SOLVING SECOND ORDER NON-LINEAR ORDINARY DIFFERENTIAL EQUATION USING ARTIFICIAL NEURAL NETWORKS.

Geremu Alemayew

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

Office of Graduate Studies

Adama Science and Technology University

August 9, 2021

Adama, Ethiopia

SOLVING SECOND-ORDER NON-LINEAR DIFFERENTIAL EQUATION USING ARTIFICIAL NEURAL NETWORKS

Geremu Alemayew

ADVISOR : Tamirat Temesgen(PhD.)

CO-ADVISOR: Yadata Chimdessa(M.Sc.)

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

Office of Graduate Studies

Adama Science and Technology University

August 9, 2021

Adama, Ethiopia

Declaration

I hereby declare that this Master Thesis entitled "Solving second order non-linear differential equation using artificial neural networks"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

We, the advisors of this thesis, hereby certify that we have read the revised version of the thesis entitled "Solving second order non-linear differential equation using artificial neural networks"prepared under our guidance by Geremu Alemayew Dureso submitted in partial fulfillment of the requirements for the degree of Master's of Science in Applied Mathematics. Therefore, we recommend that 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 second order non-linear differential equation using artificial neural networks"and developed by Geremu Alemayew Dureso, hereby certify that the recommendation and suggestion 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 Geremu Alemayew Dureso, have read and evaluated the thesis entitled "Solving second order non-linear differential equation using artificial neural networks"and examined the candidate during the open defence. This is, therefore, to certify that the thesis is accepted for partial fulfilment of the requirement of the degree of Master of Science in Applied Mathematics.

Chairperson	**Signature**	**Date**
Internal Examiner	**Signature**	**Date**
External Examiner	**Signature**	**Date**

Finally, approval and acceptance of the thesis are contingent upon submission of its final copy to the Office of Postgraduate Studies (OPGS) through the Department Graduate committee (DGC) and School Graduate Committee (SGC).

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

ACKNOWLEDGEMENT

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 Temesgen** and my **co-advisor Mr. Yadeta Chimdessa** for their helpful discussion, comments, giving direction, and providing the necessary materials in the preparation of this 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 for giving the necessary materials throughout the preparation of this thesis.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

List of Acronyms

ANN- Artificial Neural Networks

FL ANN- Functional link Artificial Neural Networks

HeNN- Hermite Neural Networks

MLP- Multilayer perceptron

ODE- Ordnary Differential Equation

IVPs- Initial value problems

RK4-Runge-Kutta fourth-order

ABSTRACT

In this study, We studied on solving a non-linear ordinary differential equations using the proposed Hermite neural network (HeNN). A single layer Hermite Functional Link Artificial Neural Network (HeNN) developed. HeNN has no hidden layer and its input pattern dimensions expanded by Hermite orthogonal polynomials. The computations become efficient because the procedure only requires to be input and output layers. A Feedforward neural networks model and an unsupervised version of error back propagations used to minimize the error function and to update the network parameters. Furthermore, we compared the HeNN results with the correct output and results obtained by the well-known numerical techniques(RK4). Two examples and one implementation (Van der Pol–Duffing oscillator equation) took to show the efficiency and power of this developed model with plotted graphs using python code.

keywords: Artificial Neural Networks, Single-layer Functional Neural Network, Single-layer Hermite Neural Network, Ordinary Differential Equation.

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 computational complexity increases with the number of sampling points, which is not a straightforward task(Wambecq, 1978).

In contrast, in recent decades, various machine intelligence procedures in particular connectionist learning or Artificial Neural Network (ANN) methods have been established as a powerful technique to solve a variety of real-world problems because of its excellent learning capacity (Schalkoff, 1997). The simplest definition of ANN is provided by the inventor of one of the first neurocomputers, Dr. Robert Hecht-Nielsen (Mukhopadhyay, 2018). He defines "a neural network as a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs".

ANNs are processing devices (algorithms) that are loosely modeled after the neuronal structure of the mammalian cerebral cortex, but on much smaller scales. Computer scientists have always been inspired by the human brain. In 1943, Warren S. McCulloch, a neuroscientist, and Walter Pitts, a logician, developed the first conceptual model of an ANN (McCulloch & Pitts, 1943). In their paper, they describe the concept of a neuron, a single cell living in a network of cells that receives inputs, processes those inputs, and generates an output. ANN is a computational model or an information processing paradigm inspired by biological nervous system (Chakraverty & Mall, 2017; Anderson, 1995; Fitch, 1944) as shown figure 1.1. ANN processes information through neurons-nodes

