have shown much promise in solving problems from scientific and engineering computing, including initial and boundary value problems of ODEs and PDEs [10, 18, 26, 31, 43, 61, 73]. Soon after universal approximation theorems to functions by neural networks was proposed [13], Chen & Chen [11] proved that there also exists a framework that could give universal approximations to nonlinear operators between Banach spaces. Based on this theory, the DeepONet [47] was constructed for learning operators where trunk net functions are used as basis and the branch net functions as mappings from the input functions to some hidden manifolds. The DeepONet replaced the one-hidden layer networks in the original proposal in Chen & Chen's paper [11] by two deep neural networks, which has been shown

to have the potential to break the curse of dimensionality from the input space. In the meantime, another approach for learning operators based on a graph kernel network [38] for PDEs has also been proposed. The nonlinear operator is decomposed by composing nonlinear activation functions with a class of integral operators with a trainable kernel. The Fourier neural operator has been proposed [37] by replacing the integral operator with the Fourier transform and a trainable mask in frequency domain. Both the graph kernel neural network and the Fourier neural operator show the capability to approximate specific operators with very good accuracy and efficiency.

In this chapter, we will study the DeepONet specifically for time-dependent operators from a physical system such as those encountered in building seismic wave response problems. The Causality-DeepONet will be proposed to ensure the causality of the retarded Green's function of the underlying differential equation between the input seismic ground accelerations and the output building responses. In addition to the causality consideration, the time homogeneity of a dynamic system will also be used in the design of the neural network by encoding the convolutional nature of the retarded Green's function in the choice of the network weights. The proposed DeepONet with built-in causality allows us to learn, accurately and with minimum requirement of training data, the mapping between the ground accelerations and the corresponding displacements of the building at the roof level excited by the seismic ground accelerations.

It could be noted that the term causality is also used in fields such as causal inference [45, 51] and causal interpretability of neural network [52], or applying the neural network to solve problem like causal reasoning [25]. Those works are referring to the logical cause-effect relationships between data. However, in our setting we focus on the physical concept of temporal causality, i.e., the state of the system at the current time is not affected by that of the future, but only by its current state and past history. For this reason, we name our framework Causality-DeepONet. In a study of crack propagation [24], past histories of tensile

energies were provided as input to the branch net of a variational energy-based DeepONet with convolution to predict the growth of fracture under quasi-static loading conditions. This approach is similar to implicit time discretization of nonlinear time evolution equation commonly used for time dependent Navier-Stokes equations where the state(s) of the system at previous time step(s) act as a forcing term for the linear system to be solved for the state of the current time.

The rest of the chapter is organized as follows. In Section 4.2, we state the problem considered in the present study. In Section 4.3, we give a short review of the universal approximation theory of nonlinear operators by neural networks, and its recent development DeepONet. Further, we provide a review of a multi-scale neural network introduced to handle high frequency functions and the POD-DeepONet for efficient basis functions in the trunk net of DeepONet. In Section 4.4, we propose two extensions of the DeepONet, one is the multi-scale DeepONet, the other is the Causality-DeepONet. In Section 4.5, a comparison of the results with all the mentioned frameworks will be carried out.

## 4.2. Problem statement: Calculation of building response due to seismic load

The problem under study is the prediction of the dynamic response of a multi-story building due to seismic loading. The equation of motion, after a finite element type discretization, for the building due to ground motions during an earthquake could be written as a dynamic system of differential equations [12],

$$\boldsymbol{M}\ddot{\boldsymbol{x}} + \boldsymbol{C}\dot{\boldsymbol{x}} + \boldsymbol{K}\boldsymbol{x} = \boldsymbol{f}(t), \tag{4.1}$$

where $\boldsymbol{M}$, $\boldsymbol{C}$ and $\boldsymbol{K}$ are the mass, damping and stiffness matrices of the system from the finite element discretization. $\boldsymbol{f}(t)$ the applied force, for our case, is due to ground motions

during an earthquake and could be written as

$$\boldsymbol{f}(t) = \boldsymbol{M}\boldsymbol{\iota}\ddot{u}_g, \tag{4.2}$$

where $\ddot{u}_g$ is the ground acceleration due to the earthquake and $\boldsymbol{\iota}$ is the influence vector. The interested reader may refer [12] for more details on formulation and solution methods.

Ground accelerations due to earthquakes are recorded at different recording stations. In the present study, we consider ground accelerations due to earthquakes for different earthquakes recorded at different stations and taken from the database of the Pacific Earthquake Engineering Research Center (https://peer.berkeley.edu/) [1]. One of the typical records of ground accelerations is shown in Figure 4.1. The earthquake record at different stations may be recorded at different sampling rates (different $\delta t$). Earthquake records recorded at finer $\delta t < 0.02$ sec are filtered using a Butterworth filter with frequency (0.1-24.9) Hz then re-sampled to $\delta t = 0.02$ sec and after that amplified to match with original PGA level. Figures before and after processing for the mentioned earthquake record in Figure 4.1 are shown in Appendix A.2.1. The building is considered at rest initially with the initial condition

$$\begin{cases} \boldsymbol{x}(0) = \boldsymbol{0}, \\ \dot{\boldsymbol{x}}(0) = \boldsymbol{0}. \end{cases} \tag{4.3}$$

[1] The earthquake ground acceleration considered are taken from the Pacific Earthquake Engineering Research Center (PEER: https://peer.berkeley.edu/)

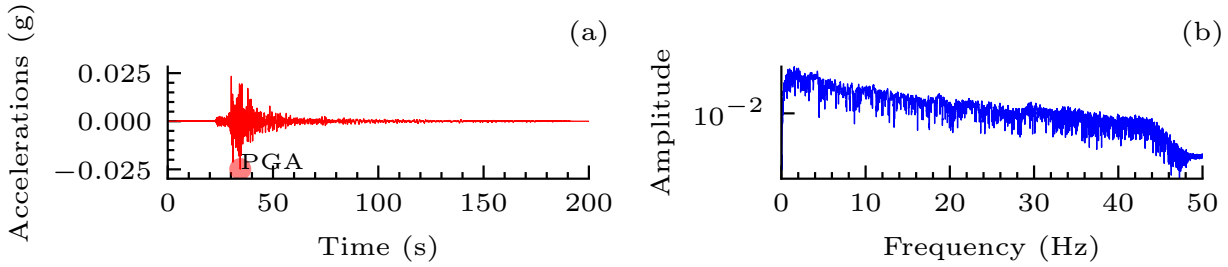



Figure 4.1: **A typical ground acceleration due to earthquake:** The ground acceleration is due to 14383980 earthquake recorded at station North Hollywood, 2008 (a) time history of the acceleration (b) frequency spectrum. The absolute maximum acceleration is indicated which is also known as Peak Ground Acceleration (PGA) of the earthquake.

Our objective is to evaluate an operator operating on the ground acceleration and predict the response of the building. Thus, it is a mapping from ground acceleration to the response of the top floor of the building.

$$\mathcal{R} : \ddot{u}_g(t) \longrightarrow x_1(t). \tag{4.4}$$

Detailed numerical study is carried out for a six-storied reinforced cement concrete (RCC) building. A 3D model of the building is generated in openseespy [74]. Apart from the dead load (beam, column, slab, wall etc.), the live load is also considered on each floor and roof. The lumped mass of the structure is calculated for a full dead load and 50% of the live load at floor level and 25% at roof level. The damping matrix of the building is calculated using model damping for 5% model damping for all the modes. Ground acceleration is applied in the major direction. The records obtained from PEER contain 3 different directions. The vertical ground accelerations are not considered. The other two horizontal ground accelerations are considered one at a time and applied only in the major direction.

In the case of a classical damped system [12], the displacement $\boldsymbol{x}(t)$ may be decomposed as the superposition of the modal contributions of undamped system:

$$\boldsymbol{x}(t) = \sum_{l=1}^{n} \phi_l q_l\,(t) \coloneqq \boldsymbol{\Phi q}, \tag{4.5}$$

where $q_l$ and $\phi_l$ are the modal coordinates and the corresponding modes, respectively, for natural frequency $\omega_l$. $\boldsymbol{\Phi} = [\phi_1, \phi_2, \cdots, \phi_n]$, $\boldsymbol{q} = [q_1, q_2, \cdots, q_n]^T$. The responses (displacement) $\boldsymbol{x}(t)$ of the system for ground accelerations $\ddot{u}_g(t)$ may be represented as

$$\boldsymbol{x}(t) = \int_0^t \ddot{u}_g(\tau)\boldsymbol{h}(t-\tau)d\tau, \tag{4.6}$$

where

$$\boldsymbol{h}(t) = -\sum_{\ell=1}^{n} \phi_\ell \frac{\Gamma_\ell}{\omega_{\ell D}} e^{-\xi_\ell \omega_\ell t} \sin \omega_{\ell D} t, \tag{4.7}$$

$\boldsymbol{h}(t)$ is the unit impulse response, also known as, the Green's function and fundamental solutions, $\Gamma_\ell = \dfrac{\phi_\ell^T \boldsymbol{M}\boldsymbol{\iota}}{\phi_\ell^T \boldsymbol{M}\phi_\ell}$ and $\boldsymbol{\iota}$ is the influence vector, $\omega_{\ell D} = \omega_\ell\sqrt{1-\xi_\ell^2}$ with $\xi_\ell$ as the damping ratio.

In the case of a non-classical damped system [12], the responses (displacement) $\boldsymbol{x}(t)$ of the system due to ground accelerations $\ddot{u}_g(t)$ may be represented as

$$\boldsymbol{x}(t) = -\sum_{\ell=1}^{n} \left[\tilde{\gamma}_\ell^\delta \omega_\ell^{\mathcal{N}} D_\ell(t) + \alpha_\ell^\delta \dot{D}_\ell(t)\right], \tag{4.8}$$

where

$$D_\ell(t) = \int_0^t \ddot{u}_g(\tau) H_\ell(t-\tau)d\tau, \tag{4.9}$$

and

$$\dot{D}_\ell(t) = \int_0^t \ddot{u}_g(\tau) \dot{H}_\ell(t-\tau)d\tau, \tag{4.10}$$

$H_\ell(t) = -\dfrac{1}{\omega_{\ell D}^{\mathcal{N}}} e^{-\xi_\ell^{\mathcal{N}}\omega_\ell^{\mathcal{N}} t} \sin \omega_{\ell D}^{\mathcal{N}} t$ is the the Green's function of the non-classical damped system, with

$$\omega_{\ell D}^{\mathcal{N}} = \omega_\ell^{\mathcal{N}}\sqrt{1-(\xi_\ell^{\mathcal{N}})^2}, \tag{4.11}$$

and

$$\omega_\ell^{\mathcal{N}} = |\lambda_\ell^{\mathcal{N}}|, \quad \xi_\ell^{\mathcal{N}} = -\frac{\mathrm{Re}\left(\lambda_\ell^{\mathcal{N}}\right)}{|\lambda_\ell^{\mathcal{N}}|}, \tag{4.12}$$

where $\lambda_\ell^{\mathcal{N}}$ is the eigenvalues of the system of first-order differential equations reduced from equation (4.1). $\omega_\ell^{\mathcal{N}}$ is a function of the amount of system damping.

Further, $\tilde{\gamma}_\ell^\delta = \left(\xi_\ell^{\mathcal{N}}\alpha_\ell^\delta - \sqrt{1-(\xi_\ell^{\mathcal{N}})^2}\gamma_\ell^\delta\right)$ with $\alpha_\ell^\delta = \mathrm{Re}(2\beta_\ell^\delta\psi_\ell), \gamma_\ell^\delta = \mathrm{Im}(2\beta_\ell^\delta\psi_\ell)$ and

$$\beta_\ell^\delta = \frac{-\psi_\ell^T \boldsymbol{M}\boldsymbol{\iota}}{2\lambda_\ell^{\mathcal{N}}\psi_\ell^T\boldsymbol{M}\psi_\ell + \psi_\ell^T\boldsymbol{C}\psi_\ell}, \tag{4.13}$$

where $\psi_\ell$ is the corresponding eigenvector of $\lambda_\ell$ .

The discussion above of equations (4.6) and (4.8) infers and inspires us to consider two phenomena in formulating an operator, the first one is that the responses at the present state is not influenced by the future ground acceleration, we understand it as causality of the system, meaning the state of the system at the current time should not be affected by the future, but only by its past history of the ground acceleration. The second one is the convolution nature of the Green's function kernel. As shown in many works with convolutional neural networks [1, 56, 68, 75], the convolution function as a specific domain knowledge for neural network to learn about the target operator. With these two insights we will construct an operator in the DeepONet framework to address both causality and convolution kernel in Section 4.4.2, and name it as Causality-DeepONet.

## 4.3. Background / Preliminary

In this section, first we will review the DeepONet in Section 4.3.1. Multi-scale deep neural networks and POD-DeepONet will be described in Sections 4.3.2 and 4.3.3, respectively.

### 4.3.1. DeepONet

Based on the universal approximation of nonlinear operators, Lu et al. [47] proposed the DeepONet by replacing the two one-hidden-layer neural networks in equation (1.14) with

two deep neural networks. For a general operator $\mathcal{G}(f)(x)$, DeepONet has form

$$\mathcal{G}(f)(x) \sim \sum_{k=1}^{N} c_k \underbrace{\sigma_{B,k}\left(\{f\left(x_j\right)\}_{j=1}^{m}\right)}_{B_k} \underbrace{\sigma_{T,k}\left(x\right)}_{T_k}, \tag{4.14}$$

where $\sigma_B\left(\cdot\right)$ with a signal $\{f(x_j)\}_{j=1}^{m}$ as input is a deep neural network with $N$ outputs, named as the branch net, $\sigma_T\left(\cdot\right)$ with input $x$ is also a deep neural network with $N$ outputs which is called the trunk net. The schematics are shown in Figure 4.2. The DeepONet has already been shown it is able to learn not only explicit mathematical operators like integration and fractional derivatives, but also PDE operators [8, 15, 16, 39, 47].

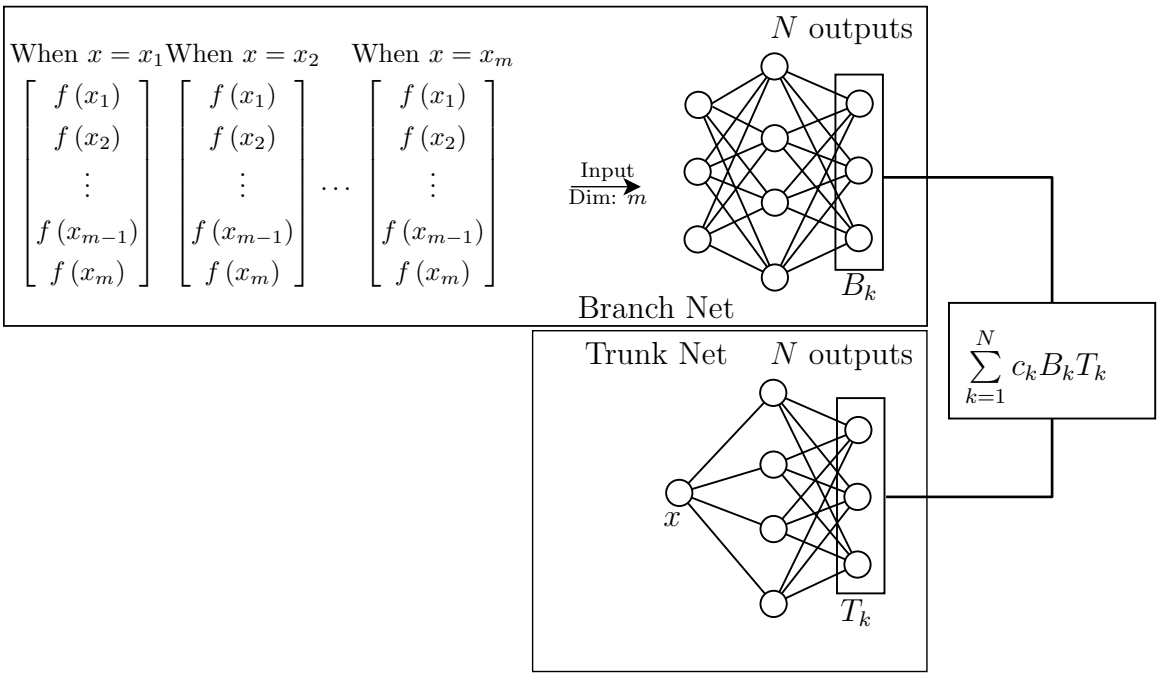



Figure 4.2: **Schematic Diagram of the DeepONet:** A schematic diagram of DeepONet showing branch and the trunk net along with the input data and output. The number of the input neurons of the branch net is equal to the number of sensor points in the input signals. The trunk net takes the input point $x$ at where the output function need to be evaluated. Thus, the first layer of trunk equal to the dimension of the problem. In the present study, it is the time point $x = t$ at which the output needs to be evaluated. Note for different time point $t$, the corresponding input of branch net is the same.

#### 4.3.2. Multi-scale deep neural network (MscaleDNN)

Multi-scale Deep Neural Network [43] is a specific framework for problem whose output function is highly oscillatory. Since the highly oscillating feature of the responses of buildings excited by ground accelerations due to earthquakes, we also propose the multi-scale DeepONet that incorporates the multi-scale neural network into the DeepONet. The general fully connected neural network could learn the low frequency content of the data quickly, but the learning process will be stalled when higher frequency components are involved in the data. This frequency bias phenomenon is considered as the Frequency Principle studied in [70]. To remedy the frequency bias in terms of learning convergence, Liu et al. [43] introduced a MscaleDNN to accelerate the convergence of neural network for fitting problems of highly oscillating data. The MscaleDNN contains several sub-neural networks, for each of which a different frequency scaling is introduced by scaling the inputs accordingly, to arrive at the following form for the MscaleDNN,

$$f_{\boldsymbol{\theta}}(x) \sim \sum_{i=1}^{S} w_i f_{\boldsymbol{\theta}_i}(\mathcal{S}_i x), \tag{4.15}$$

where $\mathcal{S}_i$ are the custom scales and $\{f_{\boldsymbol{\theta}_i}(\cdot)\}_{i=1}^{S}$ are the distinct sub-fully connected neural networks. Equation (4.15) shows a multi-scale neural network with $S$ scales. The final output of multi-scale deep neural network is the weighted sum of the outputs of the sub-neural networks with trainable weights $w_i$. The MscaleDNN has already shown its power for solving fitting problems and PDEs with high frequencies in [42, 43, 67].

#### 4.3.3. POD-DeepONet

Instead of modeling the basis of output data by training the trunk net, Lu et al. [48] propose the POD-DeepONet based on the work of Bhattacharya et al. [3]. The trunk net in the vanilla DeepONet is replaced by the basis obtained from proper orthogonal decomposition

(POD) of the outputs of training data after the mean of which is removed. Thus, the outputs of branch net are the coefficients of the precomputed basis vectors

$$\mathcal{G}(f) \sim \sum_{k=1}^{p} \sigma_{B,k} \left( \{f(x_j)\}_{j=1}^{m} \right) \mathcal{B}_k + \mathcal{B}_0, \tag{4.16}$$

where $\mathcal{B}_0$ is the mean of the output of training data, $\{\mathcal{B}_k\}_{k=1}^{p}$ are the selected $p$ basis vectors obtained by SVD or POD of the zero-mean outputs of training data, and $\sigma_B(\{f(x_j)\}_{j=1}^{m})$ is the deep neural network with $p$ outputs whose $k^{th}$ output corresponds to the $k^{th}$ singular value. In [48], it was shown that POD-DeepONet is more effective than the vanilla DeepONet and the vanilla Fourier Neural Operator [37].

## 4.4. Methodologies

### 4.4.1. Multi-scale DeepONet

As shown in Figure 4.1, the spectrum of a typical earthquake signals contains not only low frequency components, but also high frequency components, therefore, we introduce the multi-scale DeepONet to handle the oscillatory information. Since the oscillatory features are function of time $t$, it is natural that we replace the fully connected trunk net by the MscaleDNN with $S$ scales.

$$\begin{aligned} \mathcal{G}(f)(t) &\sim \sum_{k=1}^{N} c_k \sigma_{B,k} \left( \{f(t_j)\}_{j=1}^{m} \right) \sigma_{T,k}(t), \\ \sigma_T(t) &= \sum_{i=1}^{S} w_i \sigma_{\boldsymbol{\theta}_i}(\mathcal{S}_i t), \end{aligned} \tag{4.17}$$

where $\{\sigma_{\boldsymbol{\theta}_i}(\cdot)\}_{i=1}^{S}$ are $S$ distinct fully connected neural networks with $N$ outputs.

### 4.4.2. Causality-DeepONet

The DeepONet proposed in [47] is based on a proven theorem of universal approximation for nonlinear operators, as introduced in the Section 1.4.1. We will apply the universal approximation theory to nested subspaces of continuous functions indexed by the output time, which will provide a heuristic argument for the form of the Causality-DeepONet to be proposed. Rigorous mathematical justification though is still to be derived.

For a ground acceleration dynamic excitation $\ddot{u}_g(s)$, the response function $\mathcal{R}(\ddot{u}_g)(t)$ experiences a retardation effect due to the causality of the physical process. Therefore, applying the universal approximation of function (1.12) to input function space

$$C[0,t] \subseteq C[0,T], \forall t \in [0,T], \tag{4.18}$$

we have,

$$\left| \mathcal{R}(\ddot{u}_g)(t') - \sum_{k=1}^{N} c_k \left( \mathcal{R}(\ddot{u}_g), t \right) \sigma_t \left( \boldsymbol{w}_k t' + b_k \right) \right| \leq \varepsilon_1, \quad \forall\ t' \in [0,t] \subseteq [0,T]. \tag{4.19}$$

Comparing with the original universal approximation theorem of functions equation (1.12), we will require the dependence of $t$ for the parameters $c_k \left( \mathcal{R}(\ddot{u}_g), t \right)$, in which we may introduce the **causality** and **convolution**. It can be assumed that for every given interval $[0,t]$, the approximation equation (4.19) is valid within the interval based on the proof in [11]. In principle, the integer $N$, the parameters $\{\boldsymbol{w}_k\}_{k=1}^{N}, \{b_k\}_{k=1}^{N} \in \mathbb{R}$, should also have a $t$-dependence, however, due to the nested property in (4.18), for practical implementations they will be taken as global parameters to be trained for all $t \in [0,T]$.

