SOLVING NON LINEAR PARTIAL DIFFERENTIAL EQUATIONS USING PHYSICS INFORMED DEEP NEURAL NETWORK

Amanuel Yohannes Suloro

A Thesis Submitted to
The Department of Applied Mathematics
School of Applied Natural Science
Presented in Partial Fulfillment of the Requirement for the Degree of Master's in Applied Mathematics (Differential Equations)

Office of Graduate Studies
Adama Science and Technology University

June, 2022
Adama, Ethiopia

SOLVING NON LINEAR PARTIAL DIFFERENTIAL EQUATIONS USING PHYSICS INFORMED DEEP NEURAL NETWORK

Amanuel Yohannes Suloro

Advisor: Tamirat Temesgen(Ph.D.)
Co-advisor: Feyissa Kebede(Ph.D.)

A Thesis Submitted to
The Department of Applied Mathematics
School of Applied Natural Science

Presented in Partial Fulfillment of the Requirement for the Degree of Master's in Applied Mathematics (Differential Equations)

Office of Graduate Studies
Adama Science and Technology University

June, 2022
Adama, Ethiopia

Declaration

I hereby declare that this Master Thesis entitled **"Solving Non Linear Partial Differential Equations Using Physics Informed Deep Neural Network"** is my original work. That is, it has not been submitted for the award of any academic degree, diploma or certificate in any other university. All sources of materials that are used for this thesis have been duly acknowledged through citation.

Name of Student	Signature	Date

Recommendation of Advisors

we, the advisors of this thesis, hereby certify that we have read the revised version of the thesis entitled **"Solving Non Linear Partial Differential Equations Using Physics Informed Deep Neural Network"** prepared under our guidance by Amanuel Yohannes Suloro submitted in partial fulfillment of the requirements for the degree of Mater's of Science in Applied Mathematics. Therefore, we recommend the submission of revised version of the thesis to the department following the applicable procedures.

Major Advisor	Signature	Date
Co-advisor	Signature	Date

Approval Page

we, the advisors of the thesis entitled **"Solving Non Linear Partial Differential Equations Using Physics Informed Deep Neural Network"** and developed by Amanuel Yohannes Suloro, hereby certify that the recommendation and suggestions made by the board of examiners are appropriately incorporated into the final version of the thesis.

Major Advisor	Signature	Date
Co-advisor	Signature	Date

We, the undersigned, members of the Board of Examiners of the thesis by Amanuel Yohannes Suloro have read and evaluated the thesis entitled **"Solving Non Linear Partial Differential Equations Using Physics Informed Deep Neural Network"** and examined the candidate during open defense. This is, therefore, to certify that the thesis is accepted for partial fulfillment of the requirement of the degree of Master of Science in Applied Mathematics (Differential Equations).

Chairperson	Signature	Date
Internal Examiner Wubshet Ibrahim(Prof.)-	Signature	Date June 17, 2022 Date
External Examiner	Signature	Date

Final approval and acceptance of the thesis is contingent upon submission of its final copy to

the Office of Postgraduate Studies (OPGS) through the Department Graduate Council (DGC) and School Graduate Committee (SGC).

Department Head	Signature	Date
School Dean	Signature	Date
Office of Postgraduate Studies, Dean	Signature	Date

ACKNOWLEDGMENT

First, I would like to express my deepest gratitude to almighty **God** for giving me tolerance; I am also grateful to **my advisor Dr.Tamirat Temesegen** and my **co-advisor Dr.Feyisaa Kebede** for their helpful discussion, comments, giving direction, and providing the necessary material in the preparation of thesis .

Secondly, I would like to extend my thanks to my families and all who encouraged me to complete my project and their heart full help while writing the thesis . Lastly, I would like thanks to Department of Mathematics, Adama Science and Technology University and Oda Bultum University for giving the necessary materials throughout the preparation of this thesis.

Tabel of Contents

List of Tables

List of Figures

List of Acronyms

ANN :Artificial Neural Networks

NN :Neural Networks

PIDNNs :Physics Informed Deep Neural Networks

MLP :Multi-layer perceptron

MSE :Mean Squared Error

PDEs :Partial Differential Equations

Abstract

In this study, we studied on solving non linear partial differential equations using the proposed physics informed deep neural network (PIDNN). Physics-informed deep neural networks (PIDNNs) use automatic differentiation to solve non linear partial differential equations (PDEs) by penalizing the PDEs in the loss function at random set of points in the domain of boundary and initial conditions. In this study, we have used the supervised learning and multi-layer architecture to obtain approximate solutions of non linear partial differential equations. Furthermore, we compared the PIDNN results and the exact. Two implementations (schrödinger equation and Burgers equation) took to show the accuracy and the power of this developed model with plotted graphs using python code.

Keywords: Artificial Neural Networks, Automatic Differentiation, Feed Forward Neural Network, Deep Neural Network, Partial Differential Equation, Machine Learning.

CHAPTER 1

INTRODUCTION

1.1 Background of the Study

Differential equations play a vital role in various fields of science and engineering. Many real-world problems in engineering, mathematics, physics, chemistry, etc., may be modeled by ordinary or partial differential equations. In most cases, an analytical/exact solution of differential equations may not be obtained easily. Therefore, various types of numerical techniques have been developed by researchers to solve such equations, such as Euler, Runge–Kutta, etc. Although these methods provide good approximations to the solution, they require the discretization of the domain into several finite points/elements. These methods generally provide solution values at predefined points and the computational complexity increases with the number of sampling points, which is not a straightforward task. (Wambecq, 1978).
Due to their frequent presence in numerous applications in the physical, biological, and social sciences, nonlinear partial differential equations (PDEs) have been the subject of many studies. Because most nonlinear PDEs lack analytical solutions except in the most basic cases, several numerical methods have been developed over the last several decades, including finite-difference, finite-volume, and finite-element methods, and lattice Boltzmann methods, to mention a few. In addition to traditional numerical approaches, the recent advent of deep learning has provided a chance to construct surrogate models for solving nonlinear PDEs. The physics-informed neural network (PIDNN) described by (Raissi et al., 2019). has gotten a lot of attention because of its success in handling both forward and inverse nonlinear PDE problems, as well as its ease of implementation. The fundamental physical principles are encoded into the loss function of the neural network so that the governing equations are enforced by minimizing the residual loss function with automatic differentiation. The PIDNN and its variants have lately been used to solve a variety of PDEs, including the Navier-Stokes equation and the Kdv equations.

Traditional physics-based numerical methods for solving partial differential equations (PDEs) have had a lot of success in tackling issues in science and engineering. For difficult prob-

lems such as nonlinear PDEs, these methods are accurate but computationally expensive, necessitating problem-specific techniques. Data-driven methods have gotten a lot of attention in practically every field of research and engineering in the previous decade. PDE data driven approaches can aid in the identification of highly non-linear mappings (between inputs and outputs) that can be used to replace or supplement expensive physics-based simulations. Machine learning-based models can be utilized as PDE solutions in circumstances when a large number of simulations are required, such as the inverse problem and homogenization, due to their adaptability and quick assessment capabilities (Rosenblatt, 1958).

Recent advances in machine learning, combined with new data recording and sensor technologies, have the potential to revolutionize our understanding of the physical world in modern application areas like neuron science, epidemiology, finance, and dynamic network analysis, where first-principles derivations may be difficult. (Gross & Seibert, 1993).

Because of its superior learning capacity, various machine intelligence techniques, particularly constructionist learning or Artificial Neural Network (ANN) methods, have emerged as a potent methodology for solving a variety of real-world issues in recent decades (Burney et al., 2005). Robert Hecht-Nielsen, the developer of one of the first neurocomputers, provides the most basic explanation of ANN "A neural network is a computing system made up of a number of simple, highly interconnected processing components that process information based on their dynamic state responses to external inputs," he said (Mukhopadhyay, 2018).

ANNs are processing devices (algorithms) based on the neuronal organization of the mammalian cerebral cortex, but on a much smaller scale (McCulloch & Pitts, 1943). The human brain has long influenced computer scientists. In their study, they define a neuron as a single cell that accepts inputs, analyses those inputs, and generates an output in a network of cells. The biological nervous system inspired ANN, which is a computational model or information processing paradigm.

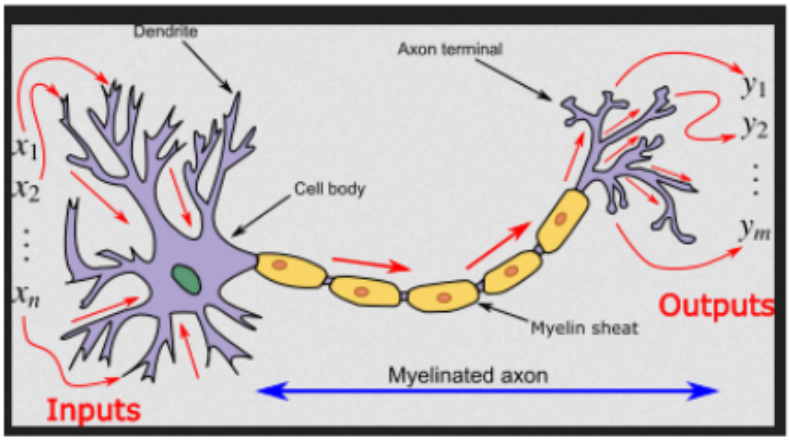


Figure 1.1: Biological Neuron
(Chakraverty & Mall, 2017)

Figure 1.1 shows the network works in the same way as a biological human brain.

The architecture is divided into three layers: input, hidden, and output. There are neurons or units in each layer when a structure contains multiple hidden layers, it is referred to as a deep neural network (DNN) (Schmidhuber, 2015).

1.2 Neural Network

A neural network is a parallel information processing system that shares some of the same properties as specific brain processes. It is made up of neurons and synaptic weights, and it learns to execute complex computations.