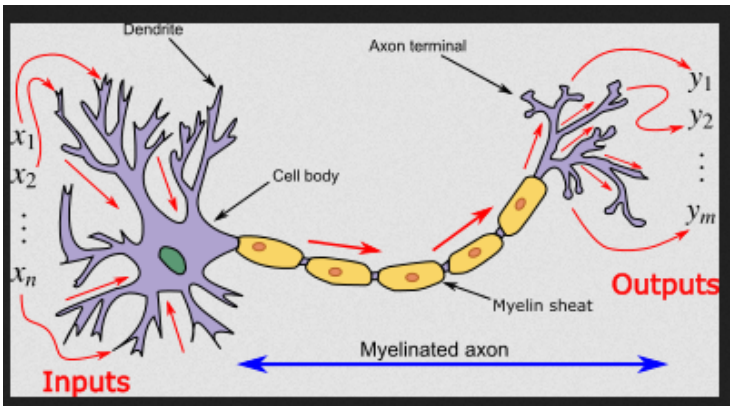

Figure 1.1: Biological Neuron

(Galinkin, 2020)

in a parallel manner to solve specific problems. It acquires knowledge through learning, and this knowledge is stored within inter-neuron connections' strength, which is expressed by numerical values called "weights." These weights are used to compute output signal values for a new testing input signal value. Patterns are presented to the network via the "input layer," which communicates to one or more "hidden layers," where the actual processing is done via a system of weighted "connections." The hidden layers then link to an "output layer," where the answer is output, as shown in Figure 1.2. 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.

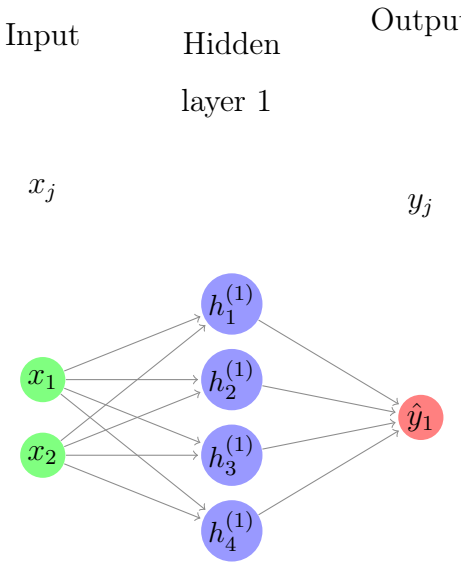

Figure 1.2: Structure of artificial neural network
(Greenwade, 1993)

There are different types of ANN such as Single-layer ANNs, Multilayer feedforward ANNs, Temporal ANNs etc. In this thesis, we have considered Single Layer artificial neural networks from different ANN types.

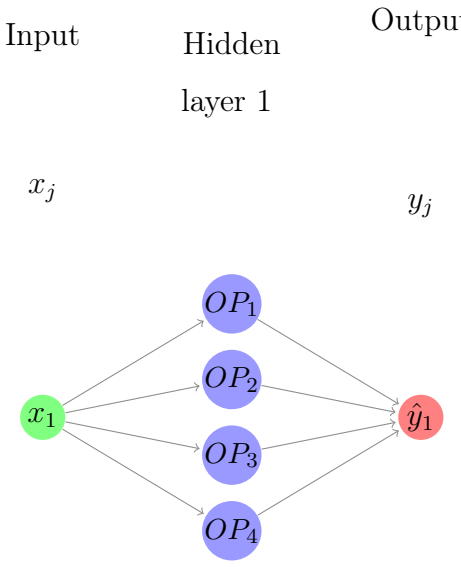

Figure 1.3: Architecture of single layer artificial neural network

FLANN models are fast-learning single-layer artificial neural network (ANN) models. The single-layer FLANN method is introduced by Pao and Philips (Chakraverty & Mall, 2017). In FLANN, the hidden layer is replaced by a functional expansion block for enhancement of the input patterns using orthogonal polynomials such as Chebyshev,

Legendre, Hermite, etc. And the total number of network parameters is less than that of the multilayer perceptron (MLP) structure. So the technique is computationally more efficient than MLP. Some advantages of the single-layer FLANN-based model for solving differential equations may be mentioned as follows: it is a single-layer neural network, so the total number of network parameters is less than that of traditional multilayer ANN,it is capable of fast learning and is computationally efficient,it is simple to implement and easy to compute, hidden layers are not required,the back propagation algorithm is unsupervised,no optimization technique is to be used (Pao & Phillips, 1995).