Following the discussion in Section 4.2, we extend the universal approximation of functionals (1.13) to approximate the functionals with causality. The functional $c_k \left( \mathcal{R}(\ddot{u}_g), t \right)$

(from a compact subset of $C[0,t]$) could be rewritten as

$$c_k\left(\mathcal{R}(\ddot{u}_g),t\right)=c_k\left(\mathcal{R}(\ddot{u}_g\chi_{[0,t]})\right)\in\mathbb{R},\quad \ddot{u}_g\in C[0,t],\quad t\in[0,T]\,, \tag{4.20}$$

where $\chi_{[0,t]}(s)$ is the characteristic function, such that

$$\chi_{[0,t]}(s)=\begin{cases}1 & s\in[0,t],\\ 0 & \text{otherwise.}\end{cases} \tag{4.21}$$

Then, following the similar approach as universal approximation of functionals (1.13), we may state that given any $\varepsilon_2>0$, there exists a positive integer $M$, $m$ equalspaced points $\{s_j\}_{j=1}^m\in[0,t]$ with real constants $c_i^k,W_{ij}^k,B_i^k\in\mathbb{R},i=1,\ldots,M,j=1,\ldots,m$, such that $c_k\left(\mathcal{R}(\ddot{u}_g),t\right)$ could be approximated by a one-hidden-layer neural network with any TW activation function $\sigma_b$

$$\left|c_k(\mathcal{R}(\ddot{u}_g),t)-\sum_{i=1}^{M}c_i^k\sigma_b\left(\sum_{j=1}^{\lfloor\frac{t}{h}\rfloor}W^k_{i,m-\lfloor\frac{t}{h}\rfloor+j}\ddot{u}_g\left(s_j\right)+B_i^k\right)\right|\leq\varepsilon_2,\forall\ddot{u}_g\in C[0,t], \tag{4.22}$$

where the $h$ is the step size and $m=\left\lfloor\dfrac{t}{h}\right\rfloor$. For any given interval $[0,t]$, based on the results of [11], there exists $m\in\mathbb{N}$ such that there are $m$ points $\{s_j\}_{j=1}^m$ that could be applied to construct the functional approximation (1.13). We further assume those points are equalspaced. The equalspaced sampling could be obtained by applying some appropriate smoothing kernel to input and output functions for the problems that are not equalspaced. And likewise, the coefficients $c_i^k,W_{ij}^k,B_i^k$ and nodes $\{s_j\}_{j=1}^m$ and $m,M$ are all independent of $\ddot{u}_g(s)$, however, $t$-dependent.

The indicator function $\chi_{[0,t]}(s)$ (4.21), implemented by the inner upper summation limit $\lfloor\frac{t}{h}\rfloor$ in equation (4.22), as a discontinuous function does not belong to the continuous function

space, which could be replaced by a smoothed version with a short transition at $s = t$ while still keeping the causality.

**Causality-DeepONet:** Combining these two desired universal approximations, we can heuristically consider the following DNN representation of an operator for retarded response for $t \in [0, T] \subset \mathbb{R}$. The basic idea is that we could find $m$ points $\{s_i\}_{i=1}^m \in [0, T]$ to approximate the functional $c_k(\mathcal{R}(\ddot{u}_g), T)$ based on the universal approximation of functionals with causality (4.22). The information to approximate the functional $c_k(\mathcal{R}(\ddot{u}_g), t)$ where $[0, t] \subseteq [0, T]$ is offered by the value of $\{\ddot{u}_g(s_i)\}_{i=1}^{\lfloor \frac{t}{h} \rfloor}$ only. To keep the input signals of the same length at different time point, we consider zero-padding $\{\ddot{u}_g(s_i)\}_{i=1}^{\lfloor \frac{t}{h} \rfloor}$ as shown in Figure 4.3. Thus the coefficients $c_i^k, W_{ij}^k, B_i^k$ and nodes $\{s_j\}_{j=1}^m$ are $t$-independent. In addition, the convolution with respect to the input signals could be implemented by shifting the signals, as shown in Figure 4.3.

Namely, we could find positive integers $M, N$, $m$ equalspaced points $\{t_j\}_{j=1}^m \in [0, T]$ with real constants $c_i^k, W_{ij}^k, B_i^k \in \mathbb{R}, i = 1, \ldots, M, j = 1, \ldots, m$, $\{\boldsymbol{w}_k\}_{k=1}^N \in \mathbb{R}, \{b_k\}_{k=1}^N \in \mathbb{R}$ that are all independent to continuous functions $\ddot{u}_g \in C[0, T]$ and $t$, such that

$$\mathcal{R}(\ddot{u}_g)(t) \sim \sum_{k=1}^{N} c_k \underbrace{\sum_{i=1}^{M} \sigma_b \left( \sum_{j=1}^{\lfloor \frac{t}{h} \rfloor} W_{i,\lfloor \frac{T}{h} \rfloor - \lfloor \frac{t}{h} \rfloor + j}^k \ddot{u}_g(s_j) + \sum_{j=\lfloor \frac{t}{h} \rfloor}^{\lfloor \frac{T}{h} \rfloor - 1} W_{i,\lfloor \frac{T}{h} \rfloor - j}^k 0 + B_i^k \right)}_{B_k} \underbrace{\sigma_t(\boldsymbol{w}_k t + b_k)}_{T_k}. \tag{4.23}$$

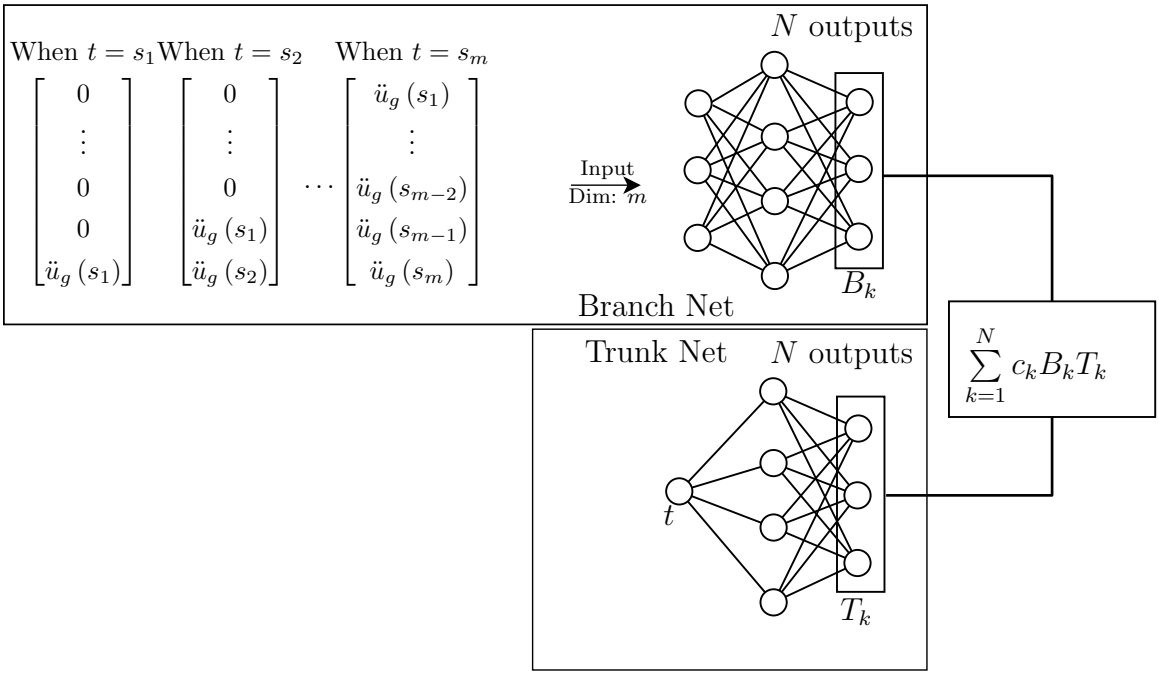



Figure 4.3: **Schematic Diagram of the Causality-DeepONet:** A schematic of the Causality-DeepONet showing branch and the trunk net along with the input data and output. Similar to DeepONet, the number of input neurons of branch of Causality-DeepONet is equal to the number of sensor points in the input signals. The input signals of branch, however, will be replaced by a zero-padding signals with a shifting window to express the causality and the convolution.

### 4.4.3. Loss function and error calculation

In the present study, we consider two loss functions depending on the method considered. Assuming $\hat{\boldsymbol{x}}_\ell(\boldsymbol{\theta})$ are the predicted response by neural network for the $\ell^{\text{th}}$ earthquake ground acceleration $\ddot{\boldsymbol{u}}_g^{(\ell)}$ with the corresponding true value $\boldsymbol{x}_\ell$, where $\boldsymbol{\theta}$ are trainable variables of the neural networks, including the weights and bias. The first loss function considered is the MSE loss function

$$\begin{aligned} \mathcal{L}(\boldsymbol{\theta}) &= \frac{1}{n}\sum_{\ell=1}^{n} \|\boldsymbol{x}_\ell - \hat{\boldsymbol{x}}_\ell(\boldsymbol{\theta})\|^2 \\ &= \frac{1}{n}\sum_{\ell=1}^{n}\left[\frac{1}{m}\sum_{i=1}^{m}(x_\ell^{(i)} - \hat{x}_\ell^{(i)}(\boldsymbol{\theta}))^2\right]. \end{aligned} \tag{4.24}$$

where $n$ is the number of samples (different earthquake accelerations) considered and $\|\cdot\|^2$ is the MSE error for one sample, $m$ is the number of points in each of the earthquake acceleration.

The second loss function considered is a weighted MSE loss function and defined as,

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}) =& \frac{1}{n}\sum_{\ell=1}^{n}\frac{1}{\max|\boldsymbol{x}_\ell|}\left\|\boldsymbol{x}_\ell-\hat{\boldsymbol{x}}_\ell(\boldsymbol{\theta})\right\|^2 \\ =& \frac{1}{n}\sum_{\ell=1}^{n}\frac{1}{\max|\boldsymbol{x}_\ell|}\left[\frac{1}{m}\sum_{i=1}^{m}(x_\ell^{(i)}-\hat{x}_\ell^{(i)}(\boldsymbol{\theta}))^2\right].\end{aligned} \tag{4.25}$$

The penalty is set to be the reciprocal of the maximum of the absolute value of the response. This act as a normalization factor when the responses have different magnitude for different earthquakes. The larger penalty is considered to the responses whose magnitude is smaller, thus it is expected that the neural network could predict the response whose magnitude is smaller accurately.

In order to check the accuracy of the predicted results we consider relative $L^2$ error,

$$\begin{aligned}\text{Relative } L^2 \text{ Error} &= \frac{1}{n}\sum_{\ell=1}^{n}\frac{\|\boldsymbol{x}_\ell-\hat{\boldsymbol{x}}_\ell(\boldsymbol{\theta})\|}{\|\boldsymbol{x}_\ell\|} \\ &= \frac{1}{n}\sum_{\ell=1}^{n}\sqrt{\frac{\sum_{i=1}^{m}(x_\ell^{(i)}-\hat{x}_\ell^{(i)}(\boldsymbol{\theta}))^2}{\sum_{i=1}^{m}(x_\ell^{(i)})^2}},\end{aligned} \tag{4.26}$$

and for the error with respect to the $\ell^{th}$ case, we consider the relative error,

$$\text{Err} = \frac{\max_i\left|x_\ell^{(i)}-\hat{x}_\ell^{(i)}(\boldsymbol{\theta})\right|}{\max_i\left|x_\ell^{(i)}\right|}. \tag{4.27}$$

The parameters $\boldsymbol{\theta}$ which include both weights and biases are optimized using the Adam optimizer [34] in a Pytorch [57] environment,

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}). \tag{4.28}$$

Once the optimized parameters of the networks (weights and biases) are obtained, these may be used for the prediction of the response of the system for an unknown input signal (earthquake ground acceleration).

### 4.5. Numerical results and discussion

In this section, we will present the numerical results of multi-scale DeepONet (MS-DeepONet) and Causality-DeepONet for the prediction of response of the multistoried building discussed in Section 4.2. We will also have a comparison study of the results with a few other DeepONet methods. First, we will study the prediction of the response with different DeepONet methods along with different sizes of networks and training samples. Then, we will present predicted responses with multi-scale DeepONet and Causality-DeepONet. We also study the different methods with different network sizes and training samples, which are discussed in subsequent sections. To avoid overfitting, dropout [28] is considered during training for a few of the cases, but is disabled during evaluation.

The testing dataset consists of different earthquakes which are not included in the training dataset. The test dataset is considered from 19 different earthquakes. One of them is recorded at three different stations. Two of them are recorded at the same station. Thus, the testing dataset consists of 44 ground accelerations (2 horizontal directions) from different earthquake recording stations. The training dataset consists of different earthquakes not considered in the testing dataset are may be from the same or different earthquakes and the same or different recording stations. Details about the training and testing dataset is shown in Table A.1 and A.2 in Appendix A.1.

### 4.5.1. DeepONet and POD-DeepONet

First, we will present the predicted results with the DeepONet method. For this purpose, we consider different trunk and branch sizes along with different training samples. As discussed in Section 4.4.3, we consider two different loss functions given by equations (4.24) and (4.25).

Different network sizes considered in the branch and trunk for DeepONet are shown in Table 4.1 along with the training samples considered. The training of DeepONet is considered with Adam optimizer for a total epoch of 5000 with ReLU($x$) activation function with a learning rate of $10^{-4}$ in the first 1000 epochs, then $10^{-5}$ in the 1000 to 3000 epochs, and $10^{-6}$ for the remaining epochs. In order to avoid overfitting, we consider using dropout with a rate of 0.01 for the branch net and $L^2$ weight regularization with $3 \times 10^{-5}$ coefficient for weights of the branch net as well. The relative $L^2$ errors for the training and testing samples after 5000 epochs are also shown in Table 4.1. The relative $L^2$ errors with epoch for training and testing are shown in Figure 4.4. A few more studies about the DeepONet are shown in the Appendix A.2.2. The relative $L^2$ errors with epoch when using DeepONet with different activation functions are shown in Figure A.3, with $\sin(x), \tanh(x), \text{Sigmoid}(x)$ considered. The relative $L^2$ error with epoch of case that $t$ is scaled to $[0, 1]$ is shown in Figure A.2(a). The relative $L^2$ error with epoch of case with fixed learning rate $10^{-4}$ is shown in Figure A.2(b). The relative $L^2$ errors with epoch of cases training up to 20000 epochs with fixed learning rate $10^{-4}$ is shown in Figure A.2(c)-(d). The predicted responses for few of testing dataset are shown in Appendix A.2.2. It could be observed that the error in predicted responses are high for both training and testing dataset in all the cases from Table 4.1 and Figure 4.4 and cases in Appendix A.2.2.

Table 4.1: Relative $L^2$ error for training and testing dataset when predicted using different sizes of DeepONet.

| Case | Branch[1] | Trunk[1] | Sample | Loss (4.24) | | Loss (4.25) | |
|---|---|---|---|---|---|---|---|
| | | | | Train | Test | Train | Test |
| 1 | [4000]-[50]×3-[50] | [1]-[50]×3-[50] | 50 | 1.0 | 0.999 | 1.001 | 1.004 |
| 2 | [4000]-[100]×3-[100] | [1]-[100]×3-[100] | 50 | 1.0 | 0.999 | 1.00 | 1.002 |
| 3 | [4000]-[200]×3-[200] | [1]-[200]×3-[200] | 50 | 1.0 | 0.999 | 1.001 | 1.003 |
| 4 | [4000]-[50]×3-[50] | [1]-[50]×3-[50] | 100 | 1.00 | 0.999 | 1.00 | 1.002 |
| 5 | [4000]-[100]×3-[100] | [1]-[100]×3-[100] | 100 | 1.0 | 0.999 | 1.0 | 1.002 |
| 6 | [4000]-[200]×3-[200] | [1]-[200]×3-[200] | 100 | 1.0 | 0.999 | 1.0 | 0.999 |

[1] The notation $[N_1]$-$[N_2]$ × 3-$[N_3]$ represents a neural network with the input size of $N_1$, 3 hidden layers with $N_2$ neurons in each layer, and the output dimension of $N_3$ neurons.

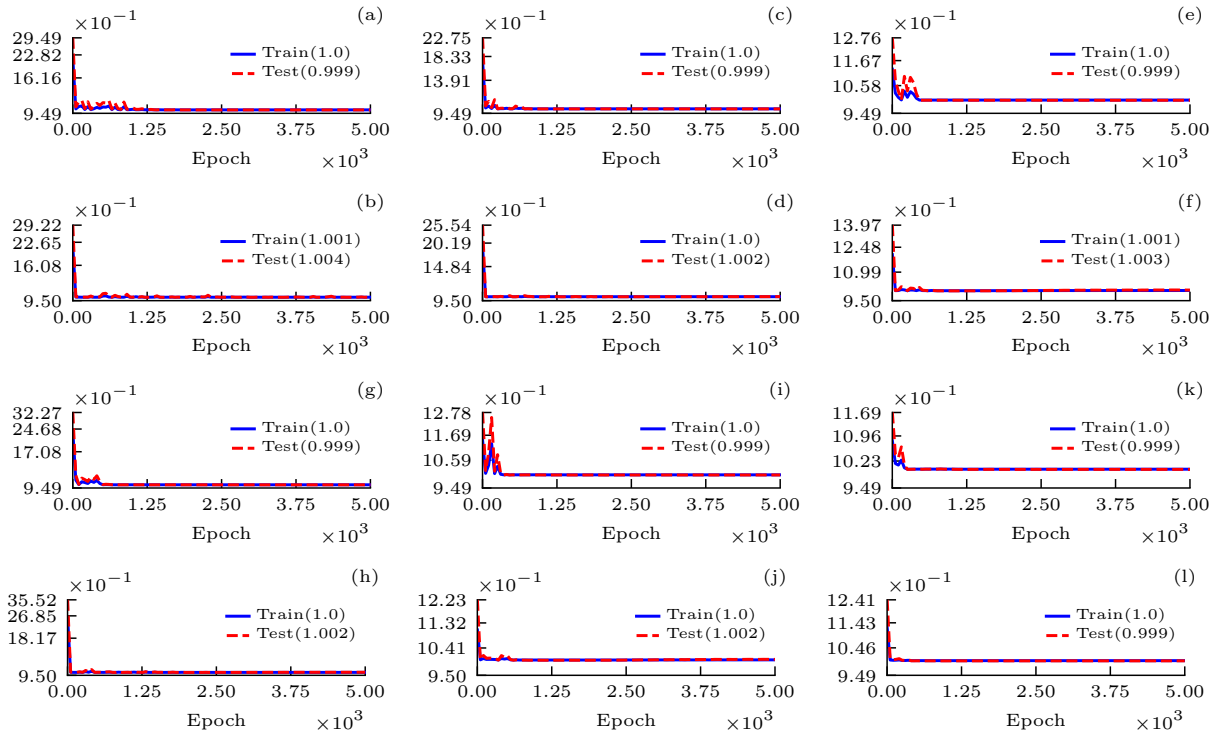



Figure 4.4: **Relative $L^2$ Error for DeepONet:** Training and testing Relative $L^2$ error with epoch for DeepONet when considered different sample size in training and with different trunk and branch sizes. The plots (a), (c), (e), (g) (i) (k) are the cases corresponding to Case-1 to Case-6 of Table 4.1, respectively, with Loss function (4.24). Similarly, the plots (b), (d), (f), (h), (j), (l) are the cases corresponding to Case-1 to Case-6 of Table 4.1, respectively, with Loss function (4.25).

To understand the effect of normalization of the input and output dataset on the accuracy of the predicted results, we consider Gaussian normalization of the input and output data,

$$x_{\text{norm}}(t) = \frac{x(t) - \mu_x(t)}{\sigma_x(t)} \tag{4.29}$$

where $x_{\text{norm}}(t)$ are the data after normalization. $\mu_x(t)$ and $\sigma_x(t)$ are the ensemble mean and standard deviation of the training dataset.

The predicted responses are decoded to the actual response with the same mean and standard deviation. Similar to the DeepONet case discussed above, we study the effect of normalization with different network sizes and the results are shown in Table 4.2 along with training and testing loss for only the MSE Loss function (4.24) considered. A ReLU($x$) activation function is considered with learning rate of $10^{-3}$ in the first 1000 epochs, then $10^{-4}$ in the 1000 to 10000 epochs, $10^{-5}$ for rest of the epoch up to 20000 epochs. The other hyperparameters considered are a dropout rate of 0.01 and a $L^2$ weight regularization with a coefficient of $10^{-5}$ for the branch net. The relative $L^2$ error with epoch for training and testing dataset are shown in Figure 4.5. It could be observed that the training relative $L^2$ error does not reduce even after the normalization of the input and output.

Table 4.2: Relative $L^2$ error for training and testing dataset when predicted using different sizes of DeepONet with Gaussian normalization for input and output.

| Case | Branch | Trunk | Sample | Loss (4.24) | |
|---|---|---|---|---|---|
| | | | | Train | Test |
| 1 | [4000]-[100]×3-[100] | [1]-[100]×3-[100] | 100 | 1.847 | 2.379 |
| 2 | [4000]-[200]×3-[200] | [1]-[200]×3-[200] | 100 | 1.841 | 2.378 |

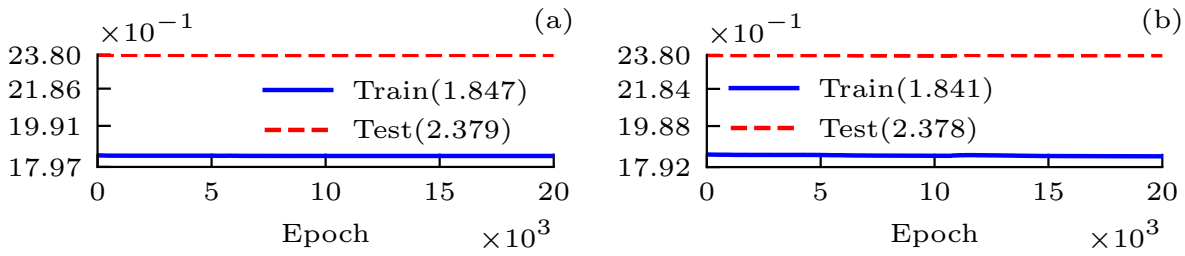



Figure 4.5: **Relative $L^2$ Error for DeepONet with Normalization:** Relative $L^2$ Error for training and testing dataset when using different sizes of DeepONet with Gaussian normalization for input and output. (a) and (b) are Case-1 and Case-2 of Table 4.2, respectively.

From the above discussion, it could be observed that the DeepONet is not able to predict the response of the building with sufficient accuracy. As discussed in Section 4.3, one of the modified versions of DeepONet is the POD-DeepONet, where the trunk net is replaced by POD modes obtained by proper orthogonal decomposition of the zero-mean training data (output data). These bases act as the trunk, and the branch is expected to learn the coefficient of the basis vectors.

Similar to DeepONet, in this case as well, we consider two loss functions given by (4.24) and (4.25) with different sizes of branch net as shown in Table 4.3. The training of POD-DeepONet is considered with Adam optimizer for a total of 20000 epochs with ReLU($x$) activation function. To avoid overfitting, we consider $L^2$ weight regularization with coefficient $10^{-6}$. The learning rate considered is $10^{-3}$ in the first 5000 epochs, then $10^{-4}$ in the 5000 to 10000 epochs, and $10^{-5}$ for the rest of the epochs up to 20000. The relative $L^2$ error with epoch is shown in Figure 4.6. It could be observed that the loss function with an additional penalty given by equation (4.25) could offer better convergence for training cases compared with the case only considering MSE loss given by equation (4.24). Further, the relative $L^2$ errors are smaller compared to DeepONet for the training dataset. However, the performance of the network is poor in the case of the testing dataset shows that 100 training samples could not offer enough information for the target response space, even

though there are slight improvements with the increase in network size. The relative $L^2$ errors for training and testing dataset with epoch when using POD-DeepONet with different activation functions are shown in Figure A.6. The predicted response for the best training and testing cases are shown in Figure A.8. The performance of POD-DeepONet highly relies on the quality of the training data. If the training dataset covers a large enough region of the space of interest, then the POD-DeepONet could have very good performance.