Neural networks can be used in the following areas:

1. Modeling complex relationships between inputs and outputs ,
2. Recognizing images including handwriting,
3. Enhancing grainy images,
4. Translating from one language to another.

To tackle specific problems, ANN processes input in parallel through neurons-nodes. It gains knowledge through learning, and this knowledge is stored in the strength of inter-neuron connections, which is expressed numerically as "weights." The output signal values for a fresh testing input signal value are computed using these weights. The "input layer" presents patterns to the network, which communicates with one or more "hidden layers," where the actual processing is done using a system of weighted "connections." As shown in Figure 1.2, the hidden layers then link to a "output layer," which outputs the answer.

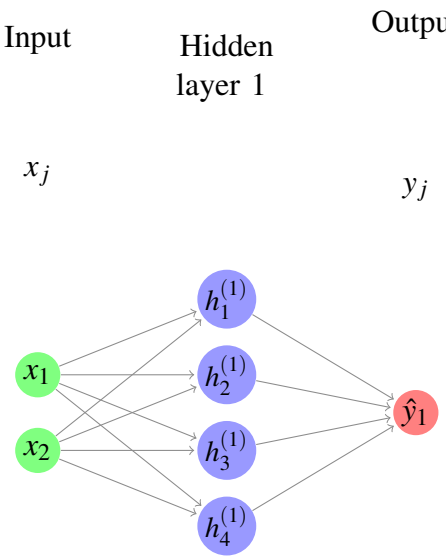


Figure 1.2: Structure of artificial neural network (Greenwade, 1993)

Here, x_j are input nodes, w_{ij} are weighting from the input layer to the hidden layer, and v_i and y_j denote the weights from the hidden layer to the output layer and the output node, respectively. Single-layer ANNs, Multilayer feedforward ANNs, Temporal ANNs, and other forms of ANNs exist.

1.3 The basic components of ANN

Neuron: It is the basic unit of a neural network.

Input Layer: This is the first layer in the neural network. It takes input signals(values) and passes them on to the next layer. It doesn't apply any operations on the input signals(values) and has no weights and biased values associated.

Hidden Layer: This is the medium layer in the neural network. Hidden layers have neurons(nodes) that apply different transformations to the input data.

Output Layer: This layer is the last in the network and receives input from the last hidden layer. With this layer, we can get the desired number of values and in the desired range.

Weights(Parameters): Weight is a parameter of ANN. It represents the strength of the connection between nodes. Initially, it begins randomly. As training continues, it adjusted toward the desired values and the correct output. A weight brings down the importance of the input value. Weights near zero means changing this input will not change the output. Positive weights mean increasing this input will increase the output. Negative weights mean increasing this input will decrease the output. A weight decides how much influence the input will have on the output.

Bias: It is an extra input neuron/node. It is used for shifting the activation function to- wards left or right, it can be referred to as a y-intercept in the line equation. A low bias means the network is making more assumptions about the form of the output, whereas a high bias means the network is making fewer assumptions about the output.

1.4 Deep Neural Networks

1.4.1 A Simple Network

The below diagram shows a simple neural network, with two inputs and one output.As it is well known, in machine learning algorithms in general, x_1,x_2 represent features for a given sample x.The $\hat{y}$ variable represents the network output approximating the target value y given a set of training data $(x_1^{(1)},x_2^{(1)}),(x_1^{(2)},x_2^{(2)}),....(x_1^{(m)},x_2^{(m)}))$ and corresponding target values $y^{(1)},y^{(2)},.....,y^{(m)}$. The algorithm search for the optimal weights (w_1,w_2) and bias (b) such that the network out put is as close as the target value. For instance, as it is indicated in the diagram, for the sample (x_1,x_2) the process goes as follows:

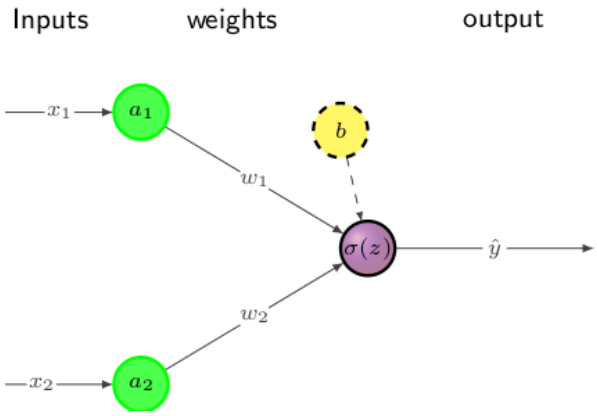


Figure 1.3: A simple neural network
(Chakraverty & Mall, 2017)

$$a_1 = x_1, a_2 = x_2$$
$$z = w_1 a_1 + w_2 a_2 + b,$$
$$y = \sigma(z)$$

One can rewrite in matrix form as;

$$z = WX + b,$$

where, $W = [w_1, w_2]$, and $X = [x_1, x_2]$. The function $\sigma(z)$ is called an activation function or transfer function. The purpose of activation function is to introduce non-linearity into the network. Some of the activation functions are; sigmoid, tan hyperbolic and Rectifiable Linear Unit. See Figure 1.4. These function are;

$$\text{Sigmoind} : \sigma(z) = \frac{1}{1+e^{-z}}$$
$$\text{Tanh} : \sigma(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$$
$$\text{ReLU} : \sigma(z) = max(z, 0)$$

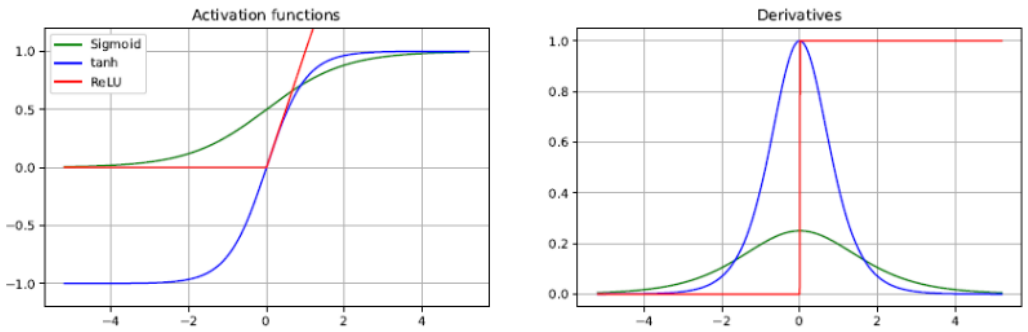


Figure 1.4: Common activation functions and their derivatives .

$$z = \sum_{j=1}^{m} x_j w_j + b = WX + b,$$
$$\hat{y} = \sigma(z)$$

To find the optimal parameters, i.e., the weights and the biases, one can apply the objective function given by the L_2 norm.

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} |\hat{y}^{(i)} - y^{(i)}|^2 = ||\hat{y} - y||^2$$

1.5 Multi Output Network, Example: Two output

The same discussion can be made with slight modification; the following network is for training data with m features and two outputs $\hat{y}_1$ and $\hat{y}_2$. which can be seen form the following Figure 1.5, the networks are densely connected with corresponding learning weights and biases.

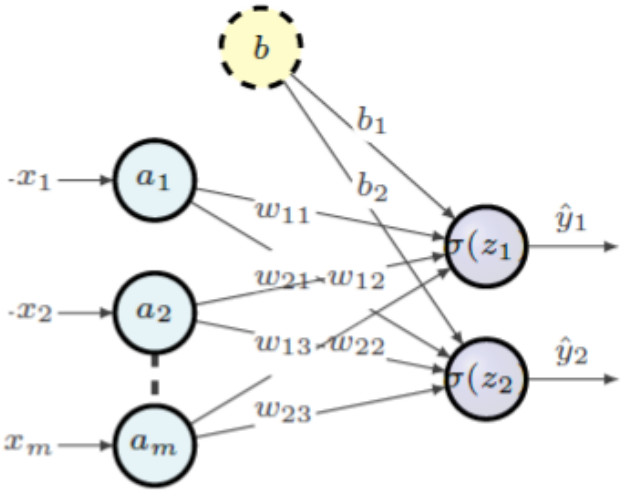


Figure 1.5: A multi-output neural network .

Similar to the case of one output network, for each training sample $x = [x_1, x_2, ..., x_m]$, the network outputs $\hat{y}_1$ and $\hat{y}_2$ are obtained as follow;

$$z_1 = \sum_{j=1}^{m} x_j w_{j,1} + b_1$$
$$z_2 = \sum_{j=1}^{m} x_j w_{j,2} + b_2,$$
$$\hat{y}_1 = \sigma(z_1),\ \hat{y}_2 = \sigma(z_2)$$

For the multiple network output say $[\hat{y}_1, \hat{y}_2, ..., \hat{y}_i, ..., \hat{y}_n]$, we have;

$$z_i = \sum_{j=1}^{m} x_j w_{j,i} + b_i,$$
$$\hat{y}_i = \sigma(z_i).$$

1.6 Single Layer Neural Network

Adding one more layer to a dataset with non-linear relationships will considerably increase the model's performance. A single layer (one hidden layer) neural network is depicted in the following schematic design. The following feed forward propagation gives the network output $\hat{y}$ for this particular model. From input to hidden layer, given a sample or training data X;

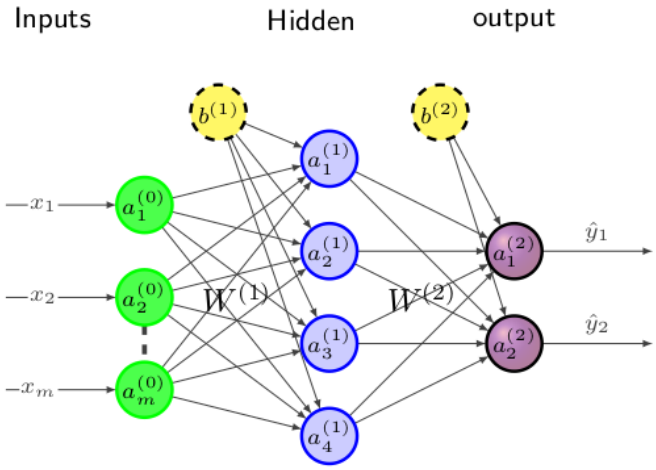


Figure 1.6: Single Layer Neural Network.

