ABSTRACT

CHEN, QIUYI. Physics Informed Learning for Dynamic Modeling of Beam Structures. (Under the direction of Dr. Fuh-Gwo Yuan).

Artificial neural networks (ANNs) have been applied to many scientific areas to approximate various mappings. These ANNs are mainly trained with conventional data-driven algorithm and haunted by overfitting problems when sampled data is insufficient. This cripples unreserved application of ANNs in scientific areas as their reliability can hardly be guaranteed. Physics informed learning (PIL) manages to incorporate human prior knowledge in the form of physics governing equations into the training process to help regularize ANNs and achieve reliable results and greater generalization. Its power in dealing with forward problems and inverse problems has been examined by a few existing works, yet not thoroughly. In this work, PIL is for the first time systematically overviewed, then employed to solve forward and inverse beam vibration problems governed by high order parabolic PDEs over large time span. We demonstrate a means to incorporate numerical methods into the training process and how to circumvent PIL's limitation using segmented time solution inspired by the spirit of Huygens' principle. Then we extend our experience in solving forward problems to inverse problems to reconstruct complicated displacement fields and identify unknown system parameters from sparse and noisy sensor data and demonstrate its robustness in achieving great generalization in the presence of noise. This work prepares the ground for extrapolating PIL to more involved and realistic problems in engineering simulation, structural health monitoring etc.

Physics Informed Learning for Dynamic Modeling of Beam Structures

by
Qiuyi Chen

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the degree of
Master of Science

Mechanical Engineering

Raleigh, North Carolina
2020

APPROVED BY:

_____	_____
Fuh-Gwo Yuan Committee Chair	Hsiao-Ying Shadow Huang
_____	
Xiaoning Jiang	

DEDICATION

To my parents and their selfless support.

BIOGRAPHY

Qiuyi Chen was born in Nanjing, China. Chen spent his life in this city until he finished his bachelor's degree at Nanjing University of Aeronautics and Astronautics in 2015. After that he moved to Shanghai and became an R&D project engineer in Radiall Inc., a French company producing telecommunication connectors. There he could anticipate his trajectory in the following decades if he devoted his youth time to mundane works and finally realized what he want for his life, so he resigned in 2018 after finishing the NEX10 project, and started pursuing his Master of Science degree at NC State, where he joined Dr. Fuh-Gwo Yuan's research group. During his stay at NC State, his research is mainly focused on physics informed neural networks' application on structural health monitoring problems. He will be pursuing his Ph.D. degree at the University of Maryland, College Park after graduation and continue his exploration within the penumbra between mechanical engineering, computer science and statistics.

ACKNOWLEDGMENTS

I would like to first show my gratitude towards Dr. Fuh-Gwo Yuan for his guidance and encouragement throughout my work. Dr. Yuan proves to be a great, caring advisor who broadened my horizons and motivated me to become a better person. Next, I would like to thank my committee members, Dr. Hsiao-Ying Shadow Huang and Dr. Xiaoning Jiang for their willingness to be in my thesis committee. I appreciate the consistent support Dr. Huang bestowed ever since she taught me, and the cozy banquet Dr. Jiang once catered.

I also feel obliged to my friends who accompanied me through this 2-year time span. I wish them Godspeed in pursuing their doctorate degree.

The rest of my gratitude shall be paid to Russian pirates who established Library Genesis, which grants students in poverty like me enormous invaluable knowledge.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

Ever since AlphaGo defeated the renowned go player Lee Sedol on the world's hardest game, artificial neural networks (ANNs) technique beneath it has drawn far more attention than ever before. Laymen are inclined to presume it as an omnipotent surreal intelligence that can be applied to anything in interest and let it do the work without any worries. Such myth is too good to be true. A brief retrospect may help dispense with this misconception.

Rome is not built in a day. ANNs originate back to some primitive ideas midway through the last century, out of which were some maladroit theoretical mimicking of the operation of animal's neurons, as indicated by the academic terms closely related to biology jargons in those papers[1]. Later on, those abstract structures were implemented in custom-built hardware as a single-layered perceptron[2]. Although research on that had been stagnated for quite a long while due to this perceptron's incapability of handling many classes of patterns, the barrier was eventually lifted with the introduction of multi-layer perceptron (MLP)[3], which has much greater processing power. As time elapses, these structures evolved into the abstract intangible MLPs implemented programmatically as seen today. MLPs, or ANNs to which they belong, are therefore still a simplified emulation of animal's intricate neural system that supposedly can replicate some behaviors of biological neurons. Considering this, it is insensible to believe that some crude and awkward duplication can suddenly gain enormous perceptual and reasoning power without guidance from human.

That intrinsic imperfection is indeed what scientists and researchers suffered throughout their research experience. From that day perceptron was invented to the present, image recognition has always been a prevalent area of ANN's application. The "fully-connectedness"

of MLPs imparts themselves great representability, which is reflected in its ability to precisely fit each input-output data pair, yet in consequence induces overfitting problems that the good behavior of the MLPs cannot be replicated when they are fed with data they never met before. This overfitting had been plaguing image recognition tasks until the introduction of convolutional neural networks (CNNs), which were creatively modelled on the receptive fields in the visual cortex, that this difficulty was mitigated by a large measure. Though appearing like a disparate thing at first sight, CNNs are essentially MLPs regularized by sparse connections and parameter sharing[4], such that they can even be equivalently transformed into MLPs whose weight matrices contain lots of zero (w.r.t sparse connection) and series of identical components (w.r.t parameter sharing). By virtue of this breakthrough achieved through informed regularization inspired by visual cortex, it proves that integrating human's prior knowledge into the design and training of ANNs is an essential part of related research and applications.

Traditionally, regularization of ANNs is accomplished by imposing constraints on its parameters, i.e. weights and biases. Apart from the regularization underlying CNNs as mentioned above, there are several regularization methods that are more straightforward to tackle overfitting, such as L1 and L2 regularization, which make the weight matrices sparse and of low-valued components respectively. Nonetheless, theses regularization approaches are carried out mostly on a trial and error basis, and there is no practical way to guarantee their positive effect. In fact, apart from their positive performance on certain tasks, no one can come up with solid explanation for that except some intuition and ambiguous statistical gibberish. Most importantly, those regularization methods all fail to work out when the samples provided are biased and insufficient, which is usually the case for most scientific regression tasks rather than those image classification or language processing works under the spotlight.

Physics informed learning (PIL) [5] recently emerged as a remedy for both the lack of data and overfitting problems. Under the fancy hood it is essentially a regularization technique but different from its precedents in two major aspects. First, instead of some statistical or probabilistic regulator that one cannot precisely interpret, the regularization terms appended to the loss function are exactly the governing equations of the physics problem in hand, such that we can make sure the training results will be in line with the physics law as long as the ANNs are successfully trained, and we can precisely determine how well the training result satisfies it. Secondly, the regularization is applied to the output of the ANNs instead of its parameters or neurons, therefore even when the datasets provided for training contain only a scarce number of noisy input-output pairs, the ANNs can still successfully achieve reliable results.

This work aims to thoroughly examine physics-informed learning's performance on mechanical engineering problems, specifically structural dynamic problems, to discover the potential deficiency of this embryonic technique and ways to overcome it, and to pave the way for applying it to more complicated structural problems and beyond.

The structure of this thesis is arranged as follows:

In Chapter 2, the background of artificial neural networks' structures, training, and problems in application is presented to serve as a preliminary material for the following chapters.

In Chapter 3, historical development of PIL and their relationships with traditional numerical methods is presented. These are the most important concepts from which this work is developed and will serve as a guidance for people who plan to delve into this emerging area.

In Chapter 4, PIL is applied to vibration problems governed by high ordered parabolic PDEs to inspect its power and limitation. Segmented time solution is also demonstrated to overcome the encountered issues.

In Chapter 5, PIL is implemented to solve simple inverse problems where the vibration displacement fields are reconstructed from noisy and sparse measurements, and the material properties of the selected beam structures are inversely determined.

Chapter 6 summarizes the accomplishments in this work and anticipate further steps in the future.

CHAPTER 2

Background of Artificial Neural Networks

2.1. Structures of Artificial Neural Networks

Artificial neural networks (ANNs) are high dimensional function compositions that were originally enlighten by the neural system of animals. Thanks to universal approximation theorem[6], they are used to approximate mappings $\boldsymbol{f}^*$ in interest so that we can use them to perform predictions instead of using the original mappings not under control. Though not possible to fully duplicate that delicate cerebral mechanism, ANNs still manage to acquire some of the ability to handle high-dimensional mappings, and that ability is further enhanced by the colossal computational power of GPUs. Several common structures of ANNs are introduced below.

2.1.1. Multi-layer Perceptron

Multi-layer perceptrons (MLPs), also called deep feedforward networks or feedforward neural networks, are the most elementary and widely used ANN architecture from which most other ANN structures stem. Its general mathematical expression is shown as follows:

$$\boldsymbol{z}_i(\boldsymbol{x}) = \boldsymbol{W}_i\boldsymbol{x} + \boldsymbol{b}_i, \tag{2.1.1}$$

$$\boldsymbol{s}_1 = \sigma_1\big(\boldsymbol{z}_1(\boldsymbol{x})\big), \;\; \boldsymbol{s}_i = \sigma_i\big(\boldsymbol{z}_i(\boldsymbol{s}_{i-1})\big), \tag{2.1.2}$$

$$\boldsymbol{o} = \boldsymbol{\Phi}(\boldsymbol{x};\boldsymbol{\theta}) = \sigma_n\Big(\boldsymbol{z}_n\Big(\dots\sigma_1\big(\boldsymbol{z}_1(\boldsymbol{x})\big)\Big)\Big), \qquad \boldsymbol{\theta} = \big\{\{\boldsymbol{W}_i\},\{\boldsymbol{b}_i\}\big\} \tag{2.1.3}$$

where to derive the ith **layer** $\boldsymbol{s}_i$ of the MLP, the affine transformation $\boldsymbol{z}_i$ first performs a linear transformation on the layer's input $\boldsymbol{x}$ using a **weight** $\boldsymbol{W}_i$, then a **bias** $\boldsymbol{b}_i$ is added to the linear transformation result. Thereafter an **activation function** σ_i, which is usually a scalar nonlinear function like logistic sigmoid, Tanh or Rectified Linear Unit (ReLU), is applied to $\boldsymbol{z}_i$ elementwise to impart nonlinearity to the transformation. An **MLP** $\boldsymbol{\Phi}$: $\mathbf{R}^m \to \mathbf{R}^n$, as its name

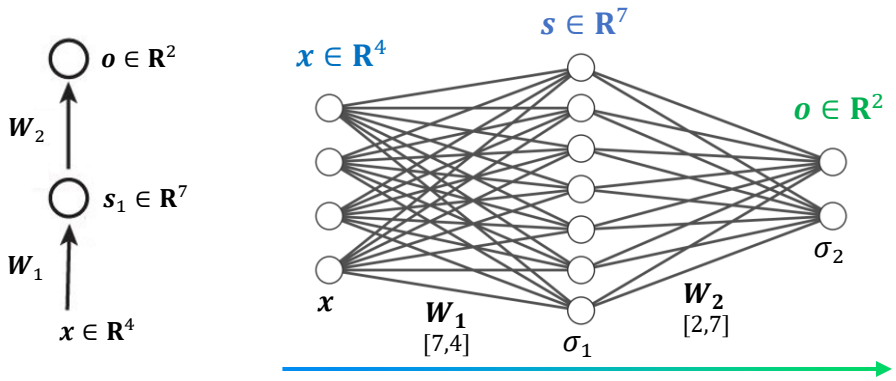

Figure 1. An MLP with single hidden layer. The input $x \in \mathrm{R}^4$ is first mapped to $s_1 \in \mathrm{R}^7$ by the fully connected hidden layer $\mathrm{R}^4 \to \mathrm{R}^7$, then mapped to $o \in \mathrm{R}^2$ by the output layer $\mathrm{R}^7 \to \mathrm{R}^2$. The hidden layer contains a weight matrix $W_1 \in \mathrm{R}^{7\times4}$, while the output layer $W_2 \in \mathrm{R}^{2\times7}$. The input of each layer, i.e. the output of previous layer, is first multiplied by the weight matrix, so that each scalar component of the layer's final output is associated with every component of the input. In this way we say every layer is fully connected with the previous layer.

indicates, contains multiple such layers $\sigma_i(\boldsymbol{z}_i(\boldsymbol{x}))$ composited in sequence to yield the desired **output** $\boldsymbol{o} \in \mathbf{R}^n$ after processing the MLP's **input** $\boldsymbol{x} \in \mathbf{R}^m$. As shown in Equation 2.1.2 and 2.1.3, the first layer takes in the MLP's input $\boldsymbol{x}$ and outputs $\boldsymbol{s}_1$, which is then fed into the subsequent layers that output $\boldsymbol{s}_2, \boldsymbol{s}_3 \ldots \boldsymbol{s}_n$ iteratively, out of which the last layer's output $\boldsymbol{s}_n$ is the desired output $\boldsymbol{o}$ of the MLP. Conventionally, we call $\boldsymbol{x}$ **input layer**, $\boldsymbol{o}$ **output layer**, and all the other layers $\boldsymbol{s}_1, \boldsymbol{s}_2 \ldots \boldsymbol{s}_{n-1}$ in between not exposing themselves **hidden layers**. The scalar components of each layer are usually referred to as **nodes** or **neurons**, resembling the intertwined biological neurons. A typical single hidden layer MLP is illustrated in Figure 1.

Since for each layer $\boldsymbol{s}_i$ of MLPs, every neuron of it is connected with all neurons of the previous layer due to the affine transformation $\boldsymbol{z}_i$, these layers are often named **fully connected layers**. This simple structure spawned many advanced offspring.

2.1.2. Convolutional Layers and Convolutional Neural Networks

Convolutional layers seem distinct from fully connected layers at first glance. In this work they are instructively denoted as follows:

$$\boldsymbol{s}_i = \sigma_i(\boldsymbol{z}_i),\ \boldsymbol{z}_i(\boldsymbol{x}) = \boldsymbol{W}_i \star \boldsymbol{x} \oplus \boldsymbol{b}_i, \tag{2.1.4}$$

Here σ_i is again element-wisely applied on $\boldsymbol{z}_i$. But unlike fully connected layer whose input is always a vector, the operation $\boldsymbol{z}_i$ in convolutional layer now processes tensors of arbitrary order and yields tensors of the same order through convolution. A high order tensor **input** $\boldsymbol{x} \in \mathbf{R}^{m_1 \times \ldots \times m_p \times c_{in}}$ whose last axis is a channel of dimension c_{in} corresponds to a **kernel** $\boldsymbol{W}_i \in \mathbf{R}^{n_1 \times \ldots \times n_p \times c_{in} \times c_{out}}$ with c_{out} **filters** and a **bias** $\boldsymbol{b}_i \in \mathbf{R}^{1_1 \times \ldots \times 1_p \times c_{out}}$, with $n \leq m$. The operator $\star$ represents cross-correlation operation which is carried out by sliding $\boldsymbol{W}_i$ along the first p axes of input $\boldsymbol{x}$ at a determined **stride** $\boldsymbol{k} \in \mathbf{R}^p$. For each stride, the components of $\boldsymbol{x}$ covered by $\boldsymbol{W}_i$ are element-wisely multiplied with the kernel, then summed up and added with $\boldsymbol{b}_i$ (The operator $\oplus$ here denotes stride-wise addition). The grid of values such yielded are then all concatenated maintaining their positional order to produce output $\boldsymbol{z}_i \in \mathbf{R}^{l_1 \times \ldots \times l_p \times c_{out}}$ where $l_i = 1 + (m_i - n_i)\backslash k_i$. A concrete example is demonstrated in Figure 2.

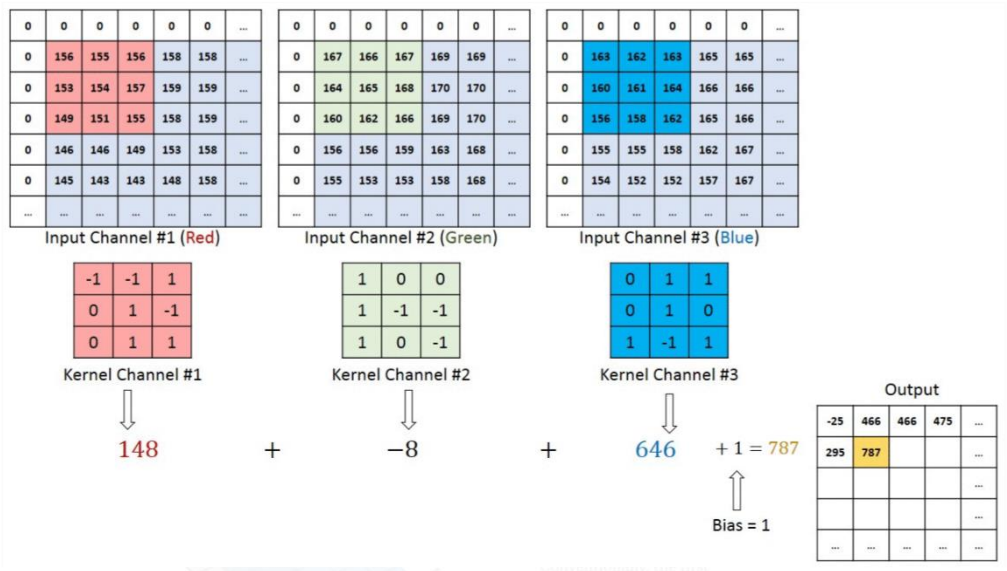

Figure 2. Convolutional layer operating on an image. A typical image tensor is of order 3, containing three axes: width, height and RGB. Here a convolutional layer with only 1 filter is used. A kernel with 3 channels corresponding to RGB is sliding over the image along its width and height axis and performing cross-correlation calculation for each stride. The results are all added with a bias 1 and eventually stitched together to form the output, which is also a third order tensor but with only 1 channel. By this operation, spatial patterns embedded in the image can be extracted more efficiently than by fully connected layers.

Despite the convolutions or cross-correlations involved, convolutional layers are in fact fully connected layers governed by sparse connection and parameter sharing. If the input and output tensor of a convolutional layer are flattened into 2 vectors, the mapping between them can be equivalently established by a fully connected layer each neuron of which is related to the inner product between the input vector and one of the weight matrix's row vectors wherein some scalar components is the same as the convolutional kernel's while the rest is zero. All row vectors of that weight matrix share the same set of kernel components, the only difference is the indices they are assigned to.

Convolutional neural networks (CNNs) are compositions involving convolutional layers, usually with convolutional layers in the front to accept high order tensors as input, a flatten layer in the middle to flatten tensors into vectors and fully connected layers in the rear to produce vector outputs. Because of the association between convolutional layers and fully connected layers, CNNs can be transformed into MLPs albeit there is no necessity to do so.

2.1.3. Recurrent Neural Networks

Recurrent neural networks (RNNs) are very suitable for handling sequences each element of which is correlated with the previous one. They are usually used to approximate the mappings $\mathbf{R}^{m\times l} \to \mathbf{R}^{n\times l}$ where l is the length of the input sequence $\boldsymbol{x} = (\boldsymbol{x}^1, \boldsymbol{x}^2, \dots \boldsymbol{x}^t, \dots \boldsymbol{x}^l)$. Their output $\boldsymbol{o} = (\boldsymbol{o}^1, \boldsymbol{o}^2, \dots \boldsymbol{o}^t, \dots \boldsymbol{o}^l)$ is generated by processing the input sequence **recursively**. RNNs' structure and recursive operation can be precisely expressed as follows:

$$\boldsymbol{z}_i^t(\boldsymbol{x}) = \boldsymbol{W}_i\boldsymbol{x} + \boldsymbol{U}_i\boldsymbol{s}_i^{t-1} + \boldsymbol{b}_i, \tag{2.1.5}$$

$$\boldsymbol{s}_1^t = \sigma_1\big(\boldsymbol{z}_1^t(\boldsymbol{x})\big), \;\; \boldsymbol{s}_i^t = \sigma_i\Big(\boldsymbol{z}_i^t(\boldsymbol{s}_{i-1}^t)\Big), \tag{2.1.6}$$

$$\boldsymbol{o}^t = \boldsymbol{\Phi}^t(\boldsymbol{x};\boldsymbol{\theta}) = \sigma_n\left(\boldsymbol{z}_n^t\left(\dots\sigma_1\big(\boldsymbol{z}_1^t(\boldsymbol{x}^t)\big)\right)\right), \qquad \boldsymbol{\theta} = \big\{\{\boldsymbol{W}_i\},\{\boldsymbol{U}_i\},\{\boldsymbol{b}_i\}\big\}, \tag{2.1.7}$$

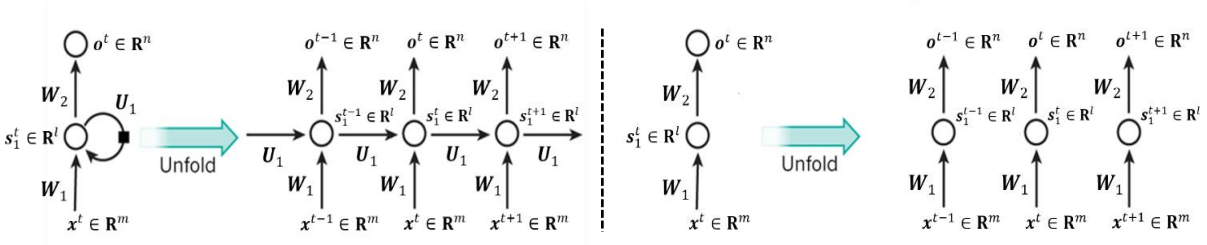

Figure 3. An RNN with single hidden layer (Left) and its comparison with its MLP counterpart (Right). For the case of MLP (Right), we first break the input sequence $\boldsymbol{x} \in \mathbf{R}^{m\times l}$ into separate elements $\boldsymbol{x}^t \in \mathbf{R}^m$, each $\boldsymbol{x}^t$ is then processed by the same MLP $\boldsymbol{\Phi}$: $\mathbf{R}^m \to \mathbf{R}^n$ that outputs the corresponding $\boldsymbol{o}^t \in \mathbf{R}^n$. Those $\boldsymbol{o}^t$ are then concatenated to obtain $\boldsymbol{o} \in \mathbf{R}^{n\times l}$ whose element $\boldsymbol{o}^t$ only depends on $\boldsymbol{x}^t$. For the case of RNN (Left), each $\boldsymbol{o}^t$ is assumed to be dependent on both $\boldsymbol{x}^t$ and the hidden state $\boldsymbol{s}_1^{t-1}$ of the previous stage $t-1$. Every $\boldsymbol{o}^t$ can be regarded as the output of an MLP $\boldsymbol{\Phi}^t$ that takes both $\boldsymbol{x}^t$ and $\boldsymbol{s}_1^{t-1}$ as input. Therefore, for the output sequence $\boldsymbol{o}$ of the RNN, each element $\boldsymbol{o}^t$ is correlated with the previous element $\boldsymbol{o}^{t-1}$. It is conventional to call this entire recursive mapping $\mathbf{R}^{m\times l} \to \mathbf{R}^{n\times l}$ an RNN.

where the superscript t denotes that the object it modifies is at its tth **stage** where the network is processing the tth element of the input sequence $\boldsymbol{x}$. The general multi-layered structure of RNNs does not differ much from that of MLPs. In fact, if we set $\boldsymbol{U}_i$ in each layer to zero, RNNs morph into MLPs. An illustration of the difference between RNNs and MLPs is shown in Figure 3.

However, due to their distinct recursive property, RNNs' layers are usually referred to as **recurrent layers**. The peculiarity is that each layer, taking $\boldsymbol{s}_i^t$ the ith layer at the tth stage as an example, is now evolving while processing the sequence because of the additional term $\boldsymbol{s}_i^{t-1}$, which stands for the hidden state of the ith layer at the previous stage $t-1$. From another perspective, if we rearrange Equation 2.1.5 and 2.1.6, we can see that:

$$\boldsymbol{s}_1^t = \sigma_1(\boldsymbol{W}_1\boldsymbol{x}^t + \boldsymbol{U}_1\boldsymbol{s}_1^{t-1} + \boldsymbol{b}_1) = \sigma_1\left([\boldsymbol{W}_1 \quad \boldsymbol{U}_1]\begin{bmatrix}\boldsymbol{x}^t \\ \boldsymbol{s}_1^{t-1}\end{bmatrix} + \boldsymbol{b}_1\right) = \sigma_1(\boldsymbol{W}_1'\boldsymbol{x}_1' + \boldsymbol{b}_1) \qquad (2.1.8)$$

$$\boldsymbol{s}_i^t = \sigma_i(\boldsymbol{W}_i\boldsymbol{s}_{i-1}^t + \boldsymbol{U}_i\boldsymbol{s}_i^{t-1} + \boldsymbol{b}_i) = \sigma_i\left([\boldsymbol{W}_i \quad \boldsymbol{U}_i]\begin{bmatrix}\boldsymbol{s}_{i-1}^t \\ \boldsymbol{s}_i^{t-1}\end{bmatrix} + \boldsymbol{b}_i\right) = \sigma_i(\boldsymbol{W}_i'\boldsymbol{x}_i' + \boldsymbol{b}_i) \qquad (2.1.9)$$

It is obvious that recurrent layers are nothing but innovative application of fully connected layers. The input of the first fully connected layer at stage t is the concatenation of current input $\boldsymbol{x}^t$ and the previous hidden state $\boldsymbol{s}_1^{t-1}$, whereas for the following layers it is the concatenation of the previous $(i-1)$th layer's output $\boldsymbol{s}_{i-1}^t$ at the current stage and the current ith layer's hidden

state $\boldsymbol{s}_i^{t-1}$ at the previous stage. Accordingly, we can make sure that the current output $\boldsymbol{o}^t$ is dependent both on the current input and the previous stage.

2.2. Training of Artificial Neural Networks

Inputting data $\boldsymbol{x}$ into ANNs to get their outputs $\boldsymbol{o}$ is designated as **forward propagation**. Before performing forward propagation to get ANNs' predictions, it is always necessary to train them beforehand to approximate the underlying function $\boldsymbol{f}^*$. "**Training**", which means finding the parameters that can minimize the specified loss function on certain ANN, is a personified paraphrase of **optimization**. For each training, a computational **graph** is first established for calculating the loss.

2.2.1. Parameters and Hyperparameters

Parameters

Parameters are variables of the graph that can be optimized by training algorithms. In most cases they are weights and biases of ANN's layers, namely $\boldsymbol{\theta} = \{\{\boldsymbol{W}_i\}, \{\boldsymbol{b}_i\}\}$, but can also have exceptions, e.g. the initial states of an RNN can be set as trainable parameters.

Hyperparameters

Hyperparameters are arguments associated with the graph and the training process that cannot be optimized by training algorithms but by hand on a trial and error basis. For instance, the depth and the width of an MLP, namely the number of layers and the number of neurons in each layer, are hyperparameters. Another common hyperparameter is activation function. Usually the activation functions of the hidden layers are set identical, while that of the output layer is set according to the expected output. For instance, if the output is a probability, sigmoid is chosen due to its $(0, 1)$ range. Identity function is usually set as activation function for the output layer if the ANN is used for regression.

2.2.2. Training Set and Loss Function

Training algorithms require objective function or **loss function** L to dictate how the parameters of the ANNs should be optimized. Training processes whose loss functions have exact outputs $\boldsymbol{o}_i$ involved are described as **supervised**. Most ANNs are trained by reducing the distances between predicted outputs $\widehat{\boldsymbol{o}}_i$ and exact outputs $\boldsymbol{o}_i$. Assume that for training an ANN $\boldsymbol{\Phi}(\boldsymbol{x};\boldsymbol{\theta})$, there exists a **training set** $\mathcal{X}$ which is a collection of input-output pairs $\{(\boldsymbol{x}_i, \boldsymbol{o}_i)\}_{i=1}^{N}$ where $\boldsymbol{x}_i$ is the ith input and $\boldsymbol{o}_i$ the corresponding **exact output**. Fed with $\boldsymbol{x}_i$, the ANN $\boldsymbol{\Phi}$ generates **predicted output** $\widehat{\boldsymbol{o}}_i = \boldsymbol{\Phi}(\boldsymbol{x}_i;\boldsymbol{\theta})$ through forward propagation. In order to make $\boldsymbol{\Phi}$ approximate the underlying function $\boldsymbol{f}^*(\boldsymbol{x})$ by which $\boldsymbol{o}_i = \boldsymbol{f}^*(\boldsymbol{x}_i)$, we minimize the distance between each $\widehat{\boldsymbol{o}}_i$ and $\boldsymbol{o}_i$ by optimizing $\boldsymbol{\theta}$. The distance function is usually chosen as L^2 norm, but other norms could also be suitable depending on situations. Therefore, the loss is calculated as:

$$L = \frac{1}{N} \sum_{\boldsymbol{x}_i, \boldsymbol{o}_i \in \mathcal{X}} |\boldsymbol{\Phi}(\boldsymbol{x}_i;\boldsymbol{\theta}) - \boldsymbol{o}_i|^2 . \qquad (2.2.1)$$

Because of the L^2 distance function, this loss function is called **mean squared error** (**MSE**).

Needless to say, there are a variety of loss functions other than MSE. Since MSE is the default loss function for regression problem, pages are saved for more important topics.

2.2.3. Training, Gradient Descent and Backpropagation

Training

Training of an ANN is specifically the process of optimizing the ANN's parameters $\boldsymbol{\theta}$ in order to minimize the loss function L. If this training process is perceived from ANNs' perspective, it can also be referred to as **learning**.

Gradient Descent

The most used optimization approach for training ANNs is gradient descent. For each iteration of gradient descent given loss function L, the parameters are updated by:

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^{k} - \epsilon \cdot \nabla_{\boldsymbol{\theta}} L, \tag{2.2.2}$$

where $\boldsymbol{\theta}^{k}$ is parameters $\boldsymbol{\theta}$ at the kth iteration, and ϵ is a hyperparameter named **learning rate**, which determines how aggressively should $\boldsymbol{\theta}$ be updated.

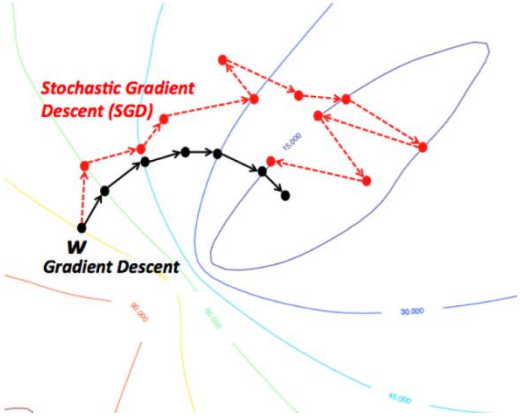

Figure 4. Gradient Descent vs Stochastic Gradient Descent

In practice, though, if L is defined over all the N data pairs as shown in Equation 2.2.1, $\boldsymbol{\theta}$ is very likely to end up stranded at local minima, and it might be expensive to evaluate L over all the N samples. To escape this trap, **stochastic gradient descent** (SGD) is usually employed. For each iteration, the loss function is evaluated instead over a mini-batch $\mathcal{X}_m = \{(\boldsymbol{x}_i, \boldsymbol{o}_i)\}_{i=1}^{N_b}$ that is uniformly resampled from the total N data pairs of $\mathcal{X}$:

$$L^R = \frac{1}{N_b} \sum_{\boldsymbol{x}_i, \boldsymbol{o}_i \in \mathcal{X}_m} |\boldsymbol{\Phi}(\boldsymbol{x}_i; \boldsymbol{\theta}) - \boldsymbol{o}_i|^2, \qquad N_b < N \tag{2.2.3}$$

Then the parameters $\boldsymbol{\theta}$ are updated using this loss L^R:

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^{k} - \epsilon \cdot \nabla_{\boldsymbol{\theta}} L^R. \tag{2.2.4}$$

This stochastic approach can be justified by $\mathbb{E}[L^R] = L$.

Backpropagation

The crucial part of gradient based method is the evaluation of the gradient $\nabla_{\boldsymbol{\theta}} L$. Since every ANN is more or less a function composition, in order to evaluate the gradient with respect to the parameters of the layers nested inside, chain rule should be implemented. For a multi-layer

structured ANN $\boldsymbol{\Phi}(\boldsymbol{x}) = \sigma_n\left(\boldsymbol{z}_n\left(\dots\sigma_1(\boldsymbol{z}_1(\boldsymbol{x}))\right)\right)$, to optimize $\boldsymbol{\theta}_i = \{\boldsymbol{W}_i, \boldsymbol{b}_i\}$ of $\boldsymbol{z}_i$ given loss function L, the gradient is evaluated as follows after forward propagation with $\boldsymbol{x}$:

$$\nabla_{\boldsymbol{\theta}_i} L = \nabla_{\boldsymbol{s}_n} L \cdot \nabla_{\boldsymbol{s}_{n-1}} \boldsymbol{s}_n \cdot \nabla_{\boldsymbol{s}_{n-2}} \boldsymbol{s}_{n-1} \dots \nabla_{\boldsymbol{s}_i} \boldsymbol{s}_{i+1} \cdot \nabla_{\boldsymbol{\theta}_i} \boldsymbol{s}_i. \tag{2.2.5}$$

Apparently, $n - i + 1$ gradients need to be evaluated in the reverse order of forward propagation ahead of evaluating $\nabla_{\boldsymbol{\theta}_i} \boldsymbol{s}_i$. After $\nabla_{\boldsymbol{\theta}_i} \boldsymbol{s}_i$ is derived, the gradient $\nabla_{\boldsymbol{s}_i} \boldsymbol{s}_{i+1}$ then takes part in the evaluation of gradients of lower layers. Such a reverse procedure of gradient computation following chain rule is embellished by the fancy word **backpropagation**.

Thankfully, there is no need to arduously derive analytical gradients by hand before compiling training algorithm. Backpropagation is the reverse mode of **automatic differentiation** and contemporary machine learning APIs employ automatic differentiation throughout their frameworks. The operations every tensor undergoes through forward backpropagation are automatically traced for later use in backpropagation. More information about automatic differentiation can be found in [7].

2.3. Performance of Artificial Neural Networks

2.3.1. Performance and Metrics

The prerequisite of performance evaluation is a clear definition of "performance". Performance is directly dependent on loss. The smaller the loss is, the better the performance, and vice versa. **Performance** is therefore a monotonically decreasing function of loss.

This suggests that any monotonic function of loss can give us a hint of how great the performance is. Traditionally, such a monotonic function of loss chosen as indicator of performance is referred to as **metrics**, which can either be a distance function more intuitive to human, like mean absolute error (MAE) for regression problems, or accuracy, which is suitable for classification problems, or something else.

2.3.2. Training Set, Validation Set and Test Set

Assume an ANN $\boldsymbol{\Phi}$ is going to be trained to approximate $\boldsymbol{f}^*(\boldsymbol{x})$ and used for prediction. In retrospect, the aforementioned **training set** $\mathcal{X}$ is nothing but finite samples extracted from the true underlying function $\boldsymbol{f}^*(\boldsymbol{x})$ whose domain is Ω. Inputs $\{\boldsymbol{x}_i\}_{i=1}^N$ of the training set is apparently a finite subset Ω_{train} of Ω. Thus, training set $\mathcal{X}$ can be expressed precisely as $\{(\boldsymbol{x}_i, \boldsymbol{o}_i)|\boldsymbol{x}_i \in \Omega_{train} \subseteq \Omega, \boldsymbol{o}_i = \boldsymbol{f}^*(\boldsymbol{x}_i)\}_{i=1}^N$. Inevitably, the performance of $\boldsymbol{\Phi}$ on the entire Ω cannot be guaranteed by its mere coincidence on Ω_{train}. Therefore, in order to evaluate $\boldsymbol{\Phi}$'s **generalization performance**, namely whether $\boldsymbol{\Phi}$ approximates $\boldsymbol{f}^*(\boldsymbol{x})$ well everywhere on Ω, samples from $\Omega \backslash \Omega_{train}$ must be used.

Validation set $\mathcal{X}_{val} = \{(\boldsymbol{x}_i, \boldsymbol{o}_i)|\boldsymbol{x}_i \in \Omega_{val} \subseteq \Omega \backslash \Omega_{train}, \boldsymbol{o}_i = \boldsymbol{f}^*(\boldsymbol{x}_i)\}_{i=1}^{N_v}$ is a dataset sampled from $\Omega \backslash \Omega_{train}$ to assess $\boldsymbol{\Phi}$'s real-time generalization performance during training. It remains fully hidden from $\boldsymbol{\Phi}$ during each optimization iteration and only reveals itself for performance evaluation after each iteration. It goes without saying that $\boldsymbol{\Phi}$'s good performance on the validation set is a necessary condition for its good performance on the entire Ω. Thus, according to Bayesian inference, if $\boldsymbol{\Phi}$ trained only on the training set can perform well on the validation set, we can expect it to achieve better performance on the entire Ω:

$$P(G|V) = P(G)\frac{P(V|G)}{P(V)} \simeq P(G)\frac{1}{P(V)} > P(G) \tag{2.3.1}$$

$G \equiv$ Good performance on Ω. $\quad$ $V \equiv$ Good performance on Ω_{val}.

If after tuning hyperparameters several times, $\boldsymbol{\Phi}$ can eventually achieve a good result on validation set, we suppose that the current hyperparameter combination can achieve good generalization performance, then merge validation set and training set to train the final $\boldsymbol{\Phi}$.

Test set $\mathcal{X}_{test} = \{(\boldsymbol{x}_i, \boldsymbol{o}_i)|\boldsymbol{x}_i \in \Omega_{test} \subseteq \Omega \backslash (\Omega_{train} \cup \Omega_{val}), \boldsymbol{o}_i = \boldsymbol{f}^*(\boldsymbol{x}_i)\}_{i=1}^{N_t}$ is also a dataset sampled from $\Omega \backslash \Omega_{train}$, but only used for performance evaluation after the final $\boldsymbol{\Phi}$ is

trained. If the final $\mathbf{\Phi}$ performs well on this dataset, we are reassured about its generalization performance and have more confidence in its prediction on $\boldsymbol{x} \in \Omega \backslash (\Omega_{train} \cup \Omega_{val} \cup \Omega_{test})$ that we have no information about.

2.3.3. Underfitting and Overfitting

If $\mathbf{\Phi}$ performs poorly on Ω_{train}, we say it suffers from **underfitting** or bias. If its performance is much poorer on any subset of $\Omega \backslash \Omega_{train}$ than on Ω_{train}, we say it endures **overfitting** or variance. Obviously, both are sign of bad generalization performance.

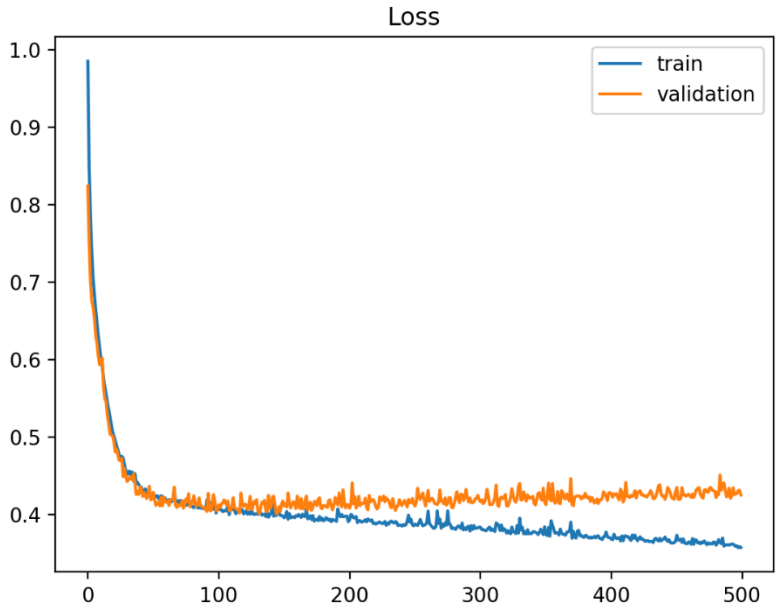

Figure 5. Typical learning curve of ANNs.

A typical learning curve of ANNs is demonstrated in Figure 5. Usually before $\mathbf{\Phi}$ is trained, unless $\mathbf{\Phi}$ is used to approximate zero function, it always has underfitting problem given that all its parameters $\boldsymbol{\theta}$ are randomly initialized to values close to zero by convention. After several training iterations, underfitting problem gets alleviated gradually while overfitting problem emerges and escalates as it goes on. If overfitting is severe, we can hardly have any confidence in $\mathbf{\Phi}$'s performance:

$$P(G|\bar{V}) = P(G)\frac{P(\bar{V}|G)}{P(\bar{V})} \simeq P(G)\frac{0}{P(\bar{V})} = 0 \qquad (2.3.2)$$

2.3.4. Regularization

Generally, for training an ANN **Φ**, most effort is devoted to tackling overfitting problem. Underfitting in most cases can be readily overcome by extending training time and increasing the representability of **Φ**, e.g. increasing its width, depth etc. Yet that in consequence induces overfitting problems given that the increased representability of **Φ** enables it to learn a much larger family of functions which can all generate the same training set. It is then natural to constrain trainable parameters to narrow **Φ** down to the vicinity of the true underlying function $\boldsymbol{f}^*(\boldsymbol{x})$. Any strategy used to alleviate overfitting but not aggravate underfitting much can be called **regularization**.

The more independent parameters an ANN has, the larger family of functions it can approximate. In light of this, regularization is about reducing independencies of ANN's parameters by imposing constraints, just like reducing degrees of freedom of a dynamic system. Due to ANNs' unique structure, constraints can be applied on:

Parameters

This kind of regularization includes parameter norm penalties, noise robustness of weights, parameter tying and sharing etc. It presumes a general relationship $R(\theta_1, \dots \theta_n) = 0$ among certain group of parameters $\theta_1, \dots \theta_n$ and enforce that relationship either by manipulating the optimization process or hard coding it into the computational graph.