Table 4.3: Relative $L^2$ error for training and testing dataset when using different sizes of POD-DeepONet.

| Case | Network Size[1] | Sample | Loss (4.24) | | Loss (4.25) | |
|---|---|---|---|---|---|---|
| | | | Train | Test | Train | Test |
| 1 | [4000]-[50]×3-[100] | 100 | 0.517 | 1.254 | 0.213 | 1.054 |
| 2 | [4000]-[100]×3-[100] | 100 | 0.469 | 1.209 | 0.085 | 1.047 |
| 3 | [4000]-[200]×3-[100] | 100 | 0.448 | 1.213 | 0.061 | 1.033 |

[1] The notation $[N_1]$-$[N_2]$×3-$[N_3]$ represents a neural network with the input size of $N_1$, 3 hidden layers with $N_2$ neurons in each layer, and the output dimension of $N_3$ neurons.

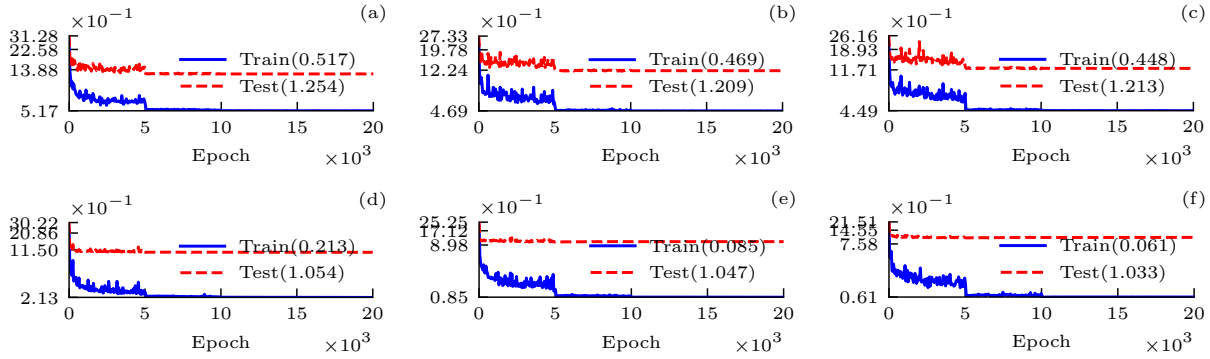



Figure 4.6: **Relative $L^2$ error for POD-DeepONet:** Relative $L^2$ Error for training and testing dataset when using POD-DeepONet with different sizes. The plots (a), (c), (e) are the cases corresponding to Case-1 to Case-3 of Table 4.3, respectively, with Loss function (4.24). Similarly, the plots (b), (d), (f) are the cases corresponding to Case-1 to Case-3 of Table 4.3, respectively, with Loss function (4.25).

### 4.5.2. Multi-scale DeepONet

In the previous section, we discussed the results of DeepONet and POD-DeepONet and observed that the results were not satisfactory. In this section, we will present and discuss the results of one of the proposed variants of DeepONet, the multi-scale DeepONet.

Table 4.4: Relative $L^2$ Error for Training and Testing Dataset when Using Different Sizes of Multi-scale DeepONet.

| Case | Branch[1] | Trunk[2] | Sample | Loss (4.24) | | Loss (4.25) | |
|---|---|---|---|---|---|---|---|
| | | | | Train | Test | Train | Test |
| 1 | [4000]-[200]×3-[200] | [1]-20×{[10]×3}-[200] | 100 | 0.349 | 1.041 | 0.304 | 0.994 |
| 2 | [4000]-[400]×3-[400] | [1]-20×{[20]×3}-[400] | 100 | 0.232 | 1.005 | 0.166 | 1.006 |

[1] The notation $[N_1]$-$[N_2]$×3-$[N_3]$ represents a neural network with the input size of $N_1$, 3 hidden layers with $N_2$ neurons in each layer, and the output dimension of $N_3$ neurons.
[2] The notation $[N_1]$-20×{$[N_2]$×3}-$[N_3]$ represents a neural network with the input size of $N_1$, and 20 sub-neural networks that contain 3 hidden layers with $N_2$ hidden neurons in each layer. The output dimension is $N_3$. To keep the number of neurons as the same as previous cases, the number of hidden neurons for each subnet at each layer are divided by 20, the number of scales.

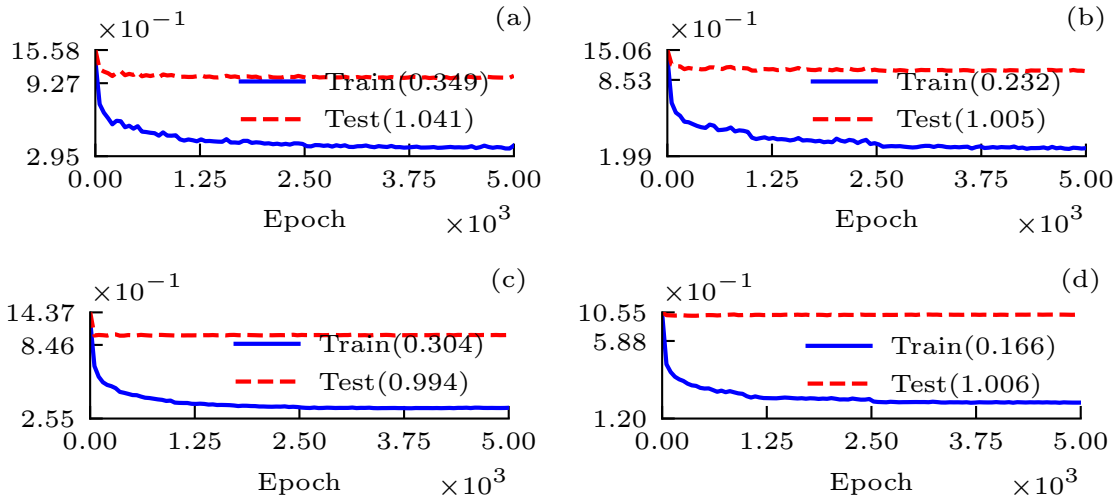



Figure 4.7: **Relative $L^2$ Error for Multi-scale DeepONet:** Relative $L^2$ Error with epoch for training and testing dataset when using different sizes of multi-scale DeepONet. The plots (a) and (b) correspond to Case-1 and Case-2 of Table 4.4, respectively, with loss function given by equation (4.24). The plots (c) and (d) correspond to Case-1 and Case-2 of Table 4.4, respectively, with loss function (4.25).

The architecture of multi-scale DeepONet is discussed in Section 4.4.1 where the fully connected deep neural network in the trunk net is replaced by a multi-scale neural network (MscaleDNN). In the present study we consider an MscaleDNN in the trunk with 20 equally spaced scales $[1, 1+20\pi, \cdots, 1+20n\pi, \cdots, 1+780\pi]$. In the meantime, the time $t$ is scaled to $t \in [0, 1]$. The activation function considered for all the cases is $\sin(x)$, according to the results in [42, 67]. To avoid overfitting, we consider dropout rate of 0.10 for the trunk net. The learning rate considered is $3 \times 10^{-4}$ in the first 1000 epochs, then $1.5 \times 10^{-4}$ in the 1000 to 2500 epochs, and $7.5 \times 10^{-5}$ for the rest of the epochs up to 5000. The relative $L^2$ errors for different network sizes with different training loss functions are shown in Table 4.4. The relative $L^2$ errors with epoch for training and testing dataset are shown in Figure 4.7. It could be observed the obtained operator is not desired based on the results of testing cases, though the MS-DeepONet accelerated the convergence for the training process.

### 4.5.3. Causality-DeepONet

As discussed in Section 4.2 and Section 4.4.2, both convolution and causality are considered in the formulation of Causality-DeepONet. In this section, we will present the numerical results and a comprehensive discussion about Causality-DeepONet for the prediction of the responses of the problem discussed in Section 4.2.

Similar to the previous numerical examples, in this study as well, we consider the two loss functions given by equations (4.24) and (4.25) with different network sizes. We also study the effect of the number of training samples on the accuracy of test results. Further, given the fact that there are multiple choices of activation functions, we test a few of the popular activation functions with the same training dataset and the same network sizes. As shown later, Causality-DeepONet with standard sigmoid activation functions converges much slower. By defining a custom sigmoid function, the performance is improved. Furthermore, to study the effect of only causality without convolution, we do a numerical study with causality only

and observed that convolution is also an indispensable component of the proposed Causality-DeepONet. Unlike the previous studies, we provide the initial conditions as additional data pairs $\{\ddot{u}_g : [0, 0, \ldots, 0], x(0) : 0\}$ in the training dataset to force the Causality-DeepONet satisfy the initial conditions.

Figure 4.8 and Figure 4.9 show the worst and the best predictions using Causality-DeepONet for the test dataset. The network is trained using 100 training samples. The network size considered is [4000]-[120]×2-[120] for branch and [1]-[120]×2-[120] for trunk. The activation function considered is $\tanh(x)$. To avoid overfitting, we consider $L^2$ weight regularization with a coefficient $1 \times 10^{-4}$ for the branch net. The learning rate considered is $10^{-3}$ in the first 2000 epochs, then $10^{-4}$ in the 2000 to 10000 epochs, and $10^{-5}$ for the rest of the training up to 20000 epochs. The loss function considered for this case is Loss (4.25). It could be observed that Causality-DeepONet can predict the responses with good accuracy for all the cases, as the error for the worst case also is within the satisfactory limit.

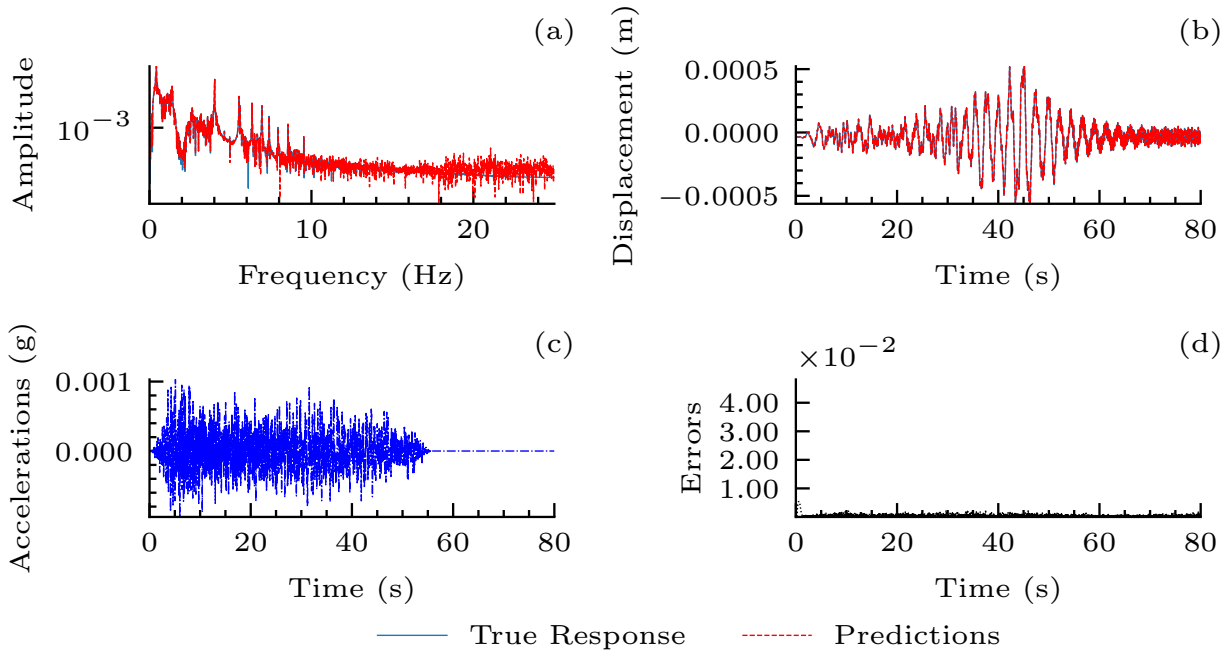



Figure 4.8: **The Worst Case of Predictions of Causality-DeepONet(Relative $L^2$ Error: 0.0042):** The worst predictions in testing dataset for the Causality-DeepONet. (a) The Amplitude of the prediction and true response in Fourier Domain, (b) The prediction and true response, (c) The corresponding input signals, (d) The relative error equation (4.27).

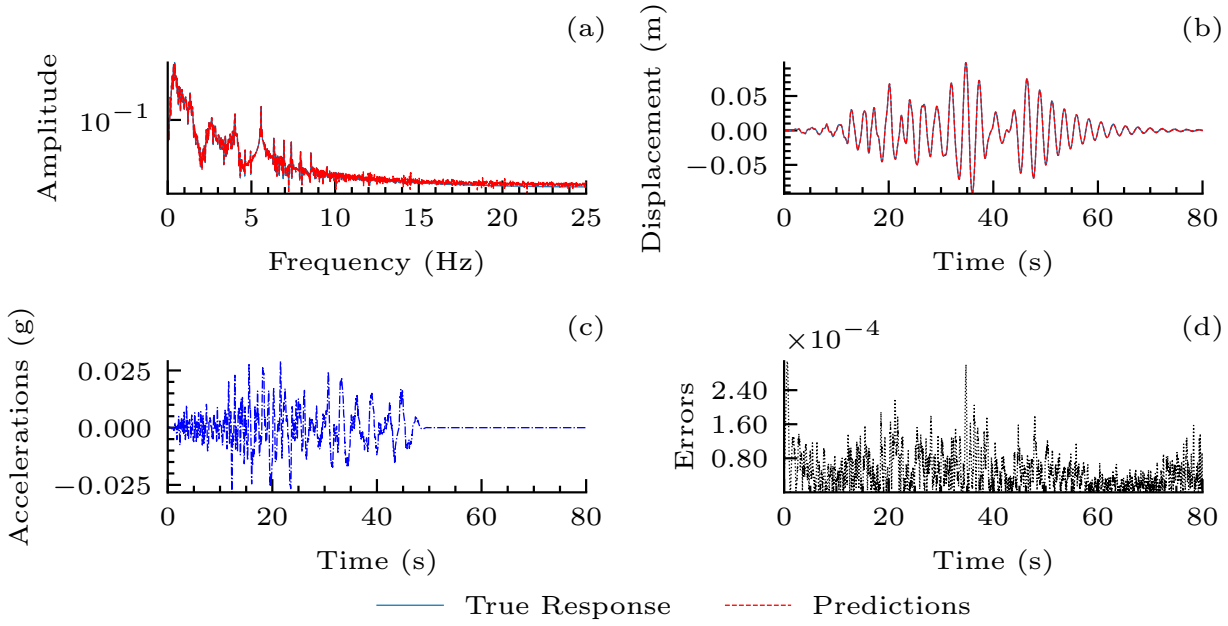



Figure 4.9: **The Best Case of Predictions of Causality-DeepONet(Relative $L^2$ Error: 0.00025)** The best predictions in testing dataset for the Causality-DeepONet. (a) The Amplitude of the prediction and true response in Fourier Domain, (b) The prediction and true response, (c) The corresponding input signals, (d) The relative error equation (4.27).

To study the effect of different network size and loss function ((4.24) and (4.25)), we consider different network sizes with the same number of training dataset. The activation function considered for all the cases is $\tanh(x)$. To avoid overfitting, we consider $L^2$ weight regularization for the branch net with a coefficient $10^{-4}$. The learning rate considered is $10^{-3}$ in the first 2000 epochs, then $10^{-4}$ in the 2000 to 10000 epochs, and $10^{-5}$ for the rest of the epochs up to 20000 epochs. The relative $L^2$ errors for both training and testing dataset after completion of training is shown in Table 4.5 and the corresponding relative $L^2$ error with epoch is shown in Figure 4.10. It could be observed that the proposed Causality-DeepONet shows a good accuracy for both loss functions. The MSE loss function given by (4.24) is more sensitive to the network size as relative $L^2$ errors for both training and testing are reduced with an increase in network sizes. The weighted loss function given by (4.25) is less sensitive to the network sizes for this numerical study. Further, it is also observed that the

relative $L^2$ error in the case of loss function (4.25) is less than that of relative $L^2$ error in the case of loss function (4.24). Thus, we conclude that the additional penalty terms in the loss function removes the bias from the magnitude of output functions/data in the present study. For the further numerical studies conducted, we consider with loss function given by (4.25) only.

Table 4.5: Relative $L^2$ error for training and testing dataset when using different sizes of Causality-DeepONet

| Case | Branch[1] | Trunk[1] | Sample | Loss (4.24) | | Loss (4.25) | |
|---|---|---|---|---|---|---|---|
| | | | | Train | Test | Train | Test |
| 1 | [4000]-[30]×2-[30] | [1]-[30]×2-[30] | 10 | 0.055 | 0.089 | 0.017 | 0.018 |
| 2 | [4000]-[60]×2-[60] | [1]-[60]×2-[60] | 10 | 0.023 | 0.040 | 0.013 | 0.014 |
| 3 | [4000]-[90]×2-[90] | [1]-[90]×2-[90] | 10 | 0.030 | 0.051 | 0.018 | 0.019 |
| 4 | [4000]-[120]×2-[120] | [1]-[120]×2-[120] | 10 | 0.018 | 0.033 | 0.012 | 0.015 |

[1] The notation $[N_1]$-$[N_2]$×2-$[N_3]$ represents a neural network with the input size of $N_1$, 2 hidden layers with $N_2$ neurons in each layer, and the output dimension of $N_3$ neurons.

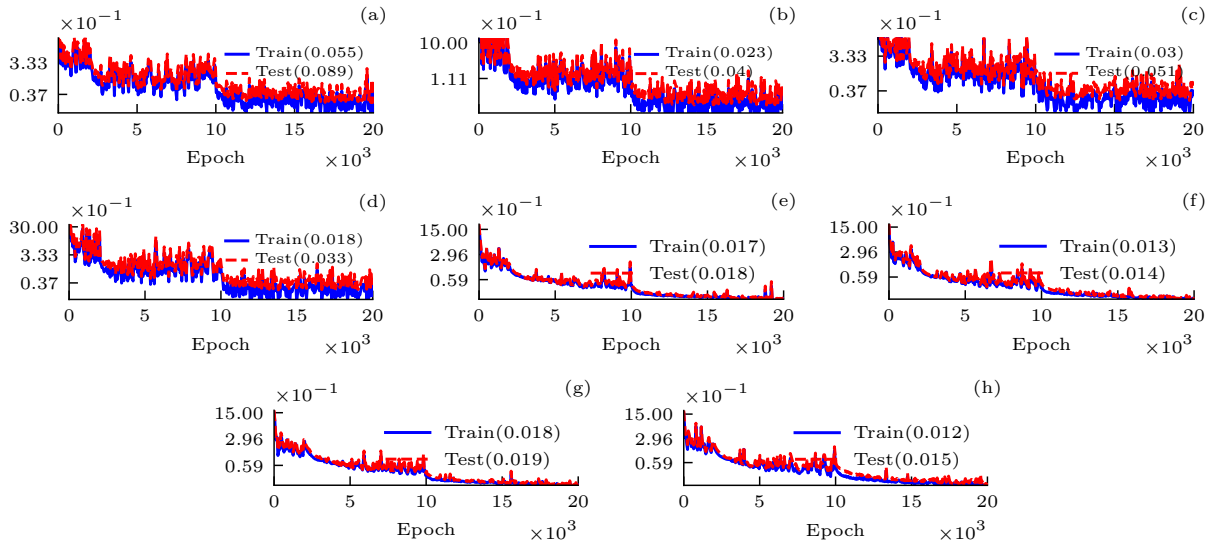



Figure 4.10: **Relative $L^2$ Error for Training and Testing Dataset when Using Causality-DeepONet with Different Branch and Trunk Sizes:** The plots (a), (b), (c), (d) are the cases corresponding to Case-1 to Case-4 of Table 4.5, respectively, with Loss function equation (4.24). Similarly, the plots (e), (f), (g), (h) are the cases corresponding to Case-1 to Case-4 of Table 4.5, respectively, with Loss function (4.25).

Table 4.6: Relative $L^2$ error for training and testing dataset when using Causality-DeepONet with different activation functions

| Case | Activation | Sample | Relative $L^2$ Error (4.26) | |
|---|---|---|---|---|
| | | | Train | Test |
| 1 | $\tanh(x)$ | 10 | 0.012 | 0.015 |
| 2 | $\sin(x)$ | 10 | 0.013 | 0.019 |
| 3 | Sigmoid$(x)$ | 10 | 0.165 | 0.143 |
| 4 | ReLU$(x)$ | 10 | 0.008 | 0.015 |
| 5 | Custom Sigmoid (4.30) | 10 | 0.02 | 0.024 |

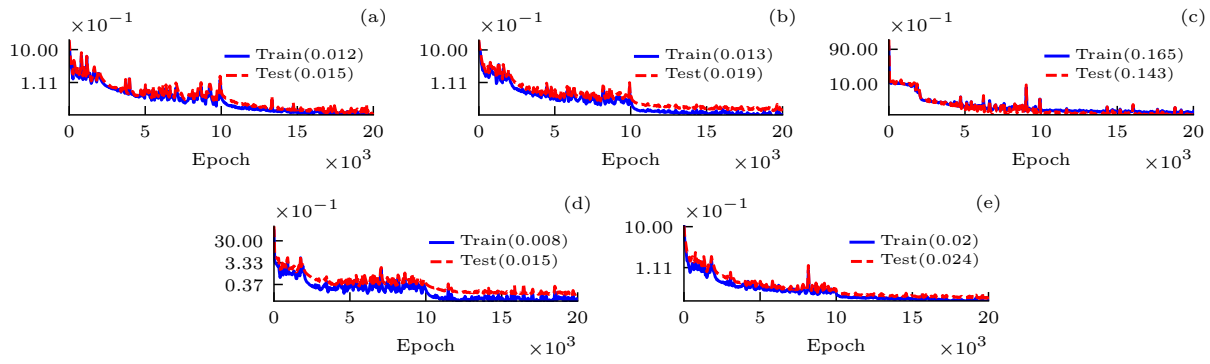



Figure 4.11: **Relative $L^2$ Error for Training and Testing Dataset when Using Causality-DeepONet With Different Activation Functions:** The plots (a)-(e) correspond to cases Case-1 to Case-5 of Table 4.6, respectively.

As discussed earlier, there are multiple choices of activation functions, and we test the performance of Causality-DeepONet with a few popular activation functions. For this purpose, we consider the same network sizes and training dataset and loss functions (4.25) for all the cases of activation function considered. The network sizes considered are [4000]-[120]×2-[120] for branch and [1]-[120]×2-[120] for trunk. To avoid overfitting, we consider $L^2$ weight regularization for the parameters in branch net with a coefficient $10^{-4}$ for case 1,2,4,5 and $5 \times 10^{-6}$ for case-3 in Table 4.6. The learning rate considered is $10^{-3}$ in the first 2000 epochs, then $10^{-4}$ in the 2000 to 10000 epochs, and $10^{-5}$ for the rest of the epoch up to 20000 epochs. As shown in Table 4.6 and Figure 4.11, It could be concluded that the Causality-DeepONet with $\tanh(x)$, $\sin(x)$ and $\mathrm{ReLU}(x)$ as activation functions obtains excellent predictions given limited training samples. However, the Causality-DeepONet with Sigmoid as activation function is not convergent as expected. By shifting the Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}} - \frac{1}{2} \tag{4.30}$$

could improve the results, as shown in Figure 4.11(e) and case 5 in Table 4.6.