$$
\begin{aligned}
X &= a^{(0)} \\
Z^{(1)} &= W^{(1)}a^{(0)} + b^{(1)} \\
a^{(1)} &= \delta^{(1)}(z^{(1)}).
\end{aligned}
$$

From hidden layer to output layer;

$$
\begin{aligned}
Z^{(2)} &= W^{(2)}a^{(1)} + b^{(2)} \\
a^{(2)} &= \delta^{(2)}(Z^{(2)}) \\
\hat{y} &= a^{(2)}
\end{aligned}
$$

Rewriting the above we get a sequence of composition of functions;

$$\hat{y} = \delta^{(2)}(w^{(2)})\delta^{(1)}(w^{(1)}a^{(0)} + b^{(1)} + b^{(2)}). \tag{1.1}$$

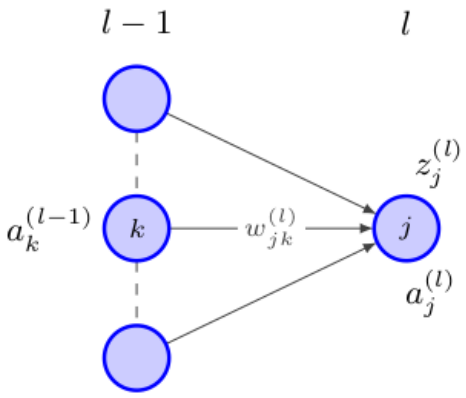
Based on the problem at hand, one can add more layer called hidden layers for better performance. A neural network with more than one hidden layer is called deep neural network.

1.7 Propagation or Delta Learning Rule

What will happen between layers? Linear combination of the values

$$
\begin{aligned}
Z_j^{(l)} &= \sum_k W_{jk}^{(l)} a_k^{(l-1)} + b_k^{(l)} \\
a_j^{(l)} &= \delta(Z_j^{(l)})
\end{aligned}
\tag{1.2}
$$

Denotes two consecutive layers by $l-1$ and l. And label nodes on layer $l-1$ by k and nodes on layer l by j. So, what value goes into the j^{th} node of layer l?



$$Z_j^{(l)} = \sum_{1}^{n_h^{(l-1)}} W_{jk}^{(l)} a_k^{(l-1)} + b_k^{(l)}, \tag{1.3}$$

where n_h denotes the number of nodes in layer l.

$$Z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)} \tag{1.4}$$

with the matrix $W^{(l)}$ containing all the multiplicative parameters, i.e. the weights $w_{(jk)}^{(l)}$, and $b^{(l)}$ is the bias.

1.8 Automatic Differentiation

The ANN model needs the human construction of functions for cost function minimization. Manual differentiation is time consuming and mistake prone. An efficient numerical technique might be considered, however it would still be prone to high inaccuracy due to rounding and truncation errors. Using symbolic differentiation is one method of addressing these issues. However, "expression swell" frequently results in elaborate and cryptic expressions. Automatic differentiation is the fourth powerful technique (AD). There is no numerical or symbolic difference in AD. It refers to a method for automatically creating a procedure for computing derivatives of a value from a program that computes that value. Backpropagation is a type of autonomous differentiation that is used to improve the performance of a system.

1.8.1 The Benefit of PIDNN over other Numerical Methods

In several cases, PIDNNs have been found to outperform traditional numerical approaches. Finite difference, finite element method(FEM), finite volume, and spectrum approaches are

examples of numerical techniques for approximating PDEs. Unlike many traditional computing methods, the computational cost of PIDNNs does not increase as the number of grid points increases. Furthermore, without retraining, the trained PDINN network can predict values on simulated grids of varied resolutions.

In FEM, a piece-wise polynomial with unknown point values approximates the solution, but in PINNs, the surrogate model is a neural network with weights and biases. Furthermore, mesh creation is normally required for FEM, whereas PDINN can use a grid or random points because it is mesh-free.

1.9 Artificial Neural Networks as Universal Approximators

A single-layer feed-forward neural network with inputs $x \in R^m$, outputs $y \in R^n$, and d hidden units is constructed as:

$$y = W^1 \delta(W^0 x + b^0) + b^1 \tag{1.5}$$

where $(W^0 \in R^{dm}, b^0 \in R^d)$, $(W^1 \in R^{nd}, b^1 \in R^n)$ are parameters of this transformation, also known as weights and biases, and δ is the activation function. As shown in this transformation can approximate any measurable function, independently of the size of input features m or the activation function δ. If we define the transformation $\sum$ as $\sum_i(\hat{x}^i) = \hat{y}^i = \delta_i(W^i\hat{x}^i + b^i)$ with $\hat{x}^i$ as the input and $\hat{y}^i$ as the output of any hidden layer $i, x = \hat{x}^0$ as the main input to the network, and $y = \sum_L(\hat{x}^L)$ as the final output of the network, we can construct a general L-layer neural network as composition of $\sum_i$ functions as:

$$y = \Sigma^L \bigcirc \Sigma^{L-1}, \bigcirc \Sigma^0(x)$$

with δ_i as activation functions that make the transformations nonlinear. Inaccurate approximation may arise due to lack of a deterministic relation between input and outputs, insufficient number of hidden units, inadequate training, or poor choice of the optimization algorithm. The parameters of the neural network, W^i and b^i of all layers $i = 0 \ldots L$, are identified through minimization using a back-propagation algorithm.

For instance, if we approximate a field variable such as temperature T with a multi-layer neural network as $T(x) \approx \hat{T}(x) = N_T(x; W, b)$, we can set up the optimization problem as

$$argminL(w,b) = ||T(x^* - \hat{T}(x^*))|| = ||T(x^*) - N_T(x^*; W, b)|| \tag{1.6}$$

where x^* is the set of discrete training points, and $||\bigcirc||_p$ is the mean squared norm. Note that one can use other choices for the loss function L, such as mean absolute error or cross entropy. The optimization problem (1.6) is non convex, which may require significant trial and error efforts to find an effective optimization algorithm and optimization parameters. We can construct deep neural networks with an arbitrary number of layers and neurons. We can

also define multiple networks and combine them to generate the final output. There are many types of neural networks that have been optimized for specific tasks. An example is the ResNet architecture introduced for image classification, consisting of many blocks each of the form:

$$z^k = \sum^{k2} \bigcirc \sum^{k1} \bigcirc \sum^{k0} (z^{k-1}) + z^{k-1} \tag{1.7}$$

where k is the block number and z^{k-1} is the output of previous block, with $x = z^0$ and $y = z^K$ as the main inputs and outputs of the network. Therefore, artificial neural networks offer a simple way of constructing very complex but dependent solution spaces (see, e.g., Figure 1.8).

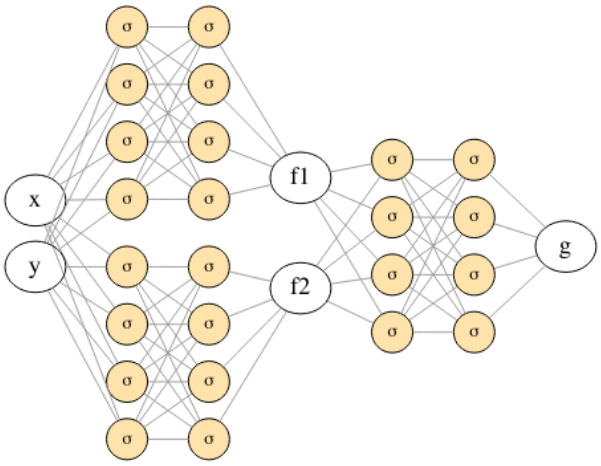


Figure 1.7: A sample multi-net architecture to construct a complex functional space g as $g(x,y) = g(f_1(x,y), f_2(x,y))$.

.

1.10 Statement of the Problem

Consider paramatrized and nonlinear partial differential equations of the general form

$$u_t + N[u;\lambda] = 0,\ x \in \Omega,\ t \in [0,T] \tag{1.8}$$

where $u(t,x)$ denotes the latent (hidden) solution, $\mathcal{N}[u]$ is a nonlinear operator parametrized by λ, and Ω is a subset of $\mathbb{R}^D$. We define $f(t,x)$ to be given by the equation;

$$f = u_t + N[u;\lambda] = 0$$

and proceed by approximating $u(t,x)$ by a deep neural network. This study is intended to answer the following basic questions;

- Can we write an efficient algorithm for the proposed method?

- Is the physics informed deep neural network method efficient to solve non linear partial differential equation?
- Is the physics informed deep neural network is efficient when we compare with the other method?

1.11 Objectives of the study

1.11.1 General objective

The general objective of this study is to solve a non linear partial differential equation using Physics Informed Deep Neural Network (PIDNN).

1.11.2 Specific objectives of the study

The specific objectives of this study are to

1. investigate the PIDNN approach to solve non linear partial differential equations,
2. develop PIDNN algorithm for solving non linear partial differential equations,
3. compare PIDNN results and the numerical result.

1.12 Significance of the study

The result of this study will help to:

- increasing the understanding of PIDNN to solve non linear partial differential equations for scientific community,
- develop the algorithm of Physics Informed deep Neural Network(PIDNN),
- know an algorithm of a new method (i.e. phyiscs informed deep neural network).
- used as a reference material for scholars who works on this area.