1.2 Statement of the Problem

We consider a non-linear second-order ordinary differential equation,

$$\frac{d^2x}{dt^2} = f\left(t, x, \frac{dx}{dt}\right), t \in [a, b]$$

with initial conditions $x(a) = A, x'(a) = A'$. Then, its HeNN trial solution may then be written as

$$x_{He}(t, p) = A + A'(t - a) + (t - a)^2 N(t, p)$$

where $N(t, p)$ is the output of the HeNN with one input node t and parameter p and the trial solution $x_{He}(t, p)$ satisfies the initial conditions (Mall & Chakraverty, 2016). This study is intended to answer the following basic questions;

- What is the Hermite neural network solution(HeNN)?
- How to formulate second-order non-linear differential equations to a proposed formula(.i.e, HeNN)?
- Is this method is efficient to solve non-linear DE?
- Can we write an efficient algorithm for the proposed method?

1.3 Objective of the study

1.3.1 General objective

The general objective of this study is to solve a non-linear second order ordinary differential equation using Hermite neural network (HeNN).

1.3.2 Specific objective of the study

The specific objectives of this study are to:-

1. formulate the Hermite neural network method for second order non-linear ordinary differential equation,
2. develop an algorithm for the Hermite neural network method,
3. compute solution of the neural network method,
4. compare the neural network solution with exact and existing numerical solution.

1.4 Significance of the study

The result of this study will help to:

- understand how to solve second-order nonlinear differential equation that may not be solved analytically,
- apply the proposed method to other non-linear DE equations to get an approximate solution,
- know an algorithm of a new method(i.e. Hermite neural network method).

CHAPTER 2

LITERATURE REVIEW

The ANN method has been established as a powerful technique to solve a variety of real-world problems because of its excellent learning capacity. This method has been successfully applied in various fields such as function approximation, clustering, prediction, identification, pattern recognition, solving ordinary and partial differential equations, etc (Schalkoff, 1997; Adomaitienė, Tamaševičius, Mykolaitis, Bumelienė, & Lindberg, 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, Spatti, Flauzino, Liboni, & dos Reis Alves, 2017). Chiaramonte et al. formulate the non-linear second-order DE to ANN and approximate using ANN method (Chiaramonte, Kiener, et al., 2013).

In (Yadav, Yadav, Kumar, et al., 2015) paper 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., is described. Backpropagation and other training algorithms have been also discussed.

A trial solution of the differential equation is written as a sum of two parts. The first part satisfies the initial/boundary conditions and contains no adjustable parameters. The second part is constructed so as not to the initial/boundary conditions. This part involves a feed-forward neural network, containing adjustable parameters (the weights). Hence, by construction, the initial/boundary conditions are satisfied, and the network is trained to satisfy the differential equation (Lagaris, Likas, & Fotiadis, 1998).

Neural networks is considered as a part of large field called neural computing or soft computing. They proposed a new solution method for the approximated solution of high order ordinary differential equations using innovative mathematical tools and neural-like systems of computation. This hybrid method can result in improved numerical methods for solving initial/boundary value problems, without using pre-assigned discretisation points. Excellent test results are obtained for the solution of lower and higher order differential equations. The model finds approximation solution for the differential equation inside and outside the domain of consideration for the close enough neighborhood of initial/boundary points. Numerical examples are described to demonstrate the method (Giorelli, Renda, Ferri, & Laschi, 2013).

FLANN models are fast-learning single-layer artificial neural network (ANN) models. The single-layer FLANN method is introduced by Pao and Philips. In FLANN, the hidden layer is replaced by a functional expansion block for enhancement of the input patterns using orthogonal polynomials such as Chebyshev, Legendre, Hermite, etc. The paper explains the structure of single-layer FLANN models, and explains the formulations and gradient computations. In FLANN, the total number of network parameters is less than that of the multilayer perceptron (MLP) structure. So the technique is computationally more efficient than MLP (Chakraverty & Mall, 2017).

A single-layer Hermite neural network (HeNN) model is used, where a hidden layer is replaced by an expansion block of the input pattern using Hermite orthogonal polynomials. A feed-forward neural network model with the unsupervised error back propagation principle is used for modifying the network parameters and minimizing the computed error function (Mall & Chakraverty, 2016).

Hermite polynomial-based functional link artificial neural network (FLANN) is proposed to solve the Van der Pol–Duffing oscillator equation, where a hidden layer is replaced by an expansion block of input pattern using Hermite orthogonal polynomials. A feed-forward neural network model with the unsupervised error back propagation principle is used for modifying the network parameters and minimizing the computed error function. The Van der Pol–Duffing and Duffing oscillator equations may not be solved exactly. Here, approximate solutions of these types of equations have been obtained by applying the HeNN model. Three mathematical example problems and two real-life application problems of Van der Pol–Duffing oscillator equation, extracting the features of early mechanical failure signal and weak signal detection problems, are solved using the proposed HeNN method. After training the HeNN model, they may used it as a black box to get numerical results at any arbitrary point in the domain (Mall & Chakraverty, 2016).