From the above discussion it could be observed that the proposed Causality-DeepONet is able to predict the response of the problem considered with a good accuracy. We also

study the effect of training samples in the accuracy of predicted response. For this purpose we consider different samples with same network size and other hyperparameters. The statistical properties of the different training samples are shown in Table **??** in Appendix A.1. It could be noted that the training samples in datasets Train-I, Train-II, Train-III are exclusively different. The training samples in datasets Train-II and Train-III are included in the dataset Train-IV. The training samples in dataset Train-I and Train-IV are included in dataset Train-V. The training samples in dataset Train-V are included in dataset Train-VI.

We consider a network size of [4000]-[120]×2-[120] for branch and [1]-[120]×2-[120] for trunk. The activation function considered is $\tanh(x)$. To avoid overfitting, we consider $L^2$ weight regularization with a coefficient $1 \times 10^{-4}$ for the branch net. The learning rate considered is $10^{-3}$ in the first 2000 epochs, then $10^{-4}$ in the 2000 to 10000 epochs, and $10^{-5}$ for the 10000 to 20000 epochs. The relative $L^2$ errors of different cases with different sample in training are shown in Table 4.7 and Figure 4.12. It could be observed that the $L^2$ error is small even with smaller number of training set and the accuracy increases with the increase in number of training set, though the improvements in accuracy is limited with increase in number of samples. On the other hand, it could also be observed that the performance of Causality-DeepONet of Case-3 in Table 4.7 with 10 training samples that have larger deviation is poor comparing with Case-4 to Case-6 in Table 4.7. The further observation from Table 4.7 is that the training relative $L^2$ error is greater than the testing relative $L^2$ error in Case-4 of Table 4.7, given the fact that dataset Train-IV contains Train-II and Train-III.

Table 4.7: Relative $L^2$ error for training and testing dataset when using Causality-DeepONet with different numbers of training samples

| Case | Sample | Relative $L^2$ Error (4.26) | |
|---|---|---|---|
| | | Train | Test |
| 1 | 7 | 0.005 | 0.017 |
| 2 | 8 | 0.003 | 0.003 |
| 3 | 10 | 0.012 | 0.015 |
| 4 | 20 | 0.006 | 0.003 |
| 5 | 50 | 0.003 | 0.003 |
| 6 | 100 | 0.002 | 0.002 |

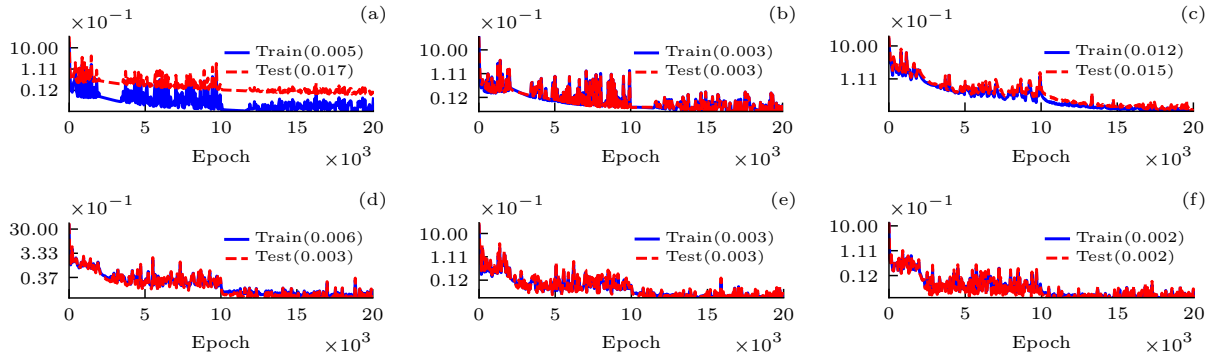



Figure 4.12: **Relative $L^2$ Error for Training and Testing Dataset when Using Causality-DeepONet with Different Number of Training Samples:** The plots (a)-(e) correspond to the cases Case-1 to Case-5 in Table 4.7, respectively.

As discussed in Section 4.2, the proposed Causality-DeepONet involves both the phenomenon of causality and convolution. To evaluate the importance of convolution on the accuracy of prediction, we study the method only with causality but without convolution. The neural network considered for this purpose has form

$$\mathcal{R}_c(\ddot{u}_g)(t) \sim \sum_{k=1}^{N} c_k \underbrace{\sum_{i=1}^{M} \sigma_b \left( \sum_{j=1}^{\lfloor \frac{t}{h} \rfloor} W_{i,j}^k \ddot{u}_g(s_j) + \sum_{j=\lfloor \frac{t}{h} \rfloor + 1}^{\lfloor \frac{T}{h} \rfloor} W_{i,j}^k 0 + B_i^k \right)}_{B_k} \underbrace{\sigma_t(\boldsymbol{w}_k t + b_k)}_{T_k} \tag{4.31}$$

The difference between the formulation given by equation (4.31) and the proposed Causality-DeepONet (4.23) is in the difference in the weights of the branch. In the case of the formulation without convolution, the weights are $W^k_{i,j}$, whereas in the case of the proposed Causality-DeepONet (with convolution) the weights are $W^k_{i,\lfloor \frac{T}{h} \rfloor - \lfloor \frac{t}{h} \rfloor + j}$.

To study the effect of convolution, we compared the results of the problem with Causality-DeepONet and Causality-DeepONet without convolution. A network size of [4000]-[120]×2-[120] for branch and [1]-[120]×2-[120] for the trunk are considered in both the cases. The activation function considered for both cases is $\tanh(x)$. To avoid overfitting, we consider $L^2$ weight regularization for the branch net with a coefficient $10^{-4}$. The learning rate considered is $10^{-3}$ in the first 2000 epochs, then $10^{-4}$ in the 2000 to 10000 epochs, and $10^{-5}$ for the rest of the epoch up to 20000 epochs. The case without convolution is considered to be trained with 100 training samples. However, the case with convolution is trained with 10 training samples only. Figure 4.14 shows the relative $L^2$ error for both the cases. It can be observed that the variant without convolution is not able to provide satisfactory accuracy.

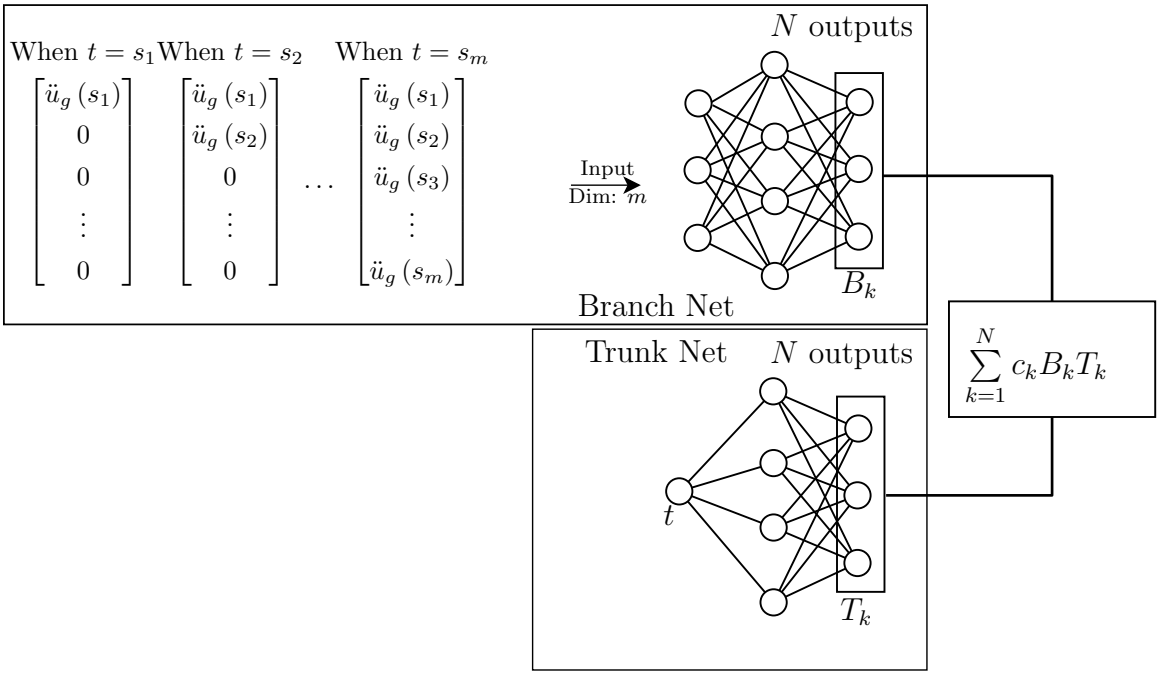



Figure 4.13: **Schematic Diagram of Causality-DeepONet without Convolution:** A schematic of the Causality-DeepONet without convolution showing branch and the trunk net along with the input data and output. The number of input neurons for the branch is equal to the number of sensor points in the input signals. The input signals for the branch, however, are replaced by a zero-padding for the future information.

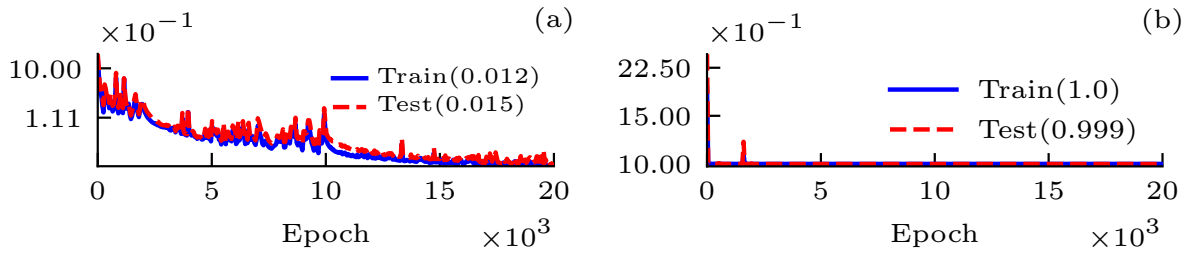



Figure 4.14: **Relative $L^2$ Error for Training and Testing Dataset when Using Causality-DeepONet with or without Convolutions:** (a) Causality-DeepONet with convolution with loss function (4.25), 10 training samples. (b) Proposed Net with causality only with loss function (4.25), 100 training samples.

# Chapter 5

## DeepPropNet - A Recursive Deep Propagator Neural Network for Learning Evolution PDE Operators

*The content in this chapter has been submitted as a preprint in collaboration with Wei Cai:*

Lizuo Liu, and Wei Cai, *DeepPropNet–A Recursive Deep Propagator Neural Network for Learning Evolution PDE Operators*, arXiv preprint arXiv:2202.13429 (2022) [40].

### 5.1. Introduction

Consider an evolution system

$$\begin{cases} \dfrac{\partial^n u}{\partial t^n} = \mathcal{L}_{\boldsymbol{x}} u + f(\boldsymbol{x}, t) \quad \boldsymbol{x} \in \mathbb{R}^d \text{ or periodic} \\ u(\boldsymbol{x}, 0) = u_0(\boldsymbol{x}) \\ u'(\boldsymbol{x}, 0) = u_1(\boldsymbol{x}) \\ \qquad \vdots \\ u^{(n-1)}(\boldsymbol{x}, 0) = u_{n-1}(\boldsymbol{x}) \end{cases} \tag{5.1}$$

where $\mathcal{L}_x$ are linear operators, $n = 1$ or 2 and $u$ could be a scalar or vector.

To obtain an intuitive sense of the problem to study, we first consider the inhomogeneous scalar wave (d'Alembert) equation

$$\frac{\partial^2 u}{\partial t^2} - c^2(x,t)\frac{\partial^2 u}{\partial x^2} = f(x,t), \quad x \in \mathbb{R}, \quad t \geq 0, \tag{5.2}$$

with the source term $f$ compactly supported on a bounded space-time domain $Q = S \times [0, T]$ where $S \subset \mathbb{R}$. This means that the source term $f(x, t)$ differs from zero only on $S$ and for the limited time interval $[0, T]$. When $c\,(x, t)$ is constant, the solution to the Cauchy problem (5.2) is given by:

$$u(x,t) = \frac{1}{2}(u_0(x-ct) + u_0(x+ct)) + \frac{1}{2c}\int_{x-ct}^{x+ct} u_1(\xi)d\xi + \frac{1}{2c}\int_0^t \int_{x-c(t-\tau)}^{x+c(t+\tau)} f(\xi,\tau)d\xi d\tau. \quad (5.3)$$

When $n = 1$, the solution $u(x, t)$ to (5.1) can also be formally written as

$$\begin{aligned} u(x,t) &= e^{t\mathcal{L}}u_0 + \int_0^t e^{(t-s)\mathcal{L}} f(x,s)ds \\ &= P(u_0, f(x,s), 0 \le s \le t). \end{aligned} \quad (5.4)$$

The solution $u(x, t)$ can be viewed through an evolution operator $P(u_0, u_1, \ldots, u_{n-1}, f(x, s), 0 \le s \le t)$, which maps the initial conditions and the source term into the solution $u\,(x, s)\,, 0 < s < t$. Learning such a map between functions has been actively studied recently with various types of operator learning methods, including DeepONet [47] as discussed previously and Fourier Neural Operator [37]. The focus of this chapter is to find an efficient way to learn this evolution operator with moderate size neural network for large time $t$. For a lack of a precise term, borrowing the term from quantum mechanics for the Green's function propagator [64], we shall name the operator $P$ as the propagator for the evolution system. From equation (5.3), it is clear that if we like to train a neural network operator for large time $t$, the size of the network will grow for increasing time $t$. And, the amount of information to be input into a network will increase dramatically as $t$ grows as well. Thus, we will propose a recursive propagator formulation for the evolution operator network.

First, the solution time interval $[0, T]$ will be divided into $N$ smaller sub-intervals

$$t_0 = 0 < t_1 < \cdots < t_i < \cdots t_N, t_i = i\Delta t, \quad \Delta t = T/N, \tag{5.5}$$

and for $t_i \leq t \leq t_{i+1}$, the solution is given by the propagator with initial condition of the solution $u_i$ and its velocity $v_i = \dot{u}_i$, i.e.,

$$u(x,t) = P(u_i, v_i, f(x,s), t_i \leq s \leq t), \tag{5.6}$$

where the initial condition $u_i$ would have been given by the propagator for the time block $t_{i-1} \leq t \leq t_i$.

The propagators in (5.6) will be approximated by a single neural network in the form of DeepONet structure [47] with modification for time causality, namely,

$$P(u_i, v_i, f(x,s), t_i \leq s \leq t) \sim P_\theta(u_i, v_i, f(x,s), t_i \leq s \leq t), 0 \leq i \leq N-1. \tag{5.7}$$

Therefore, the propagator $P_\theta(u_0(x), v_0(x), f(x,t))$ will be trained to map the initial and forcing function data into the solution $u(x,t), t_0 \leq t \leq t_1$. Moreover, this same propagator $P_\theta$ will be trained to approximate the solution for time periodic $t_1 \leq t \leq t_2$ where the initial condition at $t_1$ can be computed with the propagator for the previous time interval $[t_0, t_1]$. This procedure will be used recursively until we have trained the same propagator for the last time interval $[t_{N-1}, t_N]$. Taken all together, we arrive at a propagator neural network for the whole time interval $[0, T]$ where the building block is the single propagator of moderate size $P_\theta(u_0(x), v_0(x), f(x,t)), t_0 \leq t \leq t_1)$. By controlling the size of the $\Delta t$, the size of this propagator can be easily controlled for efficiency as well as accuracy. As the evolution PDE system has to observe the causality of the physical system, the DeepONet framework will be modified to include the causality, a previously proposed Causality-DeepONet in the study of

dynamical system for modeling building response to seismic waves [41] will be used for this purpose.

The rest of the chapter is organized as follows. In Section 5.2, we will review the DeepONet [41] with time causality and extension with proper orthogonal decomposition (POD) approach for efficient treatment of spatial dependence of the solution. Section 5.3 will give the algorithm of the DeepPropNet and numerical results of the DeepPropNet will be presented in Section 5.4.

## 5.2. DeepONet with time causality and spatial POD

A Causality-DeepONet was discussed in Chapter 4 to handle the time causality in dynamical system and was shown to be very effective to predict the seismic response of building. Here, we will just recall the final form.

**Causality-DeepONet:** A DNN representation of an operator $\mathcal{G}(f)(t)$ for any continuous function $f(t)$ with retarded response for $t \in K_2 = [0, T] \subset \mathbb{R}$ is given as

$$\mathcal{G}(f)(t) \sim \mathcal{G}_\theta(f)(t) = \sum_{k=1}^{K} \sum_{i=1}^{I} c_i^k \sigma_b \left( \sum_{j=1}^{\lceil \frac{t}{h} \rceil} \xi_{i,m-\lceil \frac{t}{h} \rceil + j}^k f(s_j) \right) \cdot \sigma_{trk}(\omega_k \cdot t + \zeta_k), \tag{5.8}$$

where $\{s_j\}_{j=1}^m \subset K_1 = [0, T] \subset \mathcal{X}$, coefficients $\theta = \{c_i^k, \xi_{ij}^k, \omega_k, \zeta_k, j = 1, \ldots, \lceil \frac{t}{h} \rceil, i = 1, \ldots, I, k = 1, \ldots K\}$-all independent of continuous function $f \in \mathcal{V} \subset C(\mathcal{F})$ and $t$.

To handle the spatial dependence of solution $u(x, t)$ for the evolution system, we will adopt the idea of separation of variables and assume the spatial coefficients to be some given basis functions, like sinusoidal or polynomials, which correspond to different boundary conditions. As for the problem discussed in this chapter, since the boundary condition is periodic, the basis functions will be sinusoidal.

Indeed, there are many other ways to construct the spatial basis. As shown in POD-DeepONet [48], the authors assumed that there is a set of global basis which could be found by singular value decomposition(SVD) or proper orthogonal decomposition(POD) of the targeted output of training data. The trunk net will be replaced by these basis functions, then the neural network is learning the mapping between input functions to the coefficients for different basis functions, or singular values if we consider SVD as an example. Thus, the quality of the training data could be the Achilles' heel of POD-DeepONets. As discussed in [3], the approximation error has order $O\left(N^{-\frac{1}{2}}\right)$ if considering doing POD/SVD on both the input functions and output functions and the mapping between the coefficients is learned, where $N$ is the number of training cases.

Furthermore, as for a problem with causality, The POD-DeepONet's philosophy guides us either do the SVD for the whole outputs regardless of the difference of temporal variables and spatial variables, or do specific SVD time step by time step to keep the causality for which the basis for all time steps are required, or consider solutions at each time step for each case as an independent target and find a common basis with respect to spatial dimensions for all time steps & all cases. These ideas either burns the high memory cost, or destroys the causality in a brutal-force way. Therefore, a modification of POD-DeepONet for problems with causality is crucial. As mentioned before, we follow the idea of POD-DeepONet but only construct the basis of spatial domain explicitly and utilize the Causality-DeepONet to handle the temporal-dependent coefficients of each spatial basis.

In the following, we consider the solution with form

$$u(x,t) = \psi_0(t) + \sum_{m=1}^{M} \psi_m(t)\cos(m\pi x) + \phi_m(t)\sin(m\pi x), \tag{5.9}$$

which is the Fourier decomposition with respect to the spatial coordinates.

We modify the Causality-DeepONet (5.8) by

$$\mathcal{P}\left(\overrightarrow{f}\right)(x,t) \sim \mathcal{P}_\theta\left(\overrightarrow{f}\right)(x,t) = \left[\overrightarrow{\sigma}_{br}\left(\overrightarrow{f}\right) \odot \overrightarrow{\sigma}_{trk}(t)\right] \cdot \overrightarrow{\sigma}_{basis}(x), \tag{5.10}$$

where $\odot$ is the elementwise multiplication, $\cdot$ is the inner product,

$$\overrightarrow{f}(s) = \left[a_0(s), a_1(s), b_1(s), \ldots, a_N(s), b_N(s)\right],$$

$$\begin{aligned}\overrightarrow{\sigma}_{br,i}\left(\overrightarrow{f}\right) = \sigma_b \Bigg( &\sum_{j=1}^{\lceil \frac{t}{h} \rceil} \xi^0_{i,m-\lceil \frac{t}{h} \rceil + j} a_0(s_j) \\ &+ \sum_{n=1}^{N} \sum_{j=1}^{\lceil \frac{t}{h} \rceil} \left( \xi^{n,a}_{i,m-\lceil \frac{t}{h} \rceil + j} a_n(s_j) + \xi^{n,b}_{i,m-\lceil \frac{t}{h} \rceil + j} b_n(s_j) \right) \Bigg),\end{aligned} \tag{5.11}$$

$$\overrightarrow{\sigma}_{trk,i}(t) = \sigma_{trk}(\omega_i \cdot t + \zeta_i), \tag{5.12}$$

$$\overrightarrow{\sigma}_{basis}(x) = \{1, \cos(\pi x), \sin(\pi x), \cdots, \cos(\pi B x), \sin(\pi B x)\}. \tag{5.13}$$

$a_0(s)$, $\ldots a_n(s)$ and $b_1(s), \ldots, b_n(s)$ are the time-dependent coefficients of the Fourier decomposition in spacial domain with respect to the right-hand side $f(x,t)$.

Note the Causality DeepONet with POD (5.10) automatically satisfies the initial condition,

$$u(x,0) = 0. \tag{5.14}$$

Thus, by multiplying $(t-t_0)^{i-1}$, it satisfies the initial conditions,

$$\frac{\partial^{i-1} u}{\partial t^{i-1}}(x,0) = 0, \quad i = 1,2. \tag{5.15}$$

So we could incorporate the non-zero initial conditions by adding several extra terms $(t-t_0)^j u_j(x)$, $j=0,\ldots,n-1$, i.e., the Causality-DeepONet (5.10) is modified by

$$\mathcal{P}_\theta\left(\overrightarrow{f}\right)(x,t,\overrightarrow{u_0},\ldots,\overrightarrow{u_{n-1}}) = \left[(t-t_0)^{n-1}\overrightarrow{\sigma}_{br}\left(\overrightarrow{f}\right)\odot\overrightarrow{\sigma}_{trk}(t) + \sum_{j=0}^{n-1}(t-t_0)^j\overrightarrow{u_j}\right]\cdot\overrightarrow{\sigma}_{basis}(x), \tag{5.16}$$

where $\overrightarrow{u_j}$ are the coefficients of initial conditions obtained by

$$\begin{aligned}\overrightarrow{u_j}_{,2k+1} &= \int_{-\infty}^{\infty} u_j(x)\sin(k\pi x)\,dx,\\ \overrightarrow{u_j}_{,2k} &= \int_{-\infty}^{\infty} u_j(x)\cos(k\pi x)\,dx\end{aligned} \qquad k=1,\ldots,B,\quad j=0,\ldots,n-1, \tag{5.17}$$

for $k=0$,

$$\overrightarrow{u_j}_0 = \frac{1}{2}\int_{-\infty}^{\infty} u_j(x)\,dx. \tag{5.18}$$

The activation functions of $\sigma_b$ and $\sigma_{trk}$ are $\tanh(x)$. The modes number $B$ in the CPOD-DeepONet(Causality-DeepONet with POD) (5.16) is not necessarily equal to the modes of the input $\overrightarrow{f}$, but needs to be greater than or equal to the number of given modes $M$ of the solution (5.9). The algorithm of the CPOD-DeepONet is shown in Algorithm 4. To obtain the memory efficiency, we will do outer product rather than computing elementwise as shown in the last line of algorithm following the fact that equation (5.9) has separation of variables,. This is one of the cruxes to handle higher dimensional problem.