CHAPTER 2

LITERATURE REVIEW

Because of its high learning capacity, the ANN approach has become a strong methodology for solving a wide range of real-world issues. This approach has been used to solve ordinary and partial differential equations, among other things (Schalkoff, 1997; Adomaitienė et al., 2008). The architecture of an artificial neural network defines how its several neurons are arranged, or placed, in relation to each other. These arrangements are structured essentially by directing the synaptic connections of the neurons (Da Silva et al., 2017). (Chiaramonte, Kiener, et al., 2013) formulate the non-linear second-order DE to ANN and approximate using ANN method .

In (Yadav et al., 2015) book brief introduction to neural network has been given along with some basic terminologies. They explained the mathematical model of neural network in terms of activation functions. Different architectures of neural network like feed forward, feed backward, radial basis function network, multi layer perceptron neural network and cellular network etc., are described. Backpropagation and other training algorithms have been also discussed.

Deep learning has achieved remarkable success in diverse applications; however, its use in-solving partial differential equations (PDEs) have emerged only recently. Here, they proposed overview of physics-informed neural networks (PINNs), which embed a PDE into the loss of the neural network using automatic differentiation. The PINN algorithm is simple, and it can be applied to different types of PDEs, including integro-differential equations,fractional PDEs, and stochastic PDEs. Moreover, from an implementation point of view, PINNs solve inverse problems as easily as forward problems. They proposed a new residual-based adaptive refinement (RAR) method to improve the training efficiency of PINNs. For pedagogical reasons, they compared the PINN algorithm to a standard finite element method(Chiaramonte et al., 2013).

The differential equation's trial solution is expressed as a two-part sum. The initial/boundary conditions are met in the first section, which has no modifiable parameters. The second section is built to avoid the initial/boundary conditions. A feed-forward neural network with changeable

parameters is used in this part (the weights). The initial/boundary conditions are thus satisfied by construction, and the network is taught to satisfy the differential equation (Lagaris et al., 1998).

Neural networks are a subset of the larger area of neural computing, often known as soft computing. They (Giorelli et al., 2013) proposed a new solution strategy based on unique mathematical tools and neural-like computation systems for the approximated solution of high order ordinary differential equations. Without employing pre-assigned discretisation points, this hybrid method can result in improved numerical methods for solving initial/boundary value problems. The solution of lower and higher order differential equations yields excellent test results. For the close enough neighborhood of initial/boundary points, the model discovers approximation solutions for the differential equation both inside and outside the domain of consideration. To exemplify the method, numerical examples are given.

(Wu et al., 2021) They developed a modified physics informed neural network method based on the conservation law constraint after investigating conservation laws as one of the major integrable characteristics of nonlinear physical models. Incorporating the conservation law into the mean square error of the loss function to train the neural network imposes physical limitations on the solution of nonlinear physical models from a global perspective. They analyzed the typical nonlinear Schrodinger problem and predicted various data-driven optical solutions using this strategy.

Neural networks based on physics using two unique types of algorithms, continuous time and discrete time models, neural networks are trained to solve supervised learning tasks while respecting any given laws of physics expressed by general nonlinear partial differential equations. (Raissi et al., 2019).

(Monterola & Saloma, 2001) With an unsupervised NN, he established a new method for solving the non linear Schrodinger problem. The trained NN has learned to solve the non linear Schrodinger partial differential equation solution at any point (z,t) in the solution space from a finite training set and without cumbersome coordinate transformations.

(Hagge et al., 2017) They were developed to solve a system of ordinary differential equations involving a sink (reaction rate) term with an unknown functional shape. They hypothesized that state variable measurements (time series) are only partially available, and that the reaction rate can be "learned" from data using a recurrent neural network. A recurrent neural network training problem using discretized ordinary differential equations is used to achieve this.

(Aarts & Van der Veer, 2001) They showed how to solve nonlinear differential equations and their boundary conditions using a neural network method in their study. The idea behind their method is to use neural networks and training sets to incorporate knowledge about the differential equation and its boundary conditions. They were able to obtain particularly structured neural networks this way. They must simultaneously train all acquired neural networks in order to solve the nonlinear differential equation and its boundary conditions.

A single-layer Hermite neural network (HeNN) model is utilized, with a hidden layer substituted by a Hermite orthogonal polynomial expansion block of the input pattern. The network parameters are modified and the computed error function is minimized using a feed-forward neural network model with the unsupervised error back propagation principle (Wambecq, 1978).

In FLANN, the hidden layer is replaced by a functional expansion block that uses orthogonal polynomials like Chebyshev, Legendre, and Hermite to improve the input patterns. The construction of single-layer FLANN models is explained, as well as the formulations and gradient computations. The overall number of network parameters in FLANN is lower than in a multilayer perceptron (MLP) structure. As a result, the approach is more computationally efficient than MLP(Chakraverty & Mall, 2017).

They proposed the Optimizing a Discrete Loss (ODIL) paradigm for employing machine learning technologies to solve partial differential equations numerically. Numerical approaches are formulated as the minimization of discrete residuals that are solved using gradient descent and Newton's methods in the framework. They demonstrated the utility of this method on equations with missing parameters or in the absence of enough data to generate a well-posed initial-value issue. The framework inherits the accuracy, convergence, and conservation qualities of mesh-based discretizations of PDEs. It keeps the sparsity of the answers and can be used to solve inverse and ill-posed problems. It is used in PDE-constrained optimization, optical flow, system identification, and data assimilation with the use of gradient descent techniques, such as those often used (Mukhopadhyay, 2018).

Physics-informed attention-based neural networks (PIANNs), is a combination of recurrent neural networks and attention mechanisms. The attention mechanism adapts the behavior of the deep neural network to the non-linear features of the solution, and break the current limitations of PINNs. They found that PIANNs effectively capture the shock front in a hyperbolic model problem, and are capable of providing high-quality solutions inside the convex hull of the training set(Gross & Seibert, 1993).

A novel hybrid method for the solution of ordinary and partial differential equations is presented here. The method creates trial solutions in neural network form using a scheme based on grammatical evolution. The trial solutions are enhanced periodically using a local optimization procedure. The proposed method is tested on a series of ordinary differential equations, systems of ordinary differential equations as well as on partial differential equations with Dirichlet boundary conditions and the results are reported(Burney et al., 2005).

Deep feed forward artificial neural networks were used to approximate partial differential equation solutions in complex geometries. They demonstrated how to weak the backpropagation algorithm to compute the partial derivatives of the network output in terms of the space variables, which is required to approximate the differential operator. The method is based on a solution approach that just requires feedforward neural networks and an unconstrained gradient-based optimization method like gradient descent or the quasi-Newton method. They demonstrated an application where traditional mesh-based approaches are ineffective and neural networks are a viable option. Finally, they discussed the advantages of deep neural networks over shallow neural networks, as well as various other convergence-enhancing strategies (Berg & Nyström, 2018).

CHAPTER 3

RESEARCH METHODOLOGY

3.1 Physics Informed Deep Neural Network (PIDNNs)

PIDNNs (Physics-Informed Deep Neural Networks) are a type of scientific machine learning technique for solving issues involving partial differential equations (PDEs). PIDNNs approximate PDE solutions by training a neural network to minimize a loss function that includes initial and boundary conditions along the space-time domain's boundary, as well as the PDE residual at selected points in the domain (called collocation point).

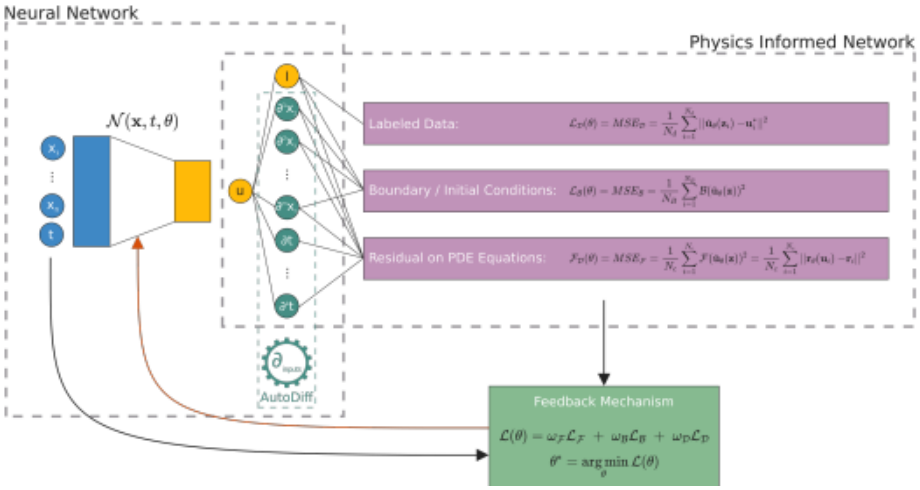


Figure 3.1: Physics-informed deep neural networks

.

Unlike typical PDE solvers, the PIDNN does not need to discretize the PDE or use costly numerical procedures to solve the equations, despite the fact that the PDE is stored in the neural network. Instead, PIDNNs use backward propagation's automatic differentiation to represent all differential operators in a PDE, as well as a neural network's training to execute a nonlinear mapping from input variables to output quantities by minimizing a loss function(Freeman & Skapura, 1991).

The PIDNN model is a grid-free approach in this regard, because no mesh is required to solve the equations. The neural network's optimization or training stage transfers all of the complexity

of solving a physical problem. As a result, a PIDNN can bring together disparate data sets. PINNs are unique in that they include a residual network that encodes the governing physics equations. The essential idea underlying PINN training is that it is an unsupervised technique that does not require labelled data, such as previous simulation or experiment findings. The PINN algorithm is a mesh-free method for finding PDE solutions by transforming the challenge of solving the governing equations directly into a loss function optimization problem.
A physics informed neural network (PINN) includes a system's physics by solving the boundary value issue using the loss function of a neural network. The PINN method has had a lot of success in estimating the map between a partial differential equation (PDE) solution and its spatio-temporal input.