In (Aarts & Van der Veer, 2001) paper they demonstrated a neural network method to solve nonlinear differential equations and its boundary conditions. The idea of their method is to incorporate knowledge about the differential equation and its boundary conditions into neural networks and the training sets. Hereby they obtained specifically structured neural networks. To solve the nonlinear differential equation and its boundary conditions they have to train all obtained neural networks simultaneously. This is realized by applying an evolutionary algorithm.

CHAPTER 3

PRELIMINARIES

This chapter includes definition of differential equation, artificial neural networks(architecture, learning methods, learning rules, components and types).

3.1 Differential Equation

Definition 3.1.1. An equation involving one dependent variable and its derivative concerning one or more independent variables is called a differential equation. It may be noted that most of the fundamental laws of nature can be formulated as DE's, namely, the growth of a population, conduction of heat in a rod, vibration of structures, etc (Mall & Chakraverty, 2016).

The following are some examples of DE's:

$$\frac{dy}{dx} + 2xy = e^{-x^2} \tag{3.1}$$

$$\frac{d^2y}{dx^2} + \frac{2}{x}\frac{dy}{dx} + y^5 = 0 \tag{3.2}$$

$$\frac{d^3y}{dx^3} - 3 + 2 = 4\tan x \tag{3.3}$$

3.1.1 Ordinary Differential Equation

Definition 3.1.2. The ordinary differential equation (ODE) is a differential equation that contains only one independent variable so that all the derivatives occurring in it are ordinary derivatives.

The general form of a nth-order ODE is,

$$f\left(x, y, \frac{dy}{dx}, \frac{d^y}{dx^2}, ..., \frac{d^n y}{dx^n}\right) = 0 \tag{3.4}$$

or

$$f\left(x, y, y', y'', y''', ..., y^n\right) = 0 \tag{3.5}$$

Equations 3.1 through 3.2 are examples of ODE's (Chakraverty & Mall, 2017).

3.1.2 Linear and Nonlinear Differential Equations

Definition 3.1.3. Linear Differential Equations are those in which (i) the dependent variable and its derivatives appear only in the first degree, and (ii) products of derivatives and the dependent variable do not occur. It expressed as

$$c_0\frac{d^n y}{dx^n} + c_1\frac{d^{n-1} y}{dx^{n-1}} + ... + c_{n-1}\frac{dy}{dx} + c_n y = f(x) \tag{3.6}$$

where the coefficient $c_0, ..., c_n$ and $f(x)$ denote constants and the functions of x, respectively.

The following are some examples of linear DE's:

$$\begin{cases} \frac{dy}{dx} + xy = sinx \\ y'' - 4xy' + (4x^2 - 1)y = -3x^2 \end{cases} \tag{3.7}$$

(Chakraverty & Mall, 2017)

Definition 3.1.4. Nonlinear differential equations are Differential equations that are not linear. That is, the products of the unknown function and derivatives are allowed and degree of dependent variable is > 1.

The following are some examples of nonlinear DE's:

$$\begin{cases} \frac{d^2y}{dx^2} + \frac{2}{x}y\frac{dy}{dx} + e^{-x} = 0 \\ \frac{d^2x}{dt^2} + \alpha\frac{dx}{dt} + \beta x + \gamma x^3 = Fcos\omega t \end{cases} \tag{3.8}$$

(Chakraverty & Mall, 2017)

3.1.3 Initial Value Problem's (IVP's)

Many problems in science and engineering can be modeled by ODE's or PDE's, along with one or more supplementary conditions. If these conditions are given at one point of

the independent variable, the problem is called the initial value problem (IVP), and these conditions are known as initial conditions. Let us consider the first-order ODE

$$\frac{dy}{dx} = f(x, y) \tag{3.9}$$

which satisfies the initial condition $y(x_0) = y_0$. Similarly, for a second-order ODE

$$\frac{d^2y}{dx^2} = f(x, y, \frac{dy}{dx}) \tag{3.10}$$

subject to initial conditions $y(x_0) = y_0, y'(x_0) = y_1$, and $n^{th}-$ order IVP with n initial conditions may be written as

$$\frac{d^ny}{dx^n} = f\left(x, y(x), \frac{dy}{dx}, \frac{d^2y}{dx^n}, ..., \frac{d^{n-1}y}{dx^{n-1}}\right) \tag{3.11}$$

with initial conditions

$$\begin{aligned} y(x_0) &= y_0 \\ y'(x_0) &= y_1 \\ y''(x_0) &= y_2 \\ &\vdots \\ y^{n-1}(x_0) &= y_{n-1}. \end{aligned}$$

(Chakraverty & Mall, 2017).