For instance, norm constraints like L^1 and L^2 can be implemented by appending a penalty term of the form $\alpha \cdot R(\theta_1, \dots \theta_n)$ to the loss function where α is a weight coefficient. L^1 forces the parameters to become sparser while L^2 makes their magnitude smaller. They can also be implemented as explicit constraints by modifying the training algorithm. On the other hand,

convolutional layer is essentially a fully connected layer regularized by repetitive parameter sharing and sparsity. These two constraints are realized by developing a new layer operating with a convolutional kernel, instead of tediously maneuvering the weight matrix.

Input and Hidden Neurons

Regularization can also be applied on neurons of input layer and hidden layers. The outputs of neurons can be taken as features extracted from the input data by the ANNs. We can compel the features to obey certain rule of thumb so that any associated weight and bias parameters can be constrained in turn. Most popular regularization methods, like noise robustness of neurons, multitask learning, sparse representations, dropout, pooling, batch normalization fall into this type.

For instance, L^1 penalty applied on neurons can induce sparsity so that many neurons will be rendered zero and cancelled out. Consequently, if the ANN $\boldsymbol{\Phi}$ wants to reduce its loss, it must learn salient features through the few remaining neurons. Dropout can increase the independence of neurons by randomly dropping out some of them during each training iteration. Multitask training leverages the common sense that a person relies on the same visual system to analyze different objects, so neurons of lower levels should perform equally well in different tasks.

Outputs

Regularizations on outputs tell ANNs what their outputs should be like on $\Omega \backslash \Omega_{train}$. Unlike the 2 types of regularizations above, this one has an unparalleled advantage: it can be interpreted by human. Traditionally, it is about extending the training dataset by means like dataset augmentation and adversarial training. Dataset augmentation is carried out by injecting simulated data points $(\boldsymbol{x}_k, \boldsymbol{o}_k)$ into the dataset, in hope that $\boldsymbol{o}_k \approx \boldsymbol{f}^*(\boldsymbol{x}_k)$. Early stopping can also be taken as a regularization of this type by which the validation set virtually augments the

training set. Adversarial training is mainly used for classification, and it is based on the reasonable intuition that when performing prediction, a sophisticated ANN $\boldsymbol{\Phi}$ shouldn't be distracted by noise that won't affect human's perception, namely $\boldsymbol{\Phi}(\boldsymbol{x}_k + \boldsymbol{\varepsilon}) = \boldsymbol{f}^*(\boldsymbol{x}_k)$, where $\boldsymbol{\varepsilon}$ is noise.

Limitations of Traditional Regularizations

As to the first 2 types of regularization, the greatest drawback is that they are qualitative rather than quantitative, while, even worse, being directly applied on abstract mechanisms we cannot fully comprehend. We may have some general empirical instincts on how the ANNs should behave but can only convey these ideas to them through some vague instructions on their neural systems, like "extract important features!", "be resistant to noise!" etc. We can mandate that a weight matrix should be **properly** sparse by minimizing its L^1 norm together with the loss function, but how **exactly** sparse should it be?

These equivocal instructions may not be an big issue for classification problems like image classification or voice recognition, because we do not require the output quantity to be much precise as long as it stays at the right side of the decision threshold; we don't pay much attention to how exactly far the output is away from the threshold. Moreover, we still cannot precisely describe how our perception distinguishes different objects, so we won't lose a dime for being ambiguous about our instructions. Half a loaf is always better than none.

For regression problems, however, we expect ANNs to output precise results lying within the small neighborhood of exact outputs, not just away from a boundary. The situation can exacerbate even more when Ω_{train} contains only a few data points highly polluted by noise and sporadically sprayed over Ω or densely concentrated somewhere. No vague intuition can work for these demanding tasks other than useful information pertaining to the exact outputs. The

aforementioned regularizations are therefore not as helpful to regression as to classification. The third kind of regularization directly applied on the outputs, such as data augmentation, seems to be a savior, yet existing methods have inherent restrictions. One ineluctable question is, how do we use fragmented, distorted information associated with the outputs to produce fake outputs? A reliable scientific simulation requires precise information about the parameters of the system, the initial and boundary conditions, and the governing equations. In reality, nevertheless, most of the time some of them are missing while the rest are garbled by noise. Dataset augmentation is therefore more likely to be futile than we thought.

Despite all that, regularization on outputs still proves to be the right direction for successfully solving regression problems. A new approach is called for.

CHAPTER 3

Literature Review of Physics Informed Learning

3.1. Chronology

3.1.1. Pre-2015 Studies, the Era of CPU and Human Labor

Artificial neural networks, especially MLPs, had been enchanting researchers from fields outside computer science ever since the end of the last century. Though most of these explorations were about MLPs' application on datasets unusual to computer scientists and those MLPs were trained using hackneyed data-driven method as widely seen today, some started to look into MLPs' inventive usage.

MLPs for Differential Equations

Because of MLPs' great capability of function approximation[6], a few researchers investigated whether they can serve as trial solution for initial-boundary value problems (IBVPs) in place of tradition trial solutions in numerical methods. Lagaris et al.[8] first proposed an MLP-based hybrid trial solution comprising two parts that transforms initial-boundary value problems governed by ordinary and partial differential equations (ODEs, PDEs) into unconstrained optimization problems, then solved them using collocation methods. Soon afterwards, they realized this hybrid trial solution's limitation on complicated boundary conditions and figured out a penalty approach that leaves the problem as constrained optimization to overcome this issue[9]. About the same time, He et al.[10] applied MLPs to the control of nonlinear systems governed by a class of first-order PDEs. Aarts and Veer[11] also independently discovered MLP-based numerical methods similar to Lagaris' work. Alli et al.[12] further extrapolated Lagaris' work to vibration control problems. Malek and Beidokhti[13] derived ways to formulate hybrid trial solutions for high order ODEs based on Lagaris' research, and later on extended their

research to PDEs[14]. Shirvany et. al[15] used these methods to solve non-linear Schrodinger equations. Effati and Pakdaman[16] solved fuzzy differential equations using MLPs. Masmoudi et. al[17] found the solution to the Hamilton-Jacobi-Bellman equation using 2 coupled MLPs. Reynaldi et. al[18] discretized PDEs with finite element method, designed n-input-m-output MLPs corresponding to n-element-m-node, and optimized the FEM loss function with Levenberg-Marquardt algorithm. Mall et.al[19] integrated Chebyshev expansion with MLP to form a Chebyshev neural network and solved Lane–Emden type equations.

MLPs for Variational Problems

Apart from differential equations, physics systems can also be described by variational formulations. However, not much research effort had been spent on MLPs' application on variational problems even today, let alone before 2015. The most substantial and comprehensive research on this topic was carried out by Lopez et. al[20] in which they used MLPs of only three hidden neurons to solve a large variety of variational problems and evaluated integrations involved using Runge-Kutta integration method.

Wearisome Backpropagation

Researchers in this period must have been hassled a lot by the tedious evaluation of analytical gradients. The loss functions of these studies are all intricate, many even contain high order derivatives of MLPs, even more exhausting is the derivation of their backpropagation formula. Software like Mathematica might mitigate some pain but compiling training algorithms is still fatiguing. Some researchers used numerical differentiation methods like finite difference instead, but their accuracy cannot always be guaranteed. Inadequate computational power of hardware at that time could also have been an impedance to the prosperity of this niche area.

3.1.2. Post-2015 Studies, the Era of GPU and TensorFlow

No sooner had Raina et. al[21] introduced GPU to training neural networks than we entered the 10s. Ever since that debut of GPU's monstrous computational power, the progress of research in machine learning area has been boosted to a great extent. At the end of 2015, Google Brain unveiled TensorFlow, a Python library with automatic differentiation embedded that liberates researchers from onerous tasks of analytical gradient derivation, and at the start of 2017 this framework eventually became full-fledged, can operate on multiple CPUs and GPUs. This could probably explain why research in PIL blossomed since 2017.

From PDE Solving to Physics Informed Learning

Raissi et. al[22] first realized the aforementioned MLP training algorithms on TensorFlow by leveraging automatic differentiation and solved a variety of non-linear PDEs including Brown equations, Schrodinger equations etc. Then they combined this purely numerical method based training approach with traditional data-driven approach and came up with the idea of PIL, by which not only can sparse dataset be used to reliably train neural networks, but also inversely determine unknown parameters of PDEs[23]. About the same time, Karpatne et. al[24] came up with a similar methodology to model lake temperature though their physics information was not PDE but inequality. Thereafter, Raissi's group further used PIL to solve high dimensional PDEs like Navier-Stokes equations[25], 100D Black-Scholes-Barenblatt and Hamilton-Jacobi-Bellman stochastic differential equations[26] etc., and integrated PIL with multistep methods to solve complicated dynamics systems like Lorenz system, Hopf bifurcation[27] etc. Sirignano and Spiliopoulos improved the static collocation approach in Raissi's work with stochastic gradient descent and proposed a Monte Carlo method for fast computation of second order derivatives[28]. Anitescu et. al[29] developed an adaptive

collocation strategy to increase the accuracy of Raissi's work. Berg and Nystrom[30] came up with a hybrid trial solution comprised of three sub-networks to overcome the issue induced by complex boundary conditions in Lagaris' work. Chen et. al[31] discretized Helmholtz equation using FEM and inversely determined space dependent relative permittivity modelled by an MLP. Kharazmi et. al[32] used sine functions and Legendre polynomials as weight functions and implemented moment method instead of collocation method to solve PDEs using MLPs. Meng and Karniadakis[33] realized a composite neural network that learns from multi-fidelity data cooperating with PIL based on Raissi's work. Zhu et. al[34] applied physic informed learning to dense convolutional encoder-decoder network and solved stochastic PDE problems like Darcy flow with high dimensional input fields. Lutter et. al[35] incorporated Lagrangian Mechanics into deep learning to learn an inverse dynamic model for robot control.

Other Important Works

There is no wonder why PIL has such an intimacy with differential equations. Differential equation is the lingua franca long developed to formulate physics phenomenon with both accuracy and succinctness. Sometimes, though, a simpler formulation can suffice to be beneficial. Stewart and Ermon[36] devised a way to instruct CNN how to tract falling object simply by embedding gravity rule in the loss function without labelling the location of object on each frame. Variational formulations from which PDEs are derived can also be an alternative way to describe physics systems. Weinan and Yu[37] developed Deep Ritz Method which can avoid the hassle of numerical integration while solving variational problems using MLPs.

It was also shown that physics laws can be learned by neural networks from data. Iten et. al[38] demonstrated how a variational autoencoder can learn physics law from dynamic systems' data then even answer questions based what it learned, and discovered the linear correlation

between autoencoder's latent variables and the parameters of the PDEs describing the systems. Dong et. al[39] proposed PDE-NET, which uses convolutional kernels to approximate differential operators and learn differential equations from data.

3.2. Physics Informed Learning with Explicit Expression

Simply put, PIL embeds human's knowledge about the underlying function $\boldsymbol{f}^*$ into the training process of the ANN $\boldsymbol{\Phi}$ pertaining to $\boldsymbol{f}^*$ to compensate for scanty training data. Though in most cases we do not know $\boldsymbol{f}^*$'s explicit expression, knowledge about $\boldsymbol{f}^*$ can usually be represented in the form of implicit expression:

$$\boldsymbol{R}\big(\boldsymbol{f}^*(\boldsymbol{x})\big) = 0 \tag{3.2.1}$$

where the operation $\boldsymbol{R}$ may or may not involve integral or differential operators and the underlying mapping $\boldsymbol{f}^*$ is assumed to be continuous. Sometimes this formula may not be an equation but an inequality.

The strategy of PIL is highly dependent on the form of the physics information provided. We shall instructively start from simpler scenarios where the explicit expression of $\boldsymbol{f}^*$ is known.

3.2.1. Trivial Case: $\boldsymbol{\Phi}(\boldsymbol{x}) = \boldsymbol{f}^*(\boldsymbol{x})$

It spares a lot of hassle when we know $\boldsymbol{f}^*$ explicitly. If the explicit expression of $\boldsymbol{f}^*(\boldsymbol{x})$ is given and we want to approximate it with ANN $\boldsymbol{\Phi}$, we can generate arbitrarily abundant dataset to train $\boldsymbol{\Phi}(\boldsymbol{x})$. The loss function can thus be designed as:

$$L = \frac{1}{N_c}\sum_{\boldsymbol{x}_i \in \Omega_c} |\boldsymbol{\Phi}(\boldsymbol{x}_i;\boldsymbol{\theta}) - \boldsymbol{f}^*(\boldsymbol{x}_i)|^2 . \tag{3.2.2}$$

where Ω_c is an N_c-element subset extracted from domain Ω with certain scheme. This seems trivial considering that it is ridiculous to not use $\boldsymbol{f}^*(\boldsymbol{x})$ already known for prediction.

Even when the coefficients $\boldsymbol{\lambda}$ of $\boldsymbol{f}^*(\boldsymbol{x}_i;\boldsymbol{\lambda})$ are undetermined while only a small training set $\mathcal{X} = \{(\boldsymbol{x}_i, \boldsymbol{o}_i)\}_{i=1}^N$ is available, it is still much more ideal to take $\boldsymbol{\lambda}$ as trainable parameters

then optimize $\boldsymbol{f}^*(\boldsymbol{x};\boldsymbol{\lambda})$ instead of $\boldsymbol{\Phi}(\boldsymbol{x};\boldsymbol{\theta})$ against $\mathcal{X}$ using the MSE loss function below because generalization performance can then be sufficiently guaranteed:

$$L = \frac{1}{N}\sum_{\boldsymbol{x}_i,\boldsymbol{o}_i\in\mathcal{X}} |\boldsymbol{f}^*(\boldsymbol{x}_i;\boldsymbol{\lambda}) - \boldsymbol{o}_i|^2 . \tag{3.2.3}$$

However, training $\boldsymbol{\Phi}$ makes great sense when $\boldsymbol{f}^*$ is not the mapping we are interested in. There are two elementary cases where this could happen.

3.2.2. Elementary Case 1: $\boldsymbol{\Phi} \circ \boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{f}^*(\boldsymbol{x})$

In this case, we need to find $\boldsymbol{\Phi}$ given an intermediate function $\boldsymbol{g}(\boldsymbol{x})$ **either continuous or discrete** for which $\boldsymbol{\Phi}(\boldsymbol{u}) = \boldsymbol{\Phi}\big(\boldsymbol{g}(\boldsymbol{x})\big) = \boldsymbol{f}^*(\boldsymbol{x})$ though the coefficients $\boldsymbol{\lambda}$ of $\boldsymbol{f}^*$ are not completely known, while barely any training data exists such that $\boldsymbol{\Phi}$ cannot be well trained with data-driven method. The loss function then becomes:

$$L = \frac{1}{N}\sum_{\boldsymbol{u}_i,\boldsymbol{o}_i\in\mathcal{X}} |\boldsymbol{\Phi}(\boldsymbol{u}_i;\boldsymbol{\theta}) - \boldsymbol{o}_i|^2 + \frac{1}{N_c}\sum_{\boldsymbol{x}_i\in\Omega_c} |\boldsymbol{\Phi}(\boldsymbol{g}(\boldsymbol{x}_i);\boldsymbol{\theta}) - \boldsymbol{f}^*(\boldsymbol{x}_i;\boldsymbol{\lambda})|^2 \tag{3.2.4}$$

where the first term is a data-driven MSE loss defined over training set $\mathcal{X}$, Ω_c is an N_c-element subset of domain Ω and $N_c \gg N$ usually. In order to minimize the loss L, both $\boldsymbol{\theta}$ and $\boldsymbol{\lambda}$ should be treated as trainable parameters and optimized by training algorithm. Under some lucky circumstances the unknown coefficients $\boldsymbol{\lambda}$ can be eliminated in advance after solving $\nabla_{\boldsymbol{\lambda}} L = \boldsymbol{0}$.

Example: Label-free Object Recognition

An exemplary work of this kind was carried out by Steward and Ermon[36], in which Φ: $\mathbf{R}^{\mathrm{m}\times\mathrm{n}\times 3} \to \mathbf{R}$ is a CNN trained with no label to recognize a falling object, a pillow to be specific, from video frame $\boldsymbol{x} \in \mathbf{R}^{\mathrm{m}\times\mathrm{n}\times 3}$ and yield the height approximation $\hat{y} \in \mathbf{R}$ of its exact height y not yet provided. Since video frame and height are both functions of time, they can be denoted as $\boldsymbol{x}(t)$ and $y(t)$, and the CNN Φ for object recognition should satisfy $\Phi\big(\boldsymbol{x}(t)\big) = y(t)$. From Newton's second law of motion, we know explicitly that:

$$y(t) = y_0 + v_0 t + at^2 \tag{3.2.5}$$

where y_0 and v_0 are unknown coefficient initial height and initial velocity respectively and a is gravitational acceleration which is known to be 9.8 m/s^2. Since digital video is shot at a specified frame rate $1 \,/\, \Delta t$, an N-frame footage $\mathbf{x}$ can be taken as a sequence of frames $\boldsymbol{x}_i$ captured at time $i\Delta t$, in other words, $\mathbf{x} = [\boldsymbol{x}_i]_{i=1}^{N}$ where $\boldsymbol{x}_i = \boldsymbol{x}(i\Delta t)$. For a perfect approximation $\hat{y}_i$ at time $i\Delta t$,

$$\hat{y}_i = y_0 + v_0 \cdot i\Delta t + a(i\Delta t)^2, \tag{3.2.6}$$

which can be rearranged into:

$$\hat{y}_i - a(i\Delta t)^2 = y_0 + v_0 \cdot i\Delta t. \tag{3.2.7}$$

Following that, we can transform this relationship into its linear algebraic form:

$$\hat{\boldsymbol{y}} - \boldsymbol{a} = \boldsymbol{A} \cdot \begin{bmatrix} v_0 \\ y_0 \end{bmatrix} \tag{3.2.8}$$

where

$$\hat{\boldsymbol{y}} = \Phi \odot \mathbf{x} = [\Phi(\boldsymbol{x}_1) \quad \Phi(\boldsymbol{x}_2) \quad \cdots \quad \Phi(\boldsymbol{x}_n)]^{\mathrm{T}},$$

$$\boldsymbol{a} = [a\Delta t^2 \quad a(2\Delta t)^2 \quad \cdots \quad a(N\Delta t)^2]^{\mathrm{T}},$$

$$\boldsymbol{A} = \begin{bmatrix} \Delta t & 2\Delta t & \cdots & N\Delta t \\ 1 & 1 & \cdots & 1 \end{bmatrix}^{\mathrm{T}}.$$

Fortunately the unknown coefficients v_0 and y_0 here can be analytically derived after using least squares method to minimize the defect $\hat{\boldsymbol{y}} - \boldsymbol{a} - \boldsymbol{A} \cdot [v_0 \quad y_0]^{\mathrm{T}}$, or equivalently finding $[v_0 \quad y_0]^{\mathrm{T}}$ that satisfies $\nabla_{[v_0 \quad y_0]^{\mathrm{T}}} L = \mathbf{0}$ where $L = \frac{1}{N}\sum_{i=1}^{N} |\Phi(\boldsymbol{x}(i\Delta t); \boldsymbol{\theta}) - y(i\Delta t; v_0, y_0)|^2$:

$$\begin{bmatrix} v_0 \\ y_0 \end{bmatrix} = (\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A})^{-1}\boldsymbol{A}^{\mathrm{T}}(\hat{\boldsymbol{y}} - \boldsymbol{a}). \tag{3.2.9}$$

Plugging 3.2.9 into 3.2.8, the unknown coefficients can be eliminated, and we get:

$$\hat{\boldsymbol{y}} = \boldsymbol{a} + \boldsymbol{A}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A})^{-1}\boldsymbol{A}^{\mathrm{T}}(\Phi \odot \mathbf{x} - \boldsymbol{a}). \tag{3.2.10}$$

Then the training process of this CNN may require no training label but merely the discretized gravitational rule 3.2.10 according to which the loss function transforms into:

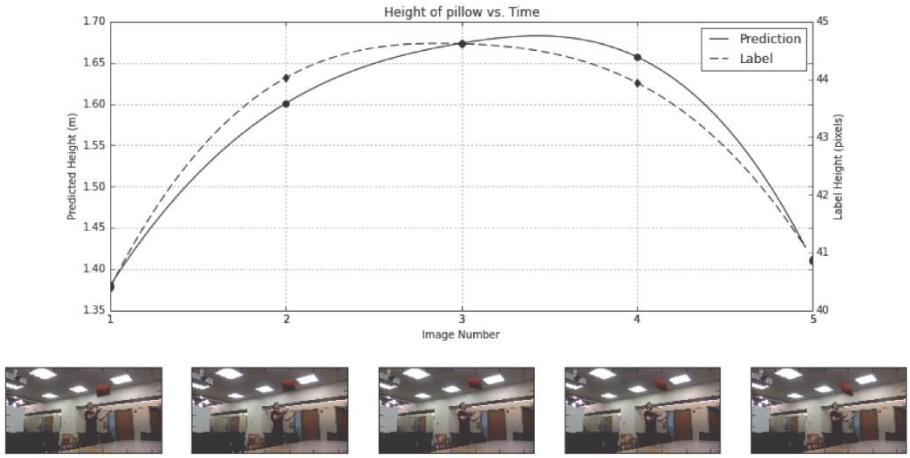

Figure 6. The prediction result of the CNN trained with a loss function with only gravitational law involved compared with the exact height.

$$L = \frac{1}{N}\sum_{i=1}^{N}|\Phi(\boldsymbol{x}_i) - \hat{y}_i|^2 . \tag{3.2.11}$$

The CNN's prediction result compared with real label is shown in Figure 6.

3.2.3. Elementary Case 2: $\boldsymbol{g} \circ \boldsymbol{\Phi}(\boldsymbol{x}) = \boldsymbol{f}^*(\boldsymbol{x})$

In this case even when the explicit expression of $\boldsymbol{f}^*$ and $\boldsymbol{g}$ are exactly known still can we not generate training set for $\boldsymbol{\Phi}$ if $\boldsymbol{g}^{-1}$ is not known. So, the loss function is designed as follows:

$$L = \frac{1}{N}\sum_{\boldsymbol{x}_i,\boldsymbol{o}_i\in\mathcal{X}}|\boldsymbol{\Phi}(\boldsymbol{x}_i;\boldsymbol{\theta}) - \boldsymbol{o}_i|^2 + \frac{1}{N_c}\sum_{\boldsymbol{x}_i\in\Omega_c}|\boldsymbol{g}(\boldsymbol{\Phi}(\boldsymbol{x}_i);\boldsymbol{\theta}) - \boldsymbol{f}^*(\boldsymbol{x}_i;\boldsymbol{\lambda})|^2 . \tag{3.2.12}$$

In fact, the function $\boldsymbol{g}$ can be regarded as a new layer which makes $\boldsymbol{g} \circ \boldsymbol{\Phi}(\boldsymbol{x})$ a new ANN, then it can reduce to a traditional data-driven training problem when $\boldsymbol{\lambda}$ is known. Unlike the previous case, however, $\boldsymbol{g}$ **should not be discrete**, otherwise backpropagation cannot be conducted.

Example: Lake Temperature Modeling

For instance, Karpatne et. al[24] modelled lake temperature T using an MLP $\Phi(\boldsymbol{D}, Y_{PHY})$ which takes both the input driver $\boldsymbol{D}$ and the output Y_{PHY} of a physics based numerical model as input to predict T. Two of $\boldsymbol{D}$'s components are depth d and time t, and it is known that the density ρ and temperature T of water are non-linearly related to each other through:

$$\rho = g(T) = 1000\left(1 - \frac{(T + 288.9414)(T - 3.9863)^2}{508929.2(T + 68.12963)}\right). \tag{3.2.13}$$

As is known, at any time t the density ρ should be a monotonic increasing function of depth d, so that $\rho(d_i, t) \leq \rho(d_{i+1}, t)$ if $d_i \leq d_{i+1}$. Then apart from training Φ with collected data using a data-driven loss function, a physics informed regularization term can also be appended to make sure of a reasonable temperature prediction abiding by the density monotonicity:

$$L = L_{data} + \frac{1}{N_d N_t}\sum_{i=1}^{N_d}\sum_{j=1}^{N_t} \text{ReLU}\left(g \circ \Phi(d_i, t_j) - g \circ \Phi(d_{i+1}, t_j)\right), \qquad d_i \leq d_{i+1} \tag{3.2.14}$$

where ReLU is rectified linear unit minimizing which to 0 can we make certain:

$$\rho(d_i, t_j) = g \circ \Phi(d_i, t_j) \leq g \circ \Phi(d_{i+1}, t_j) = \rho(d_{i+1}, t_j), \qquad d_i \leq d_{i+1}$$

3.2.4. Advanced Cases

With these 2 basic components at hand, any graph approximating $\boldsymbol{f}^*$ with $\boldsymbol{\Phi}$ engaged can be constructed, such as $\boldsymbol{g}_2 \circ \boldsymbol{\Phi} \circ \boldsymbol{g}_1(\boldsymbol{x}) = \boldsymbol{f}^*(\boldsymbol{x})$. The explicit expression of $\boldsymbol{f}^*(\boldsymbol{x}; \boldsymbol{\lambda})$ can then be used to generate dataset when $\boldsymbol{\lambda}$ is given, or otherwise regularize $\boldsymbol{\Phi}$ when data is inadequate or noisy.

3.3. Physics Informed Learning with Implicit Expression

Notwithstanding its convenience, explicit expression of the underlying function $\boldsymbol{f}^*$ is often hard to come by. Implicit expressions of the form $\boldsymbol{R}(\boldsymbol{f}^*(\boldsymbol{x})) = \boldsymbol{0}$ account for a dominant part of PIL, especially the ones with differential or integral operators.

In general, to incorporate knowledge embedded in the implicit expression of $\boldsymbol{f}^*$ into the training of ANN $\boldsymbol{\Phi}$ which approximates $\boldsymbol{f}^*$, we can simply substitute $\boldsymbol{\Phi}(\boldsymbol{x})$ for $\boldsymbol{f}^*(\boldsymbol{x})$ in the implicit expression to form a physics informed residual $\boldsymbol{R}(\boldsymbol{\Phi}(\boldsymbol{x}_i; \boldsymbol{\theta}); \boldsymbol{\lambda})$ parameterized by $\boldsymbol{\lambda}$ which is not always known, and append its L^2 norm as physics informed loss to the loss function:

$$L = \frac{1}{N}\sum_{\boldsymbol{x}_i, \boldsymbol{o}_i \in \mathcal{X}} |\boldsymbol{\Phi}(\boldsymbol{x}_i; \boldsymbol{\theta}) - \boldsymbol{o}_i|^2 + \frac{1}{N_c}\sum_{\boldsymbol{x}_i \in \Omega_c} \left|\boldsymbol{R}(\boldsymbol{\Phi}(\boldsymbol{x}_i; \boldsymbol{\theta}); \boldsymbol{\lambda})\right|^2 \tag{3.3.1}$$

where the first term is an ordinary data-driven loss defined over a training set $\mathcal{X}$, Ω_c is a subset of N_c collocation points $\boldsymbol{x}_i$ sampled over Ω. Usually $N_c \gg N$ so that physics knowledge can make up for a limited $\mathcal{X}$. We shall see later that the physics informed residual can also have other forms after proper transformation.

Apparently the case $\boldsymbol{\Phi}(\boldsymbol{x}) = \boldsymbol{f}^*(\boldsymbol{x})$ is no longer trivial for implicit expression. Of course, just like PIL with explicit expression, if a $\boldsymbol{\Phi}$ approximating $\boldsymbol{f}^*$ is not what we desire, we can simply replace $\boldsymbol{\Phi}$ with whatever composition such as $\boldsymbol{\Phi} \circ \boldsymbol{g}$, $\boldsymbol{g} \circ \boldsymbol{\Phi}$ and so forth. For most cases of implicit expression, though, $\boldsymbol{\Phi}(\boldsymbol{x}) = \boldsymbol{f}^*(\boldsymbol{x})$ is what we care about and we shall stick mainly with it in the following sections.

Most physics systems are formulated by implicit expressions with differential and integral operators. For one comprised of neither of them but some basic operations and elementary functions, minimizing loss function 3.3.2 directly is able to obtain good result. More concentration shall be exerted on differential equations and variational formulations, whose physics informed residual often have to be processed by proper numerical methods at first.

3.3.1. Strong Formulation and Weak Formulation

Almost every system in disciplines like physics, engineering, biology, economics etc. can be described by differential equations. In terms of engineering, from the simplest systems like Newton's second law of motion of a single particle, to more intricate systems like solid mechanics, magnetohydrodynamics etc., differential equations succinctly state the relations between different rates of change such as velocity, acceleration, curvature, stress, flux and so on.

Most of these seemingly intricate formulations are established on some surprisingly simple principles. For instance, convection equation $\frac{\partial U}{\partial t} + \vec{v} \cdot \nabla U = 0$ and diffusion equation $\frac{\partial U}{\partial t} - \nabla \cdot (\mu \nabla U) = 0$ are both derived from conservation law $\frac{d}{dt}\int_\Omega U \, dA + \int_{\delta\Omega} \vec{F}(U) \cdot \vec{n} \, ds =$

$\int_\Omega S(U,t)\,dA$ after applying divergence theorem. Dynamic equilibrium for elasticity $\nabla \cdot \boldsymbol{\sigma} + \boldsymbol{f} = \rho \cdot \boldsymbol{a}$ is the Euler-Lagrange equation of Hamilton's principle $\delta \int_{t1}^{t2} [T - (U + V)] dt = 0$, which is the time integration of d'Alembert's principle, a generalization of Newton's second law for multiple particles. For conciseness and generality, we move every term of a differential equation to its left and denote it as $\boldsymbol{R}\big(\boldsymbol{u}(\boldsymbol{x})\big) = 0$, where $\overline{\boldsymbol{x}}$ is the input of the solution $\boldsymbol{u}$, $\boldsymbol{R}$ is an operation involving differential operators as seen in the examples above.

Differential equations are assumed to hold true for $\forall \boldsymbol{x}$ inside the domain Ω in question. Their exact solutions $\boldsymbol{u}(\boldsymbol{x})$ are required to satisfy their boundary conditions $\boldsymbol{R}_b\big(\boldsymbol{u}(\boldsymbol{x})\big) = 0$ on $\partial\Omega$ and be of high smoothness. Because of these relatively strong, demanding requisites, they are usually referred to as **strong formulation** or strong form. It is then appealing to relieve the stringency somewhat, given that we may not care about the value of high order derivatives of the solutions, and we don't need a solution strictly satisfying the differential equation at every location. Such equivalent formulations of lower demands are called **weak formulation** or weak form, their solutions $\widehat{\boldsymbol{u}}(\boldsymbol{x}; \boldsymbol{\theta})$ are called **weak solution**, or **trial solution** as they are parameterized by $\boldsymbol{\theta}$.

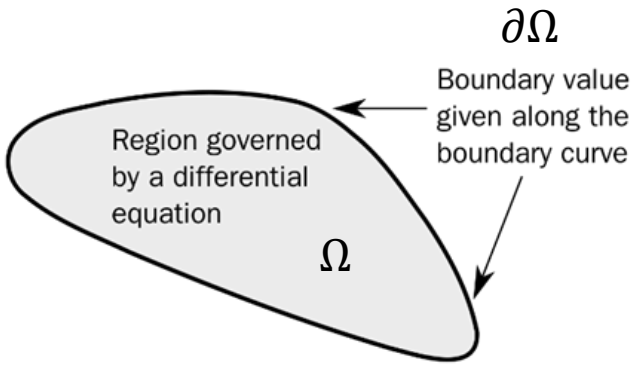

Figure 7. Domain and boundary.

The canonical method for transforming differential equations into weak form is weighted residual methods (WRM). For a differential equation with boundary condition:

$$\boldsymbol{R}\big(\boldsymbol{u}(\boldsymbol{x})\big) = 0, \qquad \boldsymbol{x} \in \Omega \tag{3.3.3}$$
$$\boldsymbol{R}_b\big(\boldsymbol{u}(\boldsymbol{x})\big) = 0, \qquad \boldsymbol{x} \in \partial\Omega \tag{3.3.4}$$

we first define the differential equation's residual by substituting trial solution $\widehat{\boldsymbol{u}}$ for $\boldsymbol{u}$ to get $\boldsymbol{R}\big(\widehat{\boldsymbol{u}}(\boldsymbol{x};\boldsymbol{\theta})\big)$ and $\boldsymbol{R}_b\big(\widehat{\boldsymbol{u}}(\boldsymbol{x};\boldsymbol{\theta})\big)$. Then, we require these residuals to be orthogonal to a finite series of weights or **test functions** $\boldsymbol{v}_i(\boldsymbol{x})$ and $\boldsymbol{w}_i(\boldsymbol{x})$, such that:

$$\langle \boldsymbol{v}_i, \boldsymbol{R} \rangle = \int_\Omega \boldsymbol{v}_i(\boldsymbol{x}) \cdot \boldsymbol{R}\big(\widehat{\boldsymbol{u}}(\boldsymbol{x};\boldsymbol{\theta})\big)\,\mathrm{d}\boldsymbol{x} = 0 \tag{3.3.5}$$
$$\langle \boldsymbol{w}_j, \boldsymbol{R}_b \rangle = \int_{\partial\Omega} \boldsymbol{w}_j(\boldsymbol{x}) \cdot \boldsymbol{R}_b\big(\widehat{\boldsymbol{u}}(\boldsymbol{x};\boldsymbol{\theta})\big)\,\mathrm{d}\boldsymbol{x} = 0. \tag{3.3.6}$$

For the very exact solution $\boldsymbol{u}$, its residuals, which are always zero, obviously should be orthogonal to any test function. This is usually dubbed **the principle of virtual work**. The finiteness of test function series here lowers down that requirement for the trial solution $\widehat{\boldsymbol{u}}$, demanding the residuals to be orthogonal only to the subspaces spanned by the test functions. On top of that, the order of differential operators in $\boldsymbol{R}$ may be reduced after implementing integration by parts such that the smoothness of $\widehat{\boldsymbol{u}}$ does not have to be as high as $\boldsymbol{u}$, and the natural boundary conditions in 3.3.4 and 3.3.6 can hereby be eliminated.

Depending on the choice of test functions $\boldsymbol{v}_i$ and $\boldsymbol{w}_j$, WRM can differentiate into several specific methods, for example:

- Collocation method: $\boldsymbol{\delta}(\boldsymbol{x} - \boldsymbol{x}_i)$;
- Galerkin method: $\partial\widehat{\boldsymbol{u}}/\partial\theta_i$;
- Least squares method: $\partial\boldsymbol{R}/\partial\theta_i$;
- Method of moment: $\boldsymbol{x}^i$.

In most cases, Galerkin method is preferable because of its precision and the symmetric coefficient matrices it generates. These methods are easy to execute if $\widehat{\boldsymbol{u}}$ is properly designed.

Traditionally, trial solutions $\widehat{\boldsymbol{u}}$ are constructed by linearly combining a finite group of functions, by which the exact solution can be approximated with a finite set of parameters θ_i:

$$\widehat{\boldsymbol{u}}(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{\phi}_0(\boldsymbol{x}) + \sum_{i=1}^{N} \theta_i \cdot \boldsymbol{\phi}_i(\boldsymbol{x}) \tag{3.3.7}$$

where $\boldsymbol{\phi}_0$ is an arbitrary function satisfying only essential boundary conditions of 3.3.4, and $\boldsymbol{\phi}_i(\boldsymbol{x})$ are called **basis functions** or shape functions, which equal zero on the boundary $\partial\Omega$ and vary with different methods. For instance, finite element method uses compactly supported piecewise polynomials as basis functions, such as chapeau-hat and piecewise cubic polynomial. Meshfree methods rely on radial basis functions like Gaussian. Spectral methods adopt sinusoidal functions to approximate periodic solutions faster. Though trial solution 3.3.7 has the advantage of making 3.3.6 trivial, sometimes $\boldsymbol{\phi}_0$ is troublesome to find. We can then delete $\boldsymbol{\phi}_0$ and use the rest as trial solution after complementing it with some additional basis functions not satisfying zero boundary conditions:

$$\widehat{\boldsymbol{u}}(\boldsymbol{x};\boldsymbol{\theta}) = \sum_{i=1}^{N} \theta_i \cdot \boldsymbol{\phi}_i(\boldsymbol{x}), \tag{3.3.8}$$

Once the basis functions and test functions are determined, after integration by parts, Equation 3.3.5 and 3.3.6 can be evaluated and translated into a system of linear equations solving which can we get $\boldsymbol{\theta}$ for the desired weak solution $\widehat{\boldsymbol{u}}$. The integral evaluation can even be carried out analytically for some low dimensional traditional trial solutions because of their linear structure and their basis functions being elementary functions well understood. Detailed information can be found in texts like [40] and [41].

3.3.2. Physics Informed Learning with Differential Equations

MLPs vs Traditional Trial Solutions

What most researchers in Section 3.1 were basically doing is replacing the summation in trial solution 3.3.7 and 3.3.8 with a well-designed MLP and solving its weak form. For 3.3.7, as can be seen in [8] [13] etc., its counterpart with an MLP $\boldsymbol{\Phi}$ integrated is of the form:

$$\widehat{\boldsymbol{u}}(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{\Psi}\big(\boldsymbol{\Phi}(\boldsymbol{x};\boldsymbol{\theta})\big) = \boldsymbol{A}(\boldsymbol{x}) + \boldsymbol{F}\big(\boldsymbol{\Phi}(\boldsymbol{x};\boldsymbol{\theta})\big) \tag{3.3.9}$$

where $\boldsymbol{A}(\boldsymbol{x})$ is equivalent to $\boldsymbol{\phi}_0(\boldsymbol{x})$ in 3.3.7, but satisfying all boundary conditions including the natural ones, and $\boldsymbol{F}$ is an operation which may contain differential operators and aims to make $\boldsymbol{F}\big(\boldsymbol{\Phi}(\boldsymbol{x};\boldsymbol{\theta})\big)$ satisfy zero boundary condition. On the other hand, for 3.3.8 its counterpart is simply the MLP itself, as presented in [5] [9] etc.:

$$\widehat{\boldsymbol{u}}(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{\Phi}(\boldsymbol{x};\boldsymbol{\theta}), \tag{3.3.10}$$

and this one, just like 3.3.8, is prevalently used because of its convenience, as we shall see later.

These replacements can be justified not only rigorously by the universal approximation theorem[6], but also intuitively by the resemblance between MLPs and traditional trial solutions. To start, we notice that trial solution 3.3.8 can be represented in its matrix form:

$$\widehat{\boldsymbol{u}}(\boldsymbol{x};\boldsymbol{\theta}) = \sum_{i=1}^{N} \theta_i \cdot \boldsymbol{\phi}_i(\boldsymbol{x}) = [\boldsymbol{\phi}_1(\boldsymbol{x}) \quad \cdots \quad \boldsymbol{\phi}_N(\boldsymbol{x})] \cdot \boldsymbol{\theta} = \boldsymbol{N} \cdot \boldsymbol{\theta}. \tag{3.3.11}$$

Convenient though this form absolutely is for deriving coefficient matrices in WRM, it blurs the fact that each dimension of $\widehat{\boldsymbol{u}}$ uses the same set of scalar basis functions $\phi_i(\boldsymbol{x})$ such that the array of θ_i for each dimension is decoupled from that for the other. For instance, the basis function matrix $\boldsymbol{N}_e$ of a 3D tetrahedra element in FEM is of the form:

$$\boldsymbol{N}_e = \begin{bmatrix} \phi_1 & 0 & 0 & \phi_2 & 0 & 0 & \phi_3 & 0 & 0 & \phi_4 & 0 & 0 \\ 0 & \phi_1 & 0 & 0 & \phi_2 & 0 & 0 & \phi_3 & 0 & 0 & \phi_4 & 0 \\ 0 & 0 & \phi_1 & 0 & 0 & \phi_2 & 0 & 0 & \phi_3 & 0 & 0 & \phi_4 \end{bmatrix} \tag{3.3.12}$$

all dimensions of which share piecewise basis functions $\{\phi_1, \phi_2, \phi_3, \phi_4\}$ in common. The 3D output field inside that very element can be easily derived by $\widehat{\boldsymbol{u}}_e(\boldsymbol{x}) = \boldsymbol{N}_e \cdot \boldsymbol{\theta}_e$, and the global

field over Ω can be assembled in a piecewise manner by $\widehat{\boldsymbol{u}}(\boldsymbol{x}) = \sum \widehat{\boldsymbol{u}}_e(\boldsymbol{x})$, which apparently uses the same set of basis functions for each dimension. Therefore, each dimension $\hat{u}_j$ of $\widehat{\boldsymbol{u}}$ is parameterized by a separate set of parameters θ_{ij}.

In light of this, traditional trial solution $\widehat{\boldsymbol{u}}$ in 3.3.11 can be equivalently written in the form below after folding the vector $\boldsymbol{\theta}$ into a matrix $\boldsymbol{W}$ and squeezing out all redundant components of $\boldsymbol{N}$ before flattening the remnant into a vector $\boldsymbol{\phi}$:

$$\widehat{\boldsymbol{u}}(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{W} \cdot \boldsymbol{\phi}(\boldsymbol{x}) = \left[\theta_{ij}\right] \cdot [\phi_1(\boldsymbol{x}) \quad \cdots \quad \phi_n(\boldsymbol{x})]^{\mathrm{T}}. \tag{3.3.13}$$

Now recall the expression 2.1.3 for MLPs. If we take $\sigma_{n-1}\left(\boldsymbol{z}_{n-1}\left(\ldots\sigma_1\left(\boldsymbol{z}_1(\boldsymbol{x})\right)\right)\right)$ as a whole, then choose identity function as activation for σ_n and set the bias of $\boldsymbol{z}_n$ to zero, and denote this MLP likewise as $\widehat{\boldsymbol{u}}$, it becomes:

$$\widehat{\boldsymbol{u}}(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{W}_n \cdot \boldsymbol{\phi}(\boldsymbol{x};\boldsymbol{\theta}'), \qquad \boldsymbol{\phi}(\boldsymbol{x};\boldsymbol{\theta}') = \sigma_{n-1}\left(\boldsymbol{z}_{n-1}\left(\ldots\sigma_1\left(\boldsymbol{z}_1(\boldsymbol{x})\right)\right)\right), \boldsymbol{\theta} = \{\boldsymbol{W}_n, \boldsymbol{\theta}'\}, \tag{3.3.14}$$

which, compared with 3.3.13, can be regarded as a trial solution with not just $\boldsymbol{W}$, but also its basis functions ϕ_i, i.e. the elements of $\boldsymbol{\phi}(\boldsymbol{x};\boldsymbol{\theta}')$ to be determined. Since ϕ_i can approximate a large family of functions, if they are trained to fit common basis functions, 3.3.14 reduces to a traditional trial solution. For example, as can be readily noticed, the chapeau-hat basis function widely employed in FEM can be produced by the sum of 2 ReLUs, so that any continuous piecewise linear function assembled by chapeau-hats can be constructed by a shallow MLP with adequate number of neurons activated by ReLU, as well explained in [42]. This resemblance also explains why the trial solutions of Meshfree method are alternatively called radial basis function networks (RBFN) and can be trained likewise on data. Indeed, all the traditional trial solutions can be taken as shallow ANNs and trained on data so long as they have proper basis functions.