**Loss function** Given batch size $\mathcal{N}$ for the training process and the total number of the test cases $N$, the loss function is defined as

$$\mathcal{L}oss(\theta) = \frac{1}{\mathcal{N}N_tN_x}\sum_{i=1}^{\mathcal{N}}\sum_{j=1}^{N_t}\sum_{k=1}^{N_x}\left(\mathcal{P}_\theta\left(\overrightarrow{f}\right)(x_k,t_j,\overrightarrow{u_0},\ldots,\overrightarrow{u_{n-1}}) - y_{ijk}\right)^2, \tag{5.19}$$

where $N_t$ is the number of time step and $N_x$ is the number of points in the $x$ direction.

**Algorithm 4** The algorithm of Causality POD-DeepONet

**Input**: $t$, $x$, $[a_0(s), a_1(s), b_1(s), \ldots, a_N(s), b_N(s)]$, $\overrightarrow{u_0}, \ldots, \overrightarrow{u_{n-1}}$
**Output**: $u(x,t)$

1: $y_t \leftarrow \mathcal{G}_\theta([a_0(s), a_1(s), b_1(s), \ldots, a_N(s), b_N(s)])(t)$ ▷ $\mathcal{G}_\theta$ is the Causality-DeepONet. The output $y_t$ has size of [batchsize × Nt × NBx]
2: $y_x \leftarrow [1, \cos(2\pi x), \sin(2\pi x), \ldots \cos(2M\pi x), \sin(2Mx)]$ ▷ The output $y_x$ has size of [batchsize × Nx × NBx]
3: $y_i \leftarrow \sum_{j=0}^{n-1} \overrightarrow{u_j}(t-t_0)^j$ ▷ The initial conditions
4: $u(x,t) \leftarrow$ Einsum ('bxn,btn → bxt', $y_x, y_t$) $+ y_i$ ▷ The Einsum is the Einstein summation convention, ('bxn,btn → bxt') means the index change in the Einstein summation convention.

### 5.3. A recursive DeepPropNet for learning evolution PDE operators

Following the semigroup formulation of evolution PDEs, the DeepPropNet computes block by block along the time direction recursively. The initial conditions for each block together with the forcing functions for that time block will be as input for DeepPropNet as presented in last section. The DeepPropNet is constructed recursively with a CPOD-DeepONet to learn the PDE evolution operator over long time interval.

• **Recursive Formulation of DeepPropNet:** To predict the time series in the time block $[t_0, t_1]$ at the beginning, the Deep Propagator(DeepPropNet) is to learn the mapping

$$\mathcal{P}_\theta : \{[u_0(x), \ldots, u_{n-1}(x)], f(x,s), t_0 \le s \le t\} \mapsto u(x,t), \quad t_0 \le t \le t_1, x \in \mathbb{R}, \tag{5.20}$$

which is exactly a CPOD-DeepONet with to solve the PDE on a small time block $[t_0, t_1]$.

Once the Deep Propagator is learned, the wave field in the next time block $[t_1, t_2]$ could be predicted by the Deep Propagator by using the initial propagator as follows

$$u(x,t) = \mathcal{P}_\theta\Big[\mathcal{P}_\theta(x,t_1), \dot{\mathcal{P}}_\theta(x,t_1), \ldots, \mathcal{P}_\theta^{(n-1)}(x,t_1), f(x,s), t_1 \le s \le t\Big], \tag{5.21}$$

where the initial condition is replaced by the prediction of DeepPropNet at time $t_1$, and $t_1 \leq t \leq t_2, x \in \mathbb{R}$.

The resulting propagator for the time block $[t_1, t_2]$ then will be again to be used to provide the initial condition for $t = t_2$ and the same initial propagator network is applied for prediction $u(x,t)$ in the time period $[t_2, t_3]$.

This procedure can be carried on recursively until the whole time period $[0, T]$ is covered, a global propagator network is thus obtained. Since it is like the initial propagator solver which tracks the waves and propagates with the solutions along the time direction and the propagator itself is a deep neural network, we call it Deep Propagator(DeepPropNet).

This is another crux to solve high dimensional problem. As illustrated in Section 5.1, the input size could explode due to the global dependence of source term, by solving it block by block recursively combining with the memory-efficient trick from separation of variables, learning high dimensional evolution operators is manageable now.

**Loss function** Similar to the loss function in Section 5.2, we will define the loss function of DeepPropNet block by block in time. Given a batch size $\mathcal{N}$ for training process, number of blocks $N_b$ and the total number of the test records $N$, the loss function is defined as

$$\begin{aligned}\mathcal{L}oss(\theta) =& \frac{1}{\mathcal{N} N_t N_x} \sum_{i=1}^{\mathcal{N}} \sum_{n=1}^{N_b} \sum_{j=N_{t,n}}^{N_{t,n+1}} \sum_{k=1}^{N_x} \\ & \left( \mathcal{P}_\theta \left( \overrightarrow{f} \right) \left( x_k, t_j, \mathcal{P}_\theta \left( x_k, t_{N_{t,n}} \right), \dot{\mathcal{P}}_\theta \left( x_k, t_{N_{t,n}} \right), \ldots, \mathcal{P}_\theta^{(n-1)} \left( x_k, t_{N_{t,n}} \right) \right) - y_{ijk} \right)^2,\end{aligned} \tag{5.22}$$

where the $n$-th block starts from $t_{N_{t,n}}$ but ends in $t_{N_{t,n+1}}$, $N_x$ is the number of points on $x$ direction, and $N_t = \sum_{n=1}^{N_b} N_{t,n}$. The inputs $\overrightarrow{f}$ for the initial conditions $\mathcal{P}_\theta^{(i-1)} \left( x_k, t_{N_{t,n}} \right)$ are omitted for simplicity of notations.

5.3.1. The evaluation of initial conditions

To propagate to the next time block from the initial time block, we need to evaluate the initial condition at the next time block, according to the recursive formulation In this section we discuss the method to estimate the initial conditions for every time blocks. Consider a 2nd Order Evolution PDE Operator as an example, $\dot{\mathcal{P}}_\theta(t)$ need to be evaluated, and we notice there is an implicit time dependence in the branch net. The auto differentiation with respect to $t$ in the trunk net is inaccurate, thus we introduce several alternatives.

• **Finite Difference for the Computation of Derivatives** To estimate the first order derivative $\dot{\mathcal{P}}_\theta(t_1)$, we consider the 4th order approximation by finite difference

$$\begin{aligned}\dot{\mathcal{P}}_\theta(t_1) \approx \frac{1}{h}\Big[&\frac{25}{12}\mathcal{P}_\theta(t_1) - 4\mathcal{P}_\theta(t_1-h) \\ &+ 3\mathcal{P}_\theta(t_1-2h) - \frac{4}{3}\mathcal{P}_\theta(t_1-3h) + \frac{1}{4}\mathcal{P}_\theta(t_1-4h)\Big] + O\left(h^4\right).\end{aligned} \tag{5.23}$$

• **Least Squares Approximation for the Computation of Derivatives** On the other hand, we could also estimate the derivatives by using the least squares approximations with polynomials to the predictions. Assume the outputs of the neural networks in one time block are listed as $[\mathcal{P}_\theta(t_0), \mathcal{P}_\theta(t_1), \ldots, \mathcal{P}_\theta(t_M)]$, we could consider using the $q$-th order polynomials

$$\tilde{\mathcal{P}}_q(t) = \sum_{i=0}^{q} \alpha_i t^i, \tag{5.24}$$

to interpolate the predictions by least squares approximation, i.e., minimizing the L2 error

$$L(\alpha_0, \alpha_1, \ldots, \alpha_q) = \sum_{j=k}^{M} \left(\mathcal{P}_\theta(t_j) - \tilde{\mathcal{P}}_q(t_j)\right)^2. \tag{5.25}$$

Note that it is not necessary to set $k$ to be the starting point of the time blocks. Thus, we could estimate the derivatives of predictions at end points $t_M$

$$\frac{\partial \mathcal{P}_\theta}{\partial t}\left(t_M\right) \approx \frac{\partial \tilde{\mathcal{P}}_n}{\partial t}\left(t_M\right) = \sum_{i=1}^{q} i\alpha_i t_M^{i-1}, \tag{5.26}$$

**Loss Function** To obtain higher accuracy of the derivatives for methods of finite difference and least squares approximation, other regularization in loss function is required. We need to apply a procedure as what PINNs did to train the DeepPropNet by using the following residue of the PDEs and the corresponding initial conditions in the next time block as further regularization for the loss function

$$\begin{aligned}
\mathcal{L}_{\text{pinn}} =& \frac{1}{\mathcal{N} N_t N_x} \sum_{i=1}^{\mathcal{N}} \sum_{n=1}^{N_b} \sum_{j=N_{t,n}}^{N_{t,n+1}} \sum_{k=1}^{N_x} \\
& \left(\mathcal{P}_\theta\left(\overrightarrow{f}\right)\left(x_k, t_j, \mathcal{P}_\theta\left(x_k, t_{N_{t,n}}\right), \dot{\mathcal{P}}_\theta\left(x_k, t_{N_{t,n}}\right), \ldots, \mathcal{P}_\theta^{(n-1)}\left(x_k, t_{N_{t,n}}\right)\right) - y_{ijk}\right)^2 \\
& + \beta \int_\Omega \left\| \mathcal{L}_t \mathcal{P}_\theta\left[\overline{u}_{0,n}, \overline{u}_{1,n}, f\right] - \mathcal{L}_x \mathcal{P}_\theta\left[\overline{u}_{0,n}, \overline{u}_{1,n}, f\right] - f \right\|^2 dxdt \\
& + \gamma \int_{x \in \mathbb{R}} \left\| \mathcal{P}_\theta\left[\overline{u}_{0,n}, \overline{u}_{1,n}, f\right](t_0, x) - \overline{u}_{0,n+1} \right\|^2 \\
& \qquad + \left\| \partial_t \mathcal{P}_\theta\left[\overline{u}_{0,n}, \overline{u}_{1,n}, f\right](t_0, x) - \overline{u}_{1,n+1}(x) \right\|^2 dx,
\end{aligned} \tag{5.27}$$

where $\beta$ and $\gamma$ are the penalty terms, $\overline{u}_{0,n}, \overline{u}_{1,n}, n = 1, \ldots N_b - 1$ are the exact initial conditions for the $n$-th time block. The derivative $\mathcal{L}_t \mathcal{P}_\theta$ is obtained by

$$\ddot{\mathcal{P}}_\theta(t) \approx \frac{1}{h^2}\left[-\frac{1}{12}\mathcal{P}_\theta(t-2h) + \frac{4}{3}\mathcal{P}_\theta(t-h) - \frac{5}{2}\mathcal{P}_\theta(t) + \frac{4}{3}\mathcal{P}_\theta(t+h) - \frac{1}{12}\mathcal{P}_\theta(t+2h)\right] + O\left(h^4\right) \tag{5.28}$$

and

$$\frac{\partial^2 \mathcal{P}_\theta}{\partial t^2}\left(t_M\right) \approx \frac{\partial^2 \tilde{\mathcal{P}}_n}{\partial t^2}\left(t_M\right) = \sum_{i=2}^{q} \frac{i(i-1)}{2} \alpha_i t_M^{i-2}. \tag{5.29}$$

for finite difference and least squares approximation, respectively.

• **Recursive Formulation with the Extra Frist Order Term:** Those methods could predict relative accurate first order derivative for noisy data, but we need higher accuracy of prediction to iterate the DeepPropNet. Thus, we consider training an extra DeepPropNet to predict the $\overrightarrow{v}_d$ at the end point of each time block. We could define $u_1(x,t) = u_t(x,t)$ as the DeepPropNet framework, where

$$u_1(x,t) = \{\overrightarrow{v}_{d-1} + \overrightarrow{\sigma}_{br,c}(\sigma_1(f_1), \sigma_2(f_2), \ldots, \sigma_m(f_m)) \odot \overrightarrow{\sigma}_{trk,c}(t)\} \cdot \overrightarrow{\sigma}_{basis}(x). \tag{5.30}$$

**Loss function** To train $u_1(x,t)$, further assumption that $u_t(x,t)$ is known during the training process is needed. Loss for $u_1(x,t)$ is defined as

$$\mathcal{L}\text{oss} = \frac{1}{\mathcal{N} N_t N_x} \sum_{i=1}^{\mathcal{N}} \sum_{n=1}^{N_b} \sum_{j=N_{t,n}}^{N_{t,n+1}} \sum_{k=1}^{N_x} \left(u_1(x_k, t_j) - u_{t,ijk}\right)^2. \tag{5.31}$$

To predict the wave field in the time block $[t_0, t_1]$ at the beginning, the two Deep Propagators are to learn the mapping

$$\mathcal{P}_\theta : \left\{[u_0(x), \ldots u_{n-1}(x)], \overrightarrow{f}(t)\right\} \mapsto u(x,t) \quad t_0 \le t \le t_1, x \in \mathbb{R}. \tag{5.32}$$

and

$$\mathcal{Q}_\theta : \left\{[u_1(x), \ldots u_{n-1}(x)], \overrightarrow{f}(t)\right\} \mapsto u_t(x,t) \quad t_0 \le t \le t_1, x \in \mathbb{R}. \tag{5.33}$$

Once the Deep Propagator is learned, the wave field in the next time block $[t_1, t_2]$ could be predicted by the Deep Propagator by using the initial propagator as follows

$$u(x,t) = \mathcal{P}_\theta\left[\mathcal{P}_\theta(t_1), \mathcal{Q}_\theta(t_1), \overrightarrow{f}(t)\right](x) \quad t_1 \le t \le t_2, x \in \mathbb{R}, \tag{5.34}$$

where the initial conditions are replaced by the predictions of DeepPropNet $\mathcal{P}_\theta$ and $\mathcal{Q}_\theta$ at time $t_1$. In the meantime, the $u_t$ is predicted by

$$u_t(x,t) = \mathcal{Q}_\theta\left[\mathcal{Q}_\theta(t_1), \overrightarrow{f}(t)\right](x) \quad t_1 \le t \le t_2, x \in \mathbb{R}. \tag{5.35}$$

### 5.3.2. Normalization and penalties

In this section, we introduce the 2 strategies utilized in this chapter to improve the performance of our method.

**Normalization of the Inputs and Outputs** According to the results of the paper [41], we set the activation function for all our cases as $\tanh(x)$. However since there are $O\left(JK\frac{N_t}{N_b}\right)$ inputs for both case 1 and case 2 in appendix B, it may cause the outputs of the first layer to be all 1. To avoid the gradient vanishing caused by the magnitude of the inputs, we shrink the inputs to a reasonable scale by normalization,

$$\overrightarrow{f}_s = \frac{\overrightarrow{f}}{J^3K\pi^2}. \tag{5.36}$$

In the meantime, A further normalization to the output could assure the Causality DeepONet with POD captured the temporal feature fast and accurately,

$$\begin{aligned}\mathcal{P}_\theta\left(\overrightarrow{f}_s\right)(x,t,\overrightarrow{u_0},\ldots,\overrightarrow{u_{n-1}}) = \Bigg[&\mathbf{C}(t-t_0)^{n-1}\overrightarrow{\sigma}_{br}\left(\overrightarrow{f}_s\right) \odot \overrightarrow{\sigma}_{trk}(t) \\ &+ \sum_{j=0}^{n-1}\overrightarrow{u_j}(t-t_0)^j\Bigg] \cdot \overrightarrow{\sigma}_{basis}(x),\end{aligned} \tag{5.37}$$

where $\mathbf{C} \sim O(JK)$. It could be noted that this is equivalent to draw the parameters of last layer of CPOD-DeepONet from a distribution with large deviation during the initialization phase of neural network.

**Extra Penalty for training** During backpropagation, the gradient of the loss with respect to the parameters $\theta$ of neural network should satisfy

$$\begin{aligned}\nabla_\theta \mathcal{L} = \mathbf{C}\left(t-t_0\right)^{n-1} \frac{1}{IJ} \sum_{i=1}^{I} \sum_{j=1}^{J} \left[\mathcal{P}_\theta\left(\overrightarrow{f}\right)\left(x_i, t_j, \overrightarrow{u_0}, \ldots, \overrightarrow{u_{n-1}}\right) - u\left(x_i, t_j\right)\right] \\ \times \overrightarrow{\sigma}_{basis}\left(x_i\right) \cdot \nabla_\theta \left[\overrightarrow{\sigma}_{br}\left(\overrightarrow{f}\right) \odot \overrightarrow{\sigma}_{trk}\left(t_j\right)\right].\end{aligned} \tag{5.38}$$

When $n = 2$ and $(t-t_0)^{n-1}$ is small, we should have $\nabla_\theta \mathcal{L} \sim O\left((t-t_0)^{n-1}\right)$ which could cause the poor prediction near the initial conditions. To avoid such issue, we consider applying penalty terms with $O\left(\frac{1}{(t-t_0)^{n-1}}\right)$ in the Loss Function

$$\mathcal{L}(x, u; \theta) = \frac{\mathbf{1}}{\left(\mathbf{t}-\mathbf{t_0}\right)^{\mathbf{n-1}}} \frac{1}{IJ} \sum_{i=1}^{I} \sum_{j=1}^{J} \left[\mathcal{P}_\theta\left(\overrightarrow{f}\right)\left(x_i, t_j, \overrightarrow{u_0}, \ldots, \overrightarrow{u_{n-1}}\right) - u\left(x_i, t_j\right)\right]^2 \tag{5.39}$$

## 5.4. Numerical results

In this section, we present numerical results to demonstrate the effectiveness of the proposed CPOD-DeepONet and DeepPropNet. We first consider the nonhomogeneous heat equation with variable coefficients on free space. Then, we consider the nonhomogeneous wave equation with variable coefficients on free space. The cases are constructed given exact solutions. For each case, the variable coefficients are given, thus the right-hand side are determined by the exact solutions and variable coefficients. The time-dependent coefficients for the right hand side term are the inputs for DeepPropNet and the Causality DeepONet with POD. The specific forms of the exact solutions are given in the Appendix B.

To evaluate the training process, the mean of the relative L2 error is considered. The relative L2 error in a complete epoch is defined as

$$\mathcal{L}_{\text{train}}^{\mathcal{R}} = \frac{1}{B}\sum_{k=1}^{B}\frac{1}{\mathcal{N}}\sum_{i=1}^{\mathcal{N}}\frac{\left\|\mathcal{P}_{\theta^{(k)}}\left(\overrightarrow{f_i}\right)(x,t) - u_{ik}(x,t)\right\|_2}{\|u_i(x,t)\|_2}, \tag{5.40}$$

where $B$ is the number of batches, $\theta^{(k)}$ means the parameters of neural network at $k$-th batch. Similarly, we define the relative L2 error for the testing dataset

$$\mathcal{L}_{\text{test}}^{\mathcal{R}} = \frac{1}{N}\sum_{i=1}^{N}\frac{\left\|\mathcal{P}_{\theta}\left(\overrightarrow{f_i}\right)(x,t) - u_i(x,t)\right\|_2}{\|u_i(x,t)\|_2}. \tag{5.41}$$

Here, $N$ is the total number of test cases. The evolutions of training and testing relative L2 error are plotted in the Appendix B.

### 5.4.1. Results of CPOD-DeepONet

In this section, we present the results of CPOD-DeepONet on both cases mentioned in the Appendix B. The network structures are the same for both cases, with 4 hidden layers and 100 hidden neurons in the trunk net and 4 hidden layers and 100 hidden neurons in the branch net. The input size of the branch net is 800 and the input size of the trunk net is 1. The learning rate considered is $10^{-4}$ for the first 30 epochs and then $5 \times 10^{-5}$ for 30 to 50 epochs, and then $2.5 \times 10^{-6}$ after 50 epochs.

**Results of Case 1** The inputs for the CPOD-DeepONet are the coefficients of basis functions of $x$ in (2.8). The shape of the inputs depends not only on $J, K$, but also on $\nu, \delta$. In this case we assume $J = 3, K = 4, \nu = 2, \delta = 3$ with 800 equal spaced time steps, and there are 800 random sampled points on x-direction for each time step. Figure 5.1 shows the results after training 100 epochs.

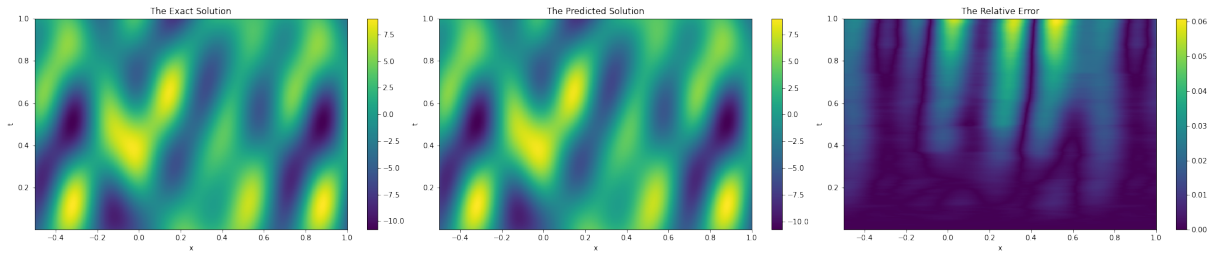



Figure 5.1: The comparison between exact solutions (left) and predictions (middle) of CPOD-DeepONet after training 100 epochs for the nonhomogeneous heat equation with time-independent variable coefficient case, where $J = 3, K = 4, \nu = 2, \delta = 3$. The right plots shows the relative error. 1000 training data will be updated for every 50 epochs. There are 800 time steps. The maximum relative error is 6%.

**Results of Case 2** Similar to case 1, the inputs for the CPOD-DeepONet are the coefficients of basis functions of $x$. The shape of the inputs depends not only on $J, K$, but also on $n$. In this case we assume $J = 3, K = 4$ and $n = 5$ with 800 equal spaced time steps, and there are 800 random sampled points on x-direction for each time step. There are two initial conditions are considered instead. Figure 5.2 shows the results after training 500 epochs.