3.2 Artificial Neural Network

ANN is a technique that seeks to build an intelligent program using models that simulate the working of the neurons in the human brain. ANN processes information similarly as the human brain does. The key element of the network is the structure of the information processing system. The network is composed of many highly interconnected processing elements (neurons) working in parallel to solve a specific problem (Mall & Chakraverty, 2016).

3.2.1 Components of the Basic Artificial Neuron:

Neuron(Node)

It is the basic unit of a neural network (Uhrig, 1995).

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 (Uhrig, 1995).

Hidden Layer

This is the medium layer in the neural network. Hidden layers have neurons(nodes) that apply different transformations to the input data (Uhrig, 1995).

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 (Uhrig, 1995).

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 (Uhrig, 1995).

Bias(Offset)

It is an extra input neuron/node. It is used for shifting the activation function towards 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 (Uhrig, 1995).

Summation Function

The work of the summation function is to bind the weights and inputs together and find their sum.

Neural networks can be considered as approximation schemes where the input data for a design of network consisting of only a set of unstructured discrete data points. Thus, an application of a neural network for solving differential equations can be regarded as a mesh-free numerical method. Recently, a lot of attention has been devoted to the study of ANN for solving differential equations. The approximate solution of differential equations by ANN is found to be advantageous, but it depends upon the ANN model that one considers (Chakraverty & Mall, 2017).

3.2.2 ANN Architecture

According to the structure of connections, different classes of neural network architecture can be identified as below (Chakraverty & Mall, 2017).

Feed Forward Neural Network

In a feed-forward neural network, the neurons are organized in the form of layers. The neurons in a layer receive input from the previous layer and feed their output to the next layer. Network connections to the same or previous layers are not allowed. Here, the data goes from input to output nodes in a strictly feed-forward way. There is no feedback (back loops) that is, the output of any layer does not affect the same layer(Mall & Chakraverty, 2016).

Feedback Neural Network

These networks can have signals traveling in both directions by the introduction of loops in the network. These are very powerful and at times get extremely complicated. They are dynamic and their state changes continuously until they reach an equilibrium point

3.2.3 Learning Methods

The ability to learn and generalize from a set of training data is one of the most powerful features of ANN. The learning situations in neural networks may be classified into two types. These are supervised and unsupervised learning (Mall & Chakraverty, 2016).

Supervised Learning or Associative Learning

In supervised or associative learning, the network is trained by providing input and output patterns. These input-output pairs can be provided by an external teacher or by the sys-

tem which contains the network. A comparison is made between the network's computed output and the corrected expected output, to determine the error. The error can then be used to change network parameters, which results in the improvement of performance.

Unsupervised or Self organization Learning

Here, the target output is not presented to the network. There is no teacher to present the desired patterns, and therefore the system learns on its own by discovering and adapting to structural features in the input patterns (Chakraverty & Mall, 2017).

3.2.4 Activation Functions

An activation function is a function that acts upon the net (input) to get the output of the network. The activation function acts as a squashing function, such that the output of the neural network lies between certain values (usually 0 and 1, or -1 and 1). The Nonlinear Activation Functions are mainly divided on the basis of their range or curves (there are many type). Sigmoid or Logistic Activation Function. The Sigmoid Function curve looks like a S-shape. it exists between (0 to 1). tanh is also like logistic sigmoid but better. The range of the tanh function is from (-1 to 1). tanh is also sigmoidal (s - shaped). The advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the tanh graph. The function is differentiable. Both tanh and logistic sigmoid activation functions are used in feed-forward nets.(Xia Wang, 2020)

In this study, we have used tangent hyperbolic activation functions only, which are continuously differentiable. The output of the tangent hyperbolic activation function lies between -1 to 1.

For example, the tangent hyperbolic activation function is defined as

$$T(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

The derivatives of the above tangent hyperbolic activation function may be formed as

$$\begin{cases} T' = 1 - T^2 \\ T'' = 2T^3 - 2T \\ T''' = 24T^5 - 36T^3 + 12T \\ \vdots \end{cases}$$

(Chakraverty & Mall, 2017)

3.2.5 ANN Learning Rules

Learning is the most important characteristic of the ANN model. Every neural network possesses knowledge that is contained in the values of the connection weights. Modifying the knowledge stored in the network as a function of experience implies a learning rule for changing the values of the weights. There are various types of learning rules for ANN such as (Eberhart, Simpson, & Dobbins, 1992)

- Hebbian learning rule
- Perceptron learning rule
- Error back propagation or Delta learning rule
- Widrow- Hoff learning rule
- Winner- Take learning rule etc.