Despite MLP's affinities to traditional trial solutions, there are some conspicuous distinctions between them. The most prominent one is that the basis functions of MLPs are

unspecified complicated nonlinear function compositions determined during training, so that there is no way to construct a coefficient matrix by WRM and solve the problem with linear algebra. In addition, MLPs don't suffer much from the curse of dimensionality. One severe drawback of methods like FEM and Meshfree is their compactly supported basis functions whose parameters are their locations in the domain Ω, such that the number of θ_i and that of basis functions required to spread over the entire domain increases exponentially with the problem's number of dimensions. MLPs, in contrast, adaptively learn a rather small set of basis functions particularly suitable for the high dimensional problems in hand. This explains why they are preferred to model high dimensional mappings. Besides, unlike trial solutions of FEM, the outputs of MLPs are infinitely differentiable. This enables us to obtain high order fields like stress field through a few differentiations without interpolation, as presented in [25].

Selection of Weighted Residual Methods

For PIL, residuals from collocation method with Kronecker delta $\boldsymbol{\delta}(\boldsymbol{x} - \boldsymbol{x}_i)$ as test function, as already demonstrated in Equation 3.3.1, is prominently used among the existing papers. This is not only because it wards off the trouble of integral evaluation in Equation 3.3.5 and 3.3.6, but also that it has a close form to data-driven loss such that many researchers have acquaintance with it. Moreover, it is very convenient to sample collocation points, and that expedites the execution of stochastic method and adaptive method proposed in [28] and [29].

One prerequisite for the implementation of collocation method is its trial solution being adequately smooth. This is never a problem for MLP trial solutions with infinitely smooth C^∞ functions such as sigmoid, tanh etc. as activation. At times, though, the high order derivative of MLP is very hard to train, so we might want to lower down the smoothness requirement somewhat. Conventionally this is accomplished using methods other than collocation methods.

However, for MLPs, Galerkin method and least squares method are no longer suitable given how complicated MLPs' derivatives with respect to their parameters are and how many parameters θ_i they have. In consequence, only test functions independent of the MLP trial solutions like Legendre polynomials, Chebyshev polynomials and trigonometric functions can be chosen, as shown in [32]. Residuals in Equation 3.3.5 and 3.3.6 after integration by parts, briefly denoted by $\boldsymbol{R}'(\boldsymbol{v}_i(\boldsymbol{x}), \boldsymbol{\Phi}(\boldsymbol{x};\boldsymbol{\theta});\boldsymbol{\lambda})$ hereafter, can then be evaluated using numerical integration methods like Gaussian quadrature, which in essence are all weighted sum of values at collocation points properly chosen. For this purpose, the loss function 3.3.1 should be modified into the generic expression below for WRMs other than collocation method:

$$L = \frac{1}{N}\sum_{\boldsymbol{x}_i,\boldsymbol{o}_i\in\mathcal{X}} |\boldsymbol{\Phi}(\boldsymbol{x}_i;\boldsymbol{\theta}) - \boldsymbol{o}_i|^2 + \frac{1}{N_V}\sum_{\boldsymbol{v}_i\in\boldsymbol{V}}\left|\sum_{\boldsymbol{x}_j,\beta_j\in\mathcal{I}} \beta_j \boldsymbol{R}'\big(\boldsymbol{v}_i(\boldsymbol{x}_j), \boldsymbol{\Phi}(\boldsymbol{x}_j;\boldsymbol{\theta});\boldsymbol{\lambda}\big)\right|^2 \qquad (3.3.15)$$

where $\boldsymbol{V} = \{\boldsymbol{v}_i\}_{i=1}^{N_V}$ is a set of N_V test functions $\boldsymbol{v}_i$, and $\mathcal{I} = \{(\boldsymbol{x}_j, \beta_j)\}$ is a set of collocation point and weight pairs determined by the numerical integration scheme selected. When for the last term $\boldsymbol{v}_i = \boldsymbol{\delta}(\boldsymbol{x} - \boldsymbol{x}_i)$ and $\mathcal{I} = (\boldsymbol{x}_i, 1)$, Equation 3.3.15 reduces to Equation 3.3.1.

Differentiation of MLPs w.r.t Inputs

It goes without saying how important the evaluation of an MLP's gradient with respect to its input is for PIL with differential equations. Because of the clear analytical expression of MLPs, the calculation's feasibility is never our concern. What is more interesting is how to obtain the computational graph of the gradient effortlessly. As can be noticed in works before 2015 like [8] and [14], researchers used to waste a lot of time on deriving the analytical expressions for those derivatives.

Thanks to modern frameworks like TensorFlow and PyTorch wherein automatic differentiation is embedded, every operation a tensor undergoes through forward propagation can

be traced by gradient tape, so the computational graph of any gradient w.r.t that tensor can be derived at ease. These tools are what researchers rely on after 2015. Once an ANN $\boldsymbol{\Phi}(\boldsymbol{x})$ is defined in the framework, its first order gradient $\nabla_{\boldsymbol{x}}\boldsymbol{\Phi}$ or even higher ordered ones can be obtained on the fly for any $\boldsymbol{x}$. These derivatives can therefore be plugged into the physics informed residuals in Equation 3.3.1 and 3.3.15 for training. In special cases, we can also resort to numerical methods like finite difference to get the derivatives though not as accurately.

Example 1: Burgers' Equation

Raissi et. al solved a series of PDEs with PIL on TensorFlow[5]. One simple problem in his work good for illustration is Burgers' equation with small viscosity governed by the nonlinear PDE, initial condition and boundary condition below:

$$R(u) = u_t + uu_x - \left(\frac{0.01}{\pi}\right)u_{xx} = 0, \qquad x \in [-1, 1], \qquad t \in [0, 1] \tag{3.3.16}$$
$$u(0, x) = -\sin(\pi x), \qquad u(t, -1) = u(t, 1) = 0.$$

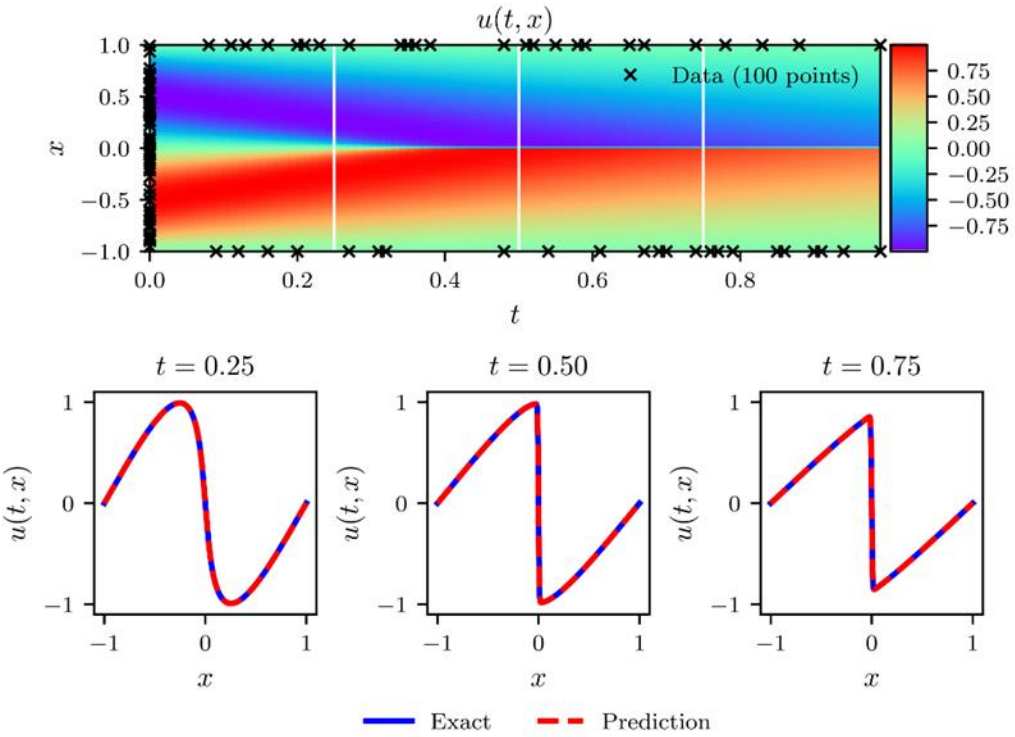

Figure 8. Burgers' equation: Top: Predicted solution $\hat{u}(t, x)$ along with the initial and boundary training data. Bottom: Comparison between predicted and exact solutions and three time points.

After constructing an MLP trial solution $\hat{u}(x,t;\boldsymbol{\theta})$ of the form in 3.3.10 and utilizing collocation method, the loss function for training $\hat{u}$ becomes:

$$L = \frac{1}{N_f}\sum_{t_i,x_i\in\Omega}^{N_f}\left|R\big(\hat{u}(x_i,t_i;\boldsymbol{\theta})\big)\right|^2 + \frac{1}{N_0}\sum_{x_i\in[-1,1]}^{N_0}|\hat{u}(x_i,0;\boldsymbol{\theta}) + \sin(\pi x_i)|^2 + \frac{1}{N_b}\sum_{t_i\in[0,1]}^{N_b}(|\hat{u}(1,t_i;\boldsymbol{\theta})|^2 + |\hat{u}(-1,t_i;\boldsymbol{\theta})|^2)\,. \tag{3.3.17}$$

where the first term is for the PDE, the second for the initial condition and the third for the boundary conditions. N_f, N_0 and N_b are the number of collocation points sampled from the entire (x,t) domain Ω, $x\in[-1,1]$ and $t\in[0,1]$ respectively. Derivatives $\hat{u}_t$, $\hat{u}_x$ and $\hat{u}_{xx}$ inside the residual $R(\hat{u})$ are obtained using the function $tf.gradient$ in TensorFlow. This loss is then minimized using gradient descent followed by L-BFGS-B, a quasi-Newton optimization algorithm. The final result is exhibited in Figure 8.

Example 2: Reflection on Examples in Section 3.2

As to the example of label-free object detection, its explicit governing equation 3.2.5 is derived from the simple ODE below:

$$\ddot{y} = a. \tag{3.3.18}$$

Since the analytical expression for $\partial\Phi\big(\boldsymbol{x}(t)\big)/\partial t$ is not achievable because $\boldsymbol{x}(t)$ is discrete, we can resort to finite difference and use the alternative loss function below to train Φ:

$$L = |\boldsymbol{D}\cdot\hat{\boldsymbol{y}} - \boldsymbol{a}|^2 \tag{3.3.19}$$

where:

$$\hat{\boldsymbol{y}} = \Phi\odot\mathbf{x} = [\Phi(\boldsymbol{x}_1)\quad\Phi(\boldsymbol{x}_2)\quad\cdots\quad\Phi(\boldsymbol{x}_n)]^{\mathrm{T}},\qquad \boldsymbol{a} = [a\quad a\quad\cdots\quad a]^{\mathrm{T}},$$

$$\boldsymbol{D} = \frac{1}{(\Delta t)^2}\begin{bmatrix} 2 & -5 & 4 & -1 & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & & \ddots & & \\ & & & -1 & 2 & -1 \\ & & -1 & 4 & -5 & 2 \end{bmatrix}.$$

The matrix $\boldsymbol{D}$ here is a finite difference operator for calculating the second order derivatives of $\widehat{\boldsymbol{y}}$.

On the other hand, the monotonicity of water density in lake temperature modeling problem can be achieved alternatively simply by forcing:

$$\frac{\partial \rho}{\partial d} \geq 0, \tag{3.3.20}$$

which in turn makes the corresponding loss function become:

$$L = L_{data} + \frac{1}{N}\sum_{d_i,t_i}^{N} \mathrm{ReLU}\left(-\frac{\partial g\big(\Phi(d_i,t_i)\big)}{\partial d}\right) \tag{3.3.21}$$

by leveraging the differentiability of $g \circ \Phi$.

3.3.3. Physics Informed Learning with Variational Formulation

Variational Formulation

Many differential equations have their corresponding variational formulations, in which a functional $J[\boldsymbol{u}]$ associated with the field $\boldsymbol{u}$ governed by the differential equation is required to be minimized. For instance, the principle of least action emerges in different areas from mechanics, electromagnetics to quantum mechanics, where the extrema of the action, basically the time integration of a Lagrangian, is obtained by finding the stationary point. A typical variational formulation is of the form:

$$J[\boldsymbol{u}] = \int_{\Omega} L\big(\boldsymbol{u}(\boldsymbol{x})\big)\,\mathrm{d}\boldsymbol{x} \tag{3.3.22}$$

$$\delta J[\boldsymbol{u}] = 0 \tag{3.3.23}$$

where $J[\boldsymbol{u}]$ is the functional associated with the field $\boldsymbol{u}$ whose argument is $\boldsymbol{x}$, and $L\big(\boldsymbol{u}(\boldsymbol{x})\big)$ is the Lagrangian of the variational formulation.

While variational formulations can be derived by applying the principle of virtual work to differential equations, differential equation is essentially the condition that the stationary point of

a variational formulation must satisfy. To illustrate this, suppose we have a variational formulation with the first order differentiation $\nabla \boldsymbol{u}$ engaged:

$$J[\boldsymbol{u}] = \int_{\Omega} L\big(\boldsymbol{x}, \boldsymbol{u}(\boldsymbol{x}), \nabla \boldsymbol{u}(\boldsymbol{x})\big)\, \mathrm{d}\boldsymbol{x}, \qquad \delta J[\boldsymbol{u}] = 0 \tag{3.3.24}$$

If we introduce an arbitrary perturbation $\varepsilon\boldsymbol{\eta}$ to $\boldsymbol{u}$, we can derive the stationary point $\boldsymbol{u}$ which satisfies $J[\boldsymbol{u}] \leq J[\boldsymbol{u} + \varepsilon\boldsymbol{\eta}]$ by forcing:

$$\int_{\Omega} \frac{dL\big(\boldsymbol{x}, (\boldsymbol{u} + \varepsilon\boldsymbol{\eta})(\boldsymbol{x}), \nabla(\boldsymbol{u} + \varepsilon\boldsymbol{\eta})(\boldsymbol{x})\big)}{d\varepsilon}\bigg|_{\varepsilon=0} \mathrm{d}\boldsymbol{x} = 0 \tag{3.3.25}$$

and we can eventually arrive at:

$$\int_{\Omega} \eta_k \left(\frac{\partial L}{\partial u_k} - \sum_i \frac{\partial}{\partial x_i} \frac{\partial L}{\partial u_{k_{(x_i)}}} \right) \mathrm{d}\boldsymbol{x} = 0. \tag{3.3.26}$$

Since $\boldsymbol{\eta}$ is arbitrary, the term inside the parenthesis must fulfill:

$$\frac{\partial L}{\partial u_k} - \sum_i \frac{\partial}{\partial x_i} \frac{\partial L}{\partial u_{k_{(x_i)}}} = 0, \tag{3.3.27}$$

which in fact is the differential equation governing $\boldsymbol{u}$, usually referred to as Euler-Lagrange equation of the variational formulation. The form of Equation 3.3.27 changes if the Lagrangian contains higher order differentiation, but the approach for finding stationary point remains the same.

Rayleigh-Ritz Method

The term on the left side of Euler-Lagrange equation is the physics informed residual we see a lot in Section 3.3.2. Since it results from Equation 3.3.23, it is viable to use 3.3.23 instead as the governing equation. This is indeed the idea behind Rayleigh-Ritz method, which is about substituting the trial solution $\widehat{\boldsymbol{u}}(\boldsymbol{x}; \boldsymbol{\theta})$ in 3.3.8 for $\boldsymbol{u}$ in Equation 3.3.23 so that:

$$\delta J[\widehat{\boldsymbol{u}}(\boldsymbol{x}; \boldsymbol{\theta})] = 0. \tag{3.3.28}$$

This is equivalent to finding $\boldsymbol{\theta}$ that satisfies:

$$\nabla_{\boldsymbol{\theta}} J[\widehat{\boldsymbol{u}}(\boldsymbol{x}; \boldsymbol{\theta})] = \mathbf{0}. \tag{3.3.29}$$

Simplifying this equation can we get the same linear algebraic equations derived from applying Galerkin method on its associated Euler-Lagrange equation.

Yet Equation 3.3.29 is not possible to be simplified if the trial solution is an MLP, whose basis functions are undetermined. To circumvent this issue, since Equation 3.3.23 is the same as finding extrema of 3.3.22, we can plug the MLP 3.3.10 into 3.3.22 and minimize it, in other words:

$$\arg\min_{\boldsymbol{\theta}} J\,,\;\; J = \int_{\Omega} L\big(\boldsymbol{\Phi}(\boldsymbol{x};\boldsymbol{\theta})\big)\,\mathrm{d}\boldsymbol{x}\,. \tag{3.3.30}$$

A loss function leveraging knowledge in the variational formulation can then be constructed as:

$$L = \rho_c \frac{1}{N} \sum_{\boldsymbol{x}_i,\boldsymbol{o}_i \in \mathcal{X}} |\boldsymbol{\Phi}(\boldsymbol{x}_i;\boldsymbol{\theta}) - \boldsymbol{o}_i|^2 + \rho_f \sum_{\boldsymbol{x}_i,\beta_i \in \mathcal{J}} \beta_i L\big(\boldsymbol{\Phi}(\boldsymbol{x}_i;\boldsymbol{\theta})\big) \tag{3.3.31}$$

where $\mathcal{J} = \{(\boldsymbol{x}_i, \beta_i)\}$ is a set of collocation point and weight pairs determined by the numerical integration scheme selected. ρ_c and ρ_f are weights for balancing these 2 terms so that the second term, which can be arbitrarily small, won't drown out the first term for data. We can get the desired solution after minimizing this loss.

Example: Catenary problem

One simple variational problem is catenary problem. In this problem the curve $y(x)$ assumed by a loose string of length l hung freely from two fixed point is to be find. Its constant length imposes a constraint below:

$$E_L\big(y(x)\big) = \int_{x_a}^{x_b} \sqrt{1 + y'(x)}\, dx - l = 0, \tag{3.3.32}$$

and the shape is found by minimizing the string's gravitational potential energy given by:

$$V\big(y(x)\big) = \int_{x_a}^{x_b} \rho y(x) \sqrt{1 + y'(x)}\, dx \tag{3.3.33}$$

The loss function for solving this problem with an MLP Φ is then constructed as follows:

$$L = \rho_L \left(E_L\big(\Phi(x)\big)\right)^2 + \rho_V V\big(\Phi(x)\big) \qquad (3.3.34)$$

where ρ_L and ρ_V are weights for balancing these 2 terms so that the constraint term always dominates over the potential energy term. The training result is shown in Figure 9.

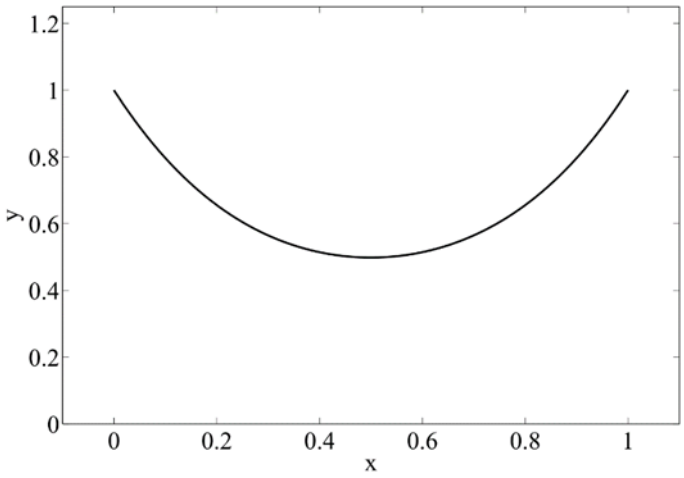

Figure 9. Neural network result of catenary problem.

CHAPTER 4

Artificial Neural Networks for Forward Vibration Problems

4.1. Motivation

Simulation is a quintessential part of product development. Specifically, it is about predicting the proposed physics system's behavior under specified external conditions via computation to estimate the system's performance before fabricating it to ensure its reliability. In essence, simulation techniques make that prediction by solving the differential equations governing the physics system after imposing specified initial and boundary conditions to obtain an approximated solution $\widehat{\boldsymbol{u}}(\boldsymbol{x}, t)$ where $\boldsymbol{x}$ is spatial location and t is time. One great drawback of these canonical simulations is their low efficiency when dealing with the same physics system under different conditions. Whenever the initial and boundary conditions alter, the whole time-consuming simulation process needs to be performed again from scratch to obtain new results. It is therefore tempting if we can attain a mapping $\widehat{\boldsymbol{u}}(\boldsymbol{x}, t, \boldsymbol{c})$ which not only takes $\boldsymbol{x}$ and t, but also some parts if not all of the initial and boundary conditions, namely $\boldsymbol{c}$ as inputs and yields the solution $\widehat{\boldsymbol{u}}$. If this mapping is attainable, then the solution for the same physics system under different conditions can be obtained almost in real time.

Surely there is a long way to go before we can eventually achieve the goal. Nevertheless, one thing we know for sure is that this proposed solution mapping $\widehat{\boldsymbol{u}}(\boldsymbol{x}, t, \boldsymbol{c})$, since it takes both the coordinates $\boldsymbol{x}, t$ and the discretized initial and boundary conditions $\boldsymbol{c}$ as input, corresponds to a high dimensional mapping. These trial solutions widely employed in traditional simulation methods, as introduced in Section 3.3.2, are constructed by linearly combining basis functions and are only suited for approximating the solution fields in low dimensional problems because they suffer from the curse of dimensionality. ANNs' unique multilayered architecture and their

resultant great representability make them so far the viable choice for this challenging task. In order to pave the way for achieving that ultimate high dimensional mapping $\widehat{\boldsymbol{u}}(\boldsymbol{x}, t, \boldsymbol{c})$, it is reasonable to first examine ANNs' performance as a low dimension solution $\widehat{\boldsymbol{u}}(\boldsymbol{x}, t)$.

This section will be focused on solving forward beam vibration problems using artificial neural networks trained with PIL technique and examining PIL's performance on high order differential equations. This part also serves as the foundation for inverse problems to be investigated shortly.

4.2. Beam Vibration Problem

Most beam vibration problems are based on Euler-Bernoulli beam theory. The dynamic beam equation is the Euler-Lagrange equation of the following action according to Hamilton's principle:

$$J[w] = \int_{t_1}^{t_2} \int_0^l \frac{1}{2}\rho A \left(\frac{\partial w}{\partial t}\right)^2 - \frac{1}{2} EI \left(\frac{\partial^2 w}{\partial x^2}\right)^2 + f(x,t) \cdot w \, \mathrm{d}x \,. \tag{4.2.1}$$

where E is the elastic modulus of the beam's material, I is the area moment of inertial, ρ is the density of the material, A is the cross section area, w the vertical displacement and $f(x,t)$ is external loading arbitrarily distributed along the beam and varies with time. The first term of the integrand stands for kinetic energy; the second one represents the potential energy caused by bending and the last one is the potential energy due to the external load $q(x)$. The dynamic beam

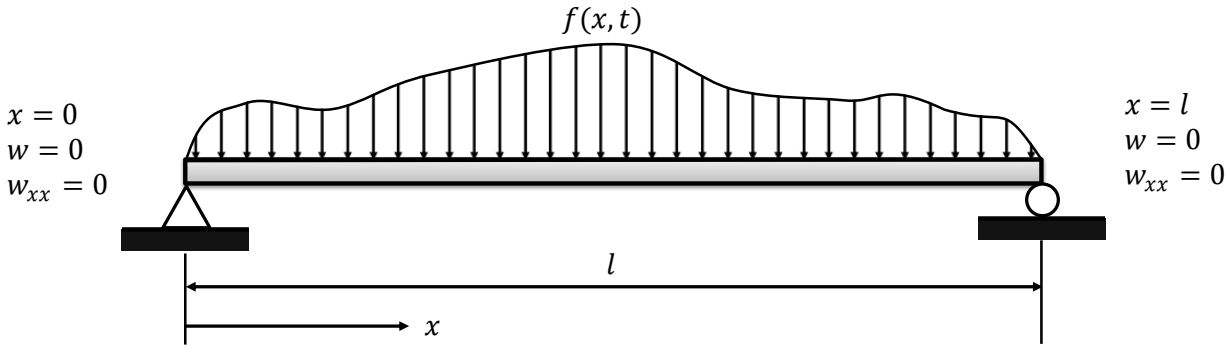

Figure 10. Setup of a simple supported beam.

equation is derived by finding the stational point of $J[w]$. For a simple supported beam as illustrated in Figure 10, if we assume that the beam is homogenous, i.e. the value of EI is independent of x, its displacement field should satisfy one dynamic beam equation which is a 4th order parabolic PDE, together with 2 initial conditions and 4 boundary conditions as follows:

$$EI\frac{\partial^4 w}{\partial x^4}+\rho A\frac{\partial^2 w}{\partial t^2}=f(x,t) \tag{4.2.2}$$
$$w(x,0)=g_1(x),\ \ w_t(x,0)=g_2(x)$$
$$w(0,t)=w(l,t)=0,\ \ w_{xx}(0,t)=w_{xx}(l,t)=0$$

As to the free vibration of a simple supported beam with unit length, the equation group above reduces to the one given below:

$$\lambda\frac{\partial^4 w}{\partial x^4}+\frac{\partial^2 w}{\partial t^2}=0,\qquad \lambda=\frac{EI}{\rho A} \tag{4.2.3}$$
$$w(x,0)=g_1(x),\ \ w_t(x,0)=g_2(x)$$
$$w(0,t)=w(1,t)=0,\ \ w_{xx}(0,t)=w_{xx}(1,t)=0$$

4.3. Collocation Method

For the time being in existing research works, PIL problems are mostly based on collocation method whose loss function is in the form of Equation 3.3.1. Though having demonstrated great accuracy in solving either linear or nonlinear differential equations in various papers, its performance on PDEs of higher order has not been adequately examined. Free vibration beam equation, which is of the 4th order, is both a significant problem in structural health monitoring area and an ideal touchstone for revealing undiscovered limitations of collocation method based PIL and inspiring ideas to further complement it.

4.3.1. Methodology

MLP Trial Solution

Different from what Alli et. al did in [12] for solving beam equation, here we use MLP trial solution in the form of Equation 3.3.10 to approximate the displacement field while integrating the initial and boundary conditions into the loss function to conduct constrained

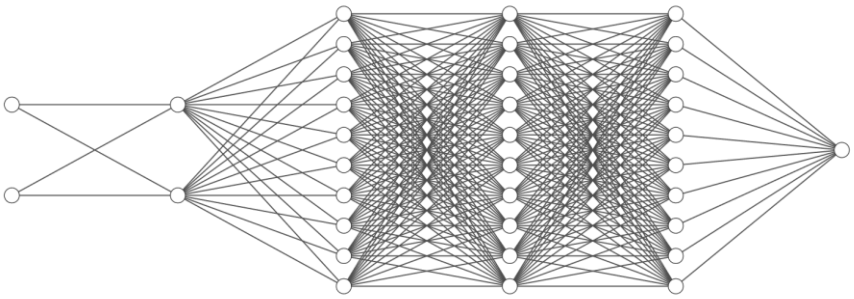

Figure 11. MLP trial solution with normalization layer for 1D vibration problems.

optimization, which is much easier to implement than the unconstrained optimization in their work and can be applied to more complicated cases, such as cantilever. In addition, we shall investigate MLP trial solution's performance on longer time span.

In this task MLP trial solutions $\widehat{w}(x,t;\boldsymbol{\theta})$ are constructed to approximate the exact displacement fields $w(x,t)$ governed by PDEs of 1D vibration problems after training. They should take time $t \in T$ and location $x \in X$ as input $\boldsymbol{x}$ and output the corresponding vertical displacement estimation $\widehat{w}$. x and t are first normalized to the range $[0,1]$ by a normalization layer before being fed to an MLP Φ. The expression for this normalization layer, which is equivalent to a fully connected layer, is:

$$\boldsymbol{f}_0(\boldsymbol{x}) = 2\frac{\boldsymbol{x}-\boldsymbol{lb}}{\boldsymbol{ub}-\boldsymbol{lb}} - \mathbf{1} = \begin{bmatrix} \frac{2}{X_{ub}-X_{lb}} & \\ & \frac{2}{T_{ub}-T_{lb}} \end{bmatrix} \begin{bmatrix} \bar{x} \\ \bar{t} \end{bmatrix} + \begin{bmatrix} \frac{X_{ub}+X_{lb}}{X_{lb}-X_{ub}} \\ \frac{T_{ub}+T_{lb}}{T_{lb}-T_{ub}} \end{bmatrix} = \boldsymbol{w}_0\boldsymbol{x} + \boldsymbol{b}_0 \quad (4.3.1)$$

$$\widehat{w} = \Phi\big(\boldsymbol{f}_0(\boldsymbol{x})\big) \quad (4.3.2)$$

Between the input layer comprising normalized x and t and the output layer yielding $\widehat{w}$ are hidden layers whose width and depth determine the representability of the MLP trial solution. Figure 11 shows the general structure of the MLP trial solution including the normalization layer for 1D vibration problems. We henceforth use the notation $[Input, Hidden, Output]$ to denote

the structure of the MLP abbreviating the normalization layer. Activation functions of hidden layers are all chosen to be $\tanh(x)$ if not mentioned otherwise.

Training

According to the general loss function for collocation method as shown in Equation 3.3.1, the loss function for solving beam vibration problem 4.2.3 using an MLP trial solution $\hat{w}(x,t;\boldsymbol{\theta})$ should be designed as follows:

$$L = \frac{\rho_f}{N_f}\sum_{x_i,t_i\in\Omega}^{N_f}\left|R\left(\hat{w}(x_i,t_i)\right)\right|^2 + \frac{\rho_0}{N_0}\sum_{x_i\in X}^{N_0}\sum_{j=1}^{2}\left|R_j^I\left(\hat{w}(x_i,0)\right)\right|^2 + \frac{\rho_b}{N_b}\sum_{t_i\in T}^{N_b}\sum_{j=1}^{4}\left|R_j^B\left(\hat{w}(x_j,t_i)\right)\right|^2 \quad (4.3.3)$$

where the first term is for minimizing the residual R of the governing equation, the second one for the residual R_j^I of the 2 initial conditions and the third one for R_j^B of the 4 boundary conditions. N_f, N_0 and N_b stand for the number of collocation points sampled from the domain Ω, X and T for governing equation, ICs and BCs respectively where $\Omega = X \times T$. ρ_f, ρ_0 and ρ_b are weights balancing these 3 terms for gradient based training algorithm.

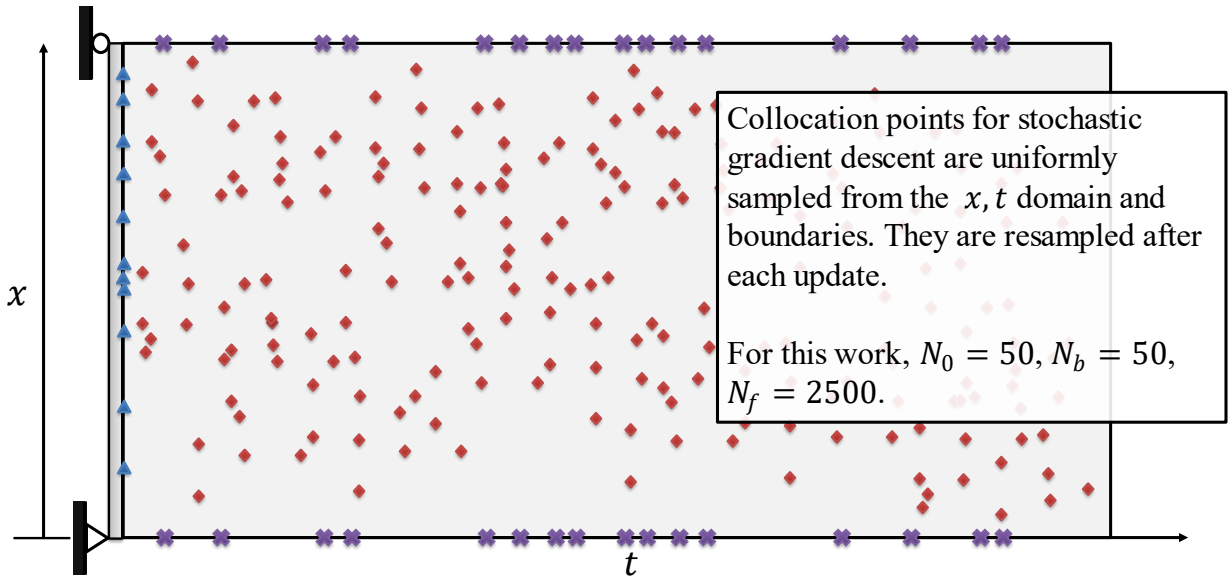

Figure 12. Collocation points for each iteration of stochastic gradient descent.

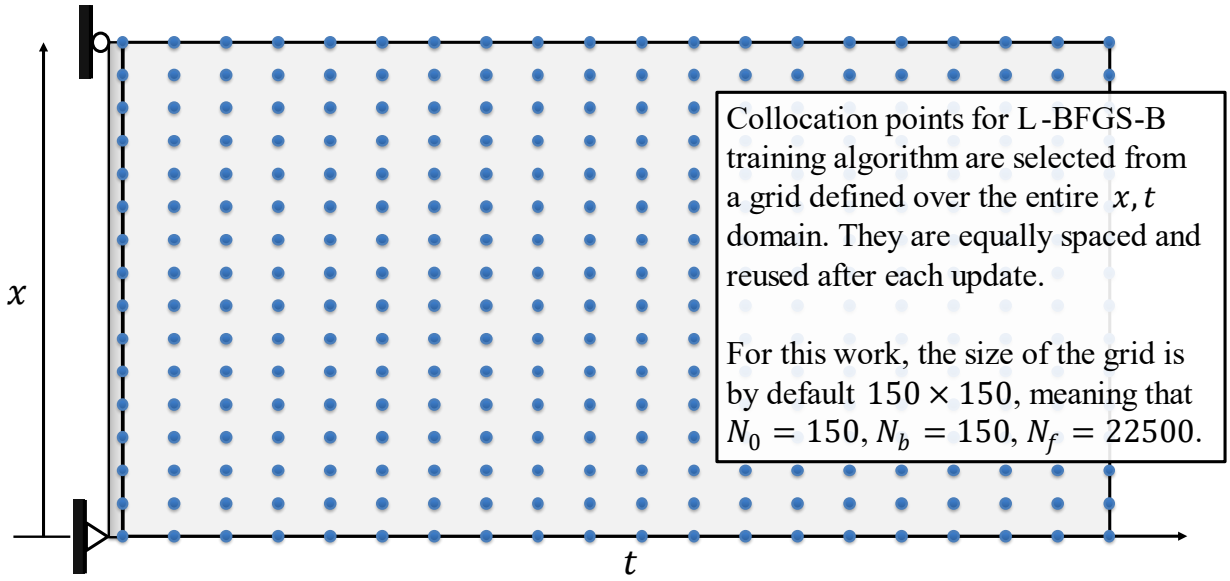

Figure 13. Collocation points for L-BFGS-B.

The parameters of the MLP $\hat{w}$ are trained with Adam optimizer with a learning rate of 0.01 for 10000 iterations at first. During each iteration, the collocation points are uniformly sampled from the domain of definition to conduct stochastic gradient descent. To be specific, $N_0 = 50$ collocation points are sampled for the initial conditions, $N_b = 50$ points are sampled for the boundary conditions and $N_f = 2500$ points are sampled for the internal domain which should satisfy the governing equation, as shown in Figure 12. Upon finished training by Adam optimizer, the parameters are then fine-tuned by L-BFGS-B on a larger number of static collocation points until the loss converges. The collocation points for L-BFGS-B optimizer are taken from an equally spaced 150×150 grid over the entire domain, so that $N_0 = N_b = 150$ and $N_f = 22500$. This is illustrated in Figure 13.

Free Vibration of Simple Supported Beam

Based on Equation 4.2.3, the loss function for $\hat{w}(x, t; \boldsymbol{\theta})$ approximating the displacement field of a simple supported beam's free vibration is:

$$L = \frac{\rho_f}{N_f}\sum_{x_i,t_i\in\Omega}^{N_f}\left|\left(\lambda\frac{\partial^4}{\partial x^4}+\frac{\partial^2}{\partial t^2}\right)\widehat{w}(x_i,t_i)\right|^2 + \frac{\rho_0}{N_0}\sum_{x_i\in X}^{N_0}(|\widehat{w}(x_i,0)-g_1(x_i)|^2+|\widehat{w}_t(x_i,0)-g_2(x_i)|^2) + \frac{\rho_b}{N_b}\sum_{t_i\in T}^{N_b}(|\widehat{w}(0,t_i)|^2+|\widehat{w}(1,t_i)|^2+|\widehat{w}_{xx}(0,t_i)|^2+|\widehat{w}_{xx}(1,t_i)|^2) \quad (4.3.4)$$

Free Vibration of String

The free vibration problem of a string shall be solved as a comparison to the beam vibration problem later. The governing equation, initial and boundary conditions for the free vibration of a string with unit length are given below:

$$\lambda\frac{\partial^2 w}{\partial x^2}-\frac{\partial^2 w}{\partial t^2}=0 \quad (4.3.5)$$
$$w(x,0)=g_1(x),\ \ w_t(x,0)=g_2(x),\qquad w(0,t)=w(1,t)=0$$

where the parameter ω makes sure that the first initial condition satisfies boundary conditions. The corresponding loss function for $\widehat{w}(x,t;\boldsymbol{\theta})$ should be:

$$L = \frac{\rho_f}{N_f}\sum_{t_i,x_i\in\Omega}^{N_f}\left|\left(\lambda\frac{\partial^2}{\partial x^2}-\frac{\partial^2}{\partial t^2}\right)\widehat{w}(x_i,t_i)\right|^2 + \frac{\rho_b}{N_b}\sum_{t_i\in T}^{N_b}(|\widehat{w}(0,t_i)|^2+|\widehat{w}(1,t_i)|^2) + \frac{\rho_0}{N_0}\sum_{x_i\in X}^{N_0}(|\widehat{w}(x_i,0)-g_1(x_i)|^2+|\widehat{w}_t(x_i,0)-g_1(x_i)|^2) \quad (4.3.6)$$

4.3.2. Results and Discussion

Baseline

A free vibration problem of simple supported beam is solved at first to ascertain PIL's feasibility to deal with high order differential equations and serve as the baseline for later investigations. Its parameters for loss function 4.3.4 are defined in Table 1.

Table 1. Problem 1: the baseline SSB problem

λ	T	X	ρ_f, ρ_b, ρ_0	Structure of $\widehat{w}$	Exact Solution $w(x,t)$
1	$[0,1]$	$[0,1]$	1, 1, 1	[2,100,100,100,1]	$\sin \pi x \cdot \cos \pi^2 t$

Figure 14 demonstrates the PIL result for this baseline problem. The first row presents the final approximated displacement field $\widehat{w}(x,t)$; the second row shows the discrepancy between the approximation $\widehat{w}(x,t)$ and the exact solution $w(x,t)$; the third row shows the residual $f_w(x,t) = \lambda \frac{\partial^4 \widehat{w}}{\partial x^4} + \frac{\partial^2 \widehat{w}}{\partial t^2}$ indicating how well $\widehat{w}$ satisfies the PDE and a detailed comparison between $\widehat{w}$ and w at three timestamps is shown in the last row. The weak solution agrees pretty well with the exact solution over the entire domain, with a maximum relative error around 2% at the very end of the time axis. Hence, PIL successfully solved this vibration problem.