3.1.1 The building blocks of a PIDNN

Physically-informed neural networks can solve problems with limited data, such as noisy experiment results. PIDNNs are neural networks that deal with supervised learning problems because they can use known data while adhering to any given physical law specified by general nonlinear partial differential equations(Rosenblatt, 1958). PINNs can solve differential equations in their most basic form, such as:

$$\begin{aligned} F(u(z);\alpha) &= f(z) \quad z \in \Omega, \\ B(u(z)) &= g(z) \quad ,z \in \partial\Omega \end{aligned} \tag{3.1}$$

defined on the domain $\Omega \subset R^d$ with the boundary $\partial\Omega$. Where $z := [x_1,...,x_n;t]$ indicates the space-time coordinate vector, u represents the unknown solution, α are the parameters related to the physics, f is the function identifying the data of the problem and F is the non linear differential operator. Finally, since the initial condition can actually be considered as a type of Dirichlet boundary condition on the spatio temporal domain, it is possible to denote B as the operator indicating arbitrary initial or boundary conditions related with the problem and g the boundary function; where the boundary conditions can be, for example, Dirichlet, Neumann, Robin, or periodic boundary conditions. With the formalization of equation (3.1) it is possible to describe a numerous physical system including both forward and inverse problems. The goal of forward problems is to find the function u for every z, where α are specified parameters. In the case of the inverse problem, α must also be determined from the data. The reader can find an operator based mathematical formulation of equation (3.1). In the PINN methodology, $u(z)$ is computationally predicted by a NN, parametrized by a set of parameters θ, giving rise to an approximation

$$\hat{u}_\theta(z) \approx u(z)$$

where $\hat{u}_\theta(z)$ denotes a NN approximation realized with θ. In such a context, the NN must learn to approximate the differential equations through finding θ that define the NN by minimizing

a loss function that depends on the differential equation L_F, the boundary conditions L_B , and eventually some known data L_{data} , each of them adequately weighted:

$$\theta^* = argmin(L_F(\theta) + L_B(\theta) + L_{data}(\theta)) \tag{3.2}$$

To summarize, PINN can be called unsupervised learning techniques when they are taught entirely utilizing physical equations and boundary conditions for forward problems; however, PINN can be regarded supervised learning methodologies when some physical attributes are determined from noisy data.

All of the PINN's building blocks mentioned in this part are summarized in Figure 3.1.

A neural network, a physics-informed network, and a feedback mechanism are the three components of a PINN. The first block is a NN, $\hat{u}_\theta$ that accepts vector variables z from the equation (3.1) and outputs the filed value u. The second block can be thought of PINN's functional component, as it computes the derivative to determine the losses of equation terms, as well as the terms of the initial and boundary conditions of equation (3.2). Generally, the first two blocks are linked using algorithmic differentiation, which is used to inject physical equations into the NN during the training phase. Thus, the feedback mechanism minimizes the loss according to some learning rate,in order to fix the NN parameters vector θ of the NN $\hat{u}_\theta$. In the following, it will be clear from the context to what network we are referring to, whether the NN or the functional network that derives the physical information.

3.2 Network architecture

It is a technique that seeks to build an intelligent program using models that simulate the working of neurons in the human brain. The key element of the network is the structure of the information processing system. ANN processes information in a similar way as the human brain does. The network is composed of a large number of highly interconnected processing elements (neurons) working in parallel to solve a specific problem. Neural computing is a mathematical model inspired by the biological model. This computing system is made up of a large number of artificial neurons and a still larger number of interconnections among them. According to the structure of these interconnections, different classes of neural network architecture can be identified, as discussed next.

The ANN consists of $L+1$ layers where layer 0 is the input layer and layer L is the output layer. The layers $0 < l < L$ are the hidden layers. The activation functions in the hidden layers can be any activation function such as for example sigmoids, rectified linear units, or hyperbolic tangents. Unless otherwise stated we will use sigmoids in the hidden layers. The output activation will be the linear activation. The ANN defines a mapping $R^N \rightarrow R^M$. Each neuron in the ANN is supplied with a bias, including the output neurons but excluding the input neurons, and the connections between neurons in subsequent layers are represented by matrices of weights. We

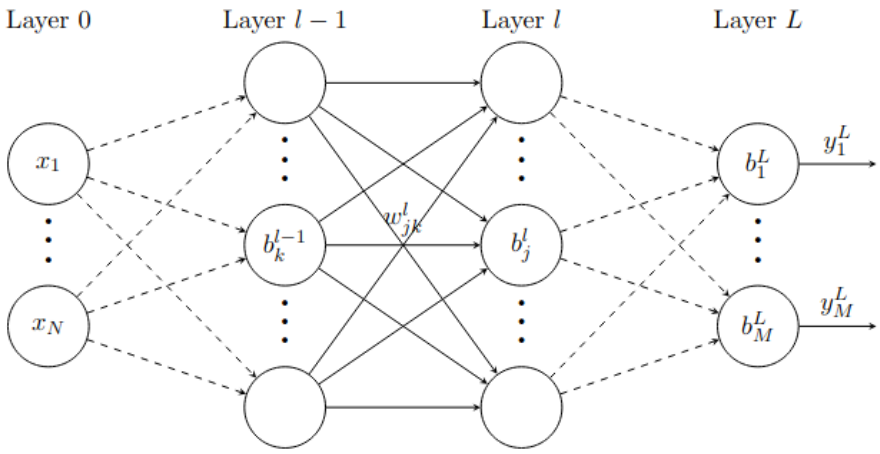


Figure 3.2: Schematic representation of a fully connected feedforward ANN.

let b_j^l denotes the bias of neuron j in layer l. The weight between neuron k in layer $l-1$ and neuron j in layer l is denoted by w_{jk}^l. The activation function in layer l will be denoted by δ_l regardless of the type of activation. We assume for simplicity that a single activation function is used for each layer. The output from neuron j in layer l will be denoted by y_j^l. See Figure 3.2 for a schematic representation of a fully connected feed forward ANN. A quantity that will be used extensively is the so-called weighted input which is defined

$$Z_j^i = \sum_k w_{jk}^l \delta_{l-1}(z_k^{l-1}) + b_j^l \tag{3.3}$$

where the sum is taken over all inputs to neuron j in layer l. That is, the number of neurons in layer $l-1$. The weighted input (3.2) can of course also be written in terms of the output from the previous layer as

$$Z_j^i = \sum_k w_{jk}^l y_k^{l-1} + b_j^l \tag{3.4}$$

where the output $y_k^{l-1} = \delta_{l-1}(z_k^{l-1})$ is the activation of the weighted input. As we will be working with deep ANNs we will prefer formula (3.2) as it naturally defines a recursion in terms of previous weighted inputs through the ANN. By definition we have

$$\delta_0(z_j^0) = y_j^0 = x_j \tag{3.5}$$

which terminates any recursion. By dropping the subscripts we can write (3.2) in the convenient vectorial form

$$Z^l = W^l \delta_{l-1}(Z^{l-1}) + b^l = W^l y^{l-1} + b^l \tag{3.6}$$

where each element in the z^l and y^l vectors are given by z^l_j and y^l_j, respectively, and the activation function is applied elementwise. The elements of the matrix W^l are given by $W^l_{jk} = w^l_{jk}$. With the above definitions the feed forward algorithm for computing the output y^L, given the input x, is given by

$$\begin{aligned} y^L &= \delta_L(z^L) \\ z^L &= W^L \delta_{l-1}(z^{L-1}) + b^L \\ z^{L-1} &= W^{L-1} \delta_{l-2}(z^{L-2}) + b^{L-1} \\ z^2 &= W^2 \delta_1(z^1) + b^2 \\ z^1 &= W^1 x + b^1 \end{aligned} \tag{3.7}$$

.

3.3 Backpropagation

Given some data $x = [x_1, \ldots, x_N]^T \in R^N$ and some target outputs $y = [y_1, \ldots, y_M]^T \in R^M$.

We wish to choose our weights and biases such that $y^L(x; w, b)$ is a good approximation of $y(x)$. We use the notation $y^L(x; w, b)$ to indicate that the ANN takes x as input and is parametrized by the weights and biases w, b. To find the weights and biases we define some cost function $C = C(y, y^L) : R^N \to R$ and compute

$$w^*, b^* = argmin C(y, y^L). \tag{3.8}$$

The minimization problem (3.8) can be solved using a variety of optimization methods, both gradient based and gradient free. In this thesis we will focus on gradient based methods and we derive the gradients that we need in order to use any gradient based optimization method. The standard method to compute gradients is backpropagation. The main ingredient is the error of neuron j in layer l defined by

$$\delta^L_j = \frac{\partial C}{\partial z^l_j}. \tag{3.9}$$

The standard back propagation algorithm for computing the gradients of the cost function is

then given by

$$\begin{aligned} \delta_j^L &= \frac{\partial C}{\partial y_j^L}\delta_L^{'}(z_j^L) \\ \frac{\partial C}{\partial w_{jk}^L} &= Y_k^{L-1}\delta_j^L \\ \delta_j^L &= \sum_k w_{kj}^{L+1}\delta_k^{L+1}\delta_L^{'}(z_j^L) \end{aligned} \tag{3.10}$$

The δ terms in (3.10) can be written in vectorial form as

$$\begin{aligned} \delta^L &= \nabla_{y^L} C \odot \delta_L^{'}(z^L), \\ \delta^l &= W^{l+1} + \delta^{l+1} \odot \delta_l^{'}(z^l) \end{aligned} \tag{3.11}$$

where we use the notation $\nabla_{y^L} C = [\frac{\partial C}{\partial y_1^L}, \ldots \frac{\partial C}{\partial y_m^L}]^T$ and $\odot$ denotes the Hadamard (component wise) product. The notation ∇C without a subscript will denote the vector of partial derivatives with respect to the input $x = [x_1, \ldots, N_t]^T$ note the transpose on the weight matrix W. The transpose W^T is the Jacobian matrix of the coordinate transformation between layer $l+1$ and l.