Error Back Propagation Learning Algorithm or Delta Learning Rule

Error propagation learning algorithm has been introduced by Rumelhart et al (Eberhart et al., 1992). It is also known as the Delta learning rule and is one of the most commonly used learning rules . It is valid for continuous activation function and is used in supervised/unsupervised training methods.
Taking the partial derivative of the error of the network to each of its weights, we can know the flow of error direction in the network. If we take the negative derivative and then proceed to add it to the weights, the error will decrease until it approaches local minima. Then we have to add a negative value to the weight or the reverse if the derivative is negative. Then apply them to each of the weights, starting from the output layer to hidden layer weights, hidden layer to input layer weights, this algorithm is called the **Back propagation algorithm**.

The training of the network involves feeding samples as input vectors, calculation of the error of the output layer, and then adjusting the weights and the bias of the network to minimize the error. The average of all the squared errors E for the outputs is computed to make the derivative simpler. After the error is computed, the weights can be updated

one by one. The weights W_j of the bias b are initialized randomly and then updated as follows(Parisi, Mariani, & Laborde, 2003).

$$W_j^{k+1} = W_j^k + \Delta W_j^k = W_j^k + \left(-\eta \frac{\partial E(t,p)}{\partial W_j^k}\right) \tag{3.12}$$

where, η is the learning parameter (which lies between 0 and 1)

k is the iteration step used to update the weights as usual in ANN

$E(x,p)$ is the error function

$$b^{k+1} = b^k + \Delta b^k = b^k + \left(-\eta \frac{\partial E(t,p)}{\partial b^k}\right) \tag{3.13}$$

Where b is bias

3.3 Single layer Artificial Neural Networks

A single-layer Artificial Neural Network is a type of ANN. FL ANN is a fast-learning single-layer Neural Network. FL ANN has own architecture (Chakraverty & Mall, 2017)

Architecture of the FL ANN

The architecture of the FL ANN models consists of two parts: the first one is the numerical transformation part and the second part is the learning part.

Numerical Transformation: In the numerical transformation part, each input data is expanded to several terms using orthogonal polynomials, namely, Chebyshev, Legendre, Hermite, etc. So orthogonal polynomials can be viewed as new input vectors.

learning algorithm: A learning algorithm is used for updating the network parameters and minimizing the error function. Here, the unsupervised error back-propagation algorithm is used to update the network parameters (weights) from the input layer to the output layer. The nonlinear tangent hyperbolic ($\tanh viz.\ (e^x - e^{-x})/(e^x + e^{-x})$) function is considered as the activation function. The gradient descent algorithm is used for learning, and the weights updated by taking a negative gradient at each iteration (Abraham, 2005; Chakraverty & Mall, 2017)

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 Formulation of the Problem

This chapter describe the general formulation of second-order non-linear differential equations using HeNN including with result and discussion.

4.1.1 Differential Equation

A general form of differential equation that represents second-order differential equations maybe written as (Mall & Chakraverty, 2016)

$$H(t, x(t), \bigtriangledown x(t), \bigtriangledown^2 x(t)) = 0, x \in \bar{D} \subset \mathbf{R} \tag{4.1}$$

where H is the function that defines the structure of the second-order differential equation, $x(t)$ denotes the solution, $\bigtriangledown$ is a differential operator, and $\bar{D}$ is the discretized domain over a finite set of points.

Types of Differential Equation Problems

Initial Value Problem(IVPs)

Let us consider a second-order IVPs may be written in general as (Chakraverty & Mall, 2017)

$$\frac{d^2x}{dt^2} = f(t, x, \frac{dx}{dt}), t \in [a, b] \tag{4.2}$$

subject to $x(a) = A$ and $x'(a) = A'$.