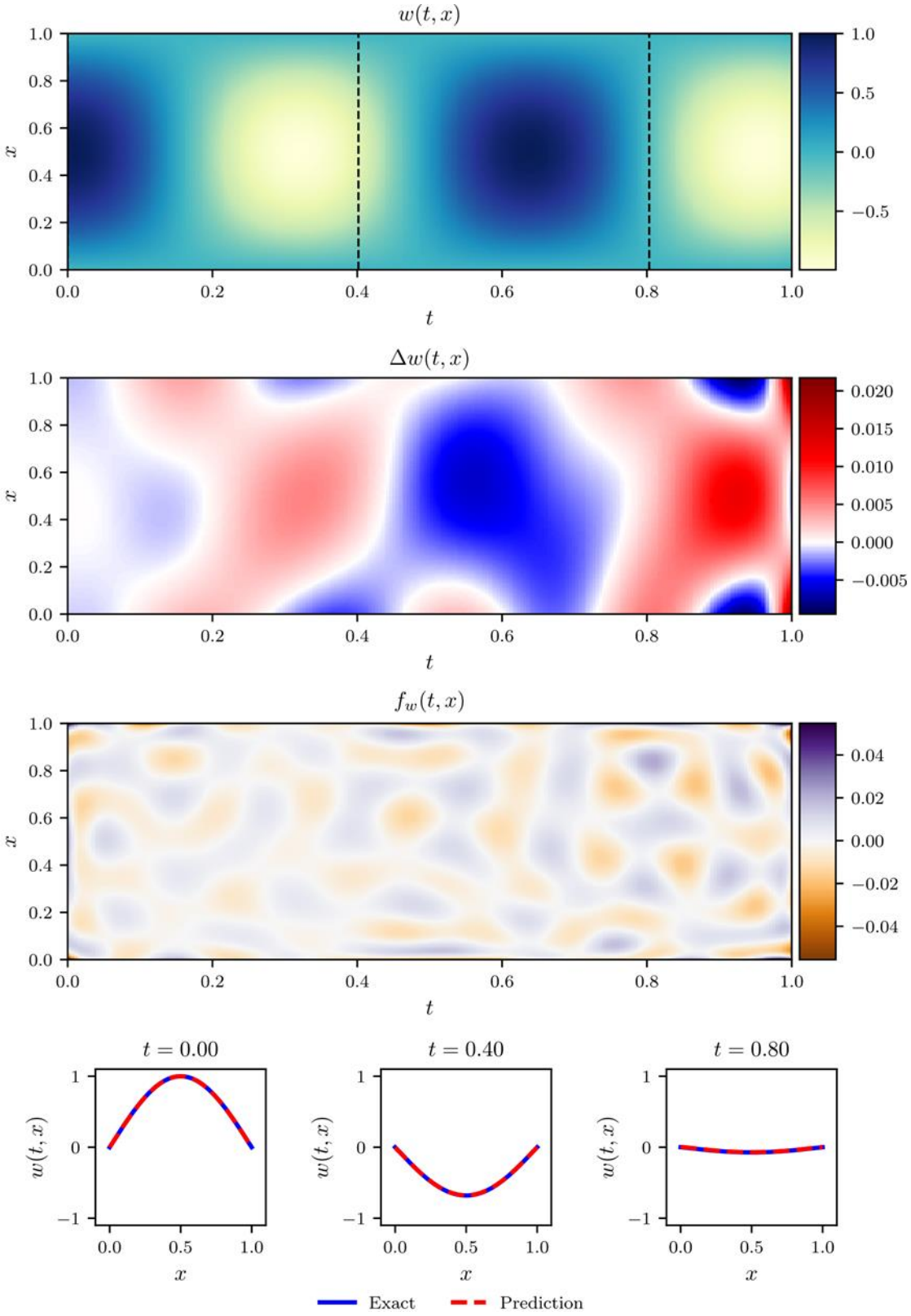

Figure 14. PIL solution for baseline SSB problem given in Table 1.

Trivial Solution Pitfall

Things go astray when the problem becomes tougher. When either the parameter λ increases or the upper bound of T extends, as shown in Table 2 and Table 3, the MLP trial solution began to suffer from a defect dubbed **trivial solution pitfall**, namely that some part of the approximated solution converges to trivial solution, as shown in Figure 15 and Figure 16. This is because constant value is a trivial solution to the governing equation $\lambda \frac{\partial^4 w}{\partial x^4} + \frac{\partial^2 w}{\partial t^2} = 0$ and the training algorithm cannot distinguish it from a meaningful solution, so that the region far away from the initial timestamp gravitates towards zero in order to lower down the overall loss, contradicting the region near $t = 0$. The collocation points locating at the rear part of the time axis then become a hindrance to the overall convergence of the MLP.

Table 2. Problem 2: SSB vibration with increased frequency

λ	T	X	ρ_f, ρ_b, ρ_0	Structure of $\widehat{w}$	Exact Solution $w(x,t)$
4	$[0, 1]$	$[0, 1]$	1, 1, 1	[2,100,100,100,1]	$\sin \pi x \cdot \cos 2\pi^2 t$

Table 3. Problem 3: SSB vibration with increased time span

λ	T	X	ρ_f, ρ_b, ρ_0	Structure of $\widehat{w}$	Exact Solution $w(x,t)$
1	$[0, 2]$	$[0, 1]$	1, 1, 1	[2,100,100,100,1]	$\sin \pi x \cdot \cos \pi^2 t$

Table 4. Problem 4: String vibration

λ	T	X	ρ_f, ρ_b, ρ_0	Structure of $\widehat{w}$	Exact Solution $w(x,t)$
1	$[0, 5]$	$[0, 1]$	1, 1, 1	[2,100,100,100,1]	$\sin 2\pi x \cdot \cos 2\pi t$

Table 5. Problem 5: SSB vibration with increased frequency and truncated time window

λ	T	X	ρ_f, ρ_b, ρ_0	Structure of $\widehat{w}$	Exact Solution $w(x,t)$
4	$[0, 0.5]$	$[0, 1]$	1, 1, 1	[2,100,100,100,1]	$\sin \pi x \cdot \cos 2\pi^2 t$

A series of deduction can be made about why this pitfall happens in problem 2 and 3 but not in 1. First, we assume that the representability of the network $[2, 100, 100, 100, 1]$ used in

problem 1 is not adequate for these 2 problems. Since for both problems the number of periods in the specified time span doubles, the displacement field becomes more undulating than before such that an MLP with more complexity might be necessary to represent this field. This guess can be refuted by solving a string vibration problem governed by a 2nd order PDE using loss function 4.3.8. The parameters for this problem are given in Table 4. Since the string vibration solution in Figure 17, which well fits the exact solution, is definitely more complicated than the solutions for problem 2 and 3, lack of representability is not the culprit for their failure. This also indicates that the order of PDE has a significant impact on the convergence of MLP solution.

Ruling out this probability, by comparing problem 1 and 3, we can infer that collocation method indeed has its coherent limitation when the trial solution's domain contains region "inappropriately" far away from its boundary where conditions are provided. Yet whether the threshold of this appropriateness is dependent of the other factors like λ still remains to be checked. To determine λ's effect, we notice that the MLPs Φ as part of the trial solutions for problem 2 and 3 should approximate the same function $f(x,t) = \sin \pi x \cdot \cos 2\pi^2 t$ because of the preceding normalization layer. They both failed to learn w so no clear inference can be made about λ. We then truncate the time window of problem 2 by half to get problem 5 given in Table 5, and it is easy to see that Φ should approximate the same function $f(x,t) = \sin \pi x \cdot \cos \pi^2 t$ for both problem 1 and 5. The approximated solution to problem 5 is given in Figure 18 and it fails to fit the exact solution. This proves that λ does have impact on the convergence of the trial solution.

Another potential factor worth investigating is the complexity of the MLP, specifically its depth and width. Though we proved that it has enough representability to approximate the displacement field, it is still possible that a simpler MLP structure can help with its convergence.

Problem 6, 7, 8 shown in Table 6, Table 7, Table 8 were solved to determine their effect and their MLP solution are given in Figure 19, Figure 20 and Figure 21. It appears that the complexity of the MLP really has great impact on the final result. If the complexity is too low, the MLP lacks the capability to represent the displacement field, as shown in Figure 21; if too high, it incurs difficulty to converge during training, as shown in Figure 18 and Figure 19; only when its complexity is in a reasonable range can it finally fall within the proximity of the exact solution, as shown in Figure 20. How to determine this range is still too difficult to address at this point.

It is also reasonable to try methods other than collocation method to see if this convergence problems can be mitigated, which we shall do next.

Table 6. Problem 6: SSB vibration with increased frequency, truncated time window and shallower MLP

λ	T	X	ρ_f, ρ_b, ρ_0	Structure of $\widehat{w}$	Exact Solution $w(x,t)$
4	$[0, 0.5]$	$[0, 1]$	1, 1, 1	[2,100,1]	$\sin \pi x \cdot \cos 2\pi^2 t$

Table 7. Problem 7: SSB vibration with increased frequency, truncated time window, narrower and shallower MLP

λ	T	X	ρ_f, ρ_b, ρ_0	Structure of $\widehat{w}$	Exact Solution $w(x,t)$
4	$[0, 0.5]$	$[0, 1]$	1, 1, 1	[2,50,1]	$\sin \pi x \cdot \cos 2\pi^2 t$

Table 8. Problem 8: SSB vibration with increased frequency, truncated time window, narrower and shallower MLP

λ	T	X	ρ_f, ρ_b, ρ_0	Structure of $\widehat{w}$	Exact Solution $w(x,t)$
4	$[0, 0.5]$	$[0, 1]$	1, 1, 1	[2,10,1]	$\sin \pi x \cdot \cos 2\pi^2 t$

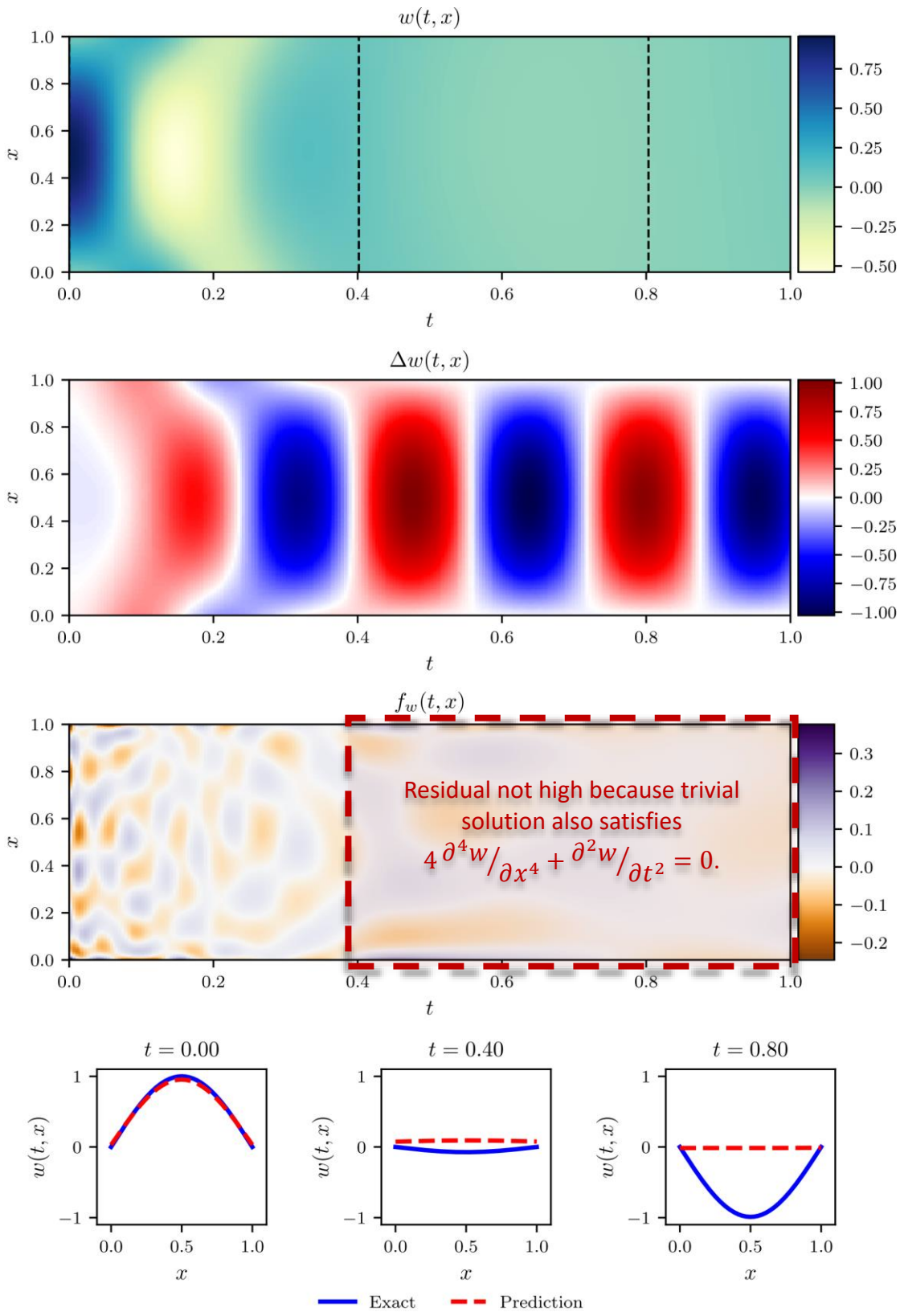

Figure 15. PIL solution for Problem 2: SSB vibration given in Table 2.

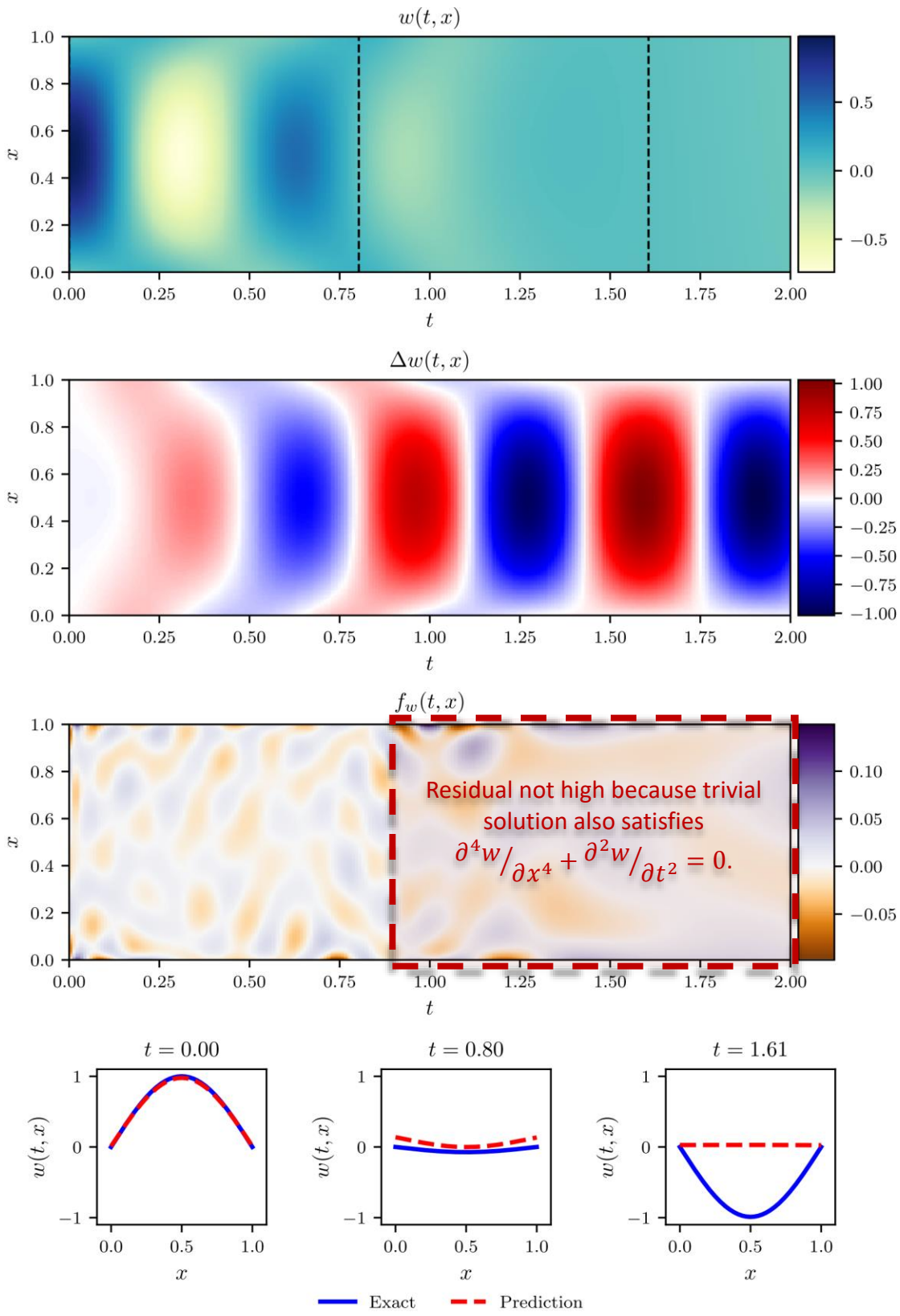

Figure 16. PIL solution for Problem 3: SSB vibration given in Table 3.

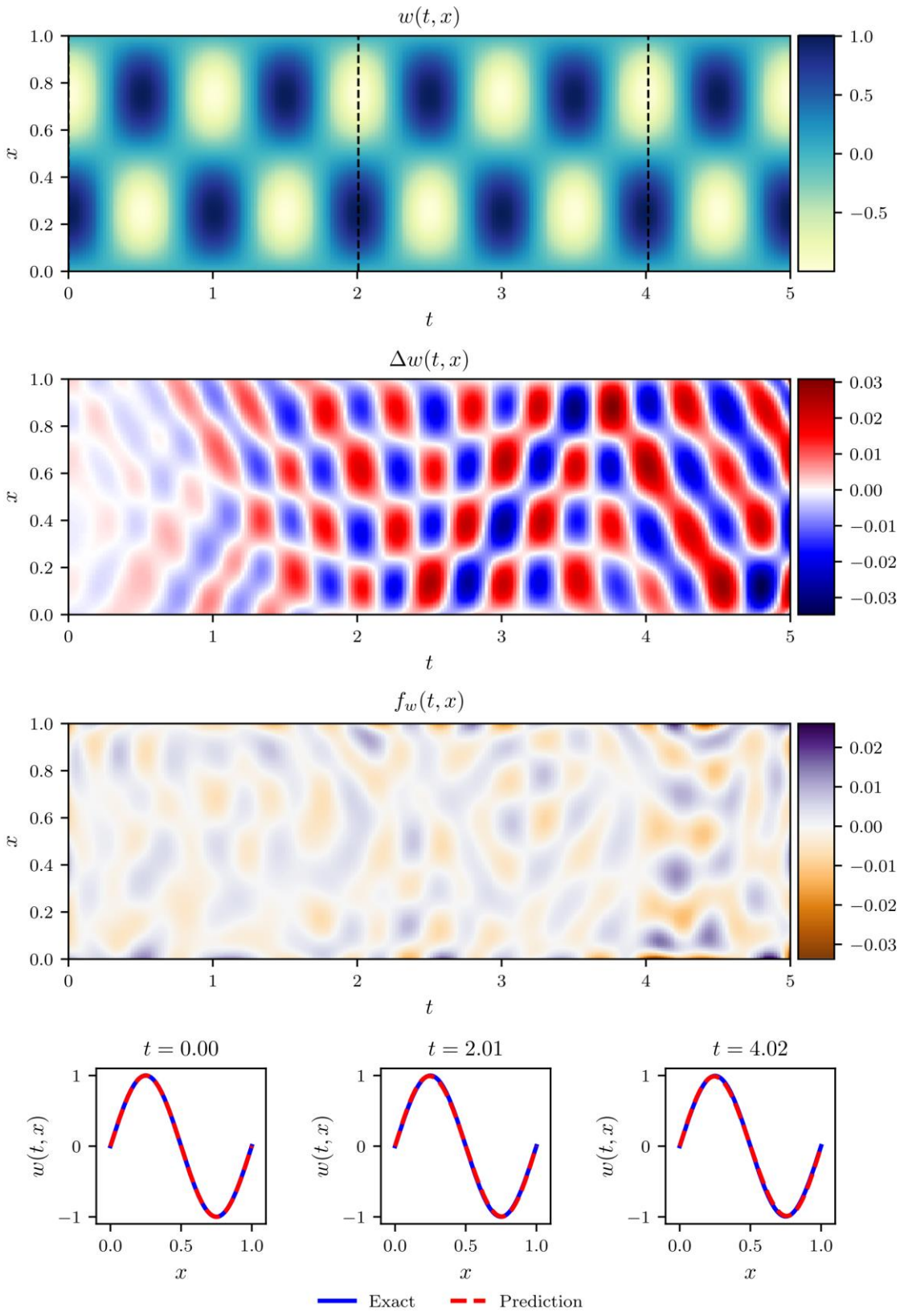

Figure 17. PIL solution for Problem 4: string vibration given in Table 4.

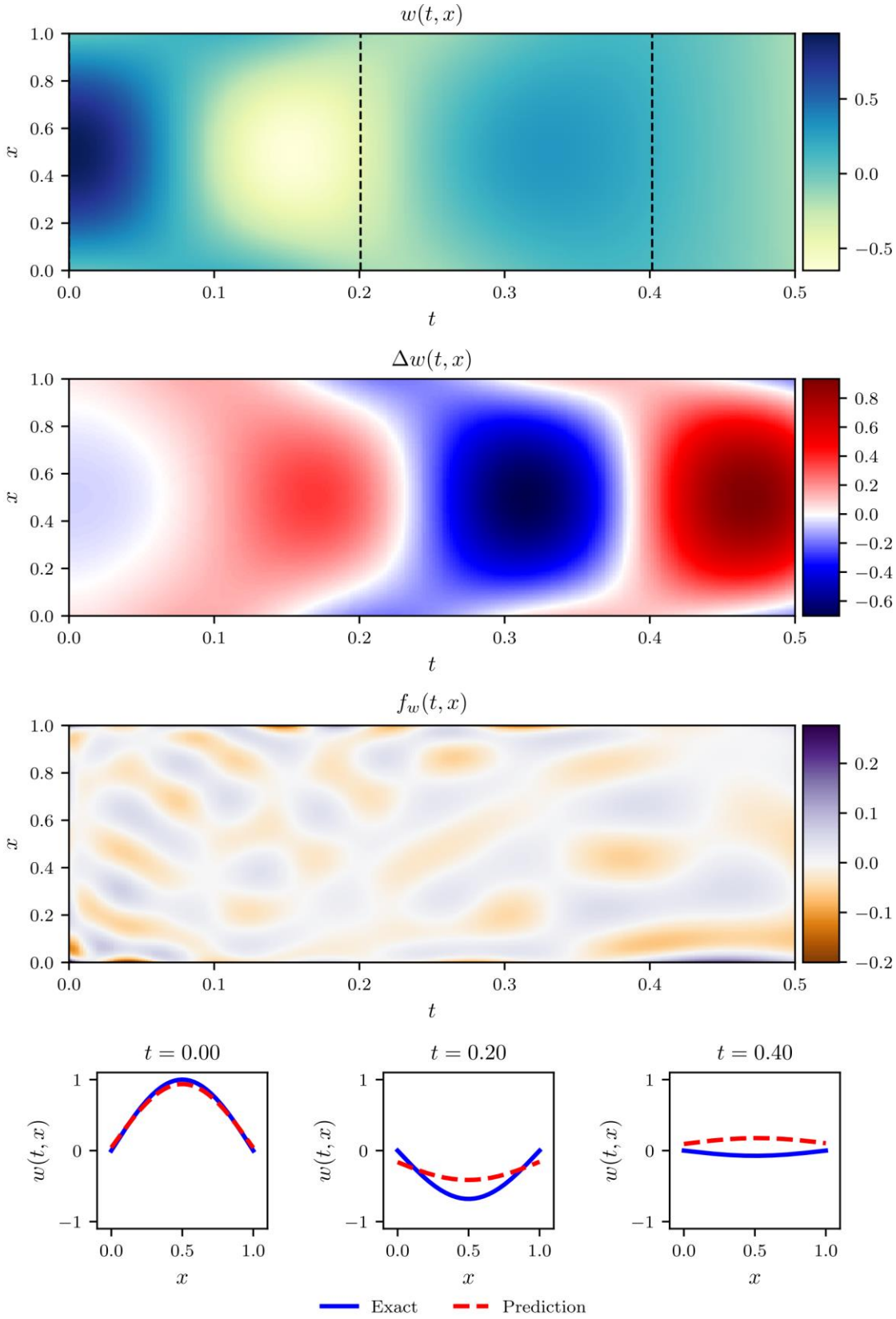

Figure 18. PIL solution for Problem 5: SSB vibration given in Table 5.

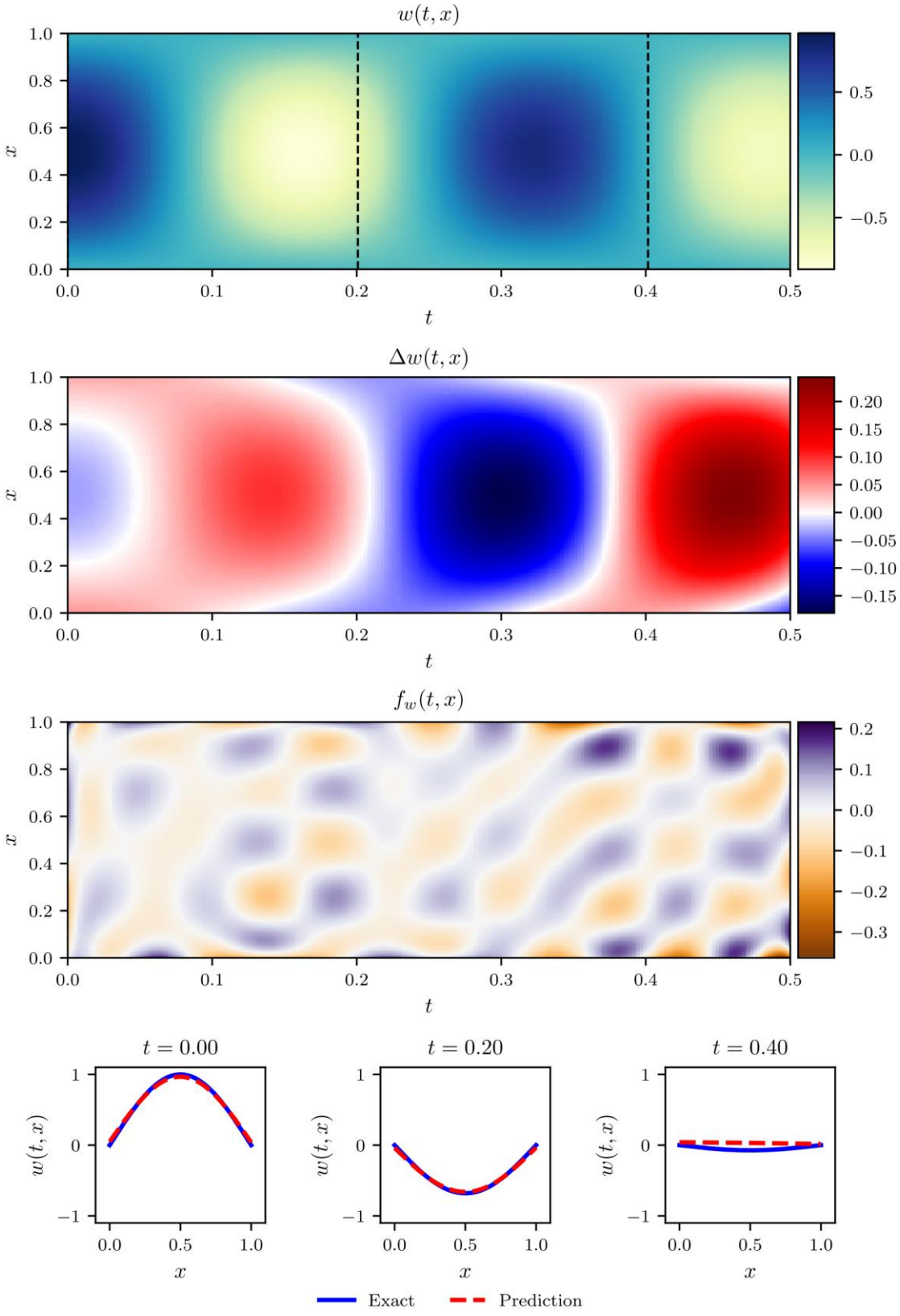

Figure 19. PIL solution for Problem 6: SSB vibration given in Table 6.

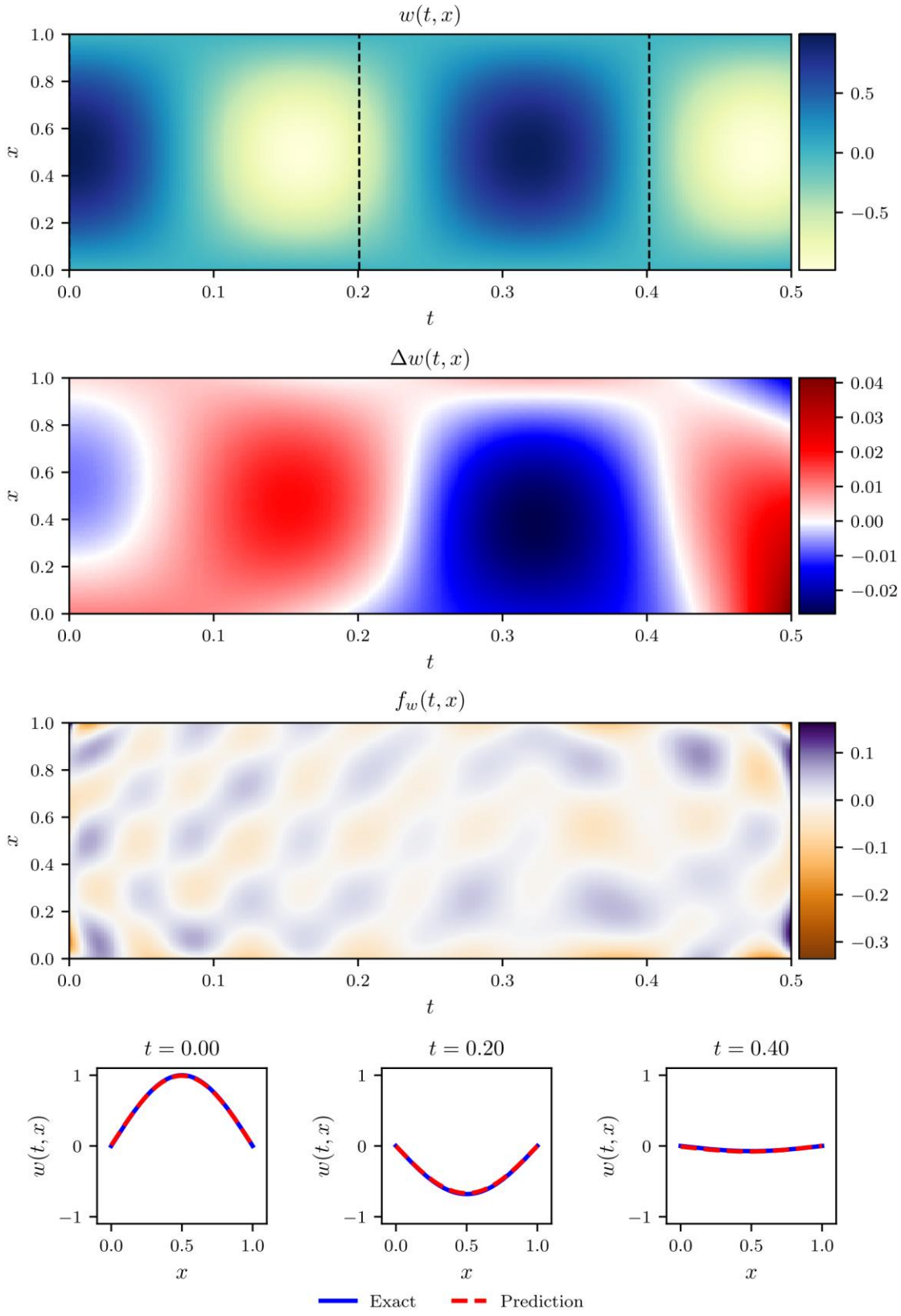

Figure 20. PIL solution for Problem 7: SSB vibration given in Table 7.

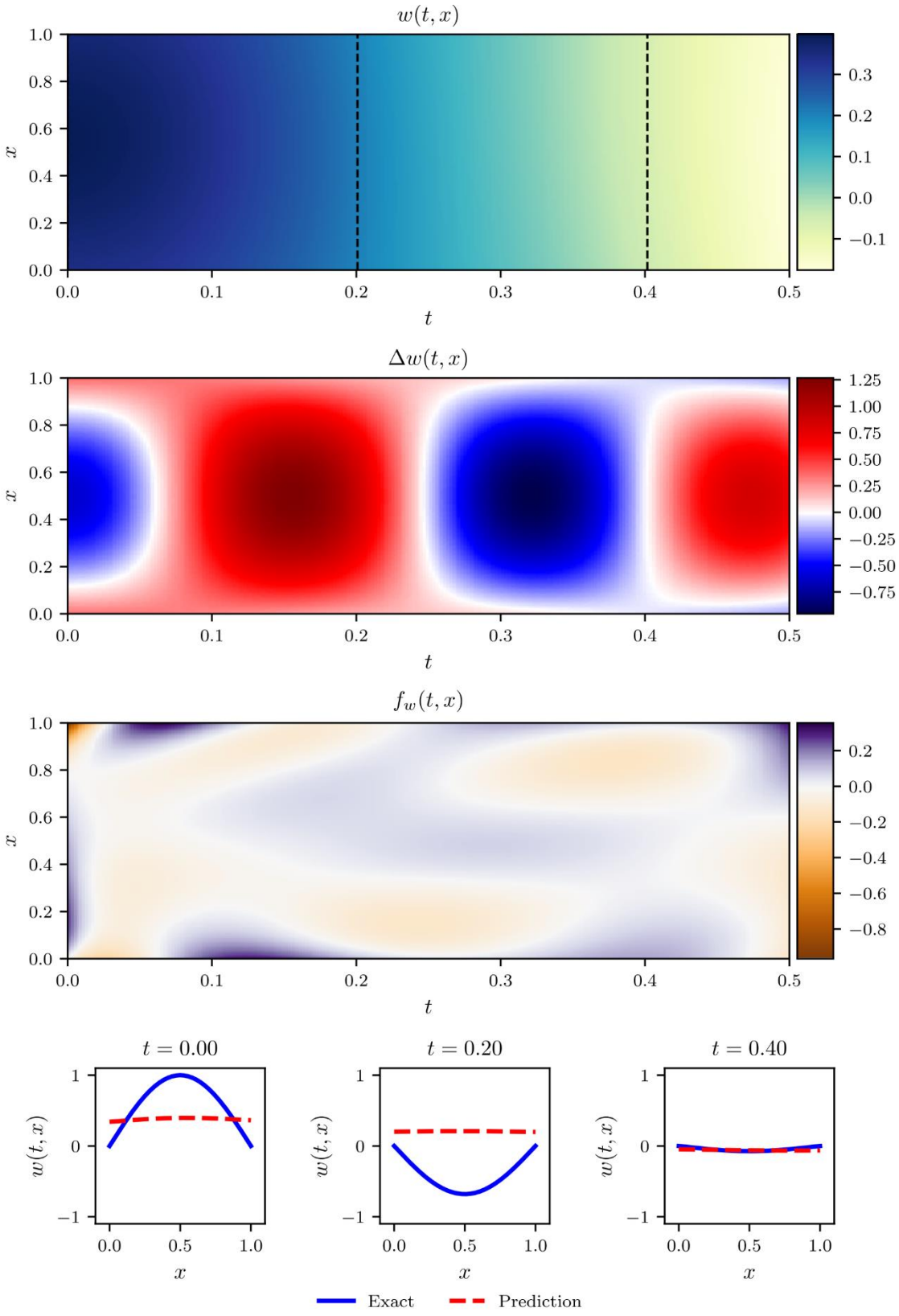

Figure 21. PIL solution for Problem 8: SSB vibration given in Table 8.

4.4. Implicit Runge-Kutta Method

Because of the limitation of collocation method, it is worth integrating other numerical method into the loss function. Multistep method and Runge-Kutta method, which are widely used to solve time related PDEs, seems to be promising given that each step is dependent on the previous step, by which the independency between collocation points in collocation method might be circumvented. Out of these 2, Runge-Kutta method is preferable because of its ability to obtain higher ordered accuracy. This section attempts to apply implicit Runge-Kutta method to physics informed training.

4.4.1. Methodology

Gauss-Legendre Implicit Runge-Kutta Method

As for an ODE of the form:

$$\boldsymbol{y}_t = f(t, \boldsymbol{y}), \qquad \boldsymbol{y}(t_0) = \boldsymbol{y}_0, \tag{4.4.1}$$

Runge-Kutta methods discretize the continuous time window into several time points t_n and iteratively derive the values $\boldsymbol{y}(t_{n+1})$ using the evaluations $\boldsymbol{\xi}_j$ at intermediate time stages $t_n + c_j h$. The corresponding Runge-Kutta formulation is:

$$\boldsymbol{\xi}_j = \boldsymbol{y}_n + h \sum_{i=1}^{\nu} a_{j,i}\, \boldsymbol{f}(t_n + c_i h, \boldsymbol{\xi}_i), \qquad j = 1,2, \dots, \nu,$$

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h \sum_{j=1}^{\nu} b_j\, \boldsymbol{f}\big(t_n + c_j h, \boldsymbol{\xi}_j\big) \tag{4.4.2}$$

where $0 \le c_i < 1$. The values of $a_{j,i}$, b_j and c_i determine the type of Runge-Kutta method. For the widely used Gauss-Legendre implicit Runge-Kutta method, c_i are selected to be the roots of νth degree Legendre polynomial which is orthogonal and has all its roots fall between $[0,\ 1]$. Its coefficients $a_{j,i}$, b_j are then determined by:

$$a_{j,i} \coloneqq \int_0^{c_j} \frac{q_i(\tau)}{q_i(c_i)}\,\mathrm{d}\tau\,,\ \ j,i = 1,2,\dots,\nu, \qquad b_j \coloneqq \int_0^1 \frac{q_j(\tau)}{q_j(c_j)}\,\mathrm{d}\tau\,,\ \ j = 1,2,\dots,\nu. \tag{4.4.3}$$

$$q(t) \coloneqq \prod_{j=1}^{\nu}(t - c_j)\,, \qquad q_l(t) \coloneqq \frac{q(t)}{t - c_l}, \qquad l = 1,2,\dots,\nu,$$

After that, c_i are linearly shifted from $[0,1]$ to $[t_n, t_{n+1}]$. This method is equivalent to first fitting $\boldsymbol{f}(t,\boldsymbol{y})$ using polynomial $\boldsymbol{p}(t)$ that coincides with $\boldsymbol{f}(t,\boldsymbol{y})$ at collocation points which are Legendre polynomial's roots, then integrating $\boldsymbol{p}(t)$ instead of $\boldsymbol{f}(t,\boldsymbol{y})$ to evaluate $\boldsymbol{y}$.

Implicit Runge-Kutta for Dynamic Beam Equation

The dynamic beam equation 4.2.3 can be discretized through implicit Runge-Kutta (IRK) method by first defining function η and f as:

$$\partial w(x,t)/_{\partial t} = \eta(x,t),$$

$$\partial \eta(x,t)/_{\partial t} = f(x,t) = -\lambda \cdot \partial^4 w(x,t)/_{\partial x^4}.$$

Then Equation 4.2.3 can be discretized by applying Gauss-Legendre IRK:

$$\boldsymbol{\eta}_j = \dot{\boldsymbol{w}}_n + h\sum_{i=1}^{\nu} a_{j,i}\, f(t_n + c_i h, \boldsymbol{\eta}_i)$$

$$\boldsymbol{\xi}_j = \boldsymbol{w}_n + h\sum_{i=1}^{\nu} a_{j,i}\, \boldsymbol{\eta}_i$$

$$\boldsymbol{w}_{n+1} = \boldsymbol{w}_n + h\sum_{i=1}^{\nu} b_i\, \boldsymbol{\eta}_i$$

where $a_{j,i}$ and b_i are coefficients defined in Equation 4.4.3, $\boldsymbol{\eta}$, $\boldsymbol{\xi}$ and $\boldsymbol{w}$ are vectors of the values of η, f and w at different x. These equations can then be expressed in matrix form as:

$$\boldsymbol{H} = \dot{\boldsymbol{W}}_n + h \cdot \boldsymbol{A} \cdot \boldsymbol{F}$$

$$\boldsymbol{\Xi} = \boldsymbol{W}_n + h \cdot \boldsymbol{A} \cdot \boldsymbol{H} = \boldsymbol{W}_n + h \cdot \boldsymbol{A} \cdot \dot{\boldsymbol{W}}_n + h^2 \cdot \boldsymbol{A}^2 \cdot \boldsymbol{F}$$

$$\boldsymbol{w}_{n+1} = \boldsymbol{w}_n + h \cdot \boldsymbol{b} \cdot \boldsymbol{H} = \boldsymbol{w}_n + h \cdot \boldsymbol{b} \cdot \dot{\boldsymbol{W}}_n + h^2 \cdot \boldsymbol{b} \cdot \boldsymbol{A}$$

The residual matrix $\boldsymbol{R}$ for the governing equation can now be constructed by assembling the equations above:

$$\boldsymbol{R} = \boldsymbol{W}_n + h \cdot \boldsymbol{A} \cdot \dot{\boldsymbol{W}}_n + h^2 \cdot \boldsymbol{B} \cdot \boldsymbol{A} \cdot \boldsymbol{F} - \boldsymbol{W}$$

$$\boldsymbol{B} = [\boldsymbol{A} \quad \boldsymbol{b}]^{\boldsymbol{T}}, \ \boldsymbol{W} = [\boldsymbol{\Xi} \quad \boldsymbol{w}_{n+1}]^{\boldsymbol{T}}$$

The loss function incorporating IRK formulation is simply:

$$L = \rho_f |\boldsymbol{R}|^2 + \frac{\rho_0}{N_0} \sum_{x_i \in X}^{N_0} \sum_{j=1}^{2} \left| R_j^I(\hat{w}(x_i, 0)) \right|^2 + \frac{\rho_b}{N_b} \sum_{t_i \in T}^{N_b} \sum_{j=1}^{4} \left| R_j^B\left(\hat{w}(x_j, t_i)\right) \right|^2 \qquad (4.4.4)$$

4.4.2. Result and Discussion

Problem 3 is solved again using the loss function 4.4.4 to ascertain IRK's effect. A 200-stage Gauss-Legendre IRK method is used to transform the governing equation into its discrete form. Its result is presented in Figure 22. In contrast with the result shown in Figure 16, PIL with IRK integrated successfully converged. This proves that the additional interrelationship between adjacent timestamps enforced by the time integration in the evaluation of $a_{j,i}$ and b_j displayed in Equation 4.4.3 really facilitated the convergence. Nevertheless, as shown in the second row of Figure 22, the defect increases as it goes further away from the initial timestamp. Indeed, as the time window T expands more, the same pitfall reemerges, as shown in Figure 23. Runge-Kutta method therefore still cannot fully resolve this convergence issue though can achieve better result than collocation method.