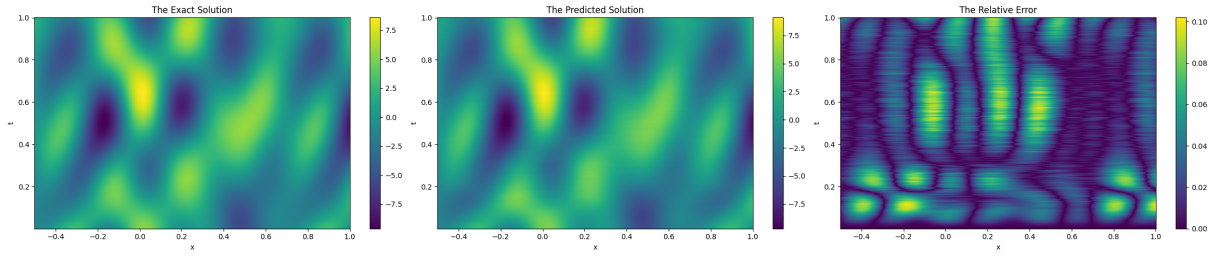



Figure 5.2: The comparison between exact solutions and the predictions of CPOD-DeepONet after training 500 epochs for the time-independent variable coefficient wave equation case, where $J = 3, K = 4, n = 5$. 1000 training data will be updated for every 50 epochs. There are 800 time steps. The maximum relative error is 11%. Left: the exact solution. Middle: the predicted solution of CPOD-DeepONet. Right: the relative error.

### 5.4.2. Results of Deep Propagator

In this section, we present the results of Deep Propagator for both the nonhomogeneous heat equation with variable coefficients and the nonhomogeneous wave equation with variable coefficients mentioned in Appendix B. The network structures are the same for both cases, with 4 hidden layers and 100 hidden neurons in the trunk net and 4 hidden layers and 100 hidden neurons in the branch net. The number of blocks considered is 8, thus the input size of the branch net is 100 and the input size of the trunk net is 1. The learning rate considered is $10^{-4}$ for the first 30 epochs and then $5 \times 10^{-5}$ for 30 to 50 epochs, and then $2.5 \times 10^{-6}$ after 50 epochs.

**Results of Case 1** The predictions of intitial conditions are exactly the last predictions at one time block for this case. The settings for $J, K, \delta, \nu$ and the number of points on x-direction are the same as in Section 5.4.1. Figure 5.3 shows the comparison between predictions from Deep Propagator after training 100 epochs and exact solutions.

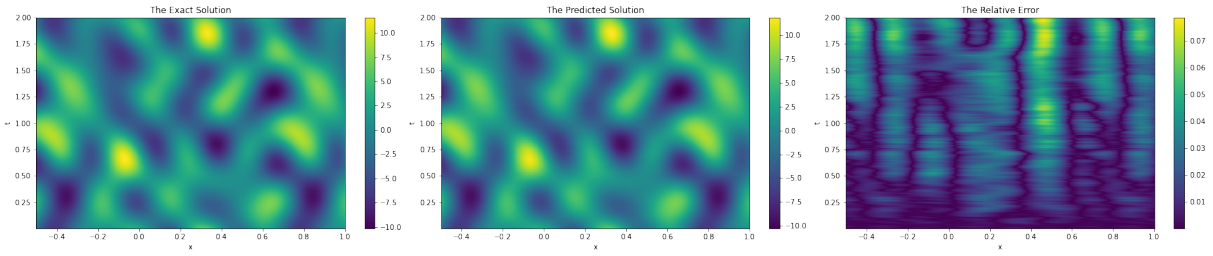



Figure 5.3: The comparison of exact solutions with predictions of Deep Propagator after training 100 epochs, given data of the first time block for the time-independent variable coefficient nonhomogeneous heat equation case, where $J = 3, K = 4, \nu = 2, \delta = 3$. 1000 training data will be updated for every 50 epochs. The DeepPropagator propagates 8 time blocks, the time scale for each time blocks is 0.25. There are 200 time steps for each time block. The current DeepPropagator contains 10 modes. The maximum relative error is 8%. Left: the exact solution. Middle: the predicted solution of Deep Propagator. Right: the relative error.

Next, we show the comparison of predictions for different methods to compute the initial conditions for case 2. The settings for $J, K, n$ and the number of points on x-direction are

the same as in Section 5.4.1. The DeepPropNet to approximate $u(x,t)$ has same shape for three different scenarios.

**Deep Propagator with Finite Difference for Initial conditions** The predictions of initial conditions are obtained by the finite difference method mentioned in Section 5.3.1. Figure 5.4 shows the comparison between predictions from Deep Propagator after training 1000 epochs and exact solutions. It could be noted that the training data are all the 8 time blocks in this case.

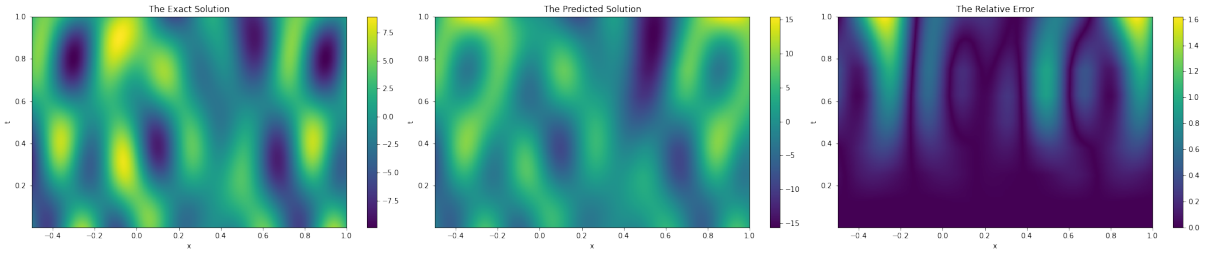



Figure 5.4: The comparison of exact solutions and predictions after training 1000 epochs, given data of the 8 time blocks for the time-independent variable coefficient wave equation case, where $J = 3, K = 4, n = 5$. The initial conditions are computed by finite difference. 1000 training data will be updated for every 50 epochs. The DeepPropagator propagates 8 time blocks, the time scale for each time blocks is 0.125. There are 100 time steps for each time block. Left: the exact solution. Middle: the predicted solution of Deep Propagator with respect to $u$. Right: the relative error. The maximum relative error for this case is 160%.

**Deep Propagator with Least Squares Interpolations for Initial conditions** The predictions of initial conditions are obtained by interpolations with respect to the predictions of $u$ for each time block and $\dot{u}$ at the last time step for current time block is obtained by taking derivatives with respect to $t$ for the interpolations in this case, as discussed in Section 5.4.1. Figure 5.5(middle) shows the predictions of Deep Propagator after training 1000 epochs. It could be noted the training data in all 8 time blocks are fed to train deep propagator in this case.

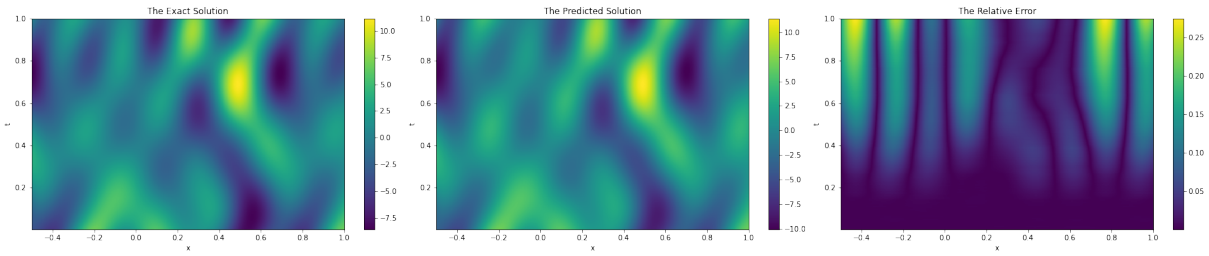



Figure 5.5: The comparison of exact solutions and predictions after training 1000 epochs, given data of the 8 time blocks for the time-independent variable coefficient wave equation case, where $J = 3, K = 4, n = 5$. The initial conditions are computed by least squares interpolations. 1000 training data will be updated for every 50 epochs. The DeepPropagator propagates 8 time blocks, the time scale for each time blocks is 0.125. There are 100 time steps for each time block. Left: the exact solution. Middle: the predicted solution of Deep Propagator with respect to $u$. Right: the relative error. The maximum relative error for this case is 30%.

**Deep Propagator with Extra DeepPropNet for Initial conditions** The predictions of initial conditions are exactly the predictions of $u$ at the last time step and the predictions of $\dot{u}$ at the last time step for current time block, respectively. Figure 5.6 shows the results of training 1000 epochs. It could be noted that the training data is only the first time block in this case.

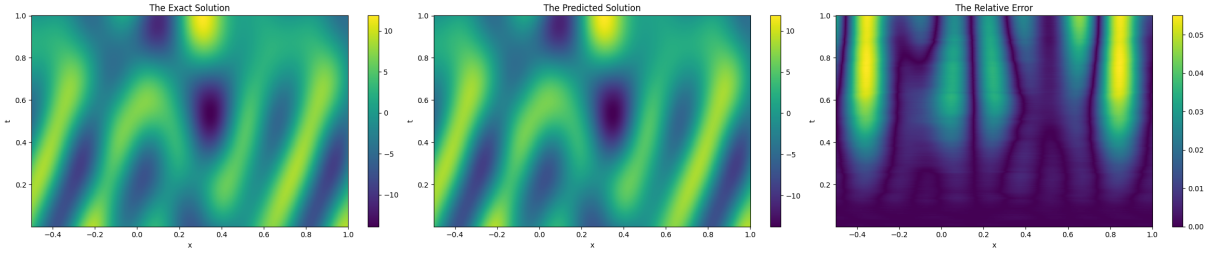



Figure 5.6: The comparison of exact solutions and predictions after training 1000 epochs, given data of the 8 time blocks for the time-independent variable coefficient wave equation case, where $J = 3, K = 4, n = 5$. The initial conditions are obtained by the predictions of two deep propagators. 1000 training data will be updated for every 50 epochs. The DeepPropagator propagates 8 time blocks, the time scale for each time blocks is 0.125. There are 100 time steps for each time block. Left: the exact solution. Middle: the predicted solution of Deep Propagator with respect to $u$. Right: the relative error. The maximum relative error for this case is 6%.

# Chapter 6

## Conclusions

In this thesis, we presented a method to accelerate the convergence of neural network learning oscillatory solutions, several linearized learning schemes for neural network solving stationary Navier-Stokes equations, a variant of DeepONet for time-dependent problems, and a corresponding time-causal framework for solving time-dependent partial differential equations.

In Chapter 2, we have proposed a phase shift DNN to learn high frequency information by using frequency shifts to convert the high frequency learning to low frequency one. As shown by various numerical tests, this approach increases dramatically the capability of the DNN as a viable tool for approximating high frequency functions and solutions of high frequency wave differential and integral equations in inhomogeneous media.

The optimization problem with the training of DNNs is complex and not much understood during the search of parameter spaces of the DNNs for a local or global minima. The specific structure of the proposed PhaseDNN seems to provide a favorable parameter structure (inspired by the mathematical or physical properties of the solutions), within which the optimization can be carried out much more efficiently than the common fully connected DNNs can provide. Moreover, our numerical results also show that the PhaseDNN with integral equation formulation of the high frequency wave problems gives better accuracy than that with differential equations. All these issues will be the subject of future theoretical analysis of the PhaseDNN. Also for future work, we will further develop the PhaseDNN to handle the high dimensionality problems from random inhomogeneous media in wave prop-

agation. The other interesting extension is to expect the neural network to capture the frequency information automatically.

In Chapter 3, we have proposed four linearized learning schemes to solve the stationary highly oscillatory Navier-Stokes flows with multiscale deep neural networks and showed the acceleration of convergence of the schemes are substantial, which demonstrate the capability of the multiscale deep neural networks and the effectiveness of the linearized schemes to solve the nonlinear Navier-Stokes equations. These schemes shed some light on the practical applications of neural network machine learning algorithms to the nonlinear equations, which are time-consuming using traditional finite element methods. The deep neural network based methods offer an alternative that doesn't require meshes and has no need to solve large-scale linear systems, as in traditional numerical methods.

There are more works to be done for these linearized learning methods, among them the most important is to understand the convergence property of these schemes. The applications of these schemes to other nonlinear PDEs will also be considered. Another challenging work is to consider the time dependent Navier-Stokes equation, which will be explored in a future work. Additionally, exploring the extension of the presented method to the 3D Navier-Stokes equation will be a topic for future research.

In Chapter 4, we have studied how to improve the accuracy of DeepONet for causal oscillatory linear dynamical systems. Two new variants of DeepONet, the multi-scale DeepONet and the Causality-DeepONet are proposed. As an application, we considered the problem of learning the mapping between earthquake ground accelerations and building's causal responses, which are both highly oscillatory. In the multi-scale DeepONet, multi-scale neural networks are used in the trunk net. Meanwhile, the Causality-DeepONet includes both causality and convolution as specific domain knowledge in its design. Though the multi-scale DeepONet improved the training of the seismic response operator, it failed to give satisfactory prediction results in the test cases. However, the Causality-DeepONet is able

to provide accurate predictions in the test cases as well. We have also studied the effect of the size of networks, the number of training samples, and the type of activation functions on the accuracy of prediction of responses using Causality-DeepONet. It is found that the proposed Causality-DeepONet can provide good accuracy in the prediction of response of the problem considered.

For future work, the Causality-DeepONet for nonlinear problems such as nonlinear dynamics, nonlinear electrical circuits etc, may be considered. Also future studies should include establishing a solid mathematical foundation for the Causality-DeepONet by extending the work of [11] to the proposed framework of the Causality-DeepONet.

In Chapter 5, we proposed the recursive way to construct DeepPropNet - a DNN propagator for evolution system over large time period by using a single building block propagator over a small time period, thus reducing the overall complexity and size of the neural network required. For the design of the DeepPropNet, we also extended the Causality DeepONet with POD with specific basis to alleviate the memory burden for large spatial variables. By separately handling the spatial and temporal domain, we gain not only the memory efficiency but also the training boost. The preliminary numerical results have shown the feasibility of this recursive DeepPropNet in predicting the time evolution of wave propagations.

The proposed DeepPropNet here is based on a supervised learning approach where the data can be generated by a separate numerical method or observation data or analytical solution when available. In theory, we could also use an unsupervised learning procedure to train the DeepPropNet by using the following residual of the PDEs as the loss function.

$$\mathcal{L}\text{oss} = \int_{\Omega} || \mathcal{L}_t P_\theta [u_0, v_0, f] - \mathcal{L}_x P_\theta [u_0, v_0, f] - f ||^2 \, dxdt. \tag{6.1}$$

It is also natural to extend this framework to learn high frequency problems, since the spatial oscillating terms will be handled explicitly by the basis, but the temporal oscillating terms could be handled by the Causality DeepONet, based on our experience in the work [41].

Future work will be conducted on more complex evolution systems, the unsupervised training, training over partial time domain, highly oscillating problems and higher dimensional problems as well as initial boundary value problems.

# Appendix A

## Appendix for Causality DeepONet

**A.1. Additional tables** In Table A.1 and A.2, we show the statistical properties of the training and testing dataset. The total number of testing dataset considered is 44 earthquake ground accelerations. These test data are considered for all the numerical examples. It is also important to note that in the case of training, dataset Train-I, Train-II and Train-III are completely different dataset. Training dataset Train-I is not included in training dataset Train-II, similarly, training dataset Train-I and Train-II are not included in training dataset Train-III. However, training dataset Train-IV includes training dataset Train-III and training dataset Train-V includes training dataset Train-IV as well. The training dataset Train-VI includes all the training data I to V. The training dataset Train-VI is used for the training of DeepONet, DeepONet with Gaussian Normalization, POD-DeepONet, MSDeepONet. The training dataset Train-I to Train-VI are the data used for the discussion in Section 4.4.2.

Table A.1: Properties of the earthquake records considered for training and testing of neural networks.

| Case | | Test | Train-I | Train-II | Train-III |
|---|---|---|---|---|---|
| Samples | | 44 | 7 | 8 | 10 |
| Mag. | Min | 4.30 | 6.30 | 4.90 | 4.92 |
| | Max | 7.90 | 7.62 | 7.62 | 7.62 |
| | Mean | 6.58 | 7.29 | 6.48 | 7.07 |
| | SD | 0.82 | 0.46 | 1.04 | 0.83 |
| PGA $\max\|\ddot{u}_g\|$ | Min | 0.00103 | 0.00966 | 0.00499 | 0.00245 |
| | Max | 0.35726 | 0.14425 | 0.11854 | 0.31313 |
| | Mean | 0.05298 | 0.05915 | 0.05262 | 0.10973 |
| | SD | 0.06428 | 0.04255 | 0.03863 | 0.11792 |
| Max $\ddot{u}_g$ | Min | 0.00103 | 0.00966 | 0.00498 | 0.00245 |
| | Max | 0.35726 | 0.14425 | 0.09805 | 0.30114 |
| | Mean | 0.04883 | 0.05610 | 0.04460 | 0.10440 |
| | SD | 0.06199 | 0.04148 | 0.03279 | 0.11204 |
| Min $\ddot{u}_g$ | Min | -0.27870 | -0.13713 | -0.11854 | -0.31313 |
| | Max | -0.00095 | -0.00671 | -0.00499 | -0.00233 |
| | Mean | -0.04820 | -0.05413 | -0.05262 | -0.09849 |
| | SD | 0.05502 | 0.04177 | 0.03863 | 0.10559 |
| Energy $\|\ddot{u}_g\|^2$ | Min | 0.00020 | 0.00632 | 0.00188 | 0.00043 |
| | Max | 3.22599 | 2.47209 | 2.05953 | 7.31542 |
| | Mean | 0.38912 | 0.57996 | 0.44703 | 1.66238 |
| | SD | 0.71029 | 0.81379 | 0.65613 | 2.40233 |

Table A.2: Properties of the earthquake records considered for training and testing of neural networks.

| Case | | Train-IV | Train-V | Train-VI |
|---|---|---|---|---|
| Samples | | 20 | 50 | 100 |
| Mag. | Min | 4.90 | 4.90 | 4.70 |
| | Max | 7.62 | 7.62 | 7.62 |
| | Mean | 6.85 | 7.01 | 7.10 |
| | SD | 0.94 | 0.83 | 0.82 |
| PGA $\max\|\ddot{u}_g\|$ | Min | 0.00245 | 0.00119 | 0.00119 |
| | Max | 0.31313 | 0.31313 | 0.31313 |
| | Mean | 0.08027 | 0.06972 | 0.07277 |
| | SD | 0.09269 | 0.07151 | 0.06338 |
| Max $\ddot{u}_g$ | Min | 0.00245 | 0.00118 | 0.00118 |
| | Max | 0.30114 | 0.30114 | 0.30114 |
| | Mean | 0.07439 | 0.06440 | 0.06661 |
| | SD | 0.08818 | 0.06795 | 0.06008 |
| Min $\ddot{u}_g$ | Min | -0.31313 | -0.31313 | -0.31313 |
| | Max | -0.00233 | -0.00119 | -0.00112 |
| | Mean | -0.07365 | -0.06312 | -0.06791 |
| | SD | 0.08314 | 0.06391 | 0.05899 |
| Energy $\|\ddot{u}_g\|^2$ | Min | 0.00043 | 0.00016 | 0.00016 |
| | Max | 7.31542 | 7.31542 | 7.31542 |
| | Mean | 1.03450 | 0.77436 | 0.82185 |
| | SD | 1.86043 | 1.34532 | 1.22943 |

## A.2. Additional figures

### A.2.1. A typical ground acceleration due to earthquake before and after processing

Figure A.1(a)-(b) show ground acceleration due to earthquake 14383980 before and after processing. The ground acceleration record are recorded with $\delta t = 0.005$ sec. The signal is passed through a Butterworth filter. It is also important to note that the length of the

signal is 200 second which is much higher than the other signal considered. Thus, we have not considered initial 23.56 second acceleration, and we have not considered the earthquake after 103.56 sec. The acceleration removed accounts for 0.7% of total energy.

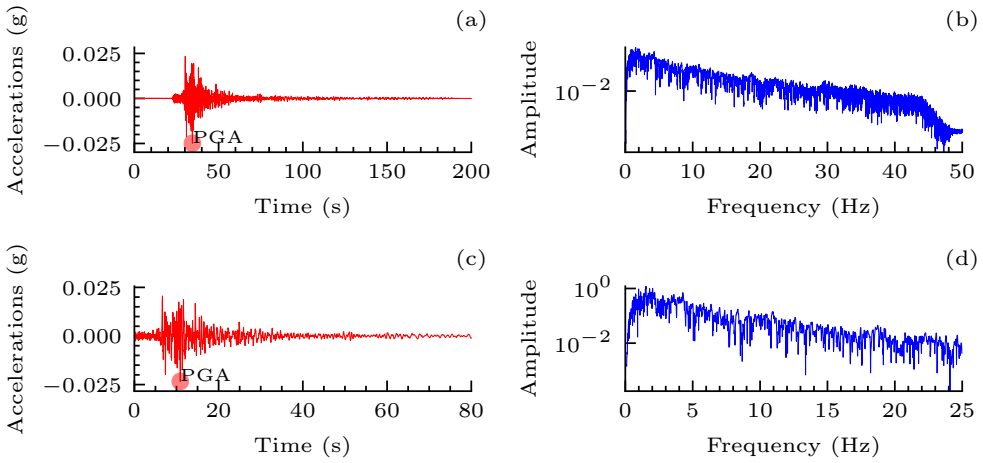



Figure A.1: **Ground Acceleration:** The ground acceleration due to 14383980 earthquake recorded at station North Hollywood, 2008 (a) Time history of the acceleration. (b) Frequency spectrum. (c) The resampled acceleration. (d) The frequency spectrum of resampled acceleration.

A.2.2. Additional results for numerical study for DeepONet

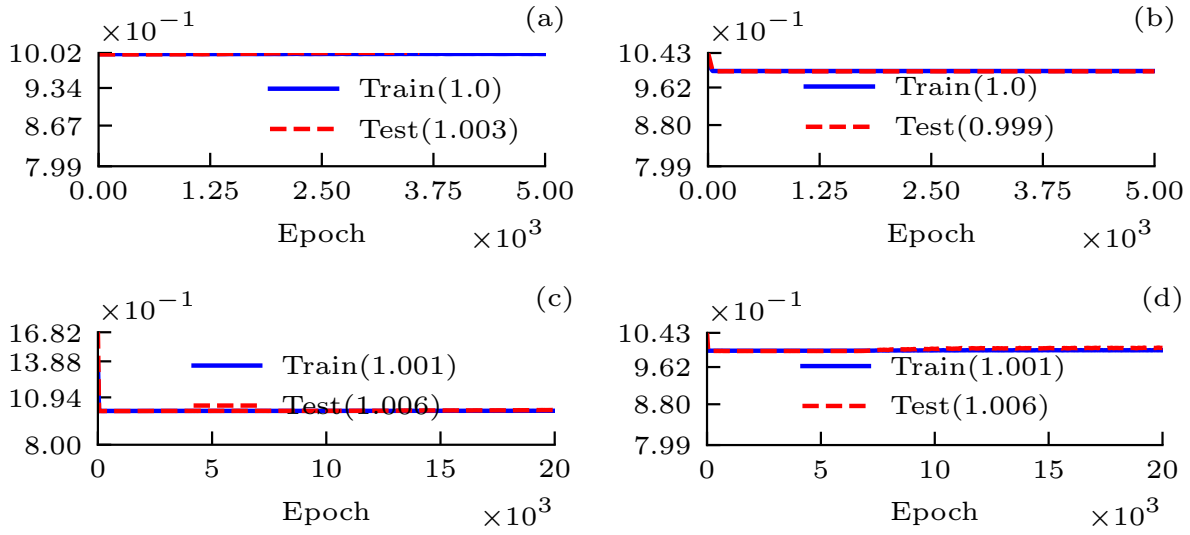



Figure A.2: **Relative $L^2$ Error for DeepONet:** All of these cases are with loss function (4.25). The activation function for all the cases are ReLU($x$). To avoid overfitting, we consider using dropout rate 0.01 for the branch net and with $3 \times 10^{-5}$ $L^2$ regularization. The network size is [4000]-[200]×3-[200] and [1]-[200]×3-[200] for the branch, the trunk net, respectively. (a) In this case, $t$ is re-scaled to $[0, 1]$, the learning rate is $10^{-4}$ in the first 1000 epochs, then $10^{-5}$ in the 1000 to 3000 epochs, at last $10^{-6}$ for the 3000 to 5000 epochs. (b) In this case, the learning rate are set to be $10^{-4}$ for all the 5000 epochs. (c) This is the case with same setting as case-3 in Table 4.1 but training up to 20000 epochs with same learning rate $10^{-4}$. (d) This is the case with same setting as case-6 in Table 4.1 but training up to 20000 epochs with same learning rate $10^{-4}$.