4.1.2 Deep Hermite Neural Network

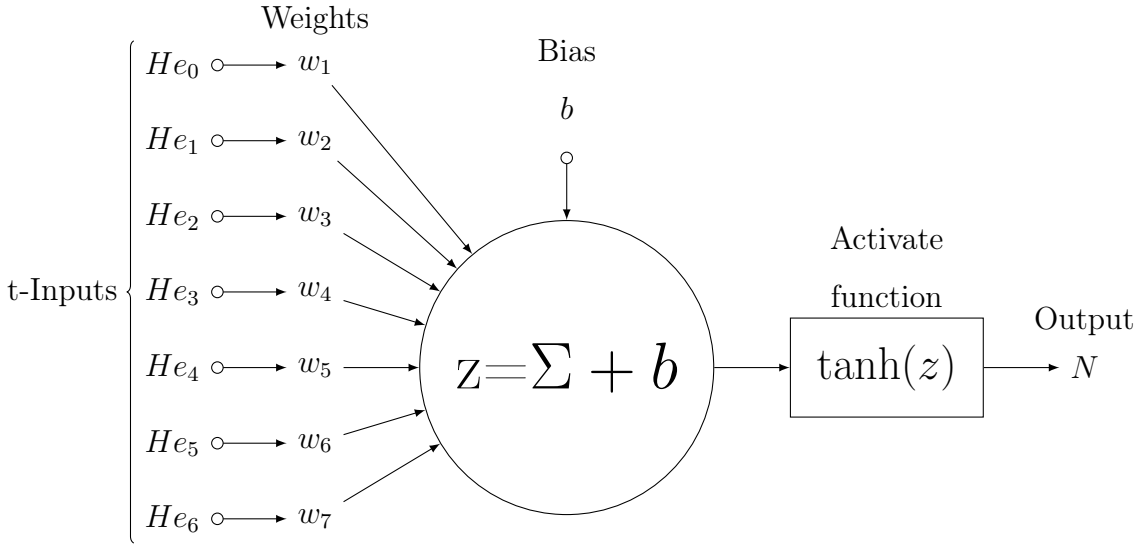

Figure 4.1: Architecture of HeNN

(Mall & Chakraverty, 2016)

$$\mathbf{Z} = \sum(\mathbf{weight} * \mathbf{input}) + \mathbf{bias}$$

Input Layer

In our network, we have one t input data.

Hidden Layer

In our network, there is no hidden layer (hidden layer is expanded using Hermite orthogonal polynomials).

Output Layer

In this network, we have single output layer N.

Weights(Parameters)

In our network, we have seven weights W_j from input to output layer.

Bias(Offset)

In our network, we have single bias node.

Figure 4.1 above depicts the structure of the HeNN model, which consists of a single input node, a single bias node, a single output node, and a Hermite orthogonal polynomial (first seven)-based functional expansion block.

The HeNN model is a single-layer neural model where each input data is expanded to several terms using Hermite polynomials. The first three Hermite polynomials may be written as (López & Temme, 1999).

$$\begin{cases} He_0(t) = 1 \\ He_1(t) = t \\ He_2(t) = t^2 - 1 \end{cases} \tag{4.3}$$

Higher-order Hermite polynomials may then be generated by the recursive formula

$$He_{n+1}(t) = t * He_n(t) - He'_n(t) \tag{4.4}$$

Transformation

Let $x_{He}(t,p)$ denote the trial solution of HeNN with adjustable parameters (weights) p. Then the above equation (4.1) general second order differential equation is transformed into the following problem:

$$H(t, x_{He}(t,p), \bigtriangledown x_{He}(t,p), \bigtriangledown^2 x_{He}(t,p)) = 0 \tag{4.5}$$

In general the HeNN trial solution $x_{He}(t,p)$ for the equation(4.5) with parameters (weights) p may be expressed as (Snehashish Chakraverty and Susmita Mall).

$$x_{He}(t,p) = A(t) + G(t, N(t,p)) \tag{4.6}$$

Where, the first term $A(t)$ does not contain adjustable parameters and satisfies only initial/boundary conditions, whereas the second term $G(t, N(t,p))$ contains the single output $N(t,p)$ of the HeNN with input t and adjustable parameters p.

Specifically, the HeNN trial solution for equation (4.2) is expressed as (Lagaris et al., 1998)

$$x_{He}(t,p) = A + A'(t-a) + (t-a)^2 N(t,p) \tag{4.7}$$

where

$N(t,p)$ is the network output of the HeNN with input t and parameter p. Thus, the trial

solution $x_{He}(t,p)$ satisfies the initial conditions.

As such, the $N(t,p)$ network output with input t and parameters (weights) p may be computed as (Mall & Chakraverty, 2016)

$$N(t,p) = tanh(z) = \begin{cases} 1, z > 0 \\ -1, z < 0 \end{cases} \tag{4.8}$$

where

$$z = \sum_{j=1}^{m} W_j He_{j-1}(t) + b \tag{4.9}$$

Here, t is input data W_j is the weight vectors of HeNN, b is a bias and $He_{j-1}(t)$ denotes the corresponding expanded input data with $j = 1, 2, 3, ..., m$.

The nonlinear tangent hyperbolic tanh(.) function is considered as the activation function.