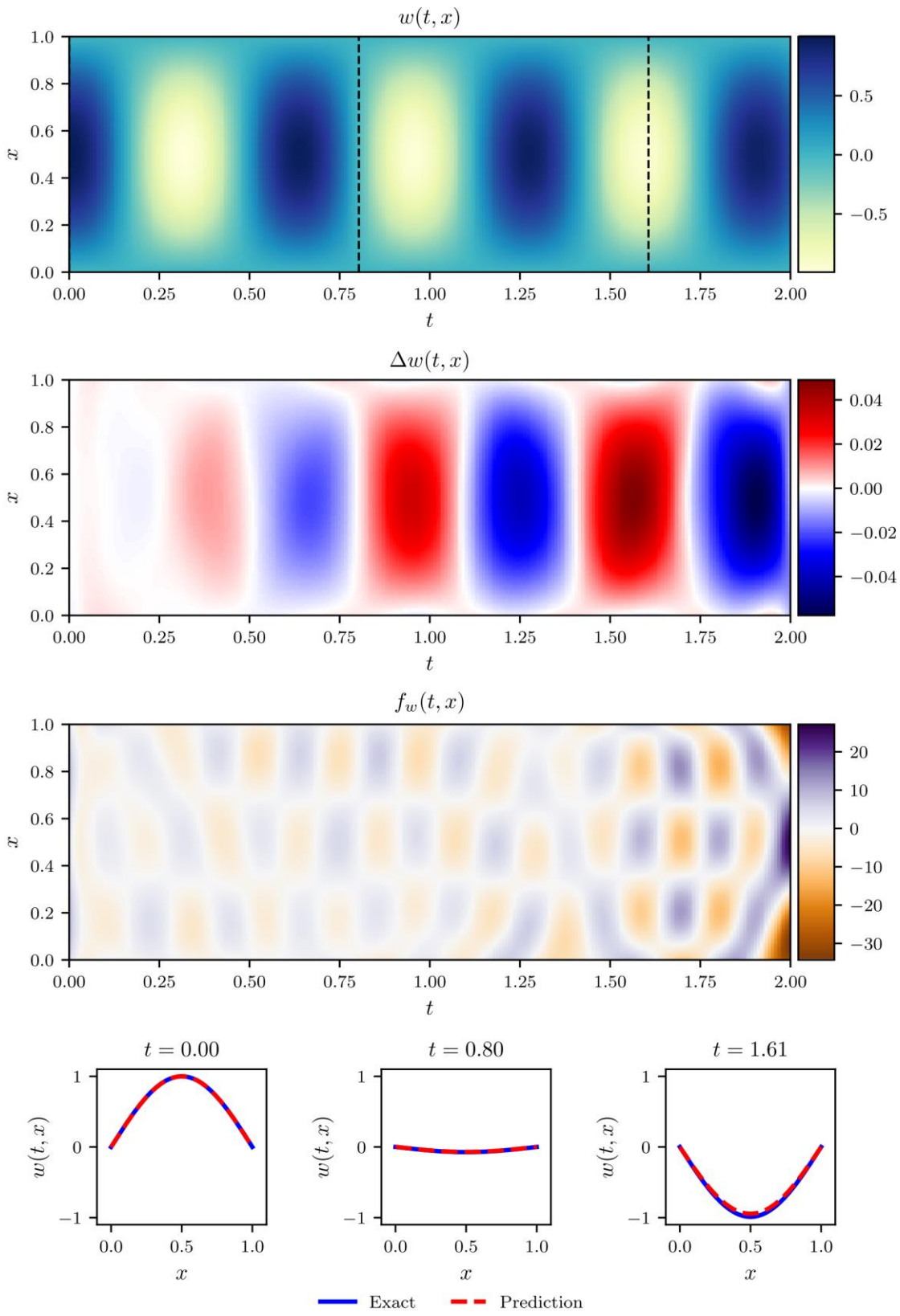

Figure 22. PIL solution for Problem 3: SSB vibration given in Table 3 incorporating IRK.

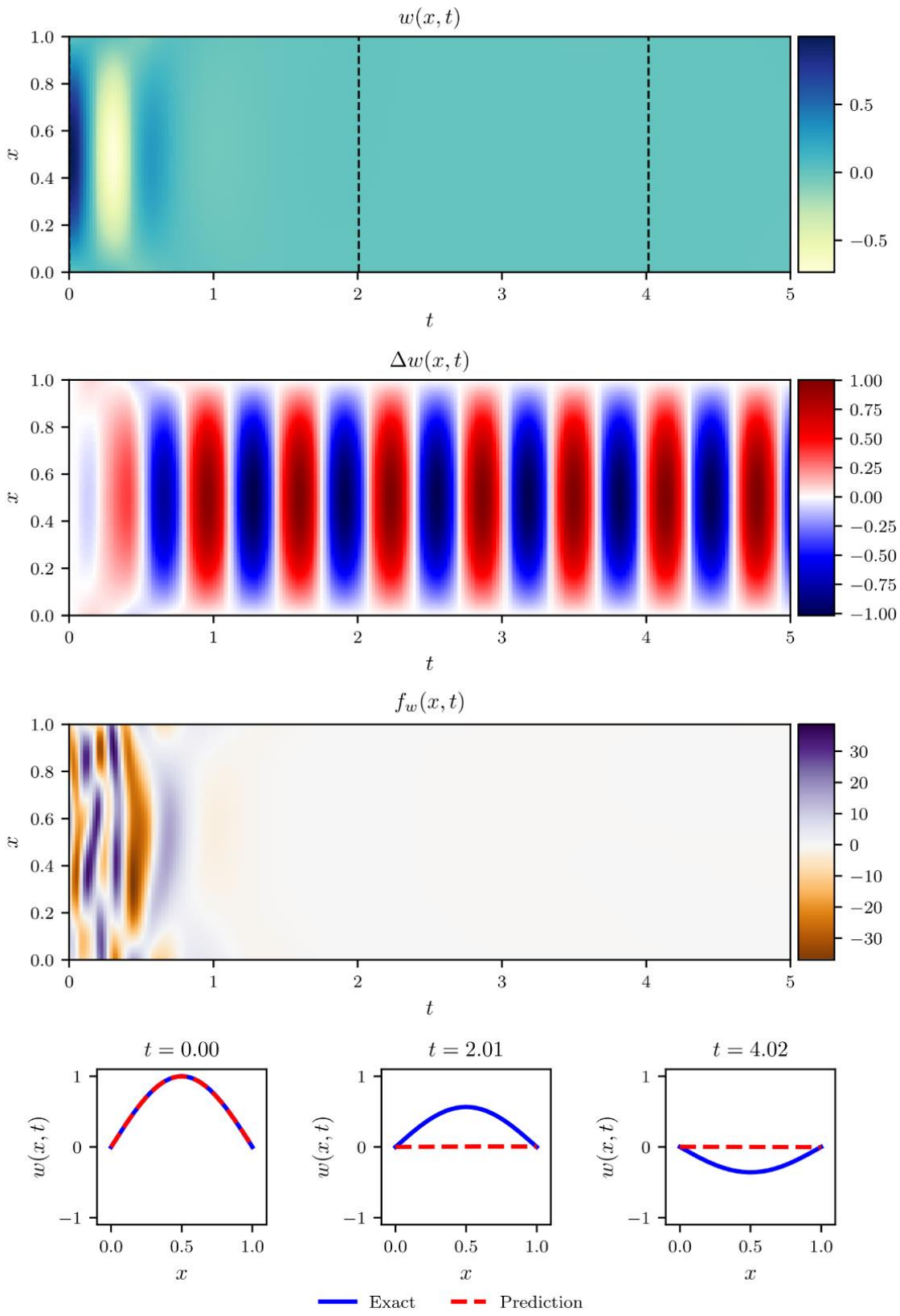

Figure 23. IRK solution for Problem 3: SSB given in Table 3 with extended time domain.

4.5. Rayleigh-Ritz Method

4.5.1. Methodology

Variational Formulation

Variational formulation of dynamic beam problem given in Equation 4.2.1 can also be used to train the MLP trial solution. For beam free vibration problem of a beam with unit length, Equation 4.2.1 reduces to:

$$J[w] = \int_{t_1}^{t_2} \int_0^1 \frac{1}{2}\left(\frac{\partial w}{\partial t}\right)^2 - \frac{1}{2}\lambda\left(\frac{\partial^2 w}{\partial x^2}\right)^2 \mathrm{d}x\,. \tag{4.5.1}$$

The physics informed loss function leveraging this formulation is:

$$L = \rho_f \sum_{\boldsymbol{x}_i,\beta_i\in\mathcal{J}} \beta_i \left\{\left(\frac{\partial \widehat{w}(\boldsymbol{x}_i)}{\partial t}\right)^2 - \lambda\left(\frac{\partial^2 \widehat{w}(\boldsymbol{x}_i)}{\partial x^2}\right)^2\right\}$$
$$+\rho_c\left(\frac{1}{N_t}\sum_{x_i\in X}^{N_t}\sum_{j=1}^{2}\left|R_j^I\left(\widehat{w}(x_i,0)\right)\right|^2 + \frac{1}{N_b}\sum_{t_i\in T}^{N_b}\sum_{j=1}^{2}\left|R_j^B\left(\widehat{w}(x_j,t_i)\right)\right|^2\right) \tag{4.5.2}$$

where $\mathcal{J}$ is the set of collocation points $\boldsymbol{x}_i$ and weights β_i of selected numerical integration scheme, ρ_V and ρ_L are weights for balancing potential energy term and constraint term.

Numerical Integration

Numerical integration, or simply quadrature, is essentially about evaluating integral using weighted sum of values at collocation points, namely:

$$\int_\Omega f(\boldsymbol{x})\,d\boldsymbol{x} \approx \sum_{i=1}^{\nu} \beta_i f(\boldsymbol{c}_i) \tag{4.5.3}$$

The most intuitive quadrature is Riemann sum, which replaces the volume below the integrand $f(\boldsymbol{x})$ with cuboids. This procedure can be taken as first fitting the function $f(\boldsymbol{x})$ with piecewise constant functions then integrating those constant functions instead as an approximation of the integral.

One widely implemented integration scheme with higher order of accuracy is Gaussian quadrature, which uses polynomial defined over the entire domain Ω instead of constant functions to approximate the integrand. Specifically, it first sets collocation points as roots of Legendre polynomial of selected degree, then fit $f(\boldsymbol{x})$ with a multivariable polynomial aligned with $f(\boldsymbol{x})$ at those points before evaluating its integral. This multivariable polynomial can be easily constructed using normalized Lagrange polynomials $\boldsymbol{p}_i(\boldsymbol{x})$ as basis so that:

$$f(\boldsymbol{x}) \approx \sum_i f(\boldsymbol{c}_i) \cdot \boldsymbol{p}_i(\boldsymbol{x}), \qquad \boldsymbol{p}_i(\boldsymbol{x}) = \prod_d \prod_{\substack{k=1 \\ k \neq i}} \frac{x^d - c_k^d}{c_i^d - c_k^d} \tag{4.5.4}$$

where the superscript d stands for the dth dimension. Then the weights can be determined by:

$$\beta_i = \int_\Omega \boldsymbol{p}_i(\boldsymbol{x})\, d\boldsymbol{x} = \prod_d \int_{\Omega^d} \prod_{\substack{k=1 \\ k \neq i}} \frac{x^d - c_k^d}{c_i^d - c_k^d}\, dx^d . \tag{4.5.5}$$

4.5.2. Result and Discussion

As to Problem 1 given in Table 1, the integration in loss function 4.5.2 is evaluated using 100×100 Gaussian quadrature and the value of weights w_V and w_L are set to be $w_V = 10^6$ and $w_L = 1$. The training result is shown in Figure 24. The displacement field approximated looks like a shattered glass with different fragments assuming ridiculous value. This is probably caused by the incompetence of gaussian quadrature, or any numerical integration algorithm relying on limited sampling points when handling an MLP of complexity way higher than a 200th degree polynomial. By the virtue of that, when minimizing the integral of the functional, the MLP tends to adjust its parameters to lower down the functional's value locally at those Legendre collocation points, leading to this bizarre phenomenon.

Lowering down the complexity of MLP might be a solution. For Problem 9 given in Table 9, the trial solution is reselected to be a shallow MLP with 100 hidden neurons. Yet its result demonstrated in Figure 25 suggest that this adjustment still doesn't work even though the

potential energy term in loss function does not exceed the constraint term in this case. It appears that variational formulation working well on elliptic PDEs like Poisson's equation in [37] has some intrinsic defects in dealing with parabolic PDEs.

Table 9. Problem 9: SSB problem with shallow MLP

λ	T	X	ρ_c, ρ_f	Structure of $\widehat{w}$	Exact Solution $w(x,t)$
1	$[0,1]$	$[0,1]$	1e10, 1	[2,100,1]	$\sin \pi x \cdot \cos \pi^2 t$

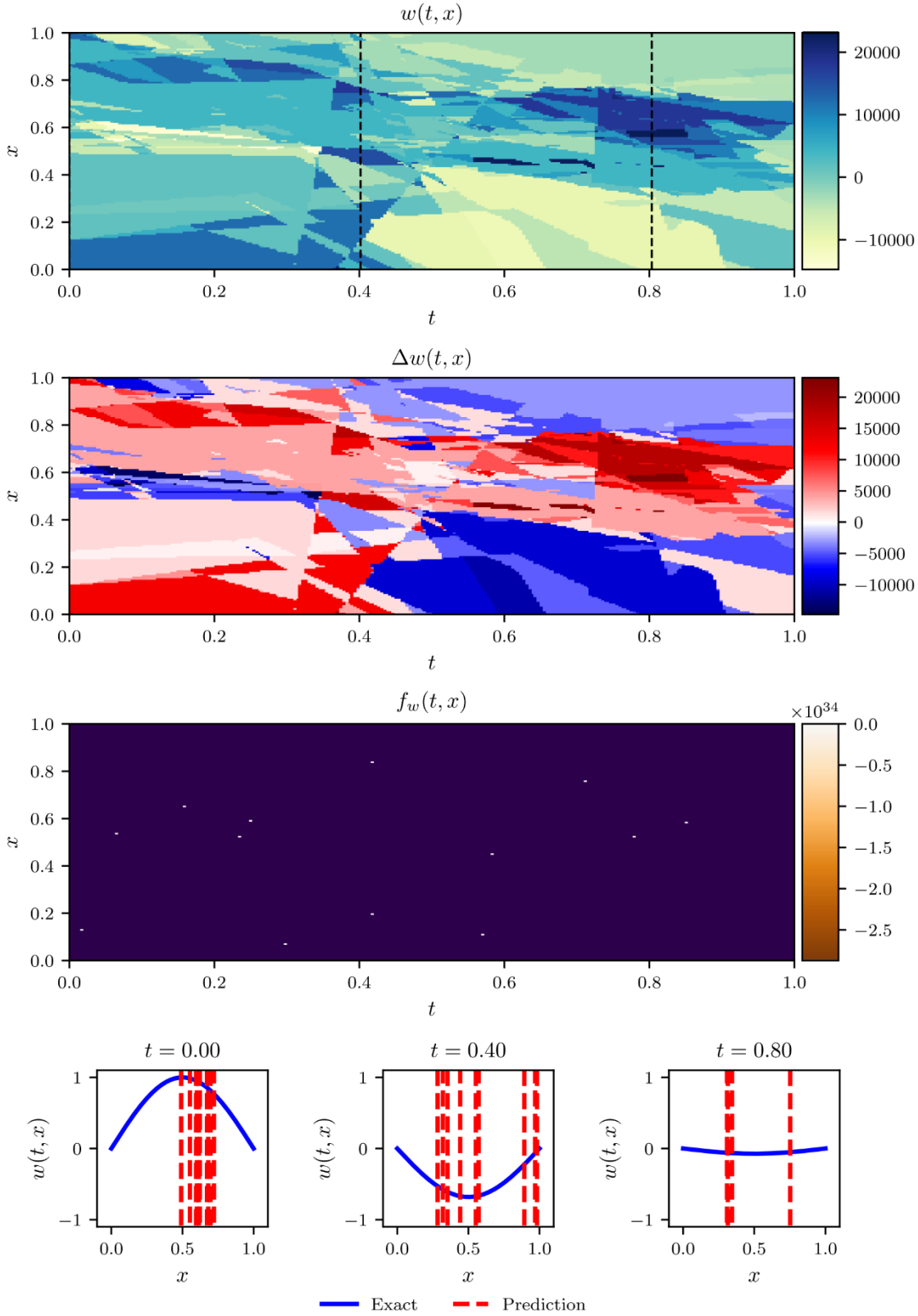

Figure 24. PIL solution trained with variational form for Problem 1 given in Table 1.

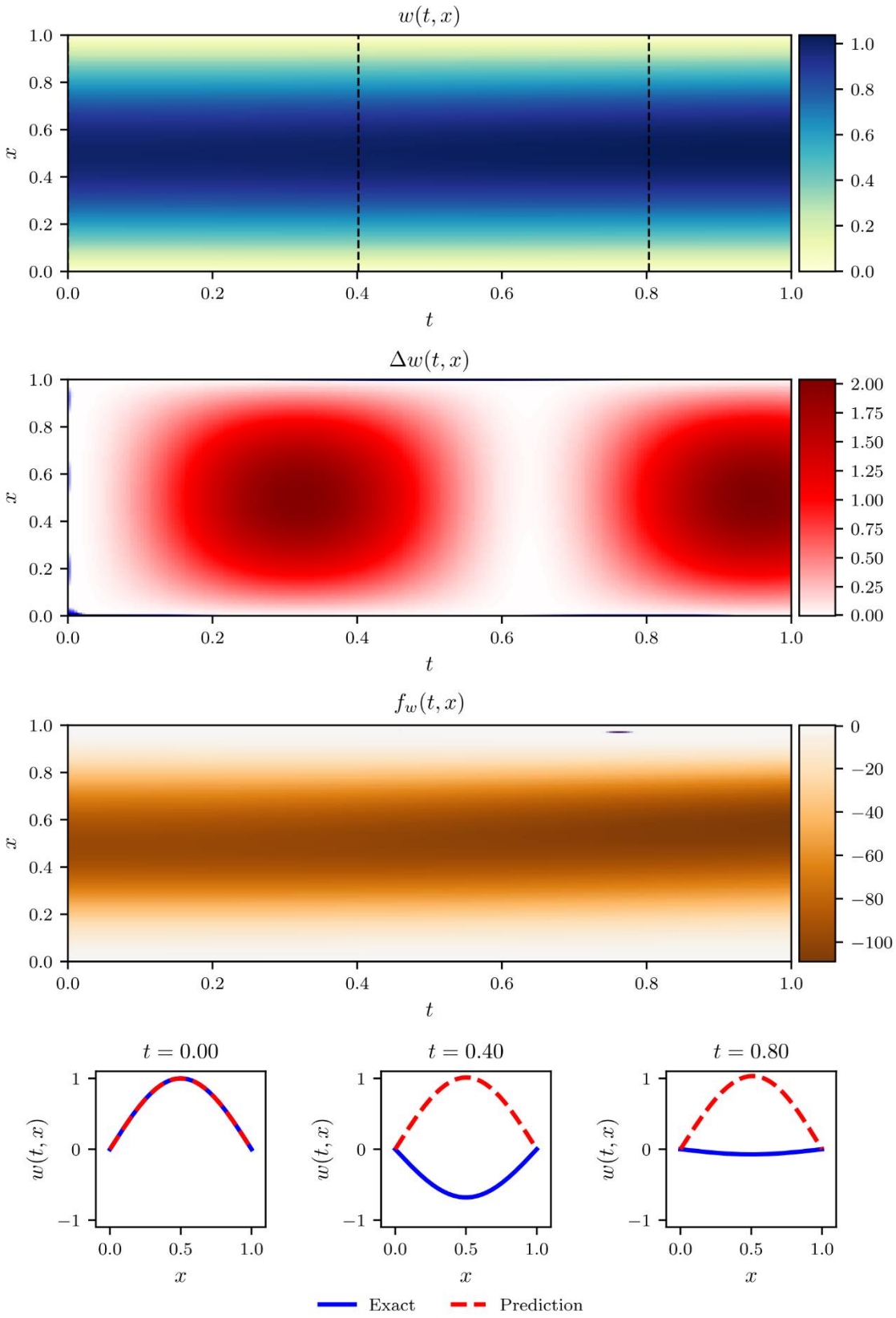

Figure 25. PIL solution trained with variational form for Problem 9 given in Table 9.

4.6. Segmented Time Solution

Huygens' principle states that all points of a wave front can be regarded as new sources of spherical wavelets that expand in every direction at a rate depending on their velocities, and the sum of these wavelets forms the wavefront. Though being more of a practical hypothesis than of a solid physics law, this old principle's spirit inspires us to come up with a method that can circumvent the long time convergence problems. For a beam vibration problem, we may treat the vertical displacement and velocity of the beam at a given time as a source for the vibration afterwards. In other words, they can be taken as the initial conditions for the vibration in the following time interval if we reset the initial time.

4.6.1. Methodology

Since both collocation and Runge-Kutta method tend to encounter convergence problem if the time domain is too large, we can circumvent this issue using segmented time solution, namely dividing the entire time domain into short time windows, then using PIL incorporating these methods to derive the result for these windows in sequence and concatenating their results

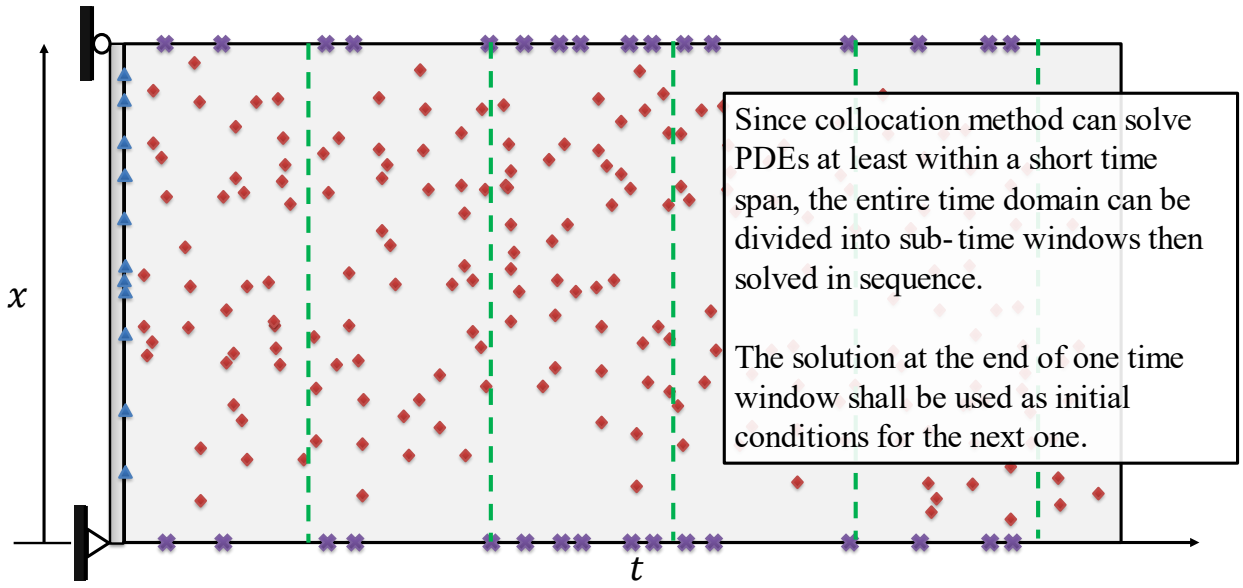

Figure 26. Piecewise solution for circumventing convergence problem.

all together afterwards, as shown in Figure 26. To do this, the time windows at the initial timestamp is solved at first, then its result at its end is used as initial condition for the neighboring window, and this process is iteratively performed until all windows are solved.

4.6.2. Result and Discussion

Problem 10, 11 and 12 given in Table 10, Table 11 and Table 12 are solved as such using collocation method to demonstrate this approach's efficacy.

Table 10. Problem 10: SSB vibration with extended time domain

λ	T	X	ρ_f, ρ_b, ρ_0	Structure of $\widehat{w}$	Exact Solution $w(x,t)$
1	$[0, 5]$	$[0, 1]$	1, 1, 1	[2,100,100,100,1]	$\sin \pi x \cdot \cos \pi^2 t$

Table 11. Problem 11: SSB vibration with further extended time domain

λ	T	X	ρ_f, ρ_b, ρ_0	Structure of $\widehat{w}$	Exact Solution $w(x,t)$
1	$[0, 10]$	$[0, 1]$	1, 1, 1	[2,64,1]	$\sin \pi x \cdot \cos \pi^2 t$

Table 12. Problem 12: SSB vibration with the 3rd mode shape

λ	T	X	ρ_f, ρ_b, ρ_0	Structure of $\widehat{w}$	Exact Solution $w(x,t)$
1	$[0, 0.15]$	$[0, 1]$	2e-4, 1, 1	$[2,64 \times 8,1]$	$\sin 3\pi x \cdot \cos 9\pi^2 t$

The results for these problems are displayed in Figure 27, Figure 28 and Figure 29. The error of the piecewise solution can be controlled by adjusting the length of the time window. Though there is currently no clear way to predetermine the proper length of the time window given a required maximum error, this is likely to be surmounted in the future.

Another thing worth mentioning is that for Problem 12, the MLP trial solution only manages to converge when the activation function is selected as $\sin(x)$ instead of $\tanh(x)$ used in other cases. Therefore, the choice of activation also has a significant impact on the MLP trial solution for certain problems.

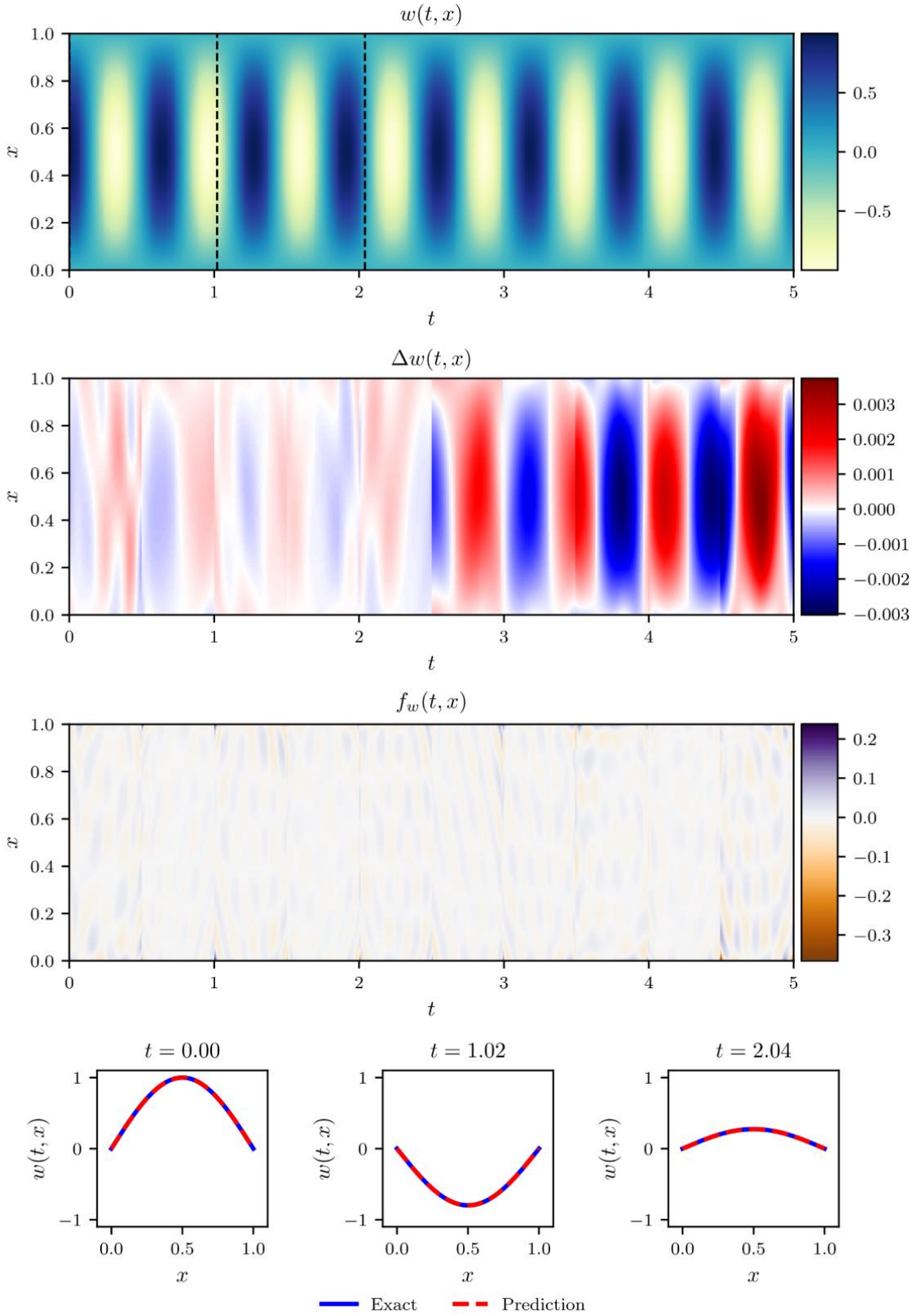

Figure 27. Piecewise solution for Problem 10 given in Table 10.

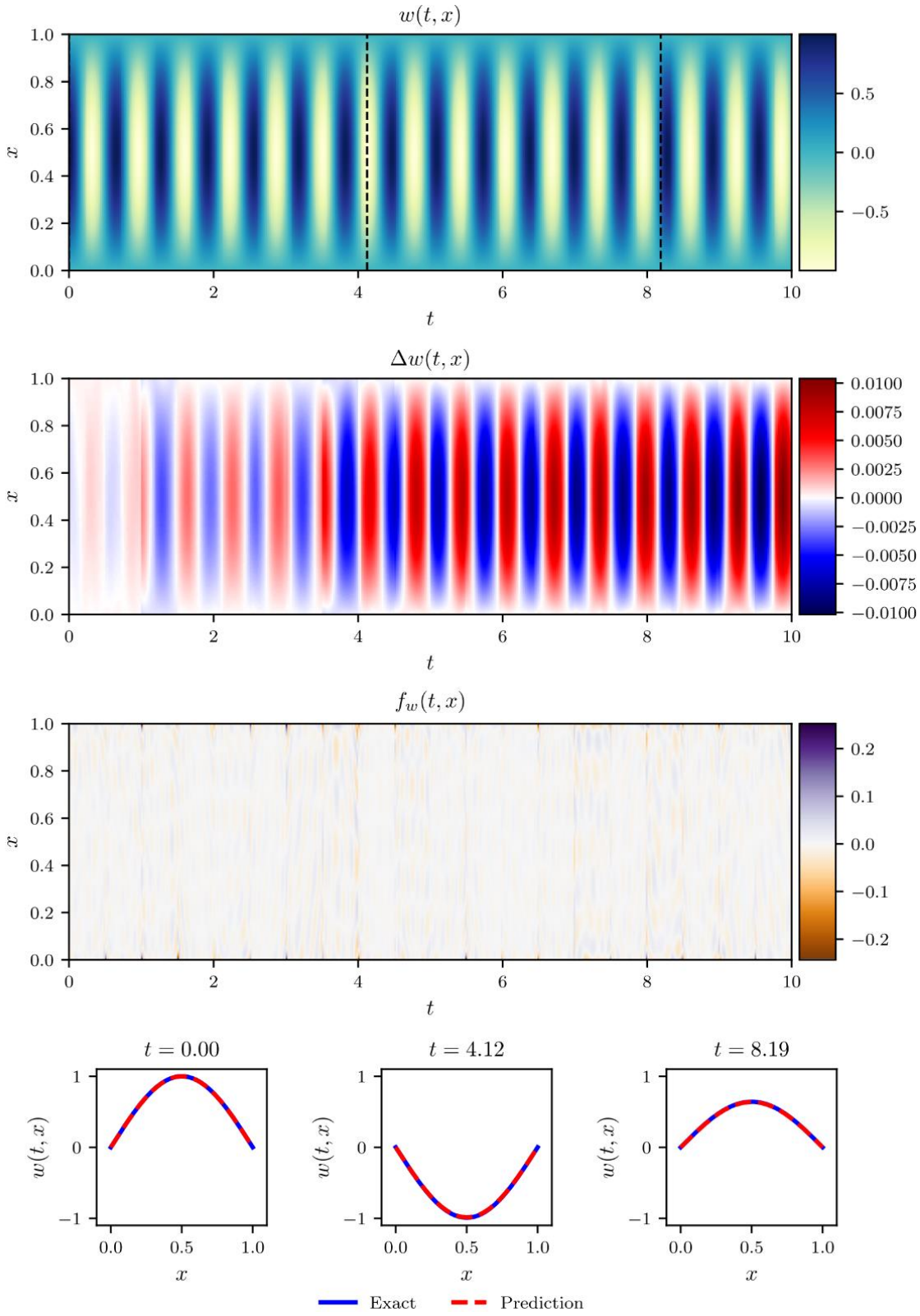

Figure 28. Piecewise solution for Problem 11 given in Table 11.

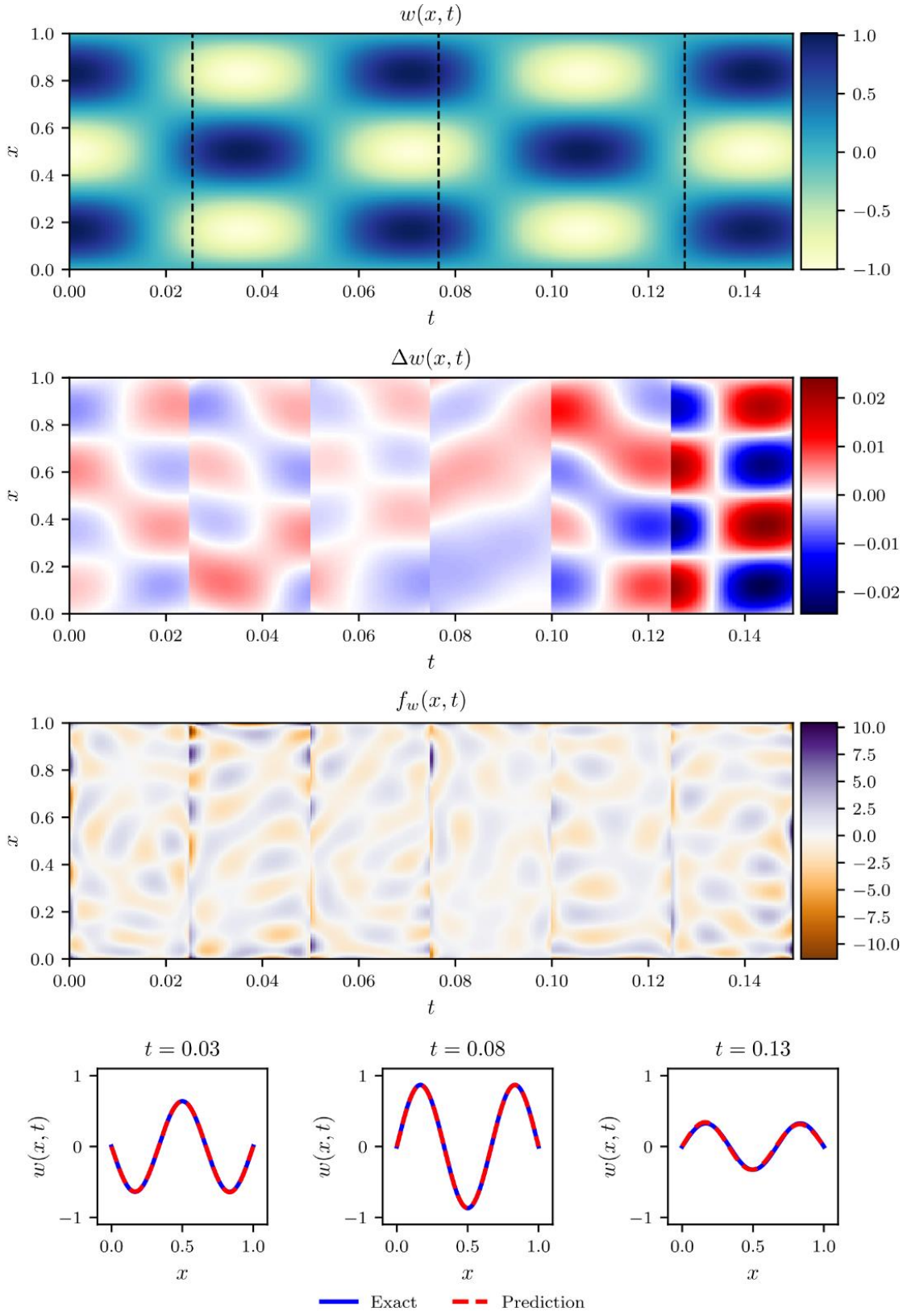

Figure 29. Piecewise solution for Problem 12 given in Table 12.

CHAPTER 5

Artificial Neural Networks for Inverse Vibration Problems

5.1. Motivation

Structural Health Monitoring (SHM) keeps incorporating computational techniques over time to help solve its inverse problems, namely localizing, evaluating and even predicting damages on large structures from the responses induced by them, such as vibration and wave transmission. Out of those responses, vibration is probably the mostly used one because it is widespread and easy to detect with devices like displacement sensors and high speed camera. The displacement field captured over time provides lots of useful information about the condition of the structures, including where the damage could be and how severe it is.

That said, the precision of vibration signals is always riddled with uncertainties resulting from error of measurement and ineluctable environmental noise. Proper preprocessing is often required to address these disturbances before deriving reliable results, not to mention that sometimes the resolution of measurement is not high enough to be used immediately. PIL, which incorporates governing equation of the physics system into the training process, has the potential to recover exact displacement field which satisfies governing equations from limited measurement data and inversely obtain unknown system parameters for vibration problems at the same time. It is also expected that we can learn from our experience of PIL for forward problems to design the PIL for inverse problems. It is worth investigating this simple application of PIL on inverse vibration problem at first to accumulate enough experience for extrapolating it to more complex problems in the future.

5.2. Signal Reconstruction

Suppose an arrays of one dimensional displacement measurements $\widetilde{w}_i = w(x_i, t_i) + z(x_i, t_i)$ are given where $w(x,t)$ stands for the exact displacement field and z is noise induced by inaccuracy of measurement. Our goal is to reconstruct the exact continuous displacement field w with an MLP $\widehat{w}$ from discrete measurements $\widetilde{w}_i$ using PIL.

5.2.1. Methodology

Training Data

The displacement measurements $\widetilde{w}_i$ are generated by superposing the exact displacement field w and noise z at different locations and timestamps. Here the continuous exact displacement field is first artificially constructed by linearly combining different mode shapes of the beam which satisfy the dynamic beam equation 4.2.3 and specified boundary and initial conditions, then data points w_i are evenly sampled from this continuous function with resolution required. The resolution specifies the number of sensors space-evenly sprayed on the beam and the sampling frequency of these sensors. The noise z is then sampled from a Gaussian distribution of zero mean and variance $\mathrm{var}(z)$ predetermined by signal-to-noise ratio (SNR) in decibel, which is defined as:

$$SNR_{dB} = 10\log\left(\frac{P_{signal}}{P_{noise}}\right) = 10\log\left(\frac{\frac{1}{N}\sum_i w_i^2}{\mathrm{var}(z)^2}\right)$$

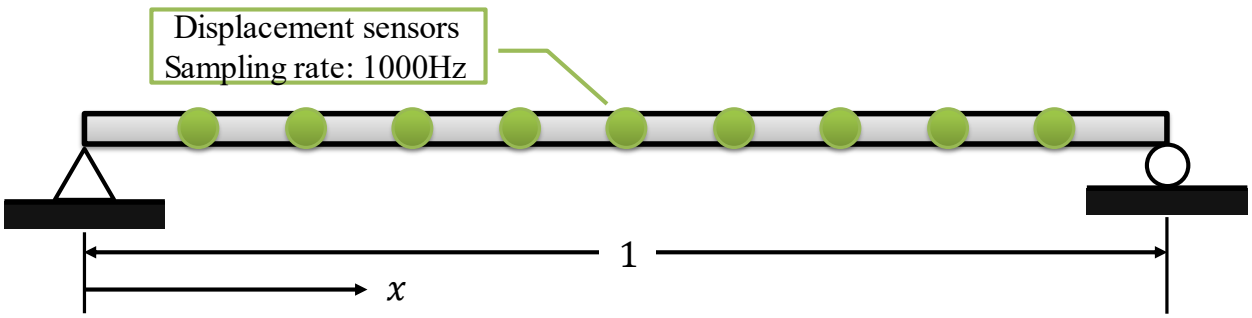

Figure 30. Setup of simple supported beam for signal reconstruction and parameter identification.

MLP for Interpolation

Traditionally, this signal reconstruction procedure is carried out by interpolating the measurements using function approximation in the form of Equation 3.3.8 with basis splines like Gaussian or chapeau-hat in order to obtain a continuous field. Likewise, these data points can also be interpolated by an MLP in the form of Equation 4.3.2, as explained in Section 3.3.2. The fidelity of this sheer interpolation based method, though, can be influenced significantly by noise and sparsity of data, such that the approximated continuous field $\widehat{w}$ does not obey physics law. Physic informed learning, whose loss functions are in the form of Equation 3.3.1 and 3.3.15, can incorporate physics law into the optimization of $\widehat{w}$ to ensure the exactness of the interpolation.