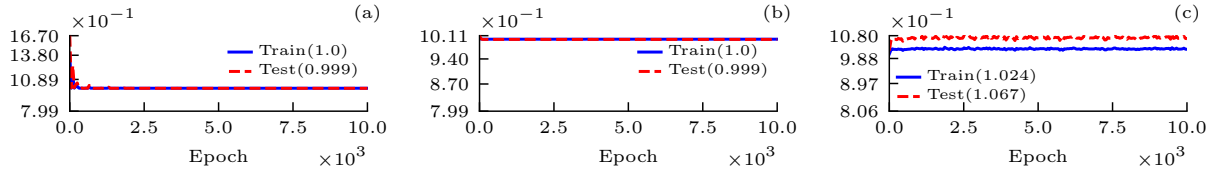



Figure A.3: **Relative $L^2$ Error for DeepONet with Different Activation Functions:** All of these cases are with loss function (4.25) and are trained 10000 epochs. To avoid overfitting, we consider using dropout rate 0.01 for the branch net and with $3 \times 10^{-5}$ $L^2$ regularization. The learning rate is $10^{-4}$ for all the cases. The network size is [4000]-[200]×3-[200] and [1]-[200]×3-[200] for the branch, the trunk net, respectively. (a) with Sigmoid($x$) as activation function. (b) with $\tanh(x)$ as activation function. (c) with $\sin(x)$ as activation function.

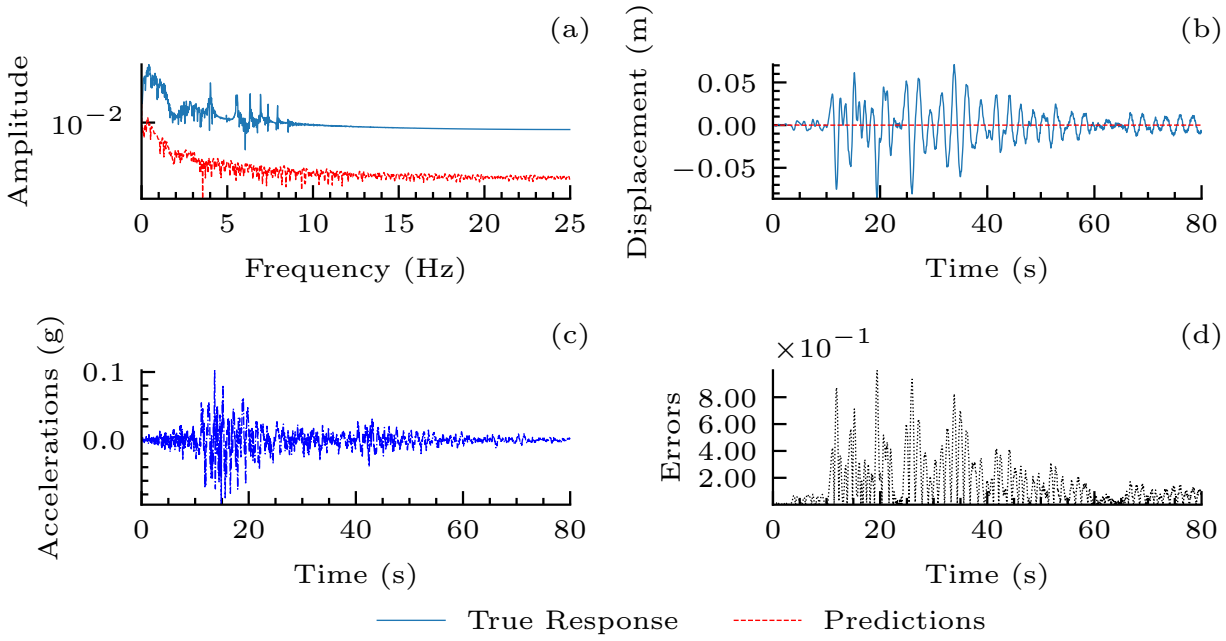



Figure A.4: **One of the Training Samples with Predictions of DeepONet:** One of the training samples of DeepONet for Case-4 in Table. 4.1 with Loss (4.25). (a) The Amplitude of the prediction and true response in Fourier Domain, (b) The prediction and true response, (c) The corresponding input signals, (d) The relative error (4.27).

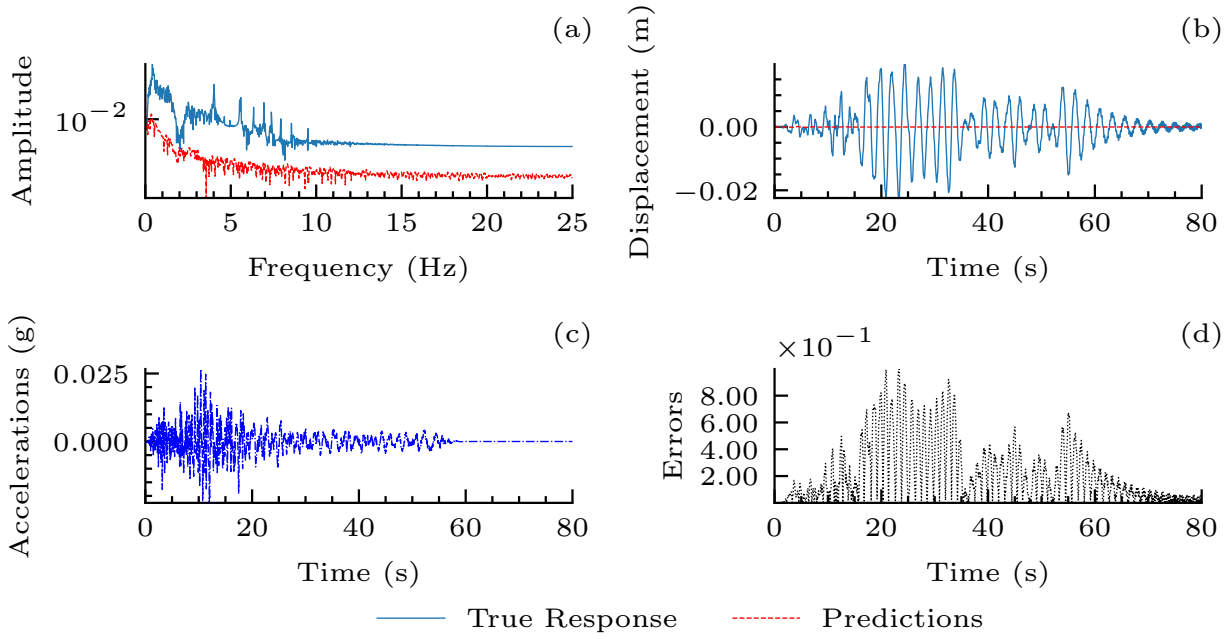



Figure A.5: **The Best Case of the Predictions of DeepONet:** The best predictions for the DeepONet for testing cases for Case-4 in Table. 4.1 with Loss (4.25). (a) The Amplitude of the prediction and true response in Fourier Domain, (b) The prediction and true response, (c) The corresponding input signals, (d) The relative error (4.27).

### A.2.3. Additional results for numerical study for POD-DeepONet

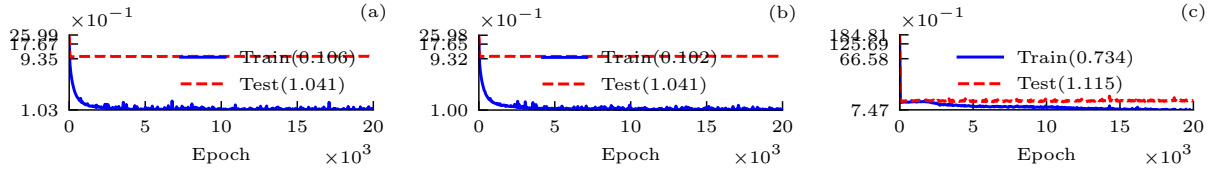



Figure A.6: **Relative $L^2$ Error for POD-DeepONet with Different Activation Functions:** All of these cases are with loss function (4.25) and are trained 20000 epochs. To avoid overfitting, we consider using $10^{-6}$ $L^2$ regularization for the net. The learning rate is $10^{-4}$ for all the cases. The network size is [4000]-[200]×3-[100]. (a) with $\sin(x)$ as activation function. (b) with $\tanh(x)$ as activation function. (c) with Sigmoid$(x)$ as activation function.

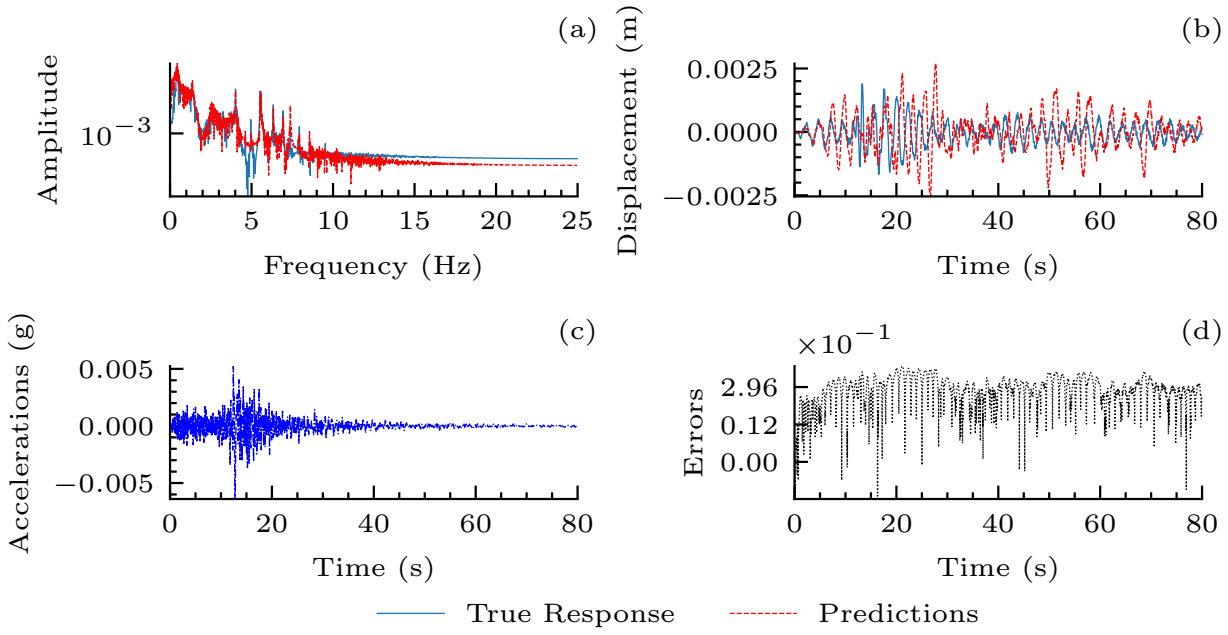



Figure A.7: **One of the Training Samples with Predictions of POD-DeepONet:** One of the training samples of POD-DeepONet with Loss (4.25). (a) The Amplitude of the prediction and true response in Fourier Domain, (b) The prediction and true response, (c) The corresponding input signals, (d) The relative error (4.27).

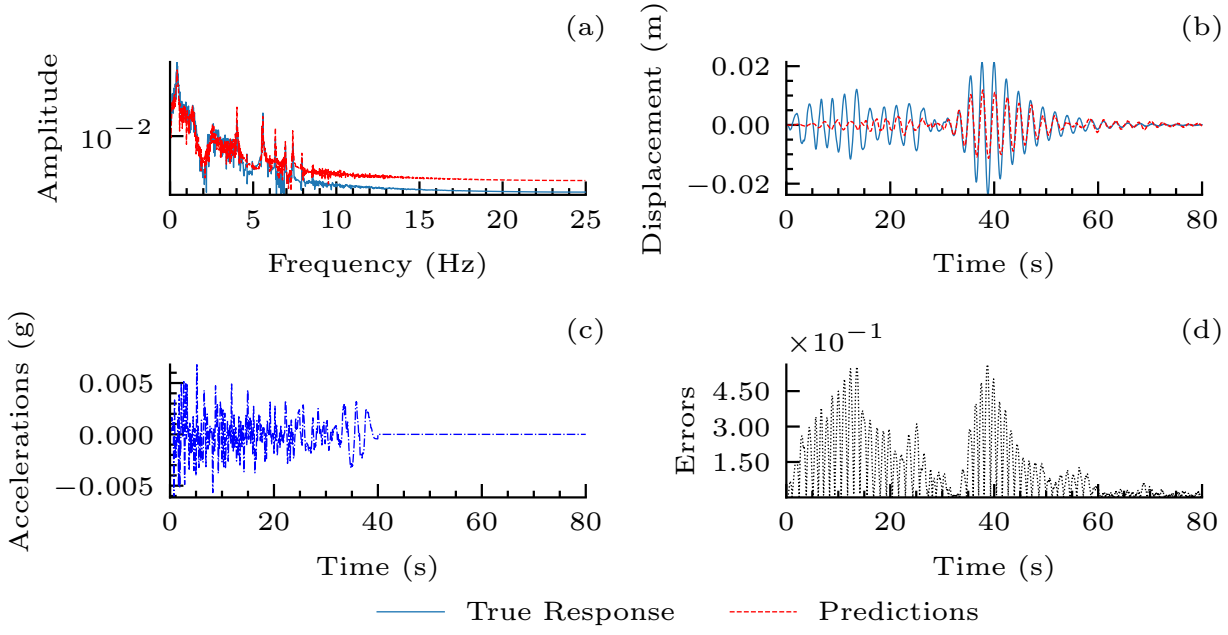



Figure A.8: **The Best Case of the Predictions of POD-DeepONet:** The best predictions for the POD-DeepONet for testing cases with Loss (4.25). (a) The Amplitude of the prediction and true response in Fourier Domain, (b) The prediction and true response, (c) The corresponding input signals, (d) The relative error (4.27).

A.2.4. Additional results for numerical study for Multi-scale DeepONet

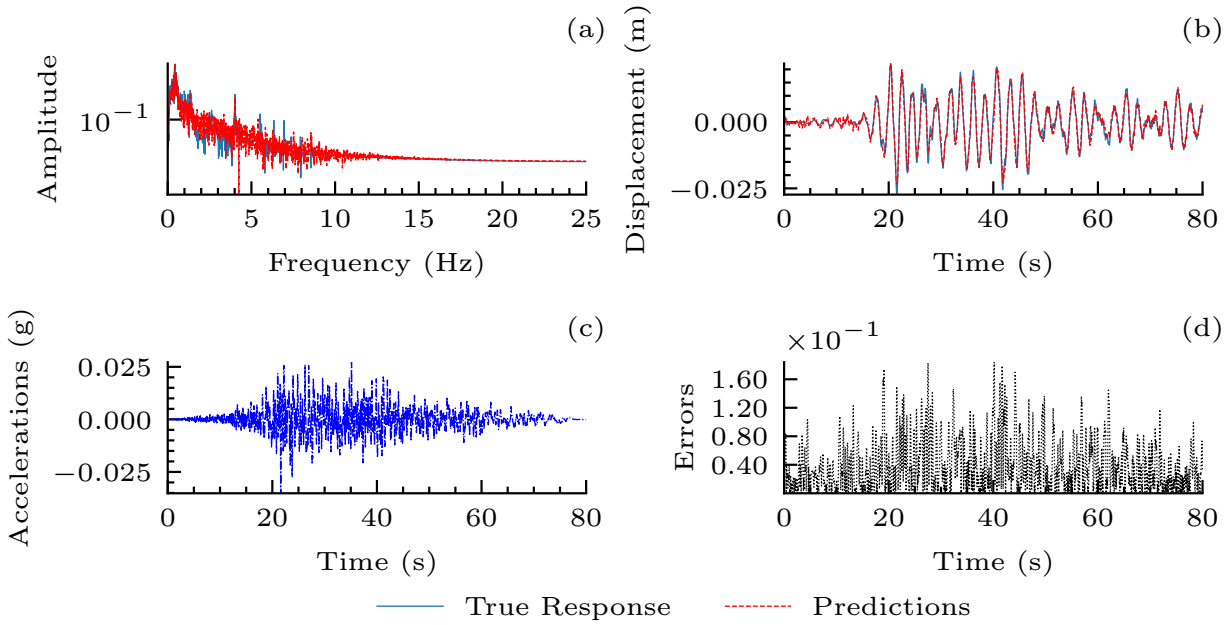



Figure A.9: **One of the Training Samples with Predictions of MS-DeepONet:** One of the training samples of MS-DeepONet with Loss (4.25). (a) The Amplitude of the prediction and true response in Fourier Domain, (b) The prediction and true response, (c) The corresponding input signals, (d) The relative error (4.27).

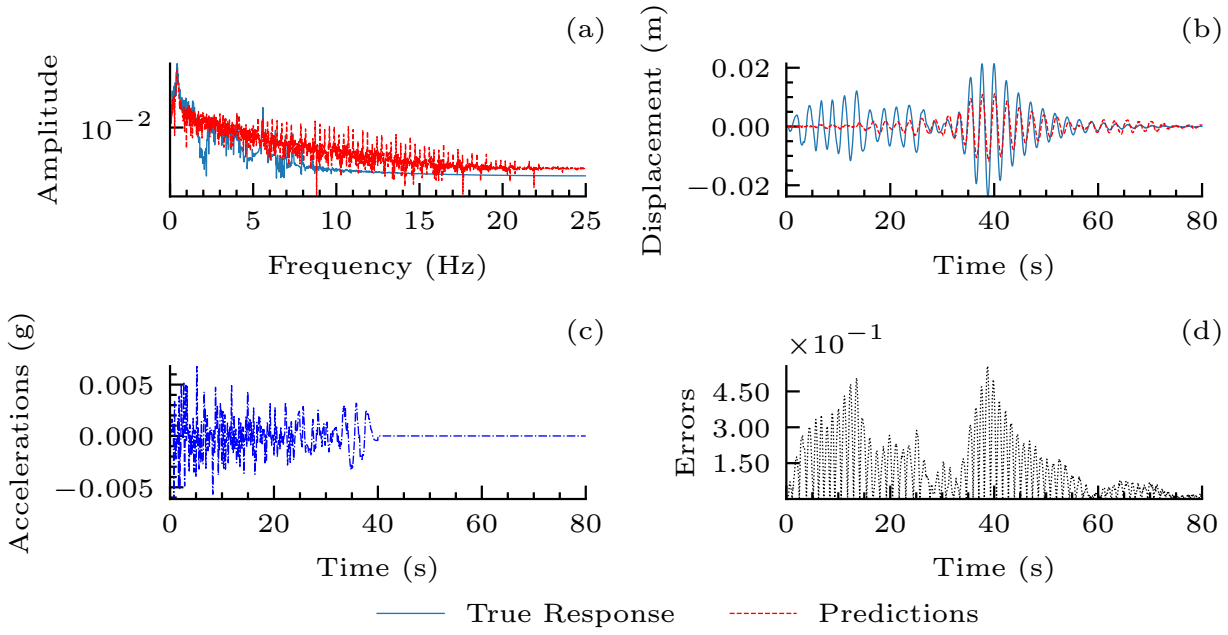



Figure A.10: **The Best Case of the Predictions of MS-DeepONet:** The best predictions for the MS-DeepONet for testing cases with Loss (4.25). (a) The Amplitude of the prediction and true response in Fourier Domain, (b) The prediction and true response, (c) The corresponding input signals, (d) The relative error (4.27).

## Appendix B

The cases considered in the experiments of DeepPropNet

**Case 1: Time-independent Variable Coefficinet Case** We consider the nonhomogeneous heat equation with variable coefficients on free space for the DeepPropNet.

$$\frac{\partial C\left(x,t\right)}{\partial t}=D\left(x\right)\frac{\partial^{2}C}{\partial x^{2}}-V\left(x\right)\frac{\partial C}{\partial x}+f\left(x,t\right),t\in\left[0,T\right] \tag{2.1}$$

with initial boundary condition.