Let us begin by concentrating on the topic of computing solutions to generalized partial differential equations.

$$u_t + N[u;\lambda] = 0, \quad x \in \Omega, \quad t \in [0,T] \tag{3.12}$$

where $u(t,x)$ denotes the solution, $N[u;\lambda]$ is a nonlinear operator parametrized by λ, and Ω is a subset of $\mathbb{R}^D$. We define $f(t,x)$ to be given by

$$f = u_t + N[u] \tag{3.13}$$

and then use a deep neural network to approximate $u(t,x)$. This assumption, together with equation (3.13), results in the $f(t,x)$ physics-informed deep neural network. The chain rule for differentiating compositions of functions using automated differentiation can be used to create this network, which has the same parameters as the network representing $u(t,x)$ but with different values. Because of the differential operator N, several activation functions exist. By reducing the mean squared error loss, the shared parameters of the neural networks $u(t,x)$ and $f(t,x)$ can be learned.

$$MSE = MSE_u + MSE_f \tag{3.14}$$

where

$$MSE_u = \frac{1}{N_u}\sum_{i=1}^{N_u}|u(t_u^i, x_u^i) - u^i|^2 \tag{3.15}$$

and

$$MSE_f = \frac{1}{N_f}\sum_{i=1}^{N_f}|u(t_f^i, x_f^i)|^2 \tag{3.16}$$

here,$\{t_u^i, x_u^i, u^i\}_{i=1}^{N_u}$ denote the initial and boundary training data on $u(t,x)$ and $\{t_f^i, x_f^i\}_{i=1}^{N_f}$ specify the collocations points for $f(t,x)$. The loss MSE_u at a finite collection of collocation locations, corresponds to the initial and boundary data while enforcing the structure given by equation (3.12). The term "physics-informed machine learning" has lately become popular(Wang et al., 2017).

Automatic differentiation in general, and the back-propagation algorithm in particular, is the most widely used method for training deep models by taking derivatives of their parameters (e.g., weights and biases). We apply the same automatic differentiation techniques used by the deep learning community to physics-inform neural networks by calculating derivatives of their input coordinates (i.e., space and time) when the physics is given by partial differential equations. This structured approach adds a regularization mechanism that allows us to use relatively simple feed-forward neural network architectures and train them with little quantities of data, as we have actually found. Because the total number of training data N_u is tiny (a few hundred to a few thousand points) in all cases involving partial differential equations, we elected to optimize all loss functions using L-BFGS, a quasi-Newton, full-batch gradient-based optimization.

3.4 Mathematical Procedure

A physics-informed neural network (PINN) is a type of machine learning model in which the governing PDE is met by the neural network's loss function. The PINN technique takes advantage of neural networks' efficient optimization and prediction capabilities. A neural network is trained in PIDNN to predict the solution anywhere in the spatial temporal domain. Consider the following general form of a partial differential equation (PDE):where x and t are the spatial and temporal coordinates, respectively.

The solution of the PDE is approximated by PIDNN as a map between points in the spatio-temporal domain. The neural network's parameters are randomly chosen and modified iteratively by minimizing the loss function that enforces the PDE. Initial condition, boundary bondition, and PDE make up the PIDNN's loss function.

1. Determining the structure of an ANN, which is defined by a specific number of layers and

neurons.

2. Using the LHS approach, create three training sets: initial condition sets, boundary condition sets, and random collocation points.

3. Define the loss function $MSE = MSE_{u,BC,IC} + MSE_f$.
We consider $f(t,x)$ to be given by

$$\begin{aligned} &ih_t + \alpha h_{xx} + \beta |h|^2 h = 0, \quad x \in [x_0, x_1], \quad t \in [t_0, t_1], \\ &h(0,x) = f(x), \\ &h(t,x_0) = h(t,x_1), \\ &h_x(t,x_0) = h_x(t,x_1), \end{aligned}$$

where β and α are arbitrary parameters, then proceed by prioritizing $h(t,x)$ with a complex-valued neural network. Indeed, if u represents the real part of h and v represents the imaginary component, we are prioritizing a multi-out neural network.

$$h(t,x) = u(t,x) + iv(t,x).$$

The physic-informed deep neural network $f(t,x)$ will be complex-valued (multi-output). By reducing the mean squared error loss, the shared parameters of the neural networks $h(t,x)$ and $f(t,x)$ can be trained.

$$MSE = MSE_0 + MSE_b + MSE_f$$

where

$$MSE_0 = \frac{1}{N_0} \sum_{i=1}^{N_0} |h(x_0, x_0^i) - h_0^i|^2$$

$$MSE_b = \frac{1}{N_b} \sum_{i=1}^{N_b} (|h^i(t_b^i, x_0) - h^i(t_b^i, x_1)| + |h_x^i(t_b^i, x_0) - h_x^i(t_b^i - x_1)|^2)$$

$$MSE_f = \frac{1}{N_b} \sum_{i=1}^{N_f} |f(t_f^i, x_f^i)|^2$$

denotes the initial data $\{t_b^i\}_{i=1}^{N_f}$ corresponds to the collocation points on the boundary, and $\left\{t_f^i, x_f^i\right\}_{i=1}^{N_f}$ represents the collocation points on $f(t,x)$.
Consequently, MSE_0 corresponds to the loss on the initial data, MSE_b enforces the periodic boundary conditions, and MSE_f . If the Schrodinger equation is not satisfied on the collocation points, it is penalized.

4. Use an appropriate optimizer, such as Adam or AdaGrad, to train the PIDNN. as well as the numerical implementation using Python. There is different kinds of optimizer for minimization of the loss function. One of the widely used minimization algorithm in machine learning is the gradient decent method. We randomly initialize the parameters and update according to the following rule; for $j = 1, 2$,

$$p_j^{k+1} = p_j^k - \eta \bigtriangledown J(p_j^k)$$

where η is the learning rate and k corresponds to iteration. The gradient decent is well suited for convex function. For non convex function, we are not grantee for global minimum point. Note that, in addition to the simple gradient decent method, currently there are more advanced optimization tools and still is an active research topics. **The moment method** is the modification of gradient decent method designed to avoid getting stuck in a local minimum. The updating rule is as follows;

$$\begin{aligned} P_j^{k+1} &= P_j^k + V_j^{k+1} \\ V_j^{k+1} &= \mu V_j^k - \eta \bigtriangledown J(P_j^k) \end{aligned}$$

where $\eta > 0$ is the learning rate and μ is a coefficient between 0 and 1 called the momentum. Here k indicates iteration and V_j is a new parameter (velocity) initialized from zero corresponding to each unknown.
The Nesterov accelerated Gradient (NAG): is obtained by modifying the mo- mentum method and the update rule is given as follow,

$$\begin{aligned} P_j^{k+1} &= P_j^k + V_j^{k+1} \\ V_j^{k+1} &= \mu V_j^k - \eta \bigtriangledown J(P_j^k + \mu V_j^k) \end{aligned}$$

The main difference from the moment method is that, the argument of the gradient is computed at the correlated value $P^k + \mu V_j k$ instead of computing it at the current position P^k .

The AdaGrad, Adaptive Gradient: the update rule have the form;

$$V_j^k = V_j^{k-1} + (\bigtriangledown J(P_j^k))^2$$
$$P_j^{k+1} = P_j^k - \frac{\mu}{\sqrt{V_j^k + \varepsilon}} \bigtriangledown J(P_j^k)$$

where ε is small number to avoid division by zero. The method changes the learning rate for the parameters in proportional to the update history. It decays the learning rate.

The Root Mean Square Propagation, or RMSProp: is family of the gradient decent method having adaptive learning rate, again following, our update rule is as follows;

$$V_j^k = \beta V_j^{k-1} + (1-\beta)(\bigtriangledown J(P_j^k))^2$$
$$P_j^{k+1} = P_j^k - \frac{\mu}{\sqrt{V_j^k + \varepsilon}} \bigtriangledown J(P_j^k)$$

where $\beta \in (0,1)$ is the forgetting factor.

Adam, Adaptive Moment: is also an adaptive learning method which combines AdaGrad and RMSProp methods. In our case the updating rule have the form;

$$M_j^k = \beta_1 V_j^{k-1} + (1-\beta)(\bigtriangledown J(P_j^k))^2$$
$$V_j^k = \beta_2 V_j^{k-1} + (1-\beta_2)(\bigtriangledown J(p_j^k))^2$$
$$\hat{M}_j^k = \frac{M_k}{1-\beta_1^k}, \hat{V}_j^k = \frac{V_k}{1-\beta_2^k}$$
$$P_j^{k+1} = P_j^k - \frac{\mu}{\sqrt{\hat{V}_j^k + \varepsilon}} \hat{M}_j^k$$

where $\beta_1, \beta_2 \in [0,1)$, are decay rates for the moment estimates, and we initialize the parameters V^j and M_j to be zero.

CHAPTER 4

RESULTS AND DISCUSSIONS

We conduct an experiment on the problem of the type described above using the proposed PIDNN technique to verify the established theoretical results in this study. To reduce errors, we employ the loss function.

Example 1. Let us consider the Burgers' equation. This equation arises in various areas of applied mathematics, including fluid mechanics, nonlinear acoustics, gas dynamics, and traffic flow . It is a fundamental partial differential equation and can be derived from the Navier-Stokes equations for the velocity field by dropping the pressure gradient term. For small values of the viscosity parameters, Burgers' equation can lead to shock formation that is notoriously hard to resolve by classical numerical methods.

$$\begin{aligned} &u_t + uu_x - (0.01/\pi)u_{xx} = 0, \quad x \in [-1,1], \quad t \in [0,1], \\ &u(0,x) = -\sin(\pi x), \\ &u(t,-1) = u(t,1) = 0. \end{aligned}$$

Let us define $f(t,x)$ to be given by

$$f := u_t + uu_x - (0.01/\pi)u_{xx},$$

and proceed by approximating $u(t,x)$ by a deep neural network.
This will result in the physics-informed neural network $f(t,x)$.
The shared parameters between the neural networks $u(t,x)$ and $f(t,x)$ can be learned by minimizing the mean squared error loss

$$MSE = MSE_u + MSE_f,$$

where

$$MSE_u = \frac{1}{N_u}\sum_{i=1}^{N_u} |u(t_u^i, x_u^i) - u^i|^2,$$

and

$$MSE_f = \frac{1}{N_f}\sum_{i=1}^{N_f} |f(t_f^i, x_f^i)|^2$$

here $\{t_u^i, x_u^i, u^i\}_{i=1}^{N_u}$ denote the initial and boundary training data on $u(t,x)$ and $\left\{t_f^i, x_f^i\right\}_{i=1}^{N_f}$ specify the collocations points for $f(t,x)$.