Loss Function

To reliably reconstruct the exact displacement field, the information we can leverage about the exact displacement field of a one dimensional beam with unit length is the scattered displacement measurements $\widetilde{w}_i$ at different locations and timestamps collected from sensors distributed along the beam shown in Figure 30, the governing equation of the beam vibration and the boundary conditions. To get the most of this information, the loss function for training the MLP $\widehat{w}$ is designed as:

$$L = \frac{\rho_d}{N_d} \sum_{t_i, x_i, \widetilde{w}_i \in \mathcal{X}}^{N_d} |\widehat{w}(x_i, t_i) - \widetilde{w}_i|^2 + \frac{\rho_f}{N_f} \sum_{t_i, x_i \in \Omega}^{N_f} \left| \left(\lambda \frac{\partial^4}{\partial x^4} + \frac{\partial^2}{\partial t^2} \right) \widehat{w}(x_i, t_i) \right|^2$$
$$+ \frac{1}{N_b} \sum_{t_i \in T}^{N_b} \left(\rho_{b1} (|\widehat{w}(0, t_i)|^2 + |\widehat{w}(1, t_i)|^2) + \rho_{b2} (|\widehat{w}_{xx}(0, t_i)|^2 + |\widehat{w}_{xx}(1, t_i)|^2) \right) \qquad (5.2.1)$$

where the first term is for fitting the training set $\mathcal{X}$ composed of measurements, the second term for satisfying the governing equation and the third for boundary conditions. ρ_d, ρ_f, ρ_{b1} and ρ_{b2} are weights for the displacement measurements, the governing equation term, the Dirichlet

boundary conditions and the second order boundary conditions respectively and should be properly chosen to redress the balance between them. Initially, they are all set to 1, but shall be adjusted on a trial and error basis.

Training

The parameters of the MLP $\widehat{w}$ are trained with Adam optimizer with a learning rate of 0.01 for 10000 iterations at first. During each iteration, while all the measurements are compiled into the data-driven loss, the collocation points for boundary conditions and governing equation are uniformly sampled from the domain of definition to conduct stochastic gradient descent. To be specific, $N_b = 50$ points are sampled for the boundary conditions and $N_f = 2500$ points are sampled for the internal domain which should satisfy the governing equation. Upon finished training by Adam optimizer, the parameters are then fine-tuned by L-BFGS-B on a larger number of static collocation points until the loss converges. The collocation points for L-BFGS-B optimizer are taken from an equally spaced 150×150 grid over the entire domain, so that $N_b = 150$ and $N_f = 22500$.

5.2.2. Result and Discussion

Baseline

The displacement field of a simple supported beam is reconstructed at first from limited data to demonstrate physic informed learning's power. The parameters for this baseline problem are given in Table 13:

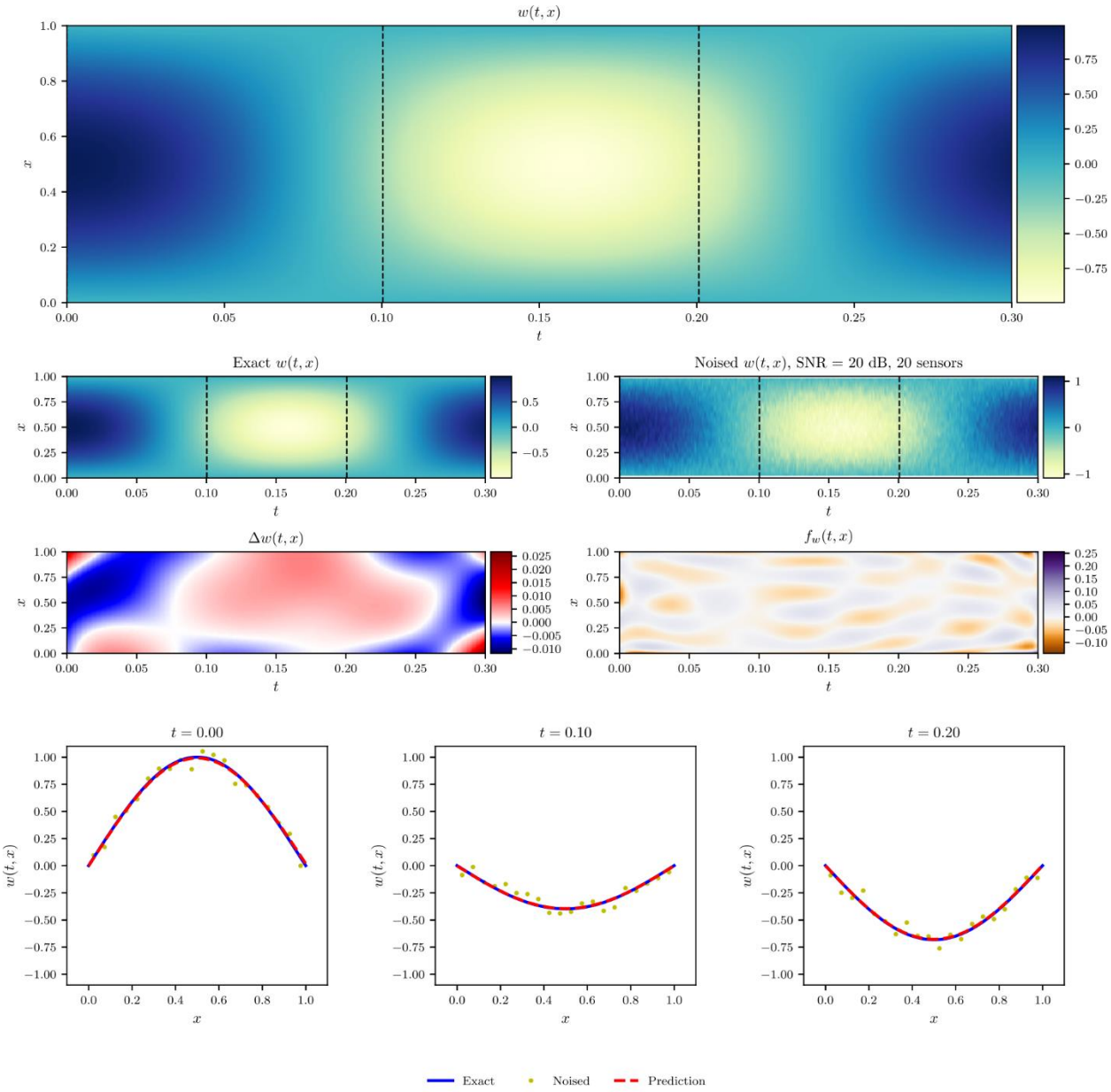

Figure 31. Physic informed result of baseline signal reconstruction problem given in Table 13.

Table 13. Problem 1: the baseline problem

λ	T	X	SNR_{dB}	Resolution	Structure of $\widehat{w}$	Exact Solution $w(x,t)$
4	$[0, 0.3]$	$[0, 1]$	20 dB	20 sensors, 1000 Hz	$[2{,}100 \times 3{,}1]$	$\sin \pi x \cdot \cos 2\pi^2 t$

The result of this reconstruction is demonstrated in Figure 31. The first row presents the reconstructed displacement field $\widehat{w}(t,x)$; the second row shows original exact displacement field

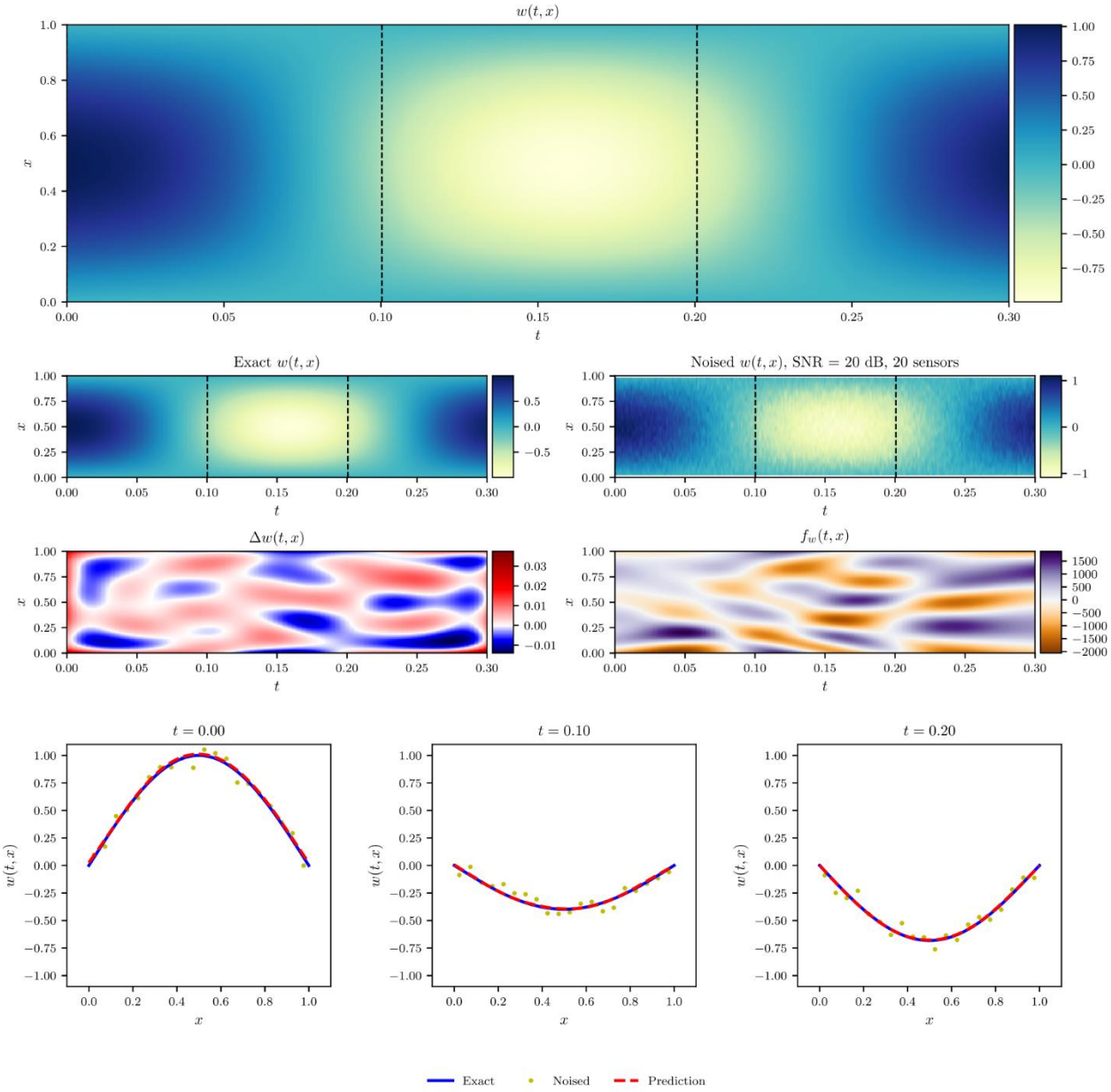

Figure 32. Data-driven result of baseline signal reconstruction problem given in Table 13.

w and the noisy displacement field measurements $\widetilde{w}_i$; the third row shows the discrepancy between $\widehat{w}$ and w on the left, and the residual $f_w(x,t) = \lambda \frac{\partial^4 \widehat{w}}{\partial x^4} + \frac{\partial^2 \widehat{w}}{\partial t^2}$ on the right indicating how well $\widehat{w}$ satisfies the PDE; a detailed comparison between $\widehat{w}$ and w at three timestamps is shown in the last row where w is illustrated in blue line while $\widehat{w}$ in red, and noisy displacement measurements in yellow dots. The reconstructed displacement field agrees well with the exact displacement field, with a maximum relative error around 2.7%. This error is calculated by:

$$\text{maximum relative error} = \frac{\max|\widehat{w}(x,t) - w(x,t)|}{\max|w(x,t)|}, \qquad (x,t) \in \Omega = X \times T$$

To prove the positive effect of the physics informed terms in loss function 5.2.1, the same problem is solved again using purely data-driven loss function, in which only the MSE defined over all the measurements is used. Its solution is given in Figure 32 and it shows that the maximum relative error of data-driven reconstruction result is around 3.9%, somewhat higher than the result of PIL. However, PIL for this problem takes 2293.3s to train whereas the data-driven method only 48.2s. The substantial amount of training time and petty improvement makes PIL not preferred for cases where data is abundant and clean.

Greater Noise and Sparser Sensors

The performance of PIL in situations where the noise is severer and the number of sensors is limited should be examined. In Problem 2 given in Table 14, the SNR decreases from 20 dB to 10 dB so that the noise becomes 10 times greater in power. Figure 34 shows the physic informed result of this problem while its data-driven counterpart is given in Figure 35. Their comparison clearly demonstrates the power of PIL given that the PIL achieved a maximum relative error 2.4% around 3 times lower than that of data-driven result which is about 7.2%, not to mention that the reconstructed displacement field of data-driven method is corrugated, a sign

of not satisfying the governing equation, as proven by the drastically high physic informed residual in Figure 35. Though the training time 1808.9s is much longer than that 48.2s of data-driven training, despite the time it consumes, PIL is preferable for the fidelity of its result.

Table 14. Problem 2: severer noise

λ	T	X	SNR_{dB}	Resolution	Structure of $\widehat{w}$	Exact Solution $w(x,t)$
4	$[0, 0.3]$	$[0, 1]$	10 dB	20 sensors, 1000 Hz	$[2,100 \times 3,1]$	$\sin \pi x \cdot \cos 2\pi^2 t$

Table 15. Problem 3: less sensors

λ	T	X	SNR_{dB}	Resolution	Structure of $\widehat{w}$	Exact Solution $w(x,t)$
4	$[0, 0.3]$	$[0, 1]$	20 dB	5 sensors, 1000 Hz	$[2,100 \times 3,1]$	$\sin \pi x \cdot \cos 2\pi^2 t$

Table 16. Problem 4: severer noise and less sensors

λ	T	X	SNR_{dB}	Resolution	Structure of $\widehat{w}$	Exact Solution $w(x,t)$
4	$[0, 0.3]$	$[0, 1]$	10 dB	5 sensors, 1000 Hz	$[2,100 \times 3,1]$	$\sin \pi x \cdot \cos 2\pi^2 t$

In Problem 3 given in Table 15, the number of equally spaced sensors on the beam decreases from 20 to 5. The physics informed and data-driven result are given in Figure 36 and Figure 37, and they take 1846.2s and 52.3s to train respectively. The positive effect of PIL becomes more apparent here as the maximum error of data-driven result 29.7% is more than 30 times higher than that 0.95% of PIL result. Training time becomes a trivial thing here.

Problem 4 where the data points are insufficient and noisy has also been solved. The PIL result is shown in Figure 38. After 1846.2s of training, it achieves a maximum error of only 1.9%. The data-driven method, on the other hand, suffers from overfitting severely after 514.87s of training as its maximum error is around 2438.1%. Even if we turn the L-BFGS-B algorithm off and only use gradient descent to optimize the network's parameter, after 37.2s, it still has an error of 30.3%, much greater than its physics informed counterpart.

Table 17. Physics informed learning vs data-driven training

		20 SNR, 20 Sensors	**10 SNR, 20 Sensors**	**20 SNR, 5 Sensors**	**10 SNR, 5 Sensors**
Physics Informed	Training Time (s)	2293.3311	1808.9215	1846.1901	1398.1148
	Max. Error	2.7%	2.4%	0.95%	1.9%
Data-driven	Training Time (s)	48.25	48.24	52.35	37.21 or 514.87
	Max. Error	3.9%	7.2%	29.7%	30.3% or 2438.1%

The training results for these 4 problems are summarized in Table 17. We can demonstrate the relationship between error, noise and resolution more clearly using linear regression, as given in Figure 33. If we set the acceptable error to be 5%, as indicated by the red plane, we can notice that only when data is abundant and clean can data-driven training be favored. Under the other circumstances, PIL proves to be a regularization method that can turn impossible to possible, rendering its long training time forgivable. These 4 cases prove that PIL has the useful ability to integrate physics information not contained in the limited measurements

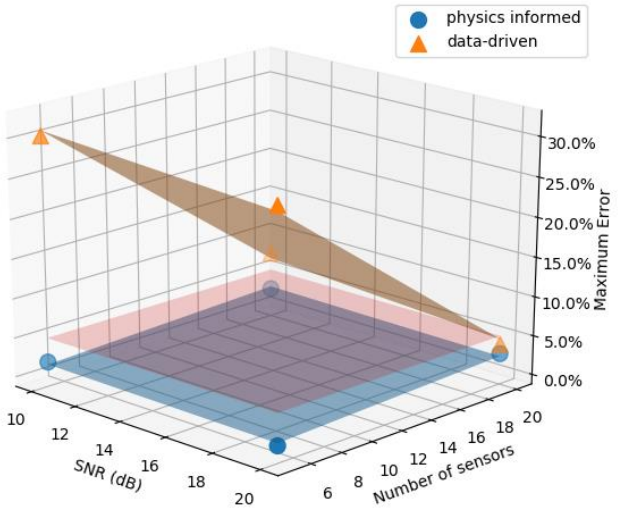

Figure 33. Comparison between physics informed learning and data-driven training.

to compensate for the scarcity of data points and mitigate the disturbance induced by terrible noise.

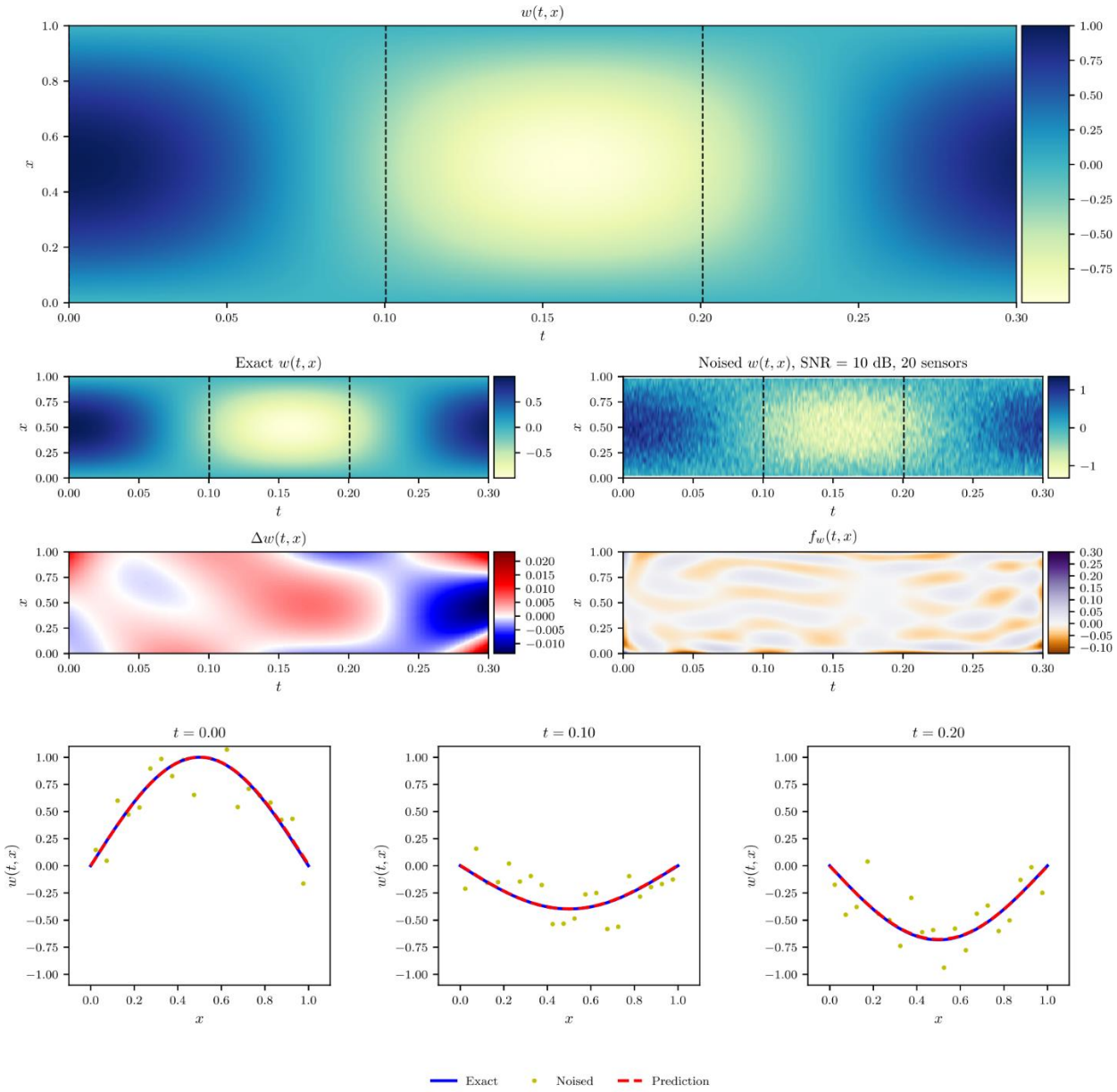

Figure 34. Physic informed result for Problem 2 given in Table 14.

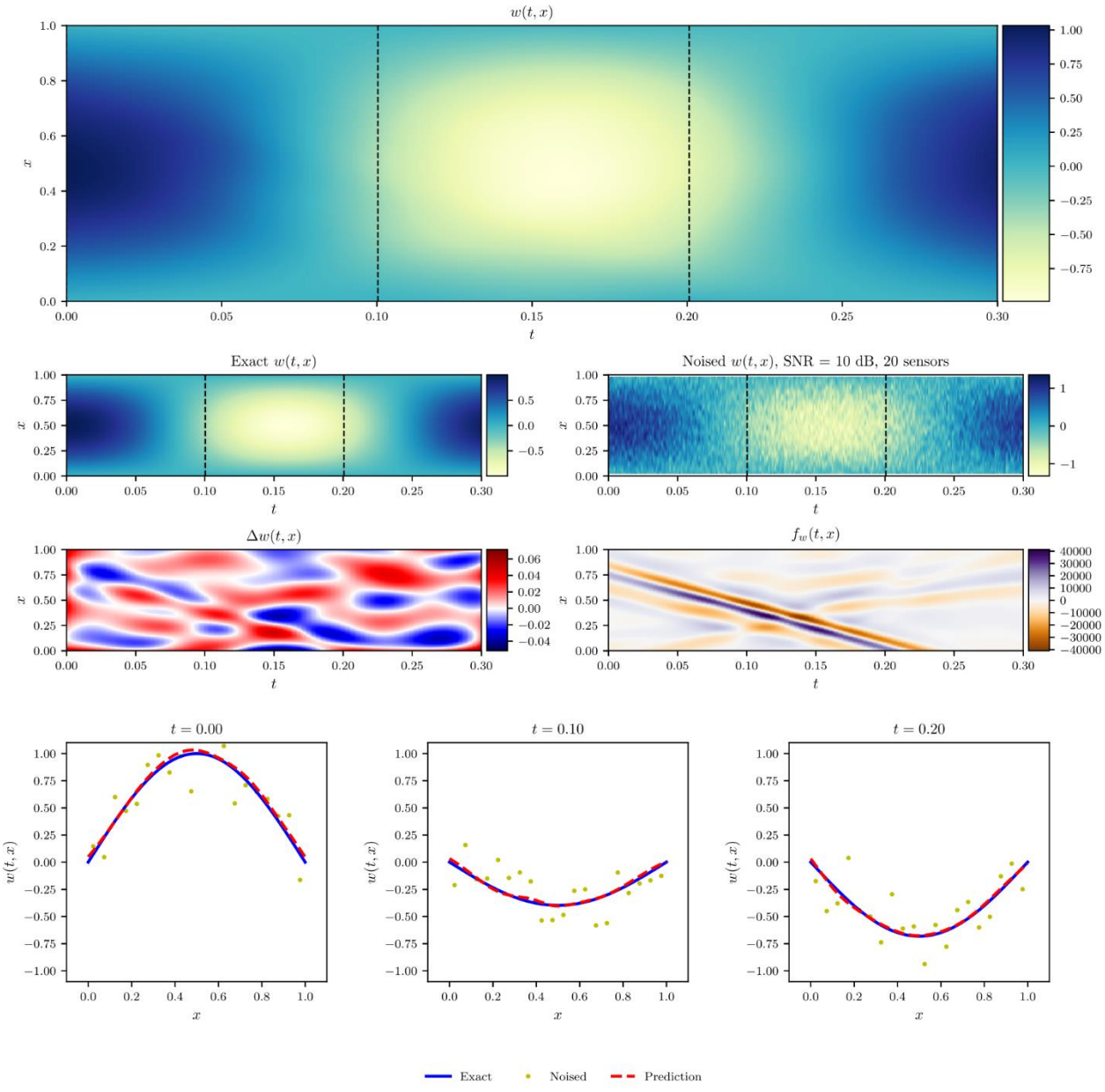

Figure 35. Data-driven result for Problem 2 given in Table 14.

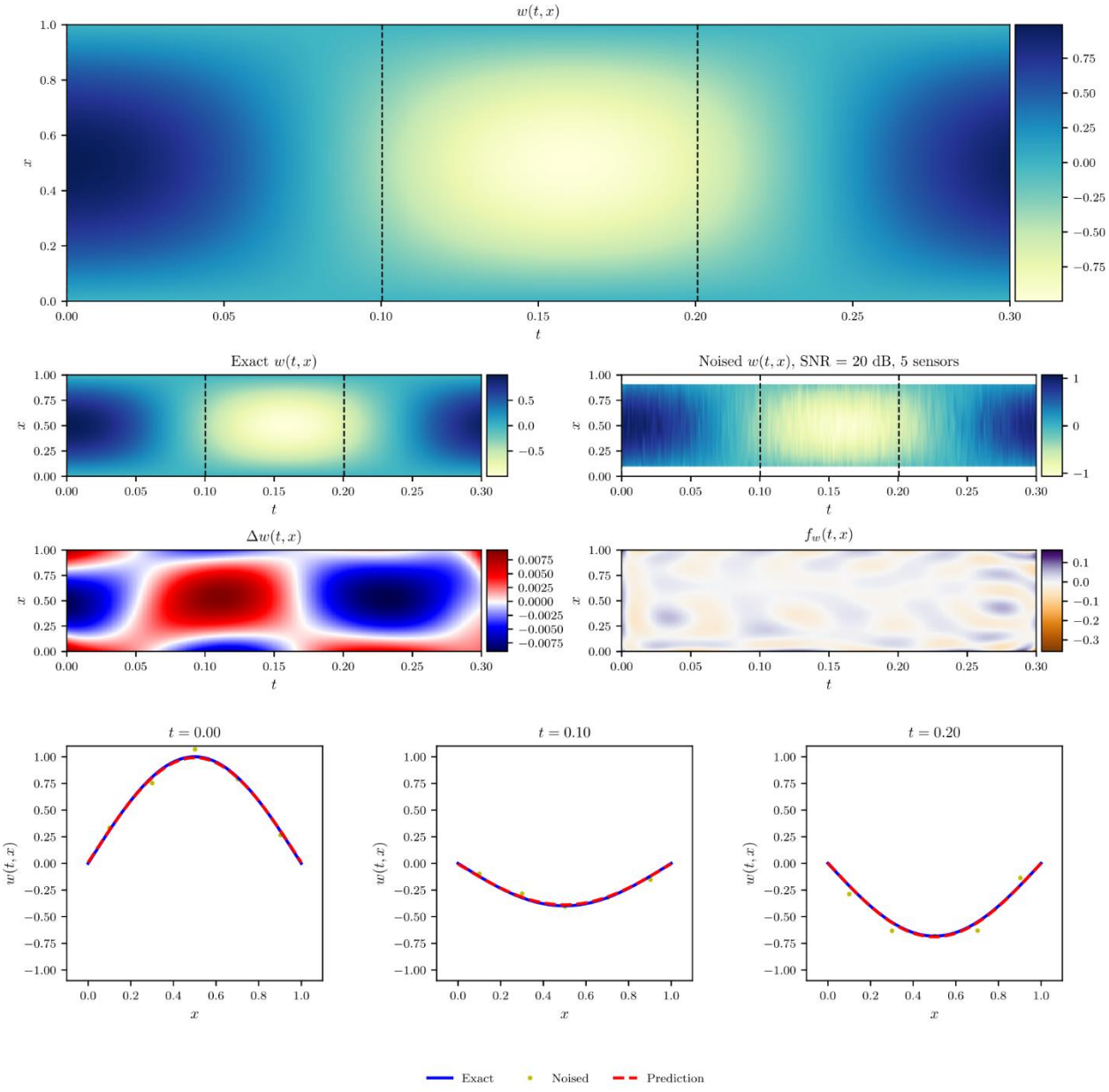

Figure 36. Physic informed result for Problem 3 given in Table 15.

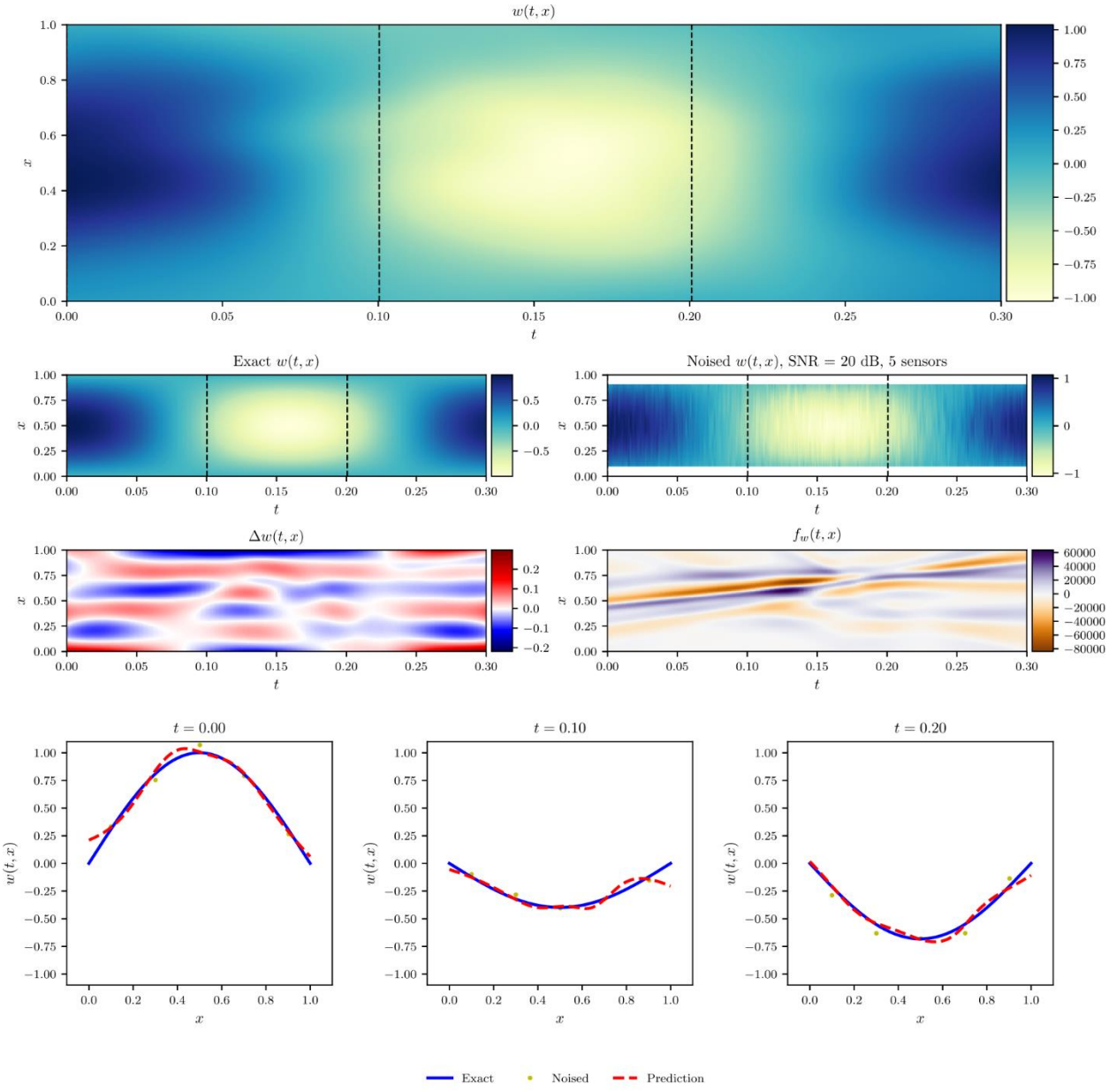

Figure 37. Data-driven result for Problem 3 given in Table 15.

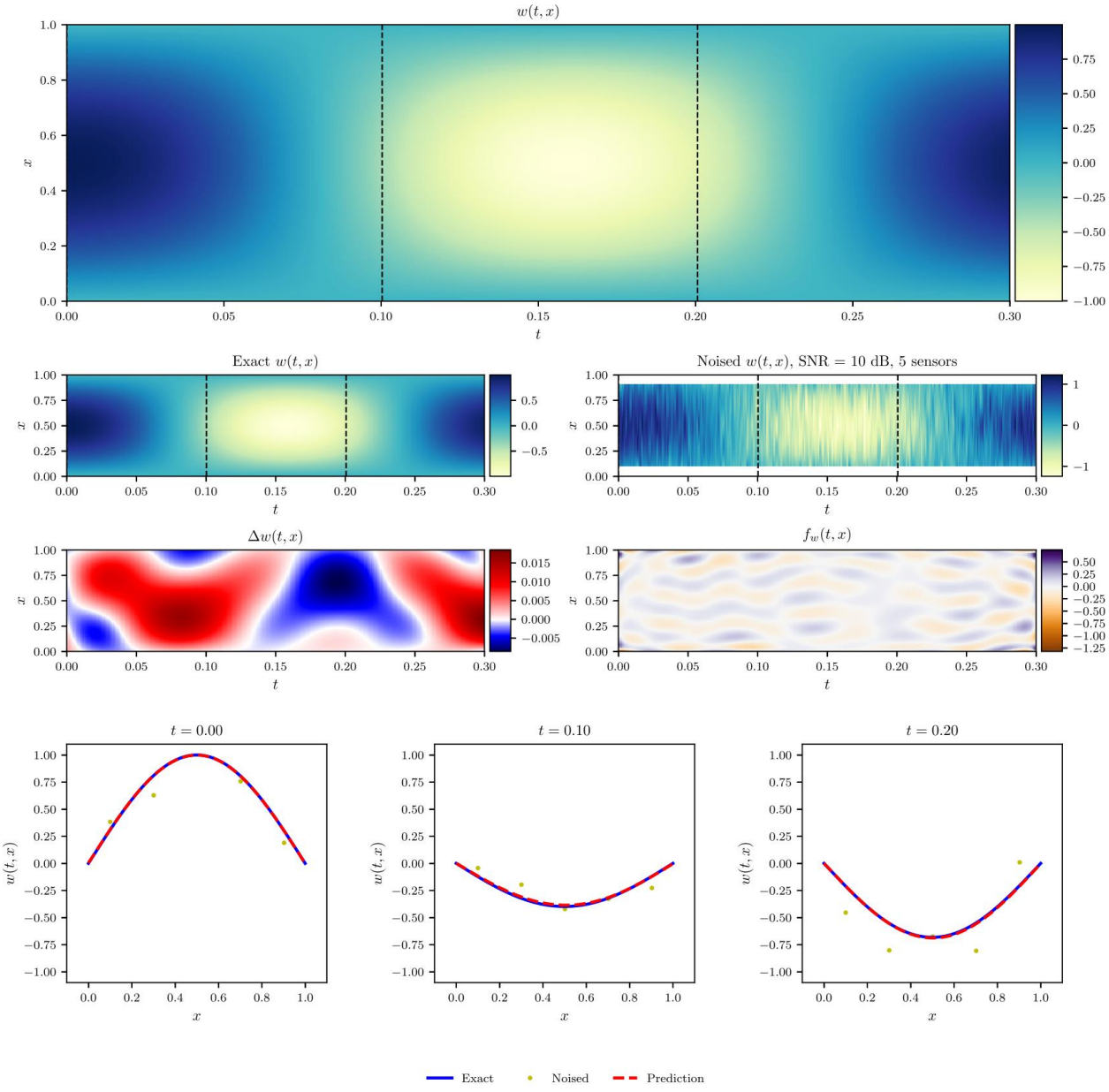

Figure 38. Physic informed result for Problem 4 given in Table 16.

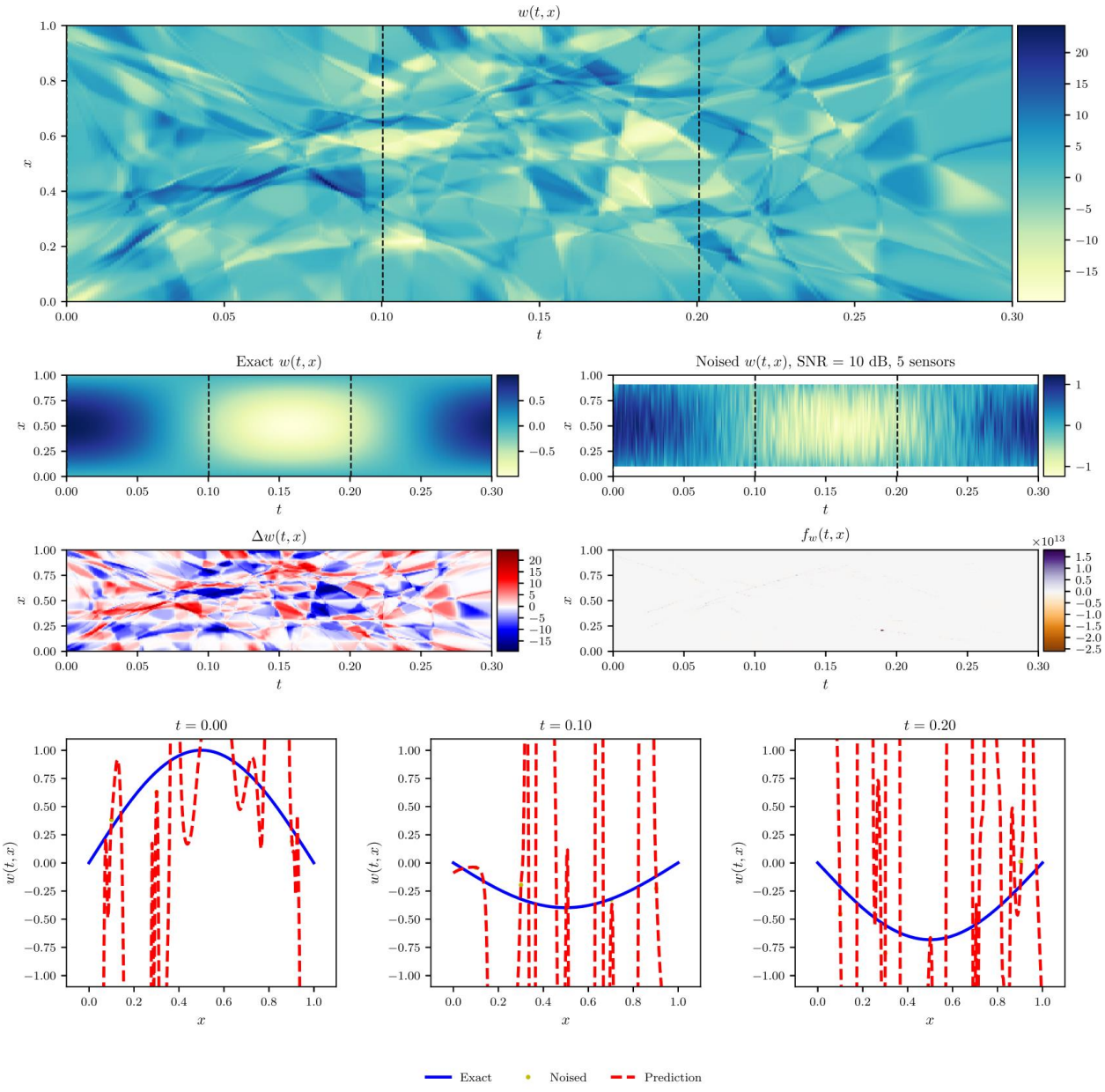

Figure 39. Data-driven result for Problem 4 given in Table 16.

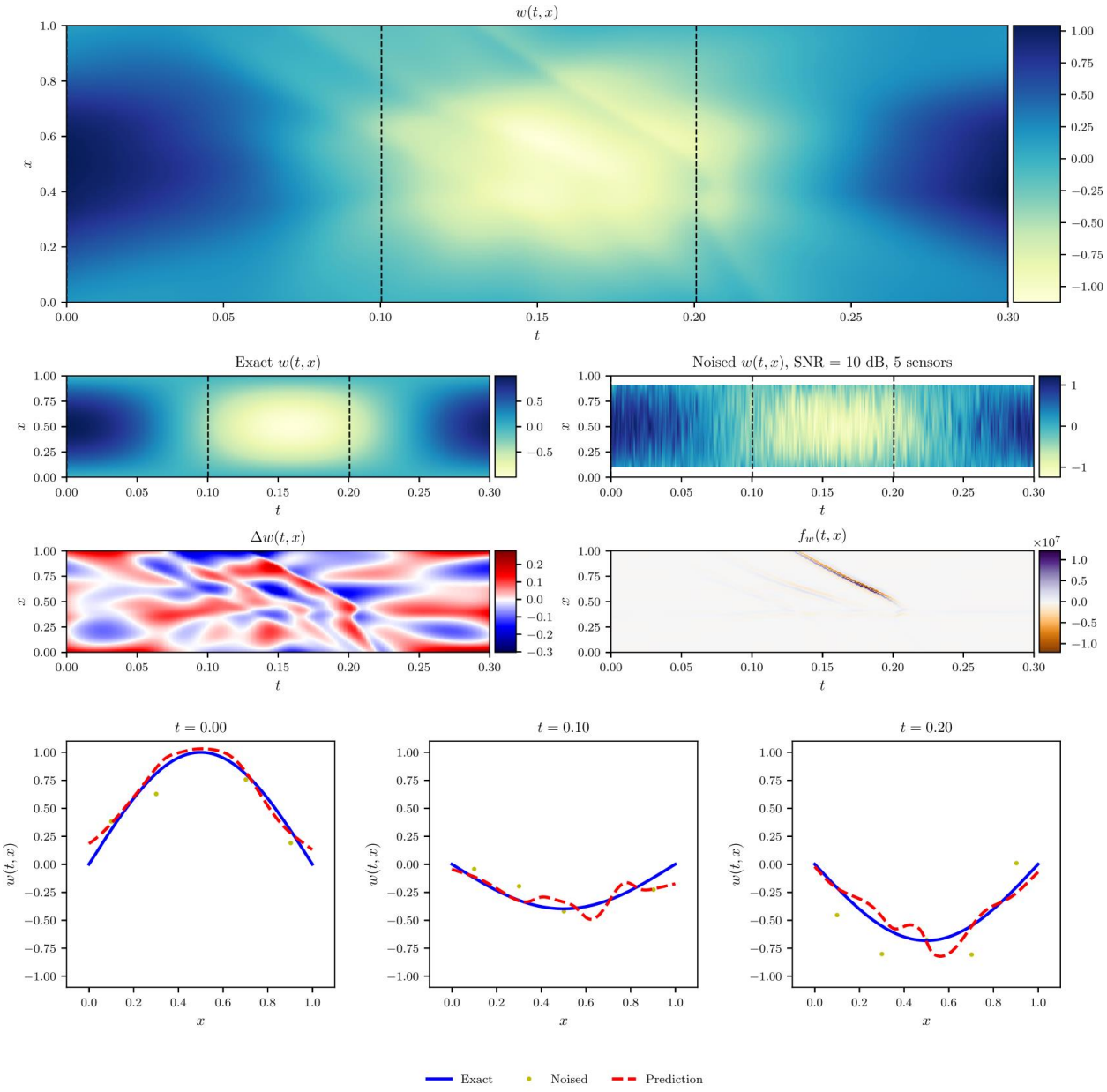

Figure 40. Gradient-descent-only data-driven result for Problem 4 given in Table 16.