The exact solution of the problem will be assumed as

$$\begin{aligned}C\left(x,t\right)=\sum_{j=1}^{J}\Bigg\{&\left[\sum_{k=1}^{K}\left(a_{jk}\cos\pi kt+b_{jk}\sin\pi kt\right)\right]\cos\pi jx\\&+\left[\sum_{k=1}^{K}\left(c_{jk}\cos\pi kt+d_{jk}\sin\pi kt\right)\right]\sin\pi jx\Bigg\}.\end{aligned} \tag{2.2}$$

Let $D(x) = \sin \delta\pi x$ and $V(x) = \cos \nu\pi x$, where $\delta, \nu \in \mathbb{N}$. We could derive the corresponding right hand side as

$$\begin{aligned} f(x,t) = \sum_{j=1}^{J} \Bigg\{ & \left[ \sum_{k=1}^{K} \left( -a_{jk} k\pi \sin \pi kt + b_{jk} k\pi \cos \pi kt \right) \right] \cos \pi jx \\ & + \left[ \sum_{k=1}^{K} \left( -c_{jk} k\pi \sin \pi kt + d_{jk} k\pi \cos \pi kt \right) \right] \sin \pi jx \\ & + \left[ \sum_{k=1}^{K} \left( a_{jk} \cos \pi kt + b_{jk} \sin \pi kt \right) \right] (j\pi)^2 \cos \pi jx \sin \delta\pi x \\ & + \left[ \sum_{k=1}^{K} \left( c_{jk} \cos \pi kt + d_{jk} \sin \pi kt \right) \right] (j\pi)^2 \sin \pi jx \sin \delta\pi x \\ & - \left[ \sum_{k=1}^{K} \left( a_{jk} \cos \pi kt + b_{jk} \sin \pi kt \right) \right] j\pi \sin \pi jx \cos \nu\pi x \\ & + \left[ \sum_{k=1}^{K} \left( c_{jk} \cos \pi kt + d_{jk} \sin \pi kt \right) \right] j\pi \cos \pi jx \cos \nu\pi x \Bigg\}. \end{aligned} \tag{2.3}$$

The corresponding initial condition is

$$\begin{aligned} C(x,t) = \sum_{j=1}^{J} \Bigg\{ & \sum_{k=1}^{K} (a_{jk}) \cos \pi jx \\ & + \sum_{k=1}^{K} (c_{jk}) \sin \pi jx \Bigg\}. \end{aligned} \tag{2.4}$$

**Case 2: Time-independent Variable Coefficient Case** In this case, we consider the inhomogeneous scalar wave (d'Alembert) equation

$$\frac{\partial^2 u}{\partial t^2} - c^2(x,t) \frac{\partial^2 u}{\partial x^2} = f(x,t), \quad x \in \mathbb{R}, \quad t \geq 0, \tag{2.5}$$

with the source term $f$ that is compactly supported on a bounded space-time domain $Q = S \times [0,T]$ where $S \subset \mathbb{R}$. This means that the source term $f(x,t)$ differs from zero only on $S$

and operates for the limited time interval $[0, T]$. The wave speed $c(x,t)$ is a function of $x$

$$c(x) = \sqrt{\sin(n\pi x) + 1}, \tag{2.6}$$

and the exact solution is given by

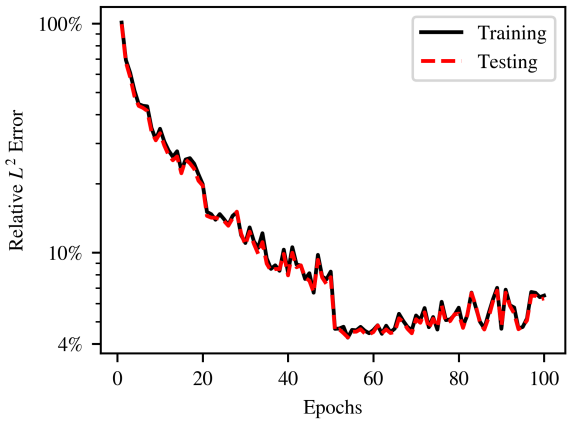
$$\begin{aligned} u(x,t) = \sum_{j=1}^{J} \Bigg\{ & \left[ \sum_{k=1}^{K} \left( a_{jk} \cos \pi k t + b_{jk} \sin \pi k t \right) \right] \cos \pi j x \\ & + \left[ \sum_{k=1}^{K} \left( c_{jk} \cos \pi k t + d_{jk} \sin \pi k t \right) \right] \sin \pi j x \Bigg\}, \end{aligned} \tag{2.7}$$

thus, the corresponding right hand side will be

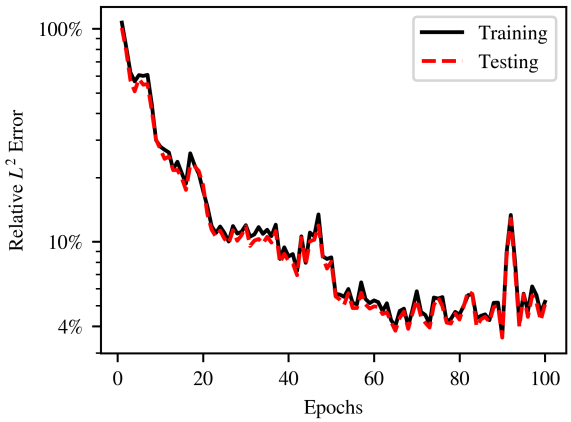
$$\begin{aligned} f(x,t) = & \sum_{j=1}^{J} \sum_{k=1}^{K} \left( j^2 - k^2 \right) \left( a_{jk} \cos \pi k t + b_{jk} \sin \pi k t \right) \cos \pi j x \\ & + \sum_{j=1}^{J} \sum_{k=1}^{K} \left( j^2 - k^2 \right) \left( c_{jk} \cos \pi k t + d_{jk} \sin \pi k t \right) \sin \pi j x \\ & + \sum_{j=1}^{J} \sum_{k=1}^{K} \frac{j^2}{2} \pi^2 \left( a_{jk} \cos \pi k t + b_{jk} \sin \pi k t \right) \sin (n+j) \pi x \\ & - \sum_{j=1}^{J} \sum_{k=1}^{K} \frac{j^2}{2} \pi^2 \left( c_{jk} \cos \pi k t + d_{jk} \sin \pi k t \right) \cos (n+j) \pi x \\ & + \sum_{j=1}^{J} \sum_{k=1}^{K} \frac{j^2}{2} \pi^2 \left( a_{jk} \cos \pi k t + b_{jk} \sin \pi k t \right) \sin (n-j) \pi x \\ & + \sum_{j=1}^{J} \sum_{k=1}^{K} \frac{j^2}{2} \pi^2 \left( c_{jk} \cos \pi k t + d_{jk} \sin \pi k t \right) \cos (n-j) \pi x. \end{aligned} \tag{2.8}$$

# Appendix C

Evolutions of relative $L^2$ errors for Deep Propagator

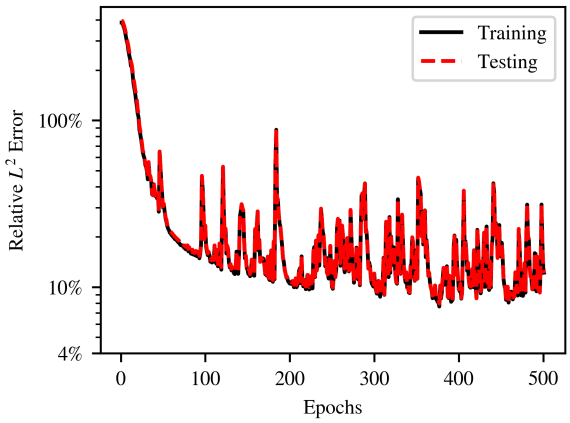



(a) The evolution of relative $L^2$ errors for the training dataset(solid line) and testing dataset (dashed line) when training Causality DeepONet with POD.

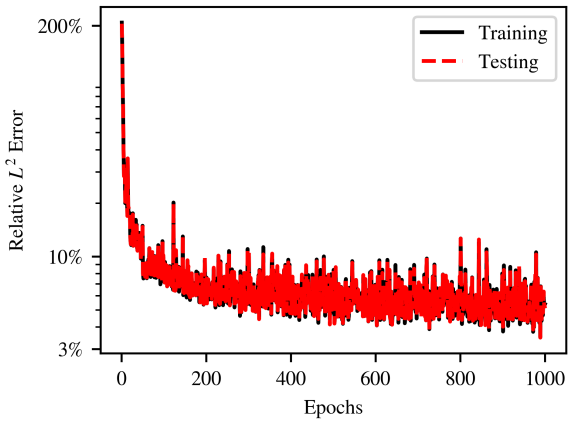



(b) The evolution of relative $L^2$ errors for the training dataset(solid line) and testing dataset (dashed line) when training Deep Propagator.

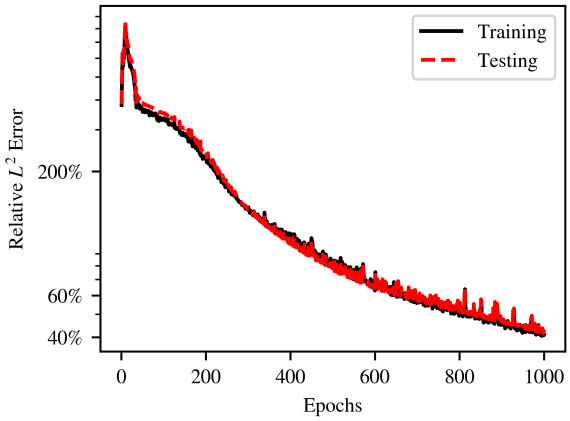
Figure C.1: The evolution of relative $L^2$ errors for Causality DeepONet with POD and Deep Propagator solving the nonhomogeneous heat equation.



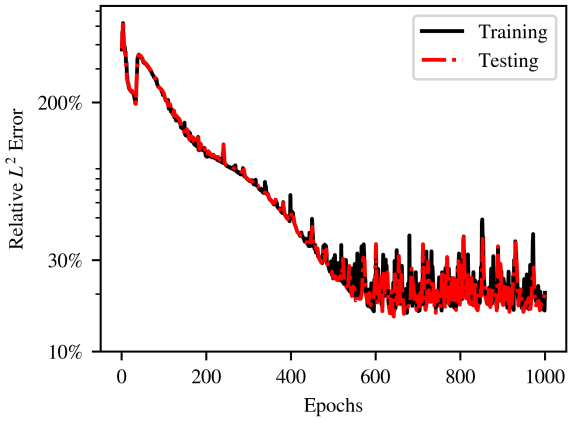

(a) The evolution of relative $L^2$ errors for the training dataset(solid line) and testing dataset (dashed line) when training Causality DeepONet with POD. Initial conditions are provided as inputs.




(b) The evolution of relative $L^2$ errors for the training dataset(solid line) and testing dataset (dashed line) when training Deep Propagator. Initial conditions $\dot{u}$ are predicted by another Deep Propagator.




(c) The evolution of relative $L^2$ errors for the training dataset(solid line) and testing dataset (dashed line) when training Deep Propagator. Initial conditions $\dot{u}$ are computed by finite difference.




(d) The evolution of relative $L^2$ errors for the training dataset(solid line) and testing dataset (dashed line) when training Deep Propagator. Initial conditions $\dot{u}$ are computed by least square interpolations.

Figure C.2: The evolution of relative $L^2$ errors for Causality DeepONet with POD and Deep Propagator solving the nonhomogeneous wave equation.

# BIBLIOGRAPHY


[1] Albawi, S., Mohammed, T. A., and Al-Zawi, S. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)* (Aug 2017), pp. 1–6. 88

[2] Allen-Zhu, Z., Li, Y., and Song, Z. A convergence theory for deep learning via over-parameterization. In *Proceedings of the 36th International Conference on Machine Learning* (09–15 Jun 2019), K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97 of *Proceedings of Machine Learning Research*, PMLR, pp. 242–252. 12

[3] Bhattacharya, K., Hosseini, B., Kovachki, N. B., and Stuart, A. M. Model Reduction And Neural Networks For Parametric PDEs. *The SMAI journal of computational mathematics 7* (2021), 121–157. 90, 119

[4] Bochev, P. B., and Gunzburger, M. D. *Least-squares finite element methods*. Springer, 2009. 32, 61

[5] Brandt, A. Multi-level adaptive solutions to boundary-value problems. *Mathematics of Computation 31*, 138 (1977), 333–390. 22

[6] Bruna, J., and Mallat, S. Invariant scattering convolution networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence 35*, 8 (2013), 1872–1886. 24

[7] Cai, S., Wang, Z., Fuest, F., Jeon, Y. J., Gray, C., and Karniadakis, G. E. Flow over an espresso cup: inferring 3-d velocity and pressure fields from tomographic background oriented schlieren via physics-informed neural networks. *Journal of Fluid Mechanics 915* (2021), A102. 59

[8] Cai, S., Wang, Z., Lu, L., Zaki, T. A., and Karniadakis, G. E. DeepM&Mnet: Inferring the electroconvection multiphysics fields based on operator approximation by neural networks. *J. Comput. Phys. 436*, C (jul 2021). 20, 89

[9] Cai, W., Li, X., and Liu, L. PhaseDNN - a parallel phase shift deep neural network for adaptive wideband learning, 2019. 27

[10] Cai, W., Li, X., and Liu, L. A phase shift deep neural network for high frequency approximation and wave problems. *SIAM Journal on Scientific Computing 42*, 5 (2020), A3285–A3312. 22, 31, 59, 82

[11] Chen, T., and Chen, H. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks 6*, 4 (July 1995), 911–917. vii, 18, 19, 82, 92, 93, 136

[12] Chopra, A. K. *Dynamics of Structures*, 4th ed. Pearson, 2011. 82, 84, 85, 86, 87

[13] Cybenko, G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS) 2*, 4 (Dec. 1989), 303–314. 9, 10, 82

[14] De Ryck, T., Jagtap, A. D., and Mishra, S. Error estimates for physics-informed neural networks approximating the Navier–Stokes equations. *IMA Journal of Numerical Analysis* (01 2023). drac085. 59

[15] Deng, B., Shin, Y., Lu, L., Zhang, Z., and Karniadakis, G. E. Approximation rates of DeepONets for learning operators arising from advection–diffusion equations. *Neural Networks 153* (2022), 411–426. 20, 89

[16] Di Leoni, P. C., Lu, L., Meneveau, C., Karniadakis, G., and Zaki, T. A. DeepONet prediction of linear instability waves in high-speed boundary layers. *arXiv preprint arXiv:2105.08697* (2021). 20, 89

[17] Du, S., Lee, J., Li, H., Wang, L., and Zhai, X. Gradient descent finds global minima of deep neural networks. In *International conference on machine learning* (2019), PMLR, pp. 1675–1685. 12

[18] E, W., and Yu, T. The Deep Ritz method: A deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics 6* (2017), 1–12. 82

[19] Eivazi, H., Tahani, M., Schlatter, P., and Vinuesa, R. Physics-informed neural networks for solving Reynolds-averaged Navier–Stokes equations. *Physics of Fluids 34*, 7 (07 2022). 075117. 59

[20] Elman, J. L. Finding structure in time. *Cognitive Science 14*, 2 (1990), 179–211. 82

[21] Fan, Y., Orozco Bohorquez, C., and Ying, L. BCR-Net: A neural network based on the nonstandard wavelet form. *Journal of Computational Physics 384* (2019), 1–15. 24

[22] Fu, X., Chang, L.-B., and Xiu, D. Learning reduced systems via deep neural networks with memory. *Journal of Machine Learning for Modeling and Computing 1*, 2 (2020), 97–118. 82

[23] Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org. 9, 32

[24] Goswami, S., Yin, M., Yu, Y., and Karniadakis, G. E. A physics-informed variational DeepONet for predicting crack path in quasi-brittle materials. *Computer Methods in Applied Mechanics and Engineering 391* (2022), 114587. 83

[25] Gowtham Reddy, A. Causality in neural networks - an extended abstract. In *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society* (New York, NY, USA, 2021), AIES '21, Association for Computing Machinery, p. 271–272. 83

[26] Han, J., Jentzen, A., and E, W. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences 115*, 34 (2018), 8505–8510. 58, 82

[27] He, Y., and Li, J. Convergence of three iterative methods based on the finite element discretization for the stationary Navier–Stokes equations. *Computer Methods in Applied Mechanics and Engineering 198*, 15 (2009), 1351 – 1359. 62, 64

[28] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Improving neural networks by preventing co-adaptation of feature detectors. *ArXiv abs/1207.0580* (2012). 97

[29] Hochreiter, S., and Schmidhuber, J. Long short-term memory. *Neural Computation 9*, 8 (11 1997), 1735–1780. 82

[30] Jiang, B.-N., and Povinelli, L. A. Least-squares finite element method for fluid dynamics. *Computer Methods in Applied Mechanics and Engineering 81*, 1 (1990), 13–37. 32

[31] Jin, X., Cai, S., Li, H., and Karniadakis, G. E. NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations. *Journal of Computational Physics 426* (2021), 109951. 60, 82

[32] Kerschen, G., Worden, K., Vakakis, A. F., and Golinval, J.-C. Past, present and future of nonlinear system identification in structural dynamics. *Mechanical Systems and Signal Processing 20*, 3 (2006), 505–592. 82

[33] Kim, H.-S. Development of seismic response simulation model for building structures with semi-active control devices using recurrent neural network. *Applied Sciences 10*, 11 (2020), 3915. 82

[34] Kingma, D. P., and Ba, J. Adam: A Method for Stochastic Optimization. *CoRR abs/1412.6980* (2015). 37, 70, 97

[35] Li, Q., and E, W. Machine learning and dynamical systems. *SIAM news* (11 2021). 82

[36] Li, Z., Han, J., E, W., and Li, Q. Approximation and optimization theory for linear continuous-time recurrent neural networks. *Journal of Machine Learning Research 23*, 42 (2022), 1–85. 82

[37] Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895* (2020). 18, 83, 91, 116

[38] Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485* (2020). 83

[39] Lin, C., Li, Z., Lu, L., Cai, S., Maxey, M., and Karniadakis, G. E. Operator learning for predicting multiscale bubble growth dynamics. *The Journal of Chemical Physics 154*, 10 (2021), 104118. 20, 89

[40] Liu, L., and Cai, W. DeepPropNet–a recursive deep propagator neural network for learning evolution pde operators. *arXiv preprint arXiv:2202.13429* (2022). 115

[41] Liu, L., Nath, K., and Cai, W. A Causality-DeepONet for causal responses of linear dynamical systems. *arXiv preprint arXiv:2209.08397* (2022). 81, 118, 127, 137

[42] Liu, L., Wang, B., and Cai, W. Linearized learning with multiscale deep neural networks for stationary Navier-Stokes equations with oscillatory solutions. *East Asian Journal on Applied Mathematics 13*, 3 (2023), 740–758. 58, 90, 104

[43] Liu, Z., Cai, W., and Xu, Z.-Q. J. Multi-scale deep neural network (MscaleDNN) for solving poisson-boltzmann equation in complex domains. *Communications in Computational Physics 28*, 5 (2020), 1970–2001. 3, 59, 65, 66, 67, 82, 90

[44] Livshits, I., and Brandt, A. Accuracy properties of the wave-ray multigrid algorithm for helmholtz equations. *SIAM Journal on Scientific Computing 28*, 4 (2006), 1228–1251. 24

[45] Louizos, C., Shalit, U., Mooij, J., Sontag, D., Zemel, R., and Welling, M. Causal effect inference with deep latent-variable models. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Red Hook, NY, USA, 2017), NIPS'17, Curran Associates Inc., p. 6449–6459. 83

[46] Lu, J., Shen, Z., Yang, H., and Zhang, S. Deep network approximation for smooth functions. *SIAM Journal on Mathematical Analysis 53*, 5 (2021), 5465–5506. 10, 11

[47] Lu, L., Jin, P., Pang, G., Zhang, Z., and Karniadakis, G. E. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence 3*, 3 (2021), 218–229. 3, 18, 20, 82, 88, 89, 92, 116, 117

[48] Lu, L., Meng, X., Cai, S., Mao, Z., Goswami, S., Zhang, Z., and Karniadakis, G. E. A comprehensive and fair comparison of two neural operators (with practical extensions) based on FAIR data. *Computer Methods in Applied Mechanics and Engineering 393* (2022), 114778. 90, 91, 119

[49] Luo, T., Ma, Z., Xu, Z.-Q. J., and Zhang, Y. Theory of the frequency principle for general deep neural networks. *CSIAM Transactions on Applied Mathematics 2*, 3 (2021), 484–507. 22

[50] Luo, T., and Yang, H. Two-layer neural networks for partial differential equations: Optimization and generalization theory, 2020. 16, 17

[51] Luo, Y., Peng, J., and Ma, J. When causal inference meets deep learning. *Nature Machine Intelligence 2*, 8 (2020), 426–427. 83

[52] Moraffah, R., Karami, M., Guo, R., Raglin, A., and Liu, H. Causal interpretability for machine learning - problems, methods and evaluation. *SIGKDD Explor. Newsl. 22*, 1 (may 2020), 18–33. 83

[53] Newmark, N. M. A method of computation for structural dynamics. *Journal of the Engineering Mechanics Division 85*, 3 (1959), 67–94. 82

[54] Oldenburg, J., Borowski, F., Öner, A., Schmitz, K.-P., and Stiehm, M. Geometry aware physics informed neural network surrogate for solving Navier-Stokes equation (GAPINN). *Advanced Modeling and Simulation in Engineering Sciences 9*, 1 (2022), 8. 59

[55] Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. Wavenet: A generative model for raw audio, 2016. 82

[56] O'Shea, K., and Nash, R. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458* (2015). 88

[57] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. 97

[58] Qin, T., Wu, K., and Xiu, D. Data driven governing equations approximation using deep neural networks. *Journal of Computational Physics 395* (2019), 620–635. 82

[59] Rahaman, N., Baratin, A., Arpit, D., Draxler, F., Lin, M., Hamprecht, F., Bengio, Y., and Courville, A. On the spectral bias of neural networks. In *Proceedings of the 36th International Conference on Machine Learning* (09–15 Jun 2019), K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97 of *Proceedings of Machine Learning Research*, PMLR, pp. 5301–5310. 13, 27

[60] Raissi, M. Forward-backward stochastic neural networks: Deep learning of high-dimensional partial differential equations, 2018. 58

[61] Raissi, M., Perdikaris, P., and Karniadakis, G. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics 378* (2019), 686–707. 14, 32, 82

[62] Raissi, M., Yazdani, A., and Karniadakis, G. E. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science 367*, 6481 (2020), 1026–1030. 59

[63] Rao, C., Sun, H., and Liu, Y. Physics-informed deep learning for incompressible laminar flows. *Theoretical and Applied Mechanics Letters 10*, 3 (2020), 207–212. 59

[64] Shankar, R. *Principles of quantum mechanics*. Springer, New York, NY, 1994. 116

[65] Sitzmann, V., Martel, J., Bergman, A., Lindell, D., and Wetzstein, G. Implicit neural representations with periodic activation functions. In *Advances in Neural Information Processing Systems* (2020), H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., pp. 7462–7473. 67

[66] Sohn, H., Farrar, C. R., Hemez, F. M., Shunk, D. D., Stinemates, D. W., Nadler, B. R., and Czarnecki, J. J. A review of structural health review of structural health monitoring literature 1996-2001. 82

[67] Wang, B., Zhang, W., and Cai, W. Multi-scale deep neural network (MscaleDNN) methods for oscillatory stokes flows in complex domains. *Communications in Computational Physics 28*, 5 (2020), 2139–2157. 2, 59, 60, 61, 67, 72, 90, 104

[68] Winovich, N., Ramani, K., and Lin, G. ConvPDE-UQ: Convolutional neural networks with quantified uncertainty for heterogeneous elliptic partial differential equations on varied domains. *Journal of Computational Physics 394* (2019), 263–279. 88

[69] Xu, Z. J. Understanding training and generalization in deep learning by fourier analysis, 2018. 22

[70] Xu, Z.-Q. J., Zhang, Y., Luo, T., Xiao, Y., and Ma, Z. Frequency principle: Fourier analysis sheds light on deep neural networks. *Communications in Computational Physics 28*, 5 (2020), 1746–1767. 13, 22, 26, 27, 59, 90

[71] Zhang, L., Cai, W., and Xu, Z.-Q. J. A correction and comments on "multi-scale deep neural network (MscaleDNN) for solving poisson-boltzmann equation in complex domains cicp, 28 (5): 1970–2001, 2020". *Communications in Computational Physics 33*, 5 (2023), 1509–1513. 67

[72] Zhang, R., Chen, Z., Chen, S., Zheng, J., Büyüköztürk, O., and Sun, H. Deep long short-term memory networks for nonlinear structural seismic response prediction. *Computers & Structures 220* (2019), 55–68. 82

[73] Zhang, W., and Cai, W. FBSDE based neural network algorithms for high-dimensional quasilinear parabolic PDEs. *Journal of Computational Physics 470* (2022), 111557. 58, 59, 66, 82

[74] Zhu, M. OpenSeesPy. https://openseespydoc.readthedocs.io/en/latest/#, 2015. [Online]. 86

[75] Zhu, Y., Zabaras, N., Koutsourelakis, P.-S., and Perdikaris, P. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics 394* (2019), 56–81. 88

[76] Zienkiewicz, O. C., Taylor, R. L., and Zhu, J. Z. *The finite element method: its basis and fundamentals*. Elsevier, 2005. 81