The resulting $u(t,x)$ and $f(t,x)$ are guaranteed to satisfy the Burgers equation and conserve all associated in variances by construction. In this work, we replace $u(t,x)$ by a neural network $u(t,x;W,b)$ and obtain a physics-informed neural network $f(t,x;W,b)$ by automatic differentiation.

Consequently, the resulting pair $u(t,x;W,b)$ and $f(t,x;W,b)$ must satisfy the burgers equation regardless of the choice of the weights W and bias b parameters. We must underline that, unlike any classical numerical method for solving partial differential equations, this prediction is obtained without any sort of discretization of the spatio-temporal domain.

Figure 4.1 summarizes our results for the data-driven solution of the Burgers equation. Specifically, given a set of $N_u = 100$ and $N_f = 10000$ randomly distributed initial and boundary data, we learn the latent solution $u(t,x)$ by training all parameters of a 9-layer deep neural network using the mean squared error loss. Each hidden layer contained 20 neurons and a hyperbolic tangent activation function. The top panel of Figure 4.1 shows the predicted spatio-temporal solution $u(t,x)$, along with the locations of the initial and boundary training data. We must underline that, unlike any classical numerical method for solving partial differential equations, this prediction is obtained without any sort of discretization of the spatio-temporal domain.

The result of solving the Burgers equation using the deep learning framework is shown in Figure 4.1

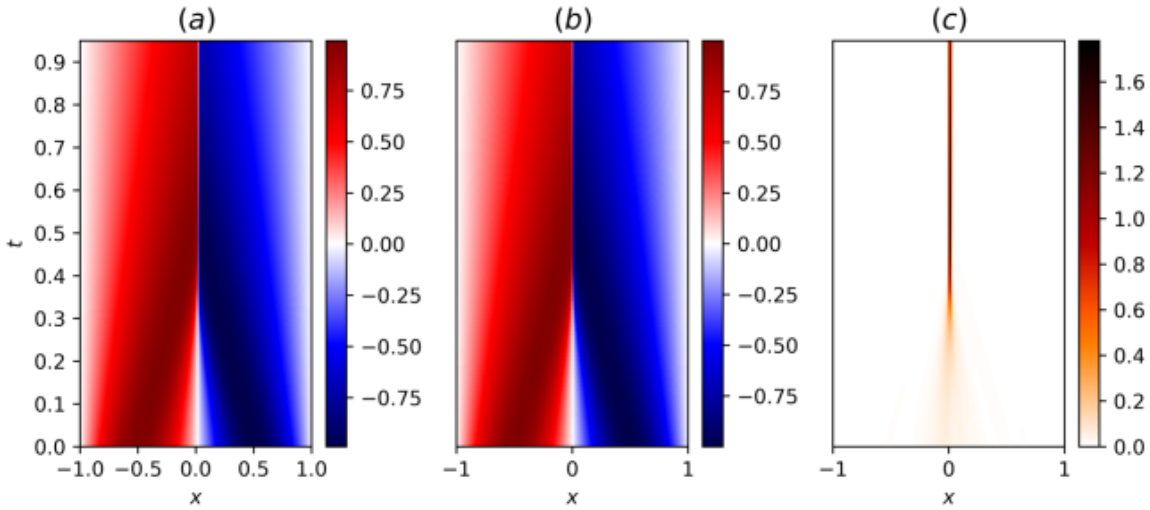


Figure 4.1: Solution of the Burgers equation using PINN. (a) True solution for $u(t,x)$; (b) PINN predicted values $u(t,x,W,b)$; (c) Absolute error between true and predicted values, $|u(t,x) - u(t,x,W,b)|$.

As expected, we observe that as the number of layers and neurons are increased (hence the capacity of the neural network to approximate more complex functions), the predictive accuracy is increased.

Table 4.1: The error between the predicted and the exact solution $u(t,x)$ for different number of initial and boundary training data N_u, and different number of collocation points N_f. Here, the network architecture is fixed to 9 layers with 20 neurons per hidden layer.

N_u	$N_f = 2000$	4000	6000	7000	8000	10000
20	2.9e-01	4.4e-01	8.9e-01	1.2e+00	9.9e-02	4.2e-02
40	6.5e-02	1.1e-02	5.0e-01	9.6e-03	4.6e-01	7.5e-02
60	3.6e-01	1.2e-02	1.7e-01	5.9e-03	1.9e-03	8.2e-03
80	5.5e-03	1.0e-03	3.2e-03	7.8e-03	4.9e-02	4.5e-03
100	6.6e-02	2.7e-01	7.2e-03	6.8e-04	2.2e-03	6.7e-04
200	1.5e-01	2.3e-03	8.2e-04	8.9e-04	6.1e-04	4.9e-04

Table 4.2: The error between the predicted and the exact solution $u(t,x)$ for different number of hidden layers and different number of neurons per layer. Here, the total number of training and collocation points is fixed to $N_u = 100$ and $N_f = 10,000$, respectively.

Layers and Neurons	10	20	40
2	7.4e-02	5.3e-02	1.0e-01
4	3.0e-03	9.4e-04	6.4e-04
6	9.6e-03	1.3e-03	6.1e-04

Example 2. We take a schrödinger differential equation with the boundary condition; this example demonstrates our method's ability to handle periodic boundary conditions, complex-valued solutions, and various sorts of non-linearities in the governing partial differential equations. The nonlinear term in optics refers to a material's intensity dependent index of refraction. The nonlinear term for bose-einstein condensates is caused by mean-field interactions in an interacting N-body system. With periodic boundary conditions, the nonlinear Schrödinger equation is given by

$$\begin{aligned} &ih_t + 0.5h_{xx} + |h|^2 h = 0, \quad x \in [-5,5], \quad t \in [0,\pi/2], \\ &h(0,x) = 2\ \mathrm{sech}(x), \\ &h(t,-5) = h(t,5), \\ &h_x(t,-5) = h_x(t,5), \end{aligned}$$

where $h(t,x)$ is the complex-valued solution .

Solution

Initial data, boundary data, and collocation points inside the domain are used to train PIDNN solutions, $t = 0$ are the initial time data. $\{x_0^i, h_0^i\}_{i=1}^{N_0}$ the boundary collocation points and $\{t_b^i\}_{i=1}^{N_b}$ the collocation points on $f(t,x)$ are $\{t_f^i, x_f^i\}_{i=1}^{N_f}$.

To be included in the training set, a total of $N_0 = 50$ beginning data points on $h(0,x)$ are randomly sampled from the entire high-resolution data set, as well as $N_b = 50$ randomly sampled boundary points to enforce the periodic bounds. Finally, $N_c = 20000$ randomly sampled collocation points are assumed for the solution domain.

Because the output of this NN is anticipated to locate the real and imaginary parts of the so-

lution, the neural network design includes two inputs, one for time t and the other for position x. The output also has length 2 rather than 1, as would generally be assumed. The network is trained to reduce losses caused by initial and boundary conditions, as well as to satisfy the Schrodinger equation on collocation points. No more data is used because we are interested in a model that is a surrogate for the PDE. In fact, we only use the known data point from the initial time step $t = 0$ to train the PIDNN. As a result, let us define $f(t,x)$ to be given by

$$f := ih_t + 0.5h_{xx} + |h|^2 h,$$

and then put a complex-valued neural network in front of $h(t,x)$. In fact, if u symbolizes the real part of h and v the imaginary part, we are prioritizing $h(t,x) = u(t,x) + iv(t,x)$ with a multi-out neural network. Two real-valued functions are u and v. The complex derivative u_x is defined as

$$u_x = \frac{\partial u}{\partial x} + i\frac{\partial u}{\partial x}$$

This will result in the complex-valued (multi-output) physic informed deep neural network $f(t,x)$. The shared parameters of the neural networks $h(t,x)$ and $f(t,x)$ can be learned by minimizing the mean squared error loss

$$MSE = MSE_0 + MSE_b + MSE_f,$$

where

$$MSE_0 = \frac{1}{50}\sum_{i=1}^{N_0} |h(0,x_0^i) - 2sech(x_0^i)|^2,$$

$$MSE_b = \frac{1}{50}\sum_{i=1}^{N_b} \left(|h^i(t_b^i,-5) - h^i(t_b^i,5)|^2 + |h_x^i(t_b^i,-5) - h_x^i(t_b^i,5)|^2\right),$$

and the governing PDE loss is

$$MSE_f = \frac{1}{20000}\sum_{i=1}^{N_f} (|ih_t + 0.5h_{xx} + |h|^2.h|^2),$$

Here, $\{x_0^i, h_0^i\}_{i=1}^{N_0}$ denote the initial data, $\{t_b^i\}_{i=1}^{N_b}$ corresponds to the collocation points on the boundary, and $\{t_f^i, x_f^i\}_{i=1}^{N_f}$ represents the collocation points on $f(t,x)$. Consequently, MSE_0 corresponds to the loss on the initial data, MSE_b enforces the periodic boundary conditions, and MSE_f penalizes the schrödinger equation not being satisfied on the collocation points.

$$ih_t + 0.5h_{xx} + |h|^2 h, = 0$$

with periodic boundary conditions as

$$x \in [-5,5], \quad t \in [0,\pi/2]$$

$$h(t,-5) = h(t,5),$$

and initial condition equal to

$$h(0,x) = 2\ \mathrm{sech}(x).$$

We only utilize real numbers, hence the real and imaginary components of the complex partial differential equation must be explicitly separated.
The lone residua has been replaced by,

$$f = ih_t + 0.5h_{xx} + |h|^2 h, = 0$$

The two (real-valued) residuals are obtained.

$$f_R = u_t + 0.5v_{xx} + (u^2 + v^2)v,$$
$$f_I = v_t - 0.5u_{xx} - (u^2 + v^2)u$$

where $u(x,t)$ and $v(x,t)$ denote the real and imaginary parts of h, respectively. A key property of physics-informed neural networks is that they can be effectively trained using small data sets; a setting often encountered in the study of physical systems for which the cost of data acquisition may be prohibitive.