Complicated Displacement Field

Though having demonstrated its power in dealing with noise and scarcity of data, the displacement field we experimented on is too simple to prove this method's efficacy. A more complicated displacement field can further unearth the potential deficiency of it. For this reason, Problem 5 given in Table 18 is solved:

Table 18. Problem 5: more complicated displacement field

λ	T	X	SNR_{dB}	Resolution	Structure of $\widehat{w}$	Exact Solution $w(x,t)$
4	$[0, 0.3]$	$[0, 1]$	20 dB	20 sensors, 1000 Hz	$[2{,}100 \times 3{,}1]$	$\sin 2\pi x \cdot \cos 8\pi^2 t$

Its physics informed result given in Figure 41 suggests that it has difficulty in convergence. As shown in the first row of Figure 41, most part of the approximated displacement field approaches zero in the end. This difficulty results from the physics informed terms, as the purely data-driven method achieved a result different from zero, though it encounters a severe overfitting problem.

Two conclusions can be drawn from this experiment. First, the complexity of the MLP is too high to properly fit those data points. It might be appealing to use a shallower and narrower network. Secondly, the physics informed term in loss function 5.2.1 seems to dominate the data-driven term, so that a slight adjustment of weights in favor of reducing the data-driven term could induce a larger increase in the physics informed term, such that the parameters of the MLP are left stagnated at local minimum. To overcome this issue, intuitively the data-driven term of the loss functions 5.2.1 should be weighted more, i.e. its weight ρ_d should be increased from their initial value 1, so that:

$$L = \frac{\rho_d}{N_d} \sum_{t_i, x_i, \tilde{w}_i \in \mathcal{X}}^{N_d} |\hat{w}(x_i, t_i) - \tilde{w}_i|^2 + \frac{1}{N_f} \sum_{t_i, x_i \in \Omega}^{N_f} \left| \left(\lambda \frac{\partial^4}{\partial x^4} + \frac{\partial^2}{\partial t^2} \right) \hat{w}(x_i, t_i) \right|^2$$
$$+ \frac{1}{N_b} \sum_{t_i \in T}^{N_b} \left((|\hat{w}(0, t_i)|^2 + |\hat{w}(1, t_i)|^2) + (|\hat{w}_{xx}(0, t_i)|^2 + |\hat{w}_{xx}(1, t_i)|^2) \right) \quad (5.2.2)$$

Problem 6 given in Table 19 are solve with those 2 propositions adopted and its result illustrated in Figure 43 shows that the convergence issue is eliminated.

Table 19. Problem 6: more complicated displacement field with weighted loss

λ	T	X	SNR_{dB}	Resolution	Structure of $\hat{w}$	ρ_d	Exact Solution $w(x,t)$
4	$[0, 0.3]$	$[0, 1]$	20 dB	20 sensors, 1000 Hz	[2,50,1]	5000	$\sin 2\pi x \cdot \cos 8\pi^2 t$

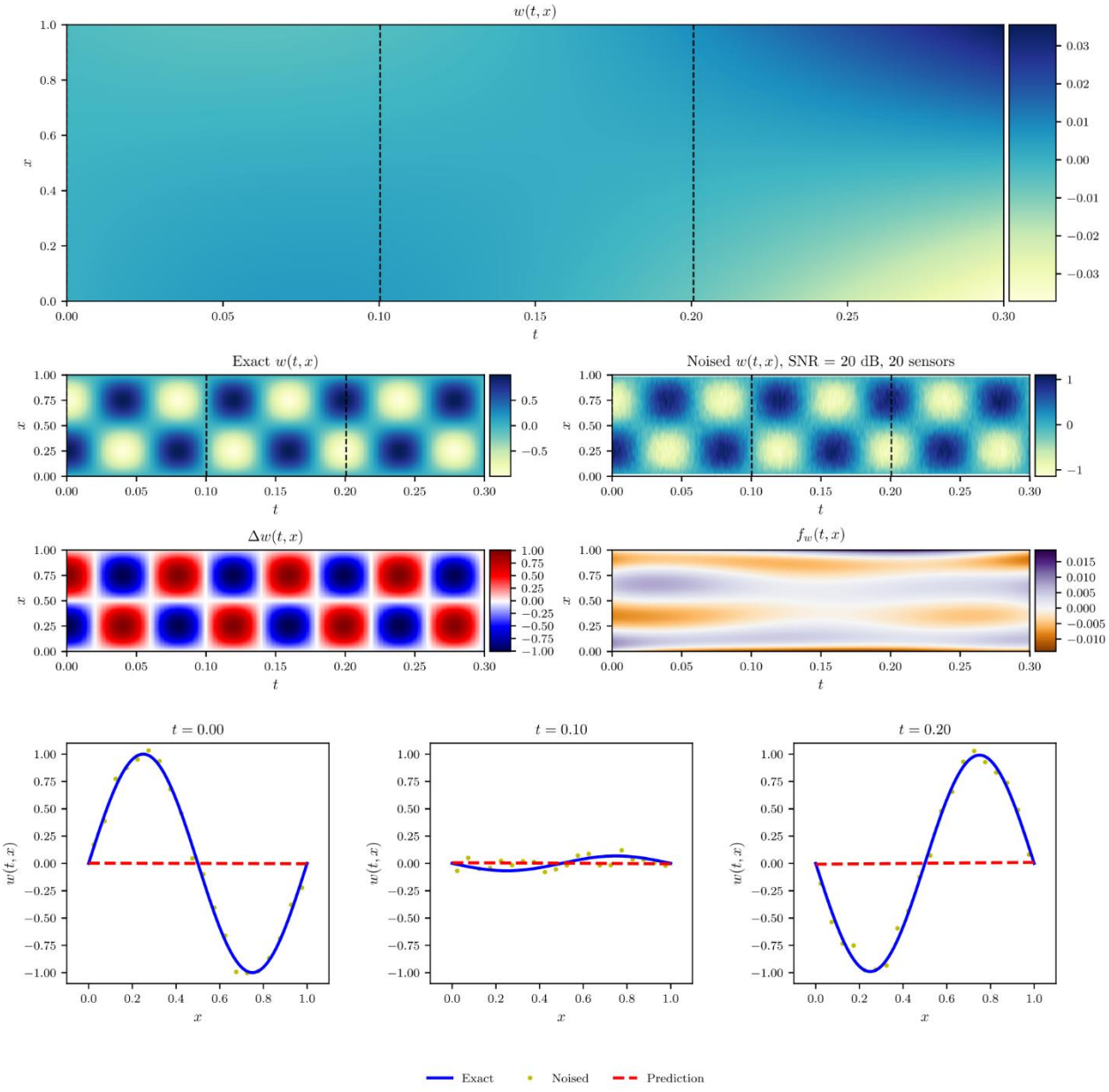

Figure 41. Physic informed result for Problem 5 given in Table 18.

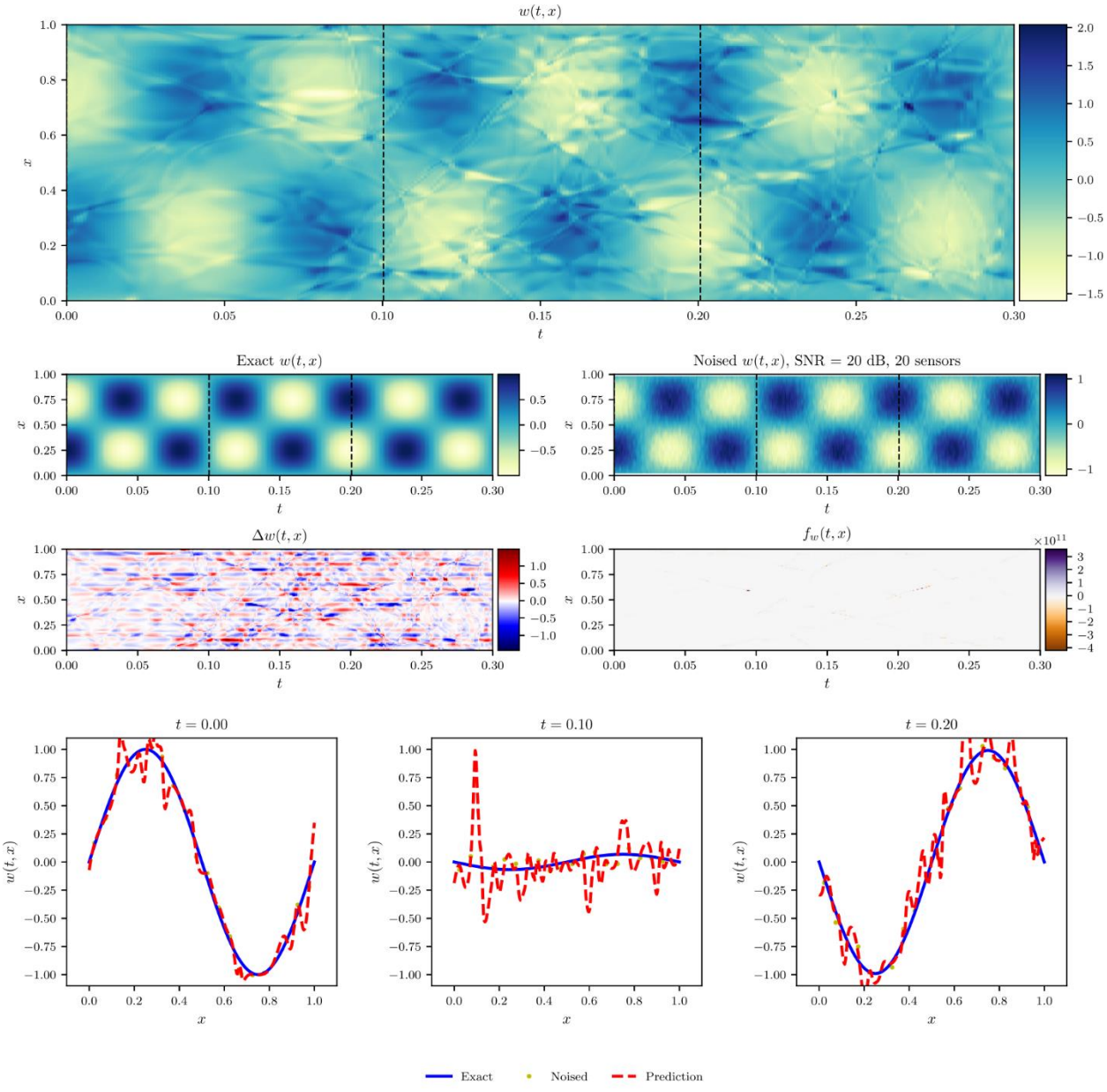

Figure 42. Data-driven result for Problem 5 given in Table 18.

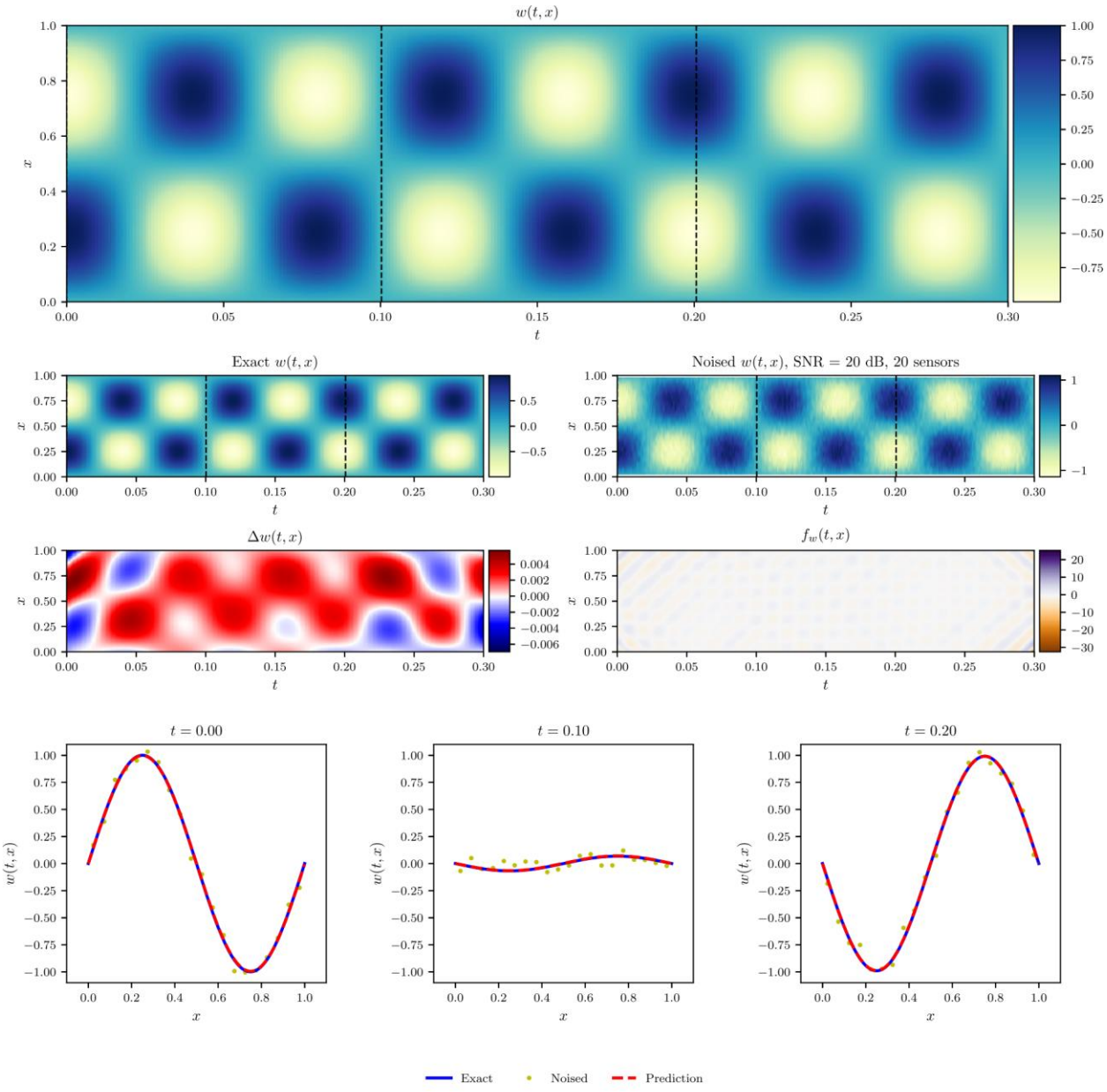

Figure 43. Physics Informed result for Problem 6 given in Table 19.

Complicated Displacement Field with Greater Noise and Sparser Sensors

Those 2 severe situations, i.e. greater noise and sparser sensors should be examined again for this complicated displacement field to further discover PIL's limitation. Problem 7 and 8 specified in Table 20 and Table 21 are established on Problem 6 with noise increased and sensors reduced respectively. Their PIL results are given in Figure 44 and Figure 45. It can be noticed that the approximated displacement field doesn't satisfy the boundary conditions well, especially the part close to the initial stage.

Table 20. Problem 7: more complicated displacement field with weighted loss and increased noise

λ	T	X	SNR_{dB}	Resolution	Structure of $\widehat{w}$	ρ_d	Exact Solution $w(x,t)$
4	$[0, 0.3]$	$[0, 1]$	10 dB	20 sensors, 1000 Hz	[2,50,1]	5000	$\sin 2\pi x \cdot \cos 8\pi^2 t$

Table 21. Problem 8: more complicated displacement field with weighted loss and less sensor

λ	T	X	SNR_{dB}	Resolution	Structure of $\widehat{w}$	ρ_d	Exact Solution $w(x,t)$
4	$[0, 0.3]$	$[0, 1]$	20 dB	10 sensors, 1000 Hz	[2,50,1]	5000	$\sin 2\pi x \cdot \cos 8\pi^2 t$

Two things could be done to mitigate this defect. The first one is to increase the weight ρ_{b1} of the terms for the Dirichlet boundary conditions accordingly. The loss function becomes:

$$L = \frac{\rho_d}{N_d} \sum_{t_i, x_i, \widetilde{w}_i \in \mathcal{X}}^{N_d} |\widehat{w}(x_i, t_i) - \widetilde{w}_i|^2 + \frac{1}{N_f} \sum_{t_i, x_i \in \Omega}^{N_f} \left| \left(\lambda \frac{\partial^4}{\partial x^4} + \frac{\partial^2}{\partial t^2} \right) \widehat{w}(x_i, t_i) \right|^2$$
$$+ \frac{1}{N_b} \sum_{t_i \in T}^{N_b} \left(\rho_{b1} (|\widehat{w}(0, t_i)|^2 + |\widehat{w}(1, t_i)|^2) + (|\widehat{w}_{xx}(0, t_i)|^2 + |\widehat{w}_{xx}(1, t_i)|^2) \right) \quad (5.2.3)$$

The second action is to divide the whole training process into 2 consecutive steps. In the first step, the MLP is trained by gradient descent on traditional data-driven loss function incorporating measurements and Dirichlet boundary conditions in order to both rapidly and

roughly determine the parameters of the MLP. Upon finished, the MLP is then finely trained by quasi-Newton L-BFGS-B algorithm on physics informed loss function incorporating all the known information.

After a series of experiments on a trial and error basis, it is finally determined that the weights for the Dirichlet boundary conditions should be 5 times as much as that for the measurements, i.e. $\rho_{b1} = 25000$. The two-step optimization is not always required but helps to overcome convergence issues in many cases. The efficacy of these 2 actions can be testified by the result of its application on Problem 7, which is given in Figure 46.

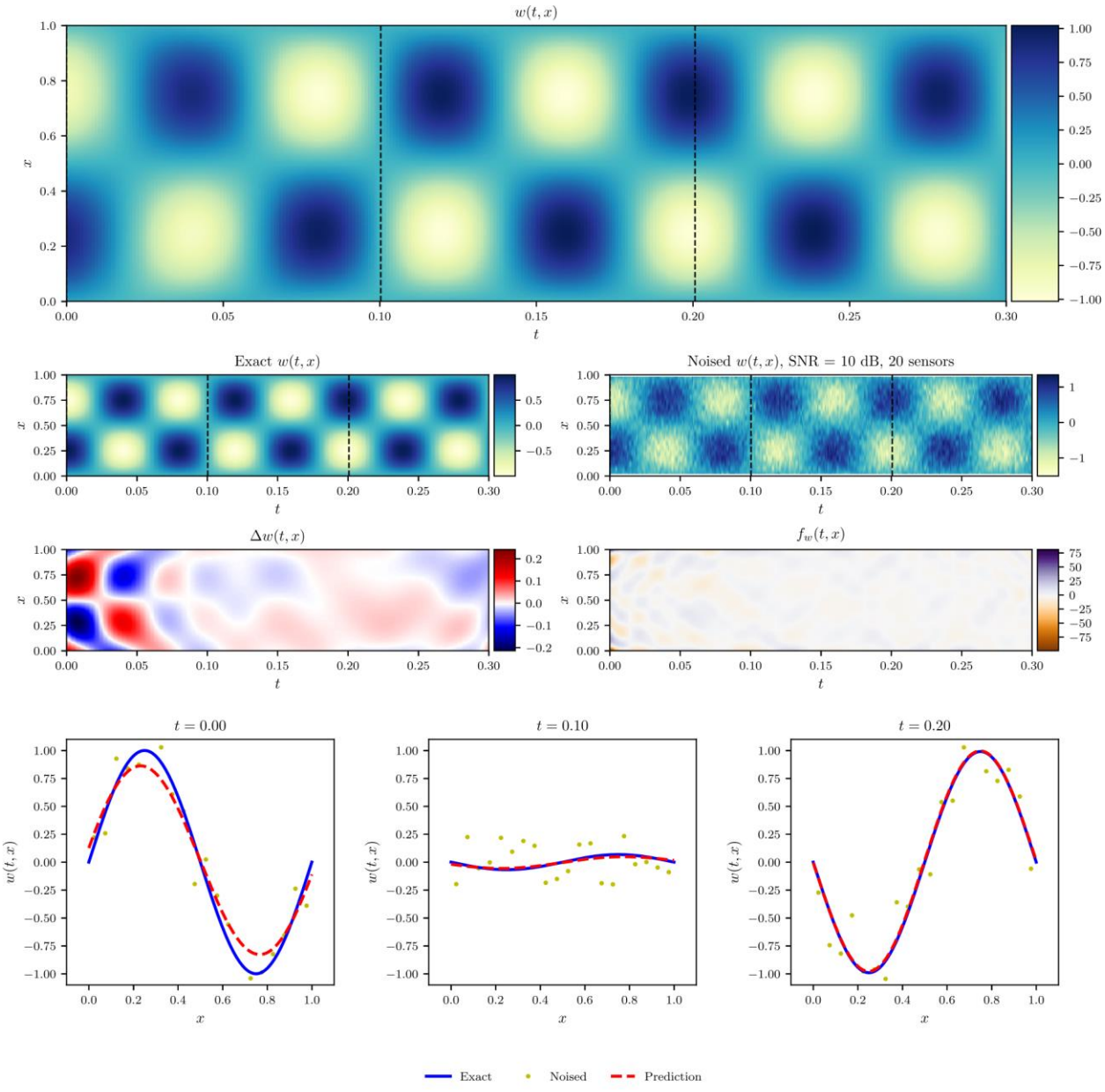

Figure 44. Physics Informed result for Problem 7 given in Table 20.

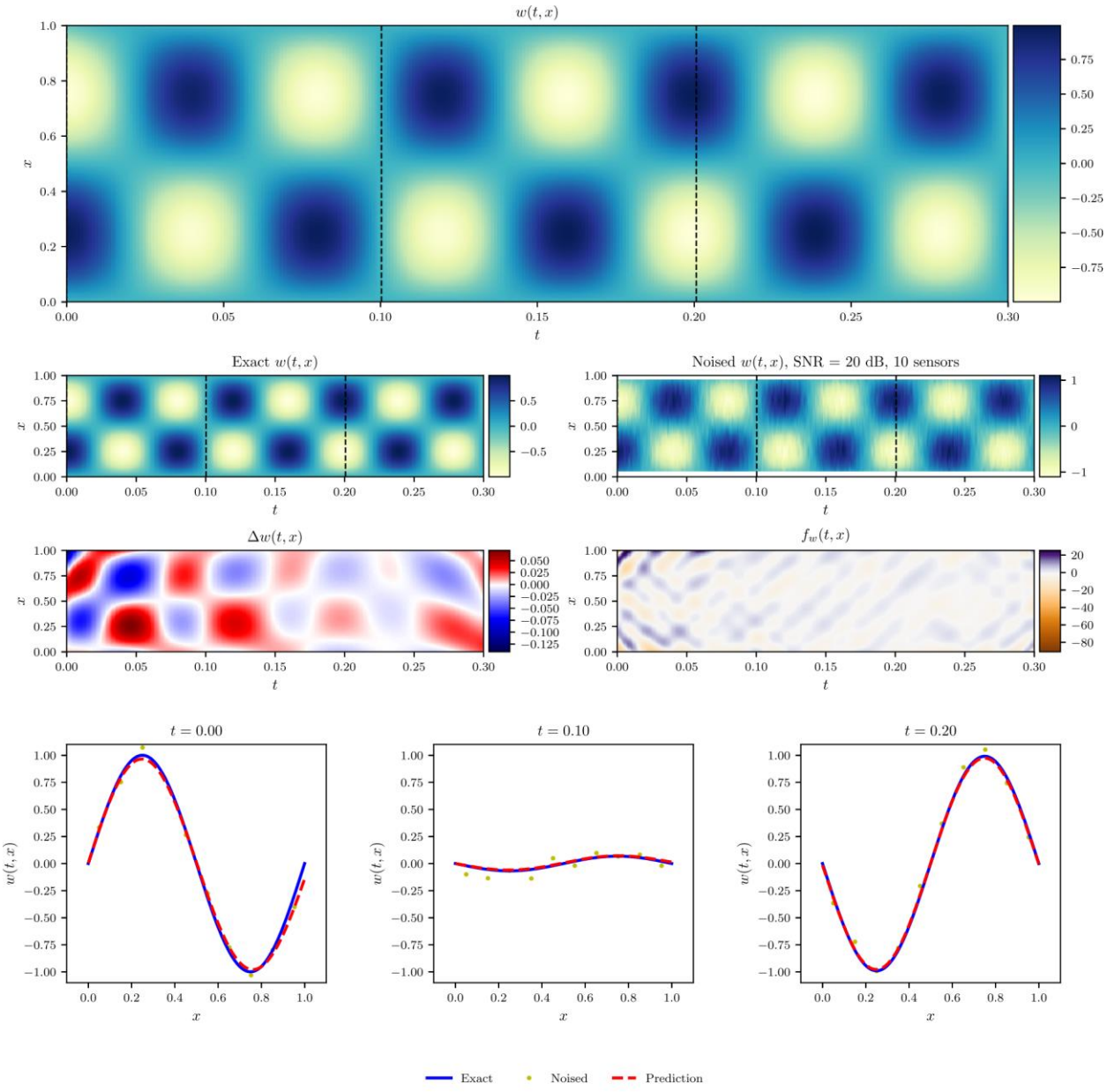

Figure 45. Physics Informed result for Problem 8 given in Table 21.

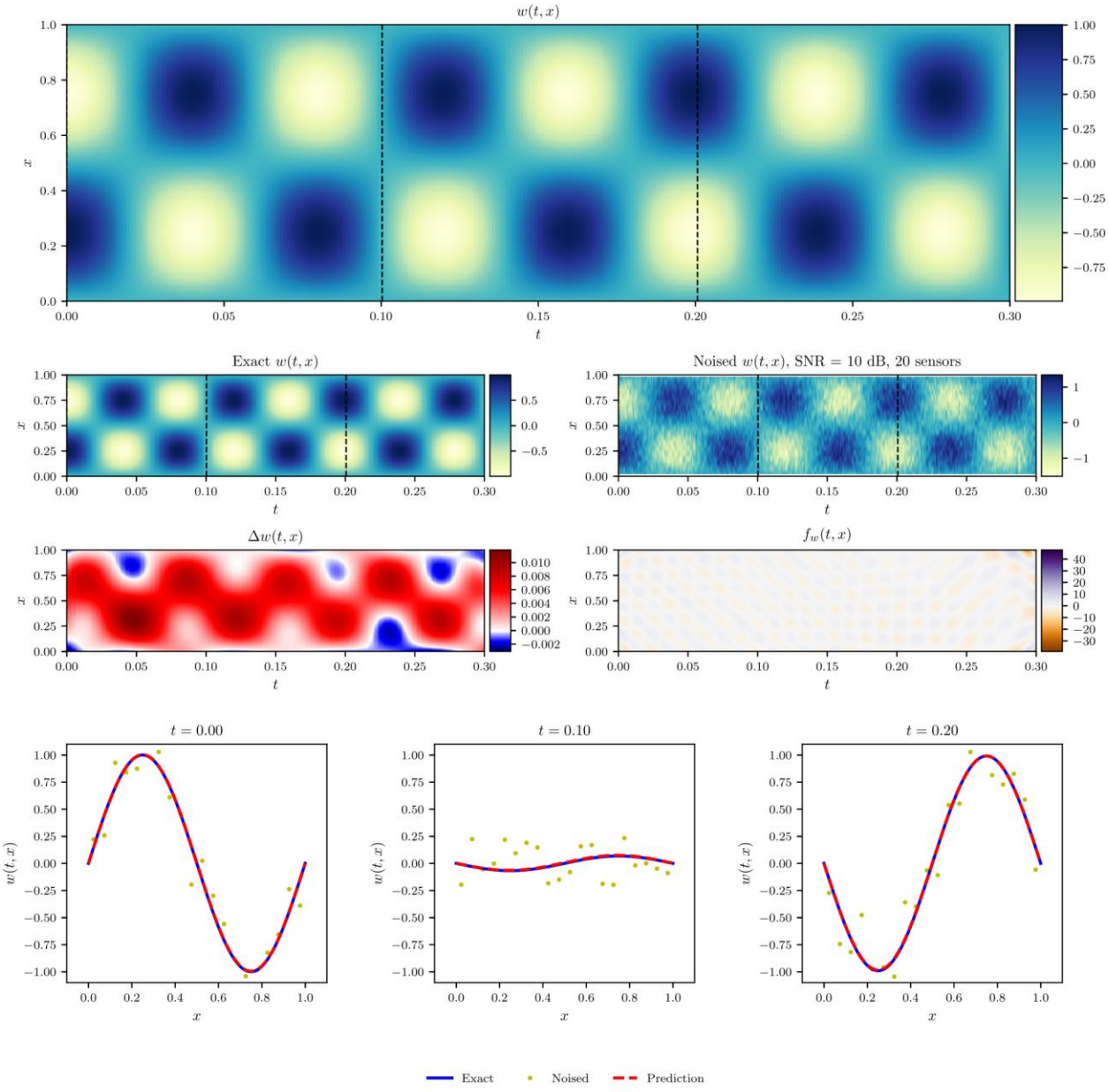

Figure 46. Improved physics informed result for Problem 7 given in Table 20.

5.3. Parameter Identification

According to Chapter 3, even if some coefficients are not provided, the governing equation can still be used to regularize the signal reconstruction process by setting the unknown coefficients as trainable parameters. Most importantly, the unknown coefficients can be derived through this process along with the displacement field. This is the foundation of PIL's application to inverse problems.

5.3.1. Methodology

All the parts except the loss function and the training process are identical to those for Section 5.2. The two exceptions are designed afresh based on the experiment results and experience obtained in the previous section.

Loss Function

To retrieve unknown coefficients while reconstructing the exact displacement field, the coefficient λ should be set as a trainable parameter to be updated during training process. As per the previous section, the loss function for training $\widehat{w}$ should be divided into 2 parts for the 2-step optimization:

$$L_{data} = 5000 \cdot \left\{ \frac{1}{N_d} \sum_{x_i,t_i,\widetilde{w}_i \in \mathcal{X}}^{N_d} |\widehat{w}(x_i,t_i) - \widetilde{w}_i|^2 + 5 \cdot \frac{1}{N_b} \sum_{t_i \in T}^{N_b} (|\widehat{w}(0,t_i)|^2 + |\widehat{w}(1,t_i)|^2) \right\} \quad (5.3.1)$$

$$L_{gvn} = \frac{1}{N_g} \sum_{x_i,t_i \in \Omega}^{N_g} \left| \left(\lambda \frac{\partial^4}{\partial x^4} + \frac{\partial^2}{\partial t^2} \right) \widehat{w}(x_i,t_i) \right|^2 + \frac{1}{N_b} \sum_{t_i \in T}^{N_b} (|\widehat{w}_{xx}(0,t_i)|^2 + |\widehat{w}_{xx}(1,t_i)|^2) \quad (5.3.2)$$

where the first one is the purely data-driven loss comprising loss for measurements and Dirichlet boundary conditions, the second one is the loss for governing equation consisted of the residuals for PDE and high order boundary conditions. The unknown parameter λ is set as trainable parameters here along with $\boldsymbol{\theta}$ of $\widehat{w}(t,x;\boldsymbol{\theta})$ and initialized with an arbitrary value. Its final value after training is taken as an approximation of the exact λ.

Training

As mentioned before, the training process is divided into 2 steps. L_{data} is used for the first step to rapidly bring parameters of the MLP into the proximity of the global minimum. To this end, the parameters of the MLP $\widehat{w}$ are trained with Adam optimizer with a learning rate of 0.01 for 10000 iterations. During each iteration, while all the N_d measurements are compiled into the first data-driven loss, the collocation points are uniformly sampled from the domain of definition to conduct stochastic gradient descent. To be specific, $N_b = 50$ points are resampled for the Dirichlet boundary conditions during each iteration. After that, the parameters are further optimized by L-BFGS-B on a larger number of static collocation points, i.e. $N_b = 150$ until the loss converges.

In the second phase, the sum $L_{data} + L_{gvn}$ is set as loss function to surmount overfitting problem. The parameters are then fine-tuned by L-BFGS-B on a larger number of static collocation points until the loss converges. The collocation points for L-BFGS-B optimizer are taken from an equally spaced 150×150 grid over the entire domain, so that $N_b = 150$ and $N_g = 22500$.

5.3.2. Result and Discussion

A series of problems in which the data is highly disturbed by noise are solved by this optimized PIL method to demonstrate its resistance against noise and biased dataset. In Problem 9, a displacement field composed of a single mode is reconstructed. Its result is shown in Figure 47. The unknown λ of the dynamic beam equation is initially set as 2 and finally updated to **0.9978**, whose relative error is only 0.22% against the exact value 1. The displacement field is also properly reconstructed as the maximum error is around 3%.

Table 22. Problem 9

λ	T	X	SNR_{dB}	Resolution	Structure of $\widehat{w}$	Exact Solution $w(x,t)$
1	$[0, 0.3]$	$[0, 1]$	10 dB	5 sensors, 1000 Hz	[2,50,1]	$\sin 2\pi x \cdot \cos 4\pi^2 t$

A more complicated case where the displacement field consists of 2 modes are examine in Problem 10 presented in Table 23 where λ_n are random variables uniformly sampled from

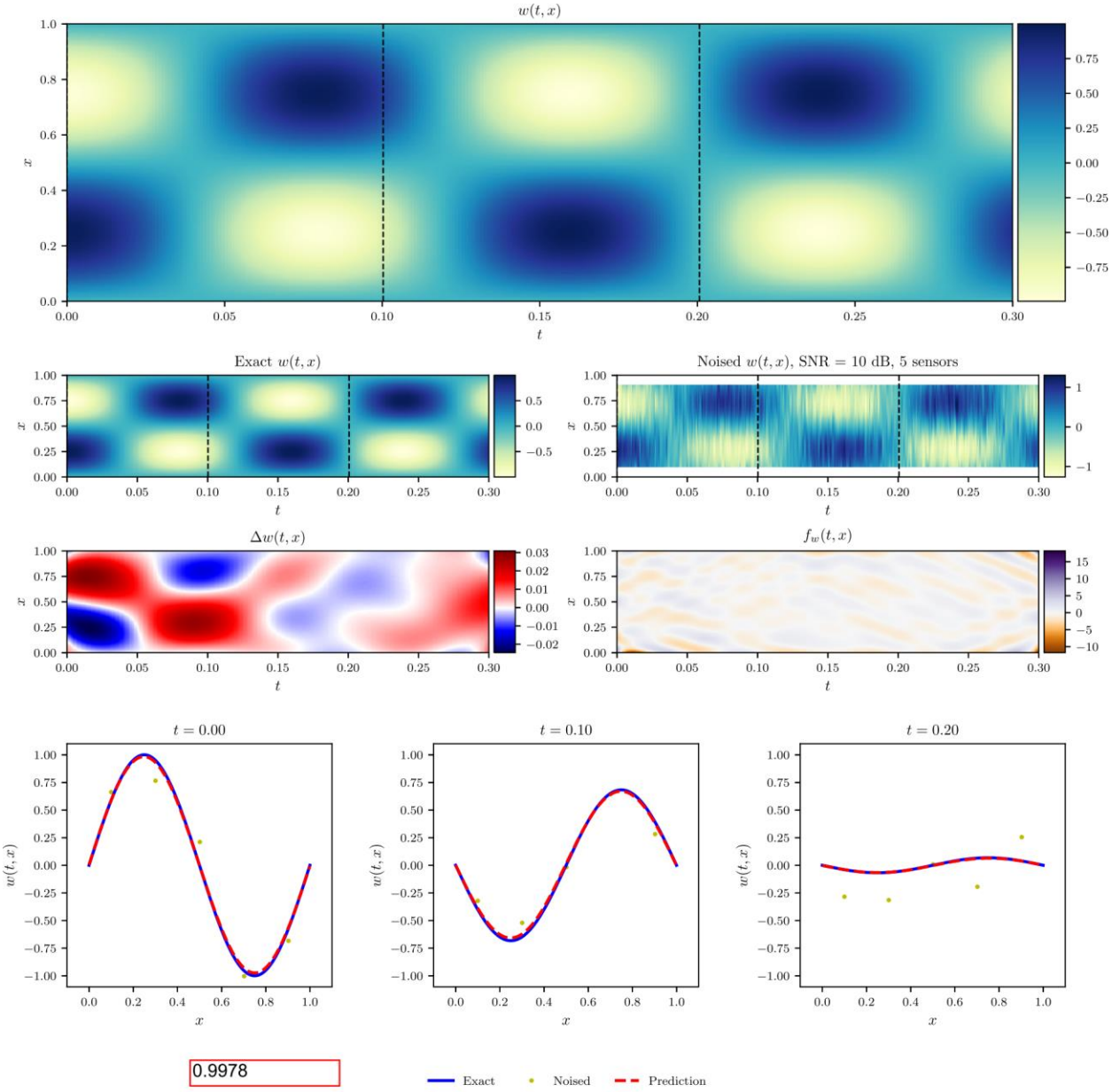

Figure 47. Result for Problem 9 given in Table 22.

$[-1, 1]$. The solution to this problem is given in Figure 48. The λ initially set to 2 is eventually determined to be **0.9964**, with a relative error of 0.36%. What is more amazing is that although the entire displacement field cannot be fully recovered because components of small wavelength cannot be retrieved when sensors are too sparsely distributed as per the Nyquist–Shannon sampling theorem, this PIL method still manages to recover the component of lower mode shape

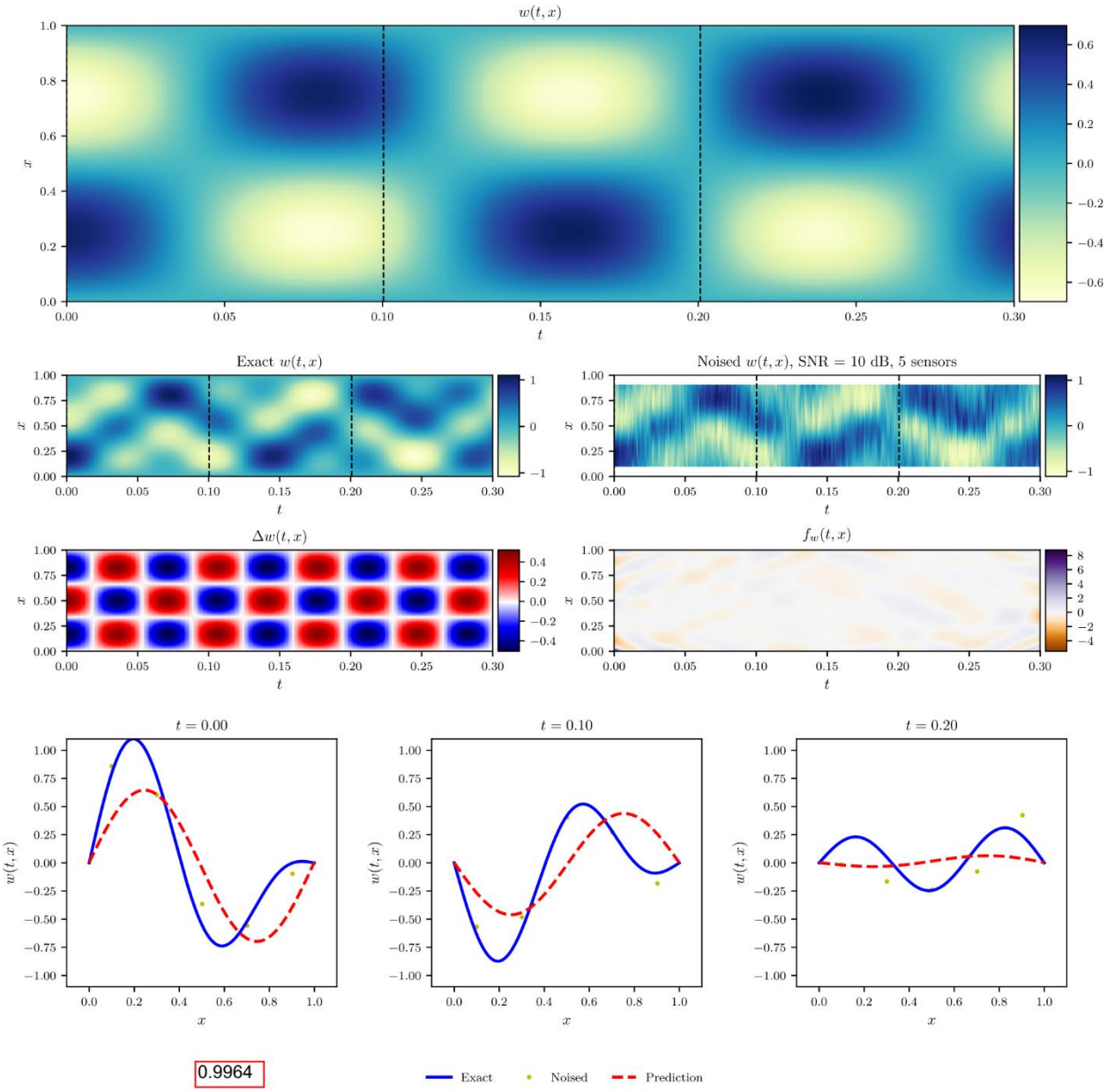

Figure 48. Result for Problem 10 given in Table 23.

from noisy measurement. This soundly proves the efficacy of PIL in obtaining physic-law-abiding result.

Table 23. Problem 10

λ	T	X	SNR_{dB}	Resolution	Structure of $\widehat{w}$	Exact Solution $w(x,t)$
1	$[0, 0.3]$	$[0, 1]$	10 dB	5 sensors, 1000 Hz	[2,200,1]	$\sum_{n=1}^{2} \lambda_n \sin n\pi x \cos n^2\pi^2 t$

Displacement field even more complicated can also be successfully processed such as Problem 11 and 12 given below:

Table 24. Problem 11

λ	T	X	SNR_{dB}	Resolution	Structure of $\widehat{w}$	Exact Solution $w(x,t)$
0.25	$[0, 0.3]$	$[0, 1]$	10 dB	20 sensors, 1000 Hz	[2,200,1]	$\sum_{n=1}^{3} \lambda_n \sin n\pi x \cos \frac{1}{4} n^2\pi^2 t$

Table 25. Problem 12

λ	T	X	SNR_{dB}	Resolution	Structure of $\widehat{w}$	Exact Solution $w(x,t)$
0.0156	$[0, 0.3]$	$[0, 1]$	10 dB	20 sensors, 1000 Hz	[2,100,100,1]	$\sum_{n=1}^{6} \lambda_n \sin n\pi x \cos \frac{1}{8} n^2\pi^2 t$

Their results are presented in Figure 49 and Figure 50 respectively. As to Problem 11, the λ initially set as 1 is determined to be 0.2492, with a relative error of 0.32% against 0.25. For Problem 12, λ is also set as 1 at first and it approaches 0.015665 with a relative error of 0.41% with respect to the exact value 0.0156. For both problems, their displacement fields are precisely reconstructed from noisy measurements.

Though performing well on the designed problems, PIL is still far from being perfect. First of all, the complexity of the MLP must increase accordingly with the complexity of the displacement field so as to be able to represent it. Yet the structure of the MLP can only be determined heuristically given that it might encounter convergence problem if the MLP's structure becomes too complex. Because of this, if the number of periods within the time

window increases, the time window needs to be properly truncated. Another way to bypass this problem is to divide the entire time span into short windows and process one at a time, just like what we accomplished in Section 4.6. Secondly, the weights of the terms in the loss function also have to be heuristically and properly chosen. The weights by far can only be determined after a few tentative experiments, and it seems that they are also problem-specific. Thirdly, not every component of the exact solution can be obtained if the amplitude of it is too low or the sensors are not densely distributed, as shown in Figure 48. More research should be conducted to determine and overcome these defects.

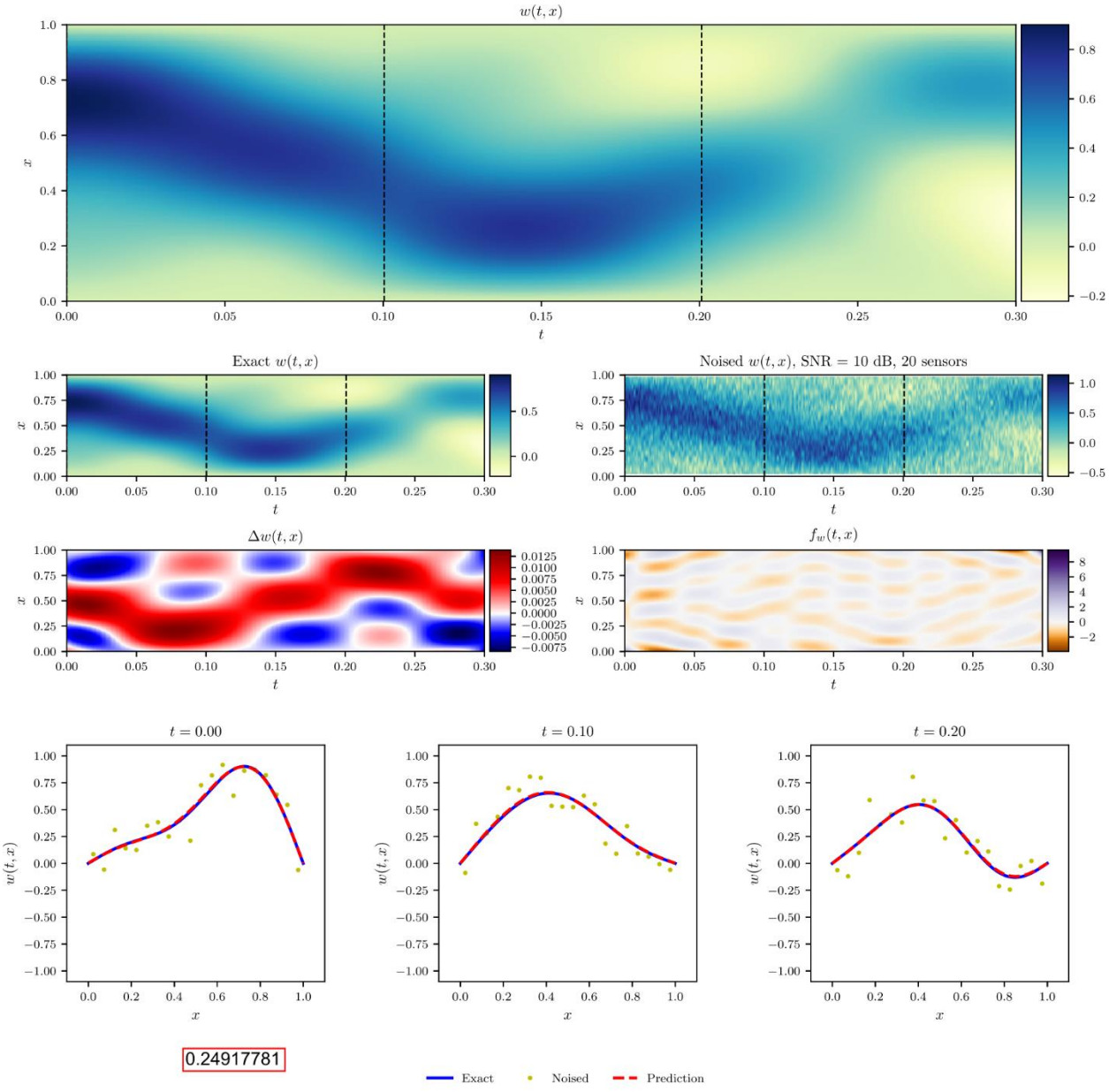

Figure 49. Result for Problem 11 given in Table 24.

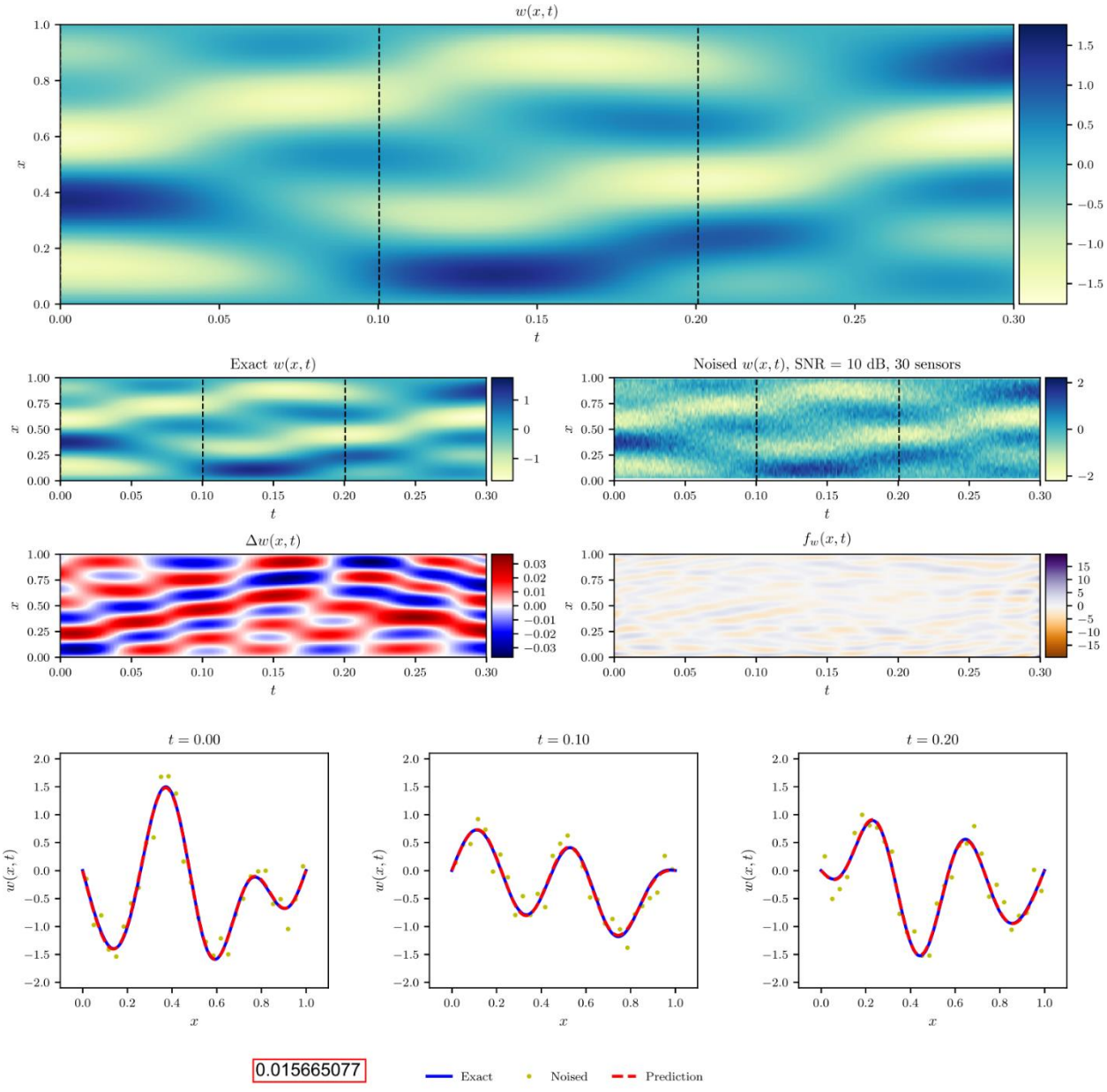

Figure 50. Result for Problem 12 given in Table 25.

CHAPTER 6

Summary and Outlook

6.1. Summary

PIL manages to blend prior knowledge in with raw data to overcome overfitting problems usually confronted by ANNs trained with purely data-driven methods. The prior knowledge usually embodied in differential equations is so informative such that the MLPs can still converge fairly well in face of the depletion of data while satisfying the governing equations. This enables it to be employed in solving both forward problems and inverse problems.

6.1.1. Summary for Forward Problems

Physics information embedded in strong formulations needs to be properly preprocessed by weighted residual methods before being incorporated into the training process. Among all the existing works, the collocation method is widely employed to cope with linear/nonlinear PDEs. It provides accurate results for forward problems governed by PDEs whose time domain is short and spatial order is no greater than 2.

In this work we extrapolated these methods to beam vibration problems over long time domain, which are governed by 4th order PDE. Unlike Alli et. al did in [12], we brought initial and boundary conditions into the loss function and turned the problem into a constrained optimization problem. We also used automatic differentiation to spare us the hassle of deriving

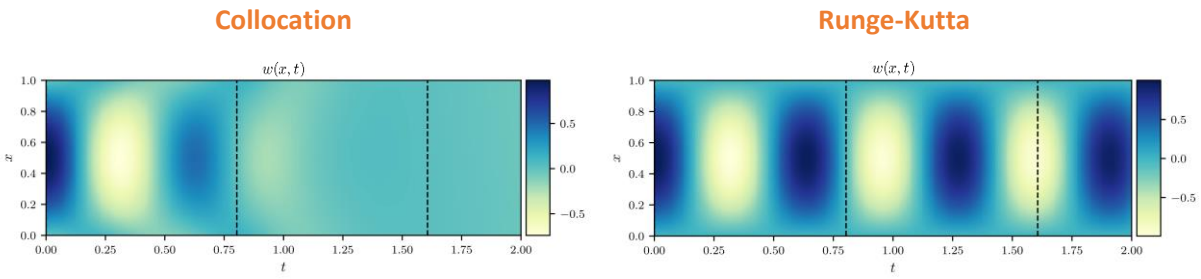

Figure 51. Comparison between collocation method and Runge-Kutta method on the same beam vibration problem.

high order derivatives. As the spatial order increases and the time domain extends, the collocation method encounters a pitfall of approaching trivial solution, which faces convergence difficulty. The Runge-Kutta method alleviates the convergence difficulty of the collocation method to some extend. Figure 51 shows that for the same beam vibration problem with a rather long time span, it does not encounter the same convergence problem as the collocation method does. Yet as the time domain extends even more, the same pitfall re-emerges. Besides, unlike these 2 methods above, PIL incorporating Ritz method faces great challenge when solving beam vibration problems.

One viable way to overcome this large time span convergence problem is to use segmented time solution inspired by the spirit of Huygens' principle, namely that we first divide the entire time domain into short time windows, then solve the PDE sequentially in time by taking the result at the end of the current time window as initial conditions for the next one, and combine them all together afterwards. As such, PIL can now be applied to arbitrarily large time span and achieve higher accuracy as the displacement field in each time window now requires less representability.

6.1.2. Summary for Inverse Problems

PIL's application in forward problems shows its great power to improve the generalization of MLP result when the data is biased, concentrated in a small region of the entire domain. This is why even when only data at the initial timestamp is provided, PIL still manages to obtain the displacement field over the entire domain. This power becomes especially useful when a set of data points scattered over the entire domain is provided, even when these points are noisy and insufficient. The result shows that data-driven method is favorable when the data is abundant and clean because of its short training time, but PIL is much more preferred because of

its robustness when the situation is less desirable. As demonstrated in Chapter 5, when we choose PIL to tackle those inverse problems, not only can the measurements being physics-law-abidingly fitted by an ANN, but also the unknown coefficient of the governing equation can be inversely determined through the 2 step training process, which costs less training time and can help us avoid convergence problem for complicated displacement field. PIL thus helps obtain much better generalization and alleviates the fret about ANNs' reliability.

Despite all the benefits demonstrated in either this work or the existing papers, PIL still has shortcomings to be addressed. Trivial solution pitfall suggests that when the dataset is too biased, this technique might encounter convergence problem and its occurrence could depend on the order and the magnitude of coefficients of the PDE. In addition, the representability of the ANN should be adequate to approximate the underlying function but shouldn't be too high as it might incur difficulty in convergence. The structure of the ANN determining representability thus should be properly designed before training. Furthermore, the terms in the loss function should be duly balanced but the weight of each can only be specified heuristically. More effort should be dedicated to these problems in the future.

6.2. Outlook

6.2.1. RNNs for Forward Problems

Semi-discretization

The MLP solutions so far implemented is quite analogous to space-time finite element whose trial solution is of the form $\widehat{w}(x,t;\boldsymbol{\theta}) = \sum_i \theta_i \cdot \phi_i(x,\ t)$ in which the parameters θ_i are constant. However, most finite element transient analyses today separate spatial and temporal variables by semi-discretization whereby the trial solution becomes $\widehat{w}(x,t;\boldsymbol{\theta}) =$

$\sum_i \theta_i(t) \cdot \phi_i(x)$, so that at each time step, the dynamic problem can be taken as a static problem, then $\boldsymbol{\theta}$ can be updated by the same algorithm at each time step after deriving $\ddot{\boldsymbol{\theta}}$ and $\dot{\boldsymbol{\theta}}$.

Inspired by semi-discretization, if the displacement field at any time step t, namely $\widehat{w}_t(x)$, can be approximated by a function $\widehat{w}(x;\boldsymbol{\theta}_t)$ parameterized by $\boldsymbol{\theta}_t$, we can assume that we can get the displacement field $\widehat{w}_{t+1}(x)$ at the next time step by deriving $\boldsymbol{\theta}_{t+1}$. $\boldsymbol{\theta}_{t+1}$ is supposed to depend on $\boldsymbol{\theta}_t$ and the current input $\boldsymbol{i}_{t+1}$. This mapping can be approximated by an RNN $\boldsymbol{\Phi}$ parameterized by $\boldsymbol{\mu}$, in other words, $\boldsymbol{\theta}_{t+1} = \boldsymbol{\Phi}(\boldsymbol{\theta}_t,\ \boldsymbol{i}_{t+1};\ \boldsymbol{\mu})$.

What the RNN Learns

A typical procedure of finite element transient analysis is comprised of three major steps:

1. For a typical transient analysis problem, e.g. vibration problem, the continuous solution $w_t(x)$ at time t is discretized into a parameter vector $\boldsymbol{w}_t$ by piecewise basis functions $\phi_i(x)$:

$$w_t(x) \cong \widehat{w}_t(x) = \widehat{w}(x;\boldsymbol{w}_t) = \sum_i w_{i,\,t} \cdot \phi_i(x).$$

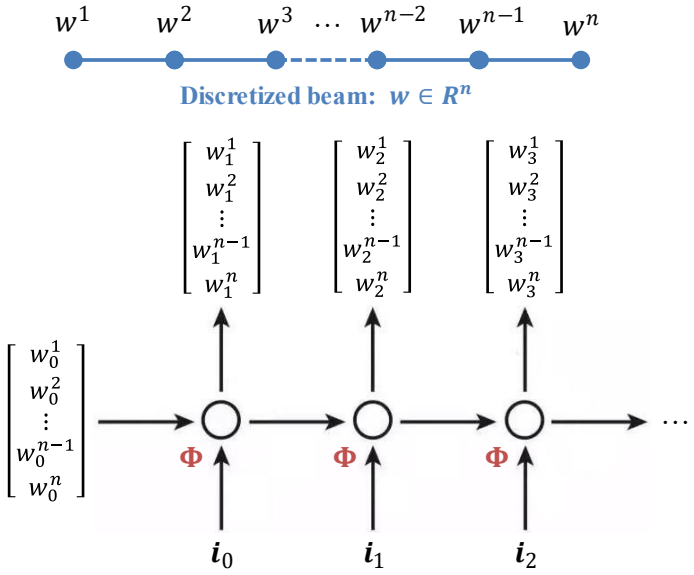

Figure 52. RNN discrete solution.

2. After that, if $\boldsymbol{w}_t$, $\dot{\boldsymbol{w}}_t$ and input $\boldsymbol{i}_t$ is known, they can be plugged into the governing equation discretized by FEM to derive $\ddot{\boldsymbol{w}}_t$.
3. $\dot{\boldsymbol{w}}_{t+1}$ and $\boldsymbol{w}_{t+1}$ can then be updated in turn by proper explicit time integration. Step 2 and 3 are then carried out recursively.

If we take a look at the procedure above, we can notice that the 2nd step can be represented as a mapping g by which $\ddot{\boldsymbol{w}}_t = g(\boldsymbol{w}_t, \dot{\boldsymbol{w}}_t, \boldsymbol{i}_t)$, and the 3rd time integration step can be represented as a mapping $\boldsymbol{h}$ by which $\boldsymbol{w}_{t+1}$, $\dot{\boldsymbol{w}}_{t+1} = \boldsymbol{h}(\boldsymbol{w}_t, \dot{\boldsymbol{w}}_t, \ddot{\boldsymbol{w}}_t)$. These 2 steps can be composited into a mapping $\boldsymbol{f}$ so that $\boldsymbol{w}_{t+1}$, $\dot{\boldsymbol{w}}_{t+1} = \boldsymbol{h}\big(\boldsymbol{w}_t \dot{\boldsymbol{w}}_t, g(\boldsymbol{w}_t \dot{\boldsymbol{w}}_t, \boldsymbol{i}_t)\big) = \boldsymbol{f}(\boldsymbol{w}_t, \dot{\boldsymbol{w}}_t, \boldsymbol{i}_t)$. This recursive procedure should be able to be approximated by an RNN $\boldsymbol{\Phi}$ parameterized by $\boldsymbol{\theta}$ as shown in Figure 52:

$$\boldsymbol{w}_{t+1}, \dot{\boldsymbol{w}}_{t+1} = \boldsymbol{\Phi}(\boldsymbol{w}_t, \dot{\boldsymbol{w}}_t, \boldsymbol{i}_t;\ \boldsymbol{\theta}), \qquad \widehat{w}_t(x) = \widehat{w}(x; \boldsymbol{w}_t)$$

Therefore, RNNs may learn the composition of governing equations and time integration.

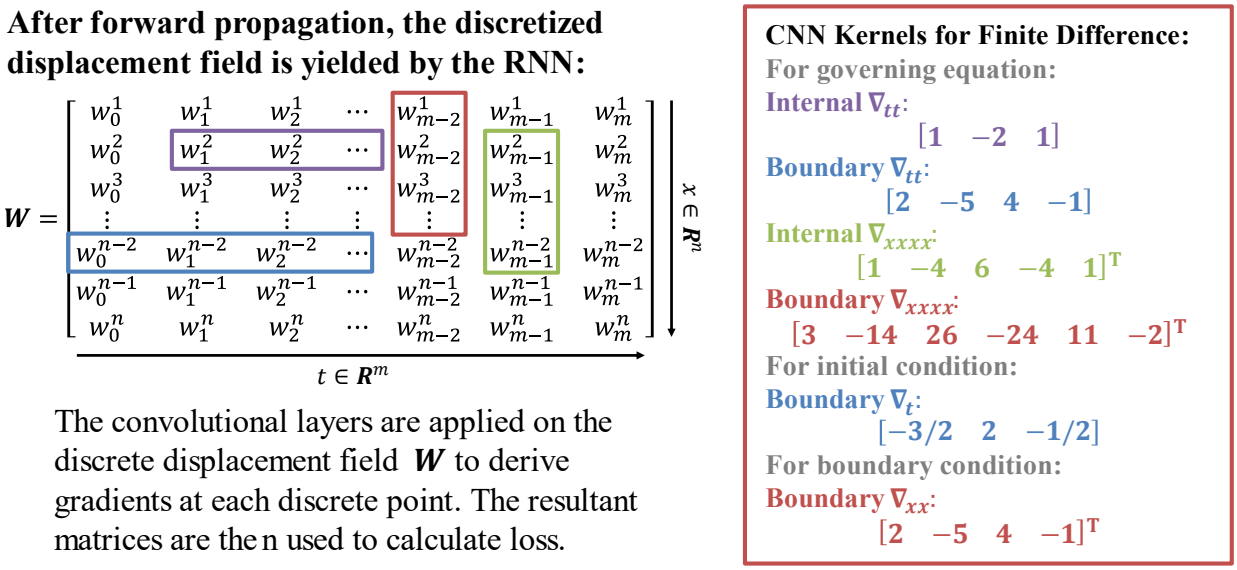

Figure 53. Convolutional layers serve as finite difference operators for the training of RNN solution.

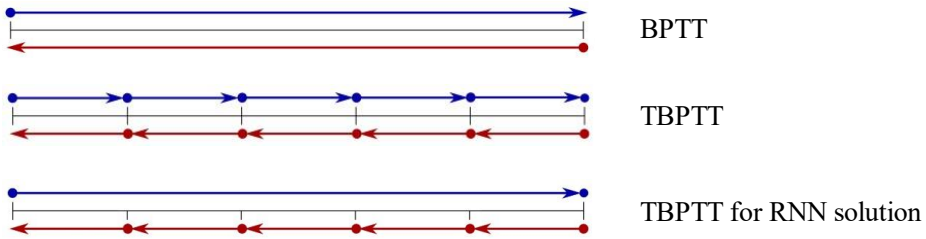

Figure 54. Different backpropagations though time.

Loss Function and Finite Difference

The general form of the loss function for this RNN solution is the same as Equation 4.3.4. The difference is that the gradients involved in that loss function can only be evaluated using finite difference here given that the output displacement field is discrete. Finite difference can be realized using convolutional layers whose kernels are set to be finite difference coefficients, as demonstrated in Figure 53.

Challenges

It appears that this solution can hardly converge in practice. One possible cause of this issue is vanishing or exploding gradient, which is usually encountered in the backpropagation through time (BPTT) of RNN if the input sequence is too long. Conventionally, the vanishing or exploding gradient problem is solved using truncated backpropagation through time (TBPTT), namely dividing the input sequence and corresponding output sequence in the dataset into small slices to form a batch, then using these short sequences to train RNNs. This cannot be done for our case because we don't have existing sequence data to slice. Yet, we can first perform a full-length forward propagation, then try to perform TBPTT by trimming the backpropagation at each time. These 3 types of BPTT are illustrated in Figure 54.

Another possible cause is the curse of dimensionality resulting from the number of dimensions of $\boldsymbol{w}_t$ and $\dot{\boldsymbol{w}}_t$, which makes $\boldsymbol{\Phi}$ hard to training. A proper encoder and decoder system may need to be developed to overcome this issue.

6.2.2. Multi-target ANN System for Damage Identification

The research in chapter 5 demonstrated PIL's power in solving simple inverse problems where the material property of a homogeneous beam is retrieved from data. Structural health monitoring problems are more advanced inverse problems in which we approximate the mapping from vibration measurements to damage location and severity. Damage makes the structure inhomogeneous, so the stiffness of the structure is quantified by a function of location and serves as an ideal indicator for the damage. The ultimate goal of this work is to identify the damage distribution on an object of higher spatial dimension, e.g. a 2D plane. The governing equation of plane's free vibration is:

$$\lambda(x,y)\cdot\Delta\Delta w+\rho w_{tt}=0,$$

where λ is the bending stiffness and ρ is density. The distribution of λ can then characterize the damage distribution of the plate. To approximate λ given measurements $\widetilde{w}_i$ of w, it is promising to approximate both λ and w with ANNs and train them with the loss function given below:

$$L=\frac{1}{N_d}\sum_{i=1}^{N_d}\left(\left|\widehat{w}\left(x_d^i,y_d^i,t_d^i\right)-\widetilde{w}_d^i\right|^2\right)+\frac{1}{N_b}\sum_{i=1}^{N_b}\begin{pmatrix}\left|\widehat{w}\left(0,y_b^i,t_b^i\right)\right|^2+\left|\nabla_{xx}\widehat{w}\left(0,y_b^i,t_b^i\right)\right|^2\\+\left|\widehat{w}\left(1,y_b^i,t_b^i\right)\right|^2+\left|\nabla_{xx}\widehat{w}\left(1,y_b^i,t_b^i\right)\right|^2\\+\left|\widehat{w}\left(x_b^i,0,t_b^i\right)\right|^2+\left|\nabla_{xx}\widehat{w}\left(x_b^i,0,t_b^i\right)\right|^2\\+\left|\widehat{w}\left(x_b^i,1,t_b^i\right)\right|^2+\left|\nabla_{xx}\widehat{w}\left(x_b^i,1,t_b^i\right)\right|^2\end{pmatrix}$$

$$+\frac{1}{N_f}\sum_{i=1}^{N_f}\left(\left|\left(\hat{\lambda}\left(x_f^i,y_f^i;\boldsymbol{\mu}\right)\cdot\Delta\Delta+\rho\frac{\partial^2}{\partial t^2}\right)\widehat{w}\left(x_f^i,y_f^i,t_f^i;\boldsymbol{\theta}\right)\right|^2\right)$$

where the first term is the MSE for fitting measurement data, the second is the loss for boundary conditions and the last one is the loss for the governing equation which integrates the desired $\hat{\lambda}$ indicating damage distribution and the displacement field approximation $\widehat{w}$. This forms a multi-target ANN system given in Figure 55. MLPs are suitable for $\widehat{w}$, but $\hat{\lambda}$ might have to be approximated by ANNs other than MLP given that structural damages are usually concentrated

at certain small regions. RBFN is worth considering for that purpose due to its compactly supported and location related basis functions. It is also worth considering integrating PIL into other SHM methods so far developed, like [43] and [44]. After all, there is still a long way to go before the method becomes full-fledged.

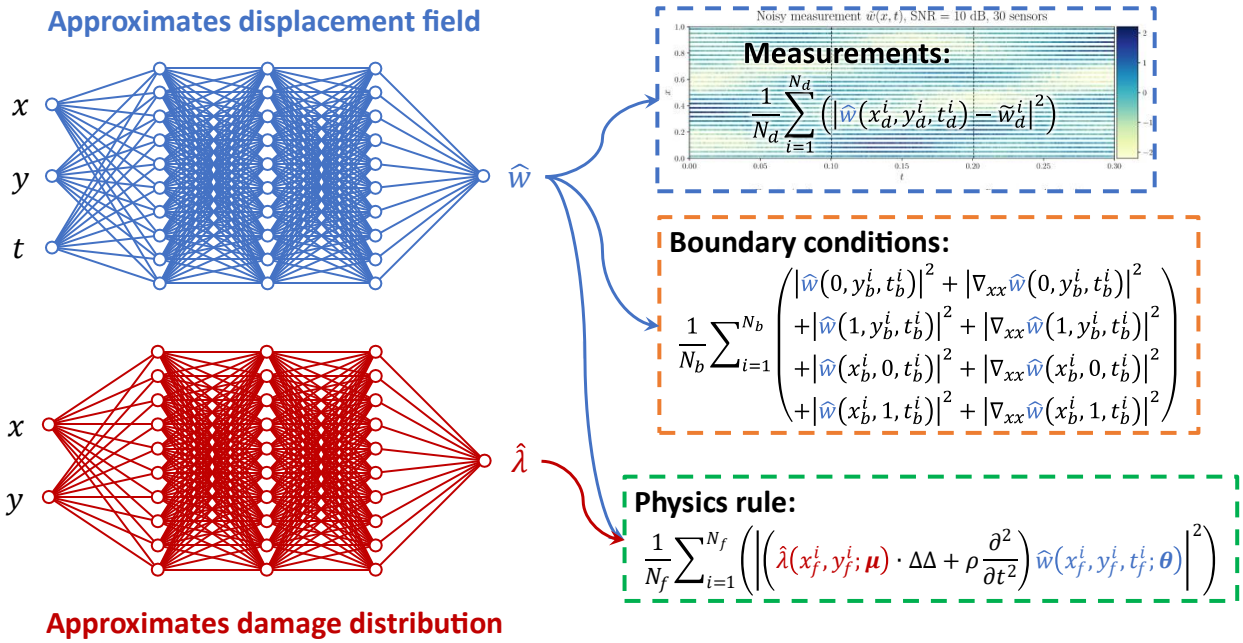

Figure 55. Multi-target ANN system.

REFERENCES

[1] S. C. Kleene, “Representation of Events in Nerve Nets and Finite Automata,” in *Automata Studies. (AM-34)*, C. E. Shannon and J. McCarthy, Eds. Princeton: Princeton University Press, 1956, pp. 3–42.

[2] F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain.,” *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958, doi: 10.1037/h0042519.

[3] A. G. Ivakhnenko, “The Group Method of Data Handling-A rival of the Method of Stochastic Approximation,” 1968.

[4] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. Cambridge, Massachusetts: The MIT Press, 2016.

[5] M. Raissi, P. Perdikaris, and G. E. Karniadakis, “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations,” *Journal of Computational Physics*, vol. 378, pp. 686–707, Feb. 2019, doi: 10.1016/j.jcp.2018.10.045.

[6] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural Networks*, vol. 2, no. 5, pp. 359–366, Jan. 1989, doi: 10.1016/0893-6080(89)90020-8.

[7] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, “Automatic differentiation in machine learning: a survey,” *arXiv:1502.05767 [cs, stat]*, Feb. 2015, Accessed: Jun. 09, 2019. [Online]. Available: http://arxiv.org/abs/1502.05767.

[8] I. E. Lagaris, A. Likas, and D. I. Fotiadis, “Artificial neural networks for solving ordinary and partial differential equations,” *IEEE Trans. Neural Netw.*, vol. 9, no. 5, pp. 987–1000, Sep. 1998, doi: 10.1109/72.712178.

[9] I. E. Lagaris, A. C. Likas, and D. G. Papageorgiou, “Neural-network methods for boundary value problems with irregular boundaries,” *IEEE Transactions on Neural Networks*, vol. 11, no. 5, pp. 1041–1049, Sep. 2000, doi: 10.1109/72.870037.

[10] S. He, K. Reif, and R. Unbehauen, “Multilayer neural networks for solving a class of partial differential equations,” *Neural Networks*, vol. 13, no. 3, pp. 385–396, Apr. 2000, doi: 10.1016/S0893-6080(00)00013-7.

[11] L. P. Aarts, “Neural Network Method for Solving Partial Differential Equations,” *Neural Processing Letters*, vol. 14, p. 11, 2001.

[12] H. Alli, A. Uçar, and Y. Demir, “The solutions of vibration control problems using artificial neural networks,” *Journal of the Franklin Institute*, vol. 340, no. 5, pp. 307–325, Aug. 2003, doi: 10.1016/S0016-0032(03)00036-X.

[13] A. Malek and R. Shekari Beidokhti, “Numerical solution for high order differential equations using a hybrid neural network—Optimization method,” *Applied Mathematics and Computation*, vol. 183, no. 1, pp. 260–271, Dec. 2006, doi: 10.1016/j.amc.2006.05.068.

[14] R. Shekari Beidokhti and A. Malek, “Solving initial-boundary value problems for systems of partial differential equations using neural networks and optimization techniques,” *Journal of the Franklin Institute*, vol. 346, no. 9, pp. 898–913, Nov. 2009, doi: 10.1016/j.jfranklin.2009.05.003.

[15] Y. Shirvany, M. Hayati, and R. Moradian, “Multilayer perceptron neural networks with novel unsupervised training method for numerical solution of the partial differential

equations," *Applied Soft Computing*, vol. 9, no. 1, pp. 20–29, Jan. 2009, doi: 10.1016/j.asoc.2008.02.003.

[16] S. Effati and M. Pakdaman, "Artificial neural network approach for solving fuzzy differential equations," *Information Sciences*, vol. 180, no. 8, pp. 1434–1457, Apr. 2010, doi: 10.1016/j.ins.2009.12.016.

[17] N. Krichen Masmoudi, C. Rekik, M. Djemel, and N. Derbel, "Two coupled neural-networks-based solution of the Hamilton–Jacobi–Bellman equation," *Applied Soft Computing*, vol. 11, no. 3, pp. 2946–2963, Apr. 2011, doi: 10.1016/j.asoc.2010.11.015.

[18] A. Reynaldi, S. Lukas, and H. Margaretha, "Backpropagation and Levenberg-Marquardt Algorithm for Training Finite Element Neural Network," in *2012 Sixth UKSim/AMSS European Symposium on Computer Modeling and Simulation*, Malta, Malta, Nov. 2012, pp. 89–94, doi: 10.1109/EMS.2012.56.

[19] S. Mall and S. Chakraverty, "Chebyshev Neural Network based model for solving Lane–Emden type equations," *Applied Mathematics and Computation*, vol. 247, pp. 100–114, Nov. 2014, doi: 10.1016/j.amc.2014.08.085.

[20] R. Lopez, E. Balsa-Canto, and E. Oñate, "Neural networks for variational problems in engineering," 2008.

[21] R. Raina, A. Madhavan, and A. Y. Ng, "Large-scale deep unsupervised learning using graphics processors," in *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*, Montreal, Quebec, Canada, 2009, pp. 1–8, doi: 10.1145/1553374.1553486.

[22] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations," *arXiv:1711.10561 [cs, math, stat]*, Nov. 2017, Accessed: Jun. 05, 2019. [Online]. Available: http://arxiv.org/abs/1711.10561.

[23] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations," *arXiv:1711.10566 [cs, math, stat]*, Nov. 2017, Accessed: Jun. 06, 2019. [Online]. Available: http://arxiv.org/abs/1711.10566.

[24] A. Karpatne, W. Watkins, J. Read, and V. Kumar, "Physics-guided Neural Networks (PGNN): An Application in Lake Temperature Modeling," *arXiv:1710.11431 [physics, stat]*, Oct. 2017, Accessed: Jun. 27, 2019. [Online]. Available: http://arxiv.org/abs/1710.11431.

[25] M. Raissi, A. Yazdani, and G. E. Karniadakis, "Hidden Fluid Mechanics: A Navier-Stokes Informed Deep Learning Framework for Assimilating Flow Visualization Data," *arXiv:1808.04327 [physics, stat]*, Aug. 2018, Accessed: Jul. 10, 2019. [Online]. Available: http://arxiv.org/abs/1808.04327.

[26] M. Raissi, "Forward-Backward Stochastic Neural Networks: Deep Learning of High-dimensional Partial Differential Equations," *arXiv:1804.07010 [cs, math, stat]*, Apr. 2018, Accessed: Jun. 27, 2019. [Online]. Available: http://arxiv.org/abs/1804.07010.

[27] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Multistep Neural Networks for Data-driven Discovery of Nonlinear Dynamical Systems," *arXiv:1801.01236 [nlin, physics:physics, stat]*, Jan. 2018, Accessed: Jun. 07, 2019. [Online]. Available: http://arxiv.org/abs/1801.01236.

[28] J. Sirignano and K. Spiliopoulos, “DGM: A deep learning algorithm for solving partial differential equations,” *Journal of Computational Physics*, vol. 375, pp. 1339–1364, Dec. 2018, doi: 10.1016/j.jcp.2018.08.029.
[29] C. Anitescu, E. Atroshchenko, N. Alajlan, and T. Rabczuk, “Artificial Neural Network Methods for the Solution of Second Order Boundary Value Problems,” *Computers, Materials & Continua*, vol. 59, no. 1, pp. 345–359, 2019, doi: 10.32604/cmc.2019.06641.
[30] J. Berg and K. Nyström, “A unified deep artificial neural network approach to partial differential equations in complex geometries,” *arXiv:1711.06464 [cs, stat]*, Nov. 2017, doi: 10.1016/j.neucom.2018.06.056.
[31] Y. Chen, L. Lu, G. E. Karniadakis, and L. D. Negro, “Physics-informed neural networks for inverse problems in nano-optics and metamaterials,” *arXiv:1912.01085 [physics]*, Dec. 2019, Accessed: Dec. 15, 2019. [Online]. Available: http://arxiv.org/abs/1912.01085.
[32] E. Kharazmi, Z. Zhang, and G. E. Karniadakis, “Variational Physics-Informed Neural Networks For Solving Partial Differential Equations,” *arXiv:1912.00873 [physics, stat]*, Nov. 2019, Accessed: Dec. 15, 2019. [Online]. Available: http://arxiv.org/abs/1912.00873.
[33] X. Meng and G. E. Karniadakis, “A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse PDE problems,” *Journal of Computational Physics*, vol. 401, p. 109020, Jan. 2020, doi: 10.1016/j.jcp.2019.109020.
[34] Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, and P. Perdikaris, “Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data,” *Journal of Computational Physics*, vol. 394, pp. 56–81, Oct. 2019, doi: 10.1016/j.jcp.2019.05.024.
[35] M. Lutter, C. Ritter, and J. Peters, “Deep Lagrangian Networks: Using Physics as Model Prior for Deep Learning,” *arXiv:1907.04490 [cs, eess, stat]*, Jul. 2019, Accessed: Nov. 16, 2019. [Online]. Available: http://arxiv.org/abs/1907.04490.
[36] R. S. S. Ermon, “Label-Free Supervision of Neural Networks with Physics and Domain Knowledge,” p. 7, 2017.
[37] E. Weinan and B. Yu, “The Deep Ritz method: A deep learning-based numerical algorithm for solving variational problems,” *arXiv:1710.00211 [cs, stat]*, Sep. 2017, Accessed: Aug. 21, 2019. [Online]. Available: http://arxiv.org/abs/1710.00211.
[38] R. Iten, T. Metger, H. Wilming, L. del Rio, and R. Renner, “Discovering physical concepts with neural networks,” *arXiv:1807.10300 [physics, physics:quant-ph]*, Sep. 2018, Accessed: Oct. 27, 2019. [Online]. Available: http://arxiv.org/abs/1807.10300.
[39] Z. Long, Y. Lu, X. Ma, and B. Dong, “PDE-Net: Learning PDEs from Data,” *arXiv:1710.09668 [cs, math, stat]*, Jan. 2018, Accessed: Nov. 15, 2019. [Online]. Available: http://arxiv.org/abs/1710.09668.
[40] K.-J. Bathe and K.-J. Bathe, *Finite element procedures*. Englewood Cliffs, N.J: Prentice Hall, 1996.
[41] G. R. Liu and Y. T. Gu, *An Introduction to Meshfree Methods and Their Programming*. Berlin/Heidelberg: Springer-Verlag, 2005.
[42] J. He, L. Li, J. Xu, and C. Zheng, “ReLU Deep Neural Networks and Linear Finite Elements,” *arXiv:1807.03973 [math]*, Jul. 2018, doi: 10.4208/jcm.1901-m2018-0160.
[43] S. A. Zargar and F.-G. Yuan, “A Deep Learning Approach for Impact Diagnosis,” *Structural Health Monitoring 2019*, vol. 0, no. 0, 2019, doi: 10.12783/shm2019/32458.

[44] S. Wang, S. A. Zargar, C. Xu, and F.-G. Yuan, “An Efficient Augmented Reality (AR) System for Enhanced Visual Inspection,” *Structural Health Monitoring 2019*, vol. 0, no. 0, 2019, doi: 10.12783/shm2019/32278.