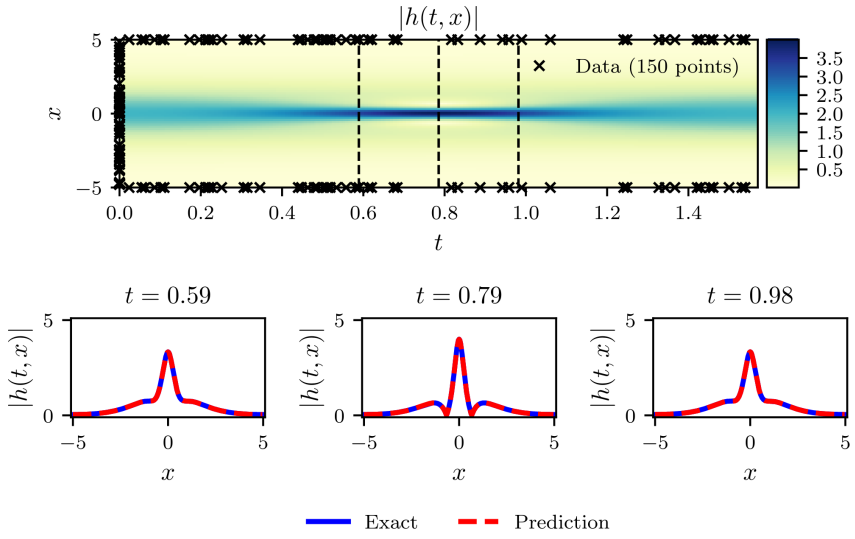


Figure 4.2: The solution of schrödinger equation

Figure 4.2 summarizes the results of our experiment. Specifically, the top panel of figure 4.2 shows the magnitude of the predicted spatio-temporal solution $|h(t,x)| = \sqrt{u^2(t,x) + v^2(t,x)}$,

along with the locations of the initial and boundary training data. A more detailed assessment of the predicted solution is presented in the bottom panel of Figure 4.2. In particular, we present a comparison between the exact and the predicted solutions at different time instants t = 0.59, 0.79, 0.98. Using only a handful of initial data, the physics-informed neural network can accurately capture the intricate nonlinear behavior of the Schrödinger equation. Finally, Table 4.3 shows the results of a few trials in which we examine how the training loss, *MAE*, and *MSE* change as the network architecture, number of collocation points, boundary data, and initial value data are changed.

Furthermore, if the number of training points or epochs is increased or decreased, we can see how the loss error changes.

Table 4.3: The number of layers and neurons determine the expressiveness of the PINN in solving the 1D nonlinear shrödinger (NLS) equation; the higher the number, the better it may approximate the function; however, a higher number of layers necessitates a greater number of epochs and thus a longer training time.

hidden layer	neurons per layer	N_0	N_b	N_f	Epoch	Training loss	MAE	MSE
3	10	40	120	3000	15000	8,35E-03	1,42E-01	9,02E-02
3	50	40	90	3000	15000	7,55E-04	4,22E-02	7,62E-03
3	100	40	90	3000	1500	3,53E-04	2,95E-02	3,76E-03
3	200	40	120	20000	1500	2,35E-04	2,33E-02	2,26E-03
3	200	200	120	20000	15000	2,04E-04	1,89E-02	1,67E-03
4	70	50	50	5000	15000	3,48E-04	3,13E-02	4,47E-03
4	100	100	100	1000	15000	3,30E-05	1,76E-01	1,53E-01
4	100	150	150	1000	15000	5,61E-05	8,96E-02	4,53E-02
4	100	100	100	5000	15000	1,56E-04	1,88E-02	1,68E-03
4	100	50	50	5000	15000	2,54E-04	2,87E-02	3,89E-03
4	100	100	100	10000	30000	8,20E-05	1,12E-02	5,17E-04
4	100	100	100	10000	15000	2,03E-04	1,85E-02	1,35E-03
4	100	50	50	20000	10000	6,33E-04	4,24E-02	7,51E-03
4	100	50	50	20000	20000	9,20E-05	1,51E-02	9,25E-04
4	100	50	50	20000	30000	7,56E-05	1,37E-02	7,81E-04
4	100	50	50	20000	15000	2,78E-04	2,59E-02	2,77E-03

Conclusions and Recommendations

Conclusions

This thesis describes a strategy for combining physical data with physics-informed deep neural networks to solve nonlinear partial differential equations. In this study, the physical principles found in partial differential equations are applied to neural networks as a regularization. This breakthrough allows the neural network to learn partial differential equation solutions with less observations. The method proposed in this thesis yields good experimental results because to the powerful function approximation capacity of neural networks and the physical information included in partial differential equations. Deep neural networks based on physics are utilized to investigate two nonlinear partial differential equations in this thesis.

Recommendations

This thesis specifically focused on application of PIDNN method to solve approximate solution of non-linear partial differential equations. PIDNN method is very interesting and easy to find approximate solution on nonlinear partial differential equations. Hence, we recommended that, future researchers should try to compare PIDNN method with other method to generalize the efficiency, validity of this method. Also one can extend the PIDNN method to non-linear fifth order PDE, 3-dimensional PDE and etc.

Research Fund Acknowledgment

This research project is funded by Adama Science and Technology University under the grant number ASTU/SM-R/420/22, Adama, Ethiopia.

References

Aarts, L. P., & Van der Veer, P. (2001). Solving nonlinear differential equations by a neural network method. In *International conference on computational science* (pp. 181–189).

Adomaitienė, E., Tamaševičius, A. V., Mykolaitis, G., Bumelienė, S., & Lindberg, E. (2008). Analogue electrical circuit for simulation of the duffing-holmes equation. *Nonlinear Analysis: Modelling and Control*, *13*(2), 241–252.

Berg, J., & Nyström, K. (2018). A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, *317*, 28–41.

Burney, S. M. A., Jilani, T. A., Ardil, C., & Muhammad, S. (2005). Levenberg-marquardt algorithm for karachi stock exchange share rates forecasting. *International Journal of Computational Intelligence*, *1*(3), 144–149.

Chakraverty, S., & Mall, S. (2017). *Artificial neural networks for engineers and scientists: solving ordinary differential equations*. CRC Press.

Chiaramonte, M., Kiener, M., et al. (2013). Solving differential equations using neural networks. *Machine Learning Project*, *1*.

Da Silva, I. N., Spatti, D. H., Flauzino, R. A., Liboni, L. H. B., & dos Reis Alves, S. F. (2017). Artificial neural network architectures and training processes. In *Artificial neural networks* (pp. 21–28). Springer.

Freeman, J. A., & Skapura, D. M. (1991). Algorithms, applications, and programming techniques. *Neural Networks*.

Giorelli, M., Renda, F., Ferri, G., & Laschi, C. (2013). A feed-forward neural network learning the inverse kinetics of a soft cable-driven manipulator moving in three-dimensional space. In *2013 ieee/rsj international conference on intelligent robots and systems* (pp. 5033–5039).

Greenwade, G. D. (1993). The comprehensive tex archive network (ctan). *TUGBoat*, *14*(3), 342–351.

Gross, M. H., & Seibert, F. (1993). Visualization of multidimensional image data sets using a neural network. *The Visual Computer*, *10*(3), 145–159.

Hagge, T., Stinis, P., Yeung, E., & Tartakovsky, A. M. (2017). Solving differential equations with unknown constitutive relations as recurrent neural networks. *arXiv preprint arXiv:1710.02242*.

Lagaris, I. E., Likas, A., & Fotiadis, D. I. (1998). Artificial neural networks for solving ordinary

and partial differential equations. *IEEE transactions on neural networks*, *9*(5), 987–1000.

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, *5*(4), 115–133.

Monterola, C., & Saloma, C. (2001). Solving the nonlinear schrodinger equation with an unsupervised neural network. *Optics express*, *9*(2), 72–84.

Mukhopadhyay, S. (2018). Deep learning and neural networks. In *Advanced data analytics using python* (pp. 99–119). Springer.

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, *378*, 686–707.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, *65*(6), 386.

Schalkoff, R. J. (1997). *Artificial neural networks*. McGraw-Hill Higher Education.

Schmidhuber, J. (2015). Deep learning. *Scholarpedia*, *10*(11), 32832.

Wambecq, A. (1978). Rational runge-kutta methods for solving systems of ordinary differential equations. *Computing*, *20*(4), 333–342.

Wang, J.-X., Wu, J., Ling, J., Iaccarino, G., & Xiao, H. (2017). A comprehensive physics-informed machine learning framework for predictive turbulence modeling. *arXiv preprint arXiv:1701.07102*.

Wu, G.-Z., Fang, Y., Wang, Y.-Y., & Dai, C.-Q. (2021). Modified physics-informed neural network method based on the conservation law constraint and its prediction of optical solitons. *arXiv preprint arXiv:2108.13192*.

Yadav, N., Yadav, A., Kumar, M., et al. (2015). *An introduction to neural network methods for differential equations*. Springer.