Gradient Computation of the HeNN

For minimizing the error function $E(t,p)$, we differentiate $E(t,p)$ with respect to the network parameters. Thus, the gradient of the network output with respect to its input is computed as follows (Mall & Chakraverty, 2016).

$$\frac{dN(t,p)}{dt} = \sum_{j=1}^{m} \left\{ \left[\frac{(e^{W_j He_{j-1}(t)+b} + e^{-(W_j He_{j-1}(t)+b)})^2 - (e^{W_j He_{j-1}(t)+b} - e^{-(W_j He_{j-1}(t)+b)})^2}{(e^{W_j He_{j-1}(t)+b} + e^{-(W_j He_{j-1}(t)+b)})^2} \right] W_j He'_{j-1}(t)$$

Similarly, the second derivative of $N(t,p)$ is computed as (Mall & Chakraverty, 2016)

$$\frac{d^2N(t,p)}{dt^2} = \sum_{j=1}^{m} \left[\left\{ 2 \left(\frac{e^{W_j He_{j-1}(t)+b} - e^{-(W_j He_{j-1}(t)+b)}}{e^{W_j He_{j-1}(t)+b} + e^{-(W_j He_{j-1}(t)+b)}} \right)^3 - 2 \left(\frac{e^{W_j He_{j-1}(t)+b} - e^{-(w_j He_{j-1}(t)+b)}}{e^{W_j He_{j-1}(t)+b} + e^{-(W_j He_{j-1}(t)+b)}} \right) \right\} (W_j$$

$$+ \sum_{j=1}^{m} \left[\left\{ 1 - \left(\frac{e^{W_j He_{j-1}(t)+b} - e^{-(W_j He_{j-1}(t)+b)}}{e^{w_j He_{j-1}(t)+b} + e^{-(W_j He_{j-1}(t)+b)}} \right)^2 \right\} (W_j He''_{j-1}(t)) \right]$$

where W_j denote weights and $He'_{j-1}(t), He''_{j-1}(t)$ denote first and second derivatives of Hermite polynomials and b bias.

Let $N_\delta = dN/dt$ denote the derivative of the network output with respect to input t. The derivative of $N(t,p)$ and N_δ with respect to other parameters (weights) may be formulated as

$$\frac{\partial N}{\partial w_j} = \sum_{j=1}^{m} \left[1 - (\tanh(z))^2\right] (He_{j-1}(t)) \tag{4.10}$$

$$\frac{\partial N}{\partial b} = \sum_{j=1}^{m} \left[1 - (\tanh(z))^2\right] \tag{4.11}$$

$$\frac{\partial N_\delta}{\partial w_j} = \left[(\tanh(z))''(He_{j-1}(t))(w_j He'_{j-1}(t))\right] + \left[(\tanh(z))'(He'_{j-1}(t))\right] \tag{4.12}$$

After getting all the derivatives, we can find out the gradient of error. Using the unsupervised error back-propagation learning algorithm.

From equation (4.7), we have (by differentiating)

$$\frac{dx_{He}(t,p)}{dt} = A' + 2(t-a)N(t,p) + (t-a)^2\frac{dN}{dt} \tag{4.13}$$

and

$$\frac{d^2x_{He}(t,p)}{dt^2} = 2N(t,p) + 4(t-a)\frac{dN}{dt} + (t-a)^2\frac{d^2N}{dt^2} \tag{4.14}$$

The error function to be minimized for the second-order ODE is found to be

$$E(t,p) = \sum_{i=1}^{h}\frac{1}{2}\left[\frac{d^2x_{He}(t_i,p)}{dt^2} - f\left(t_i, x_{He}(t_i,p), \frac{dx_{He}(t_i,p)}{dt}\right)\right]^2 \tag{4.15}$$

Updating weights and bias

To update the weights and bias we required to find the derivatives of the error function with respect to w_j and is written as

$$\frac{\partial E(t,p)}{\partial W_j} = \frac{\partial}{\partial W_j}\left(\sum_{i=1}^{h}\frac{1}{2}\left[\frac{d^2x_{He}(t_i,p)}{dt^2} - f\left(t_i, x_{He}(t_i,p), \frac{dx_{He}(t_i,p)}{dt}\right)\right]^2\right) \tag{4.16}$$

Now, the weights are updated by taking negative gradient at each iteration as (Zurada, 1994)

$$W_j^{k+1} = W_j^k + \Delta W_j^k = W_j^k + \left(-\eta\frac{\partial E(t,p)}{\partial W_j^k}\right) \tag{4.17}$$

And the bias are updated by taking negative gradient at each iteration as follows

$$b^{k+1} = b^k + \Delta b^k = b^k + \left(-\eta\frac{\partial E(t,p)}{\partial b^k}\right) \tag{4.18}$$

4.1.3 Algorithm of HeNN

Following the same procedure as (Dufera, 2021);

Step 1: Take discrete points $t^i \in [a,b]$, where $i = 1,2,\ldots,m$ and form an input vector $T = [t^1, t^2, ..., t^m]$.

Step 2: Using the Hermite polynomials expansion (4.4) to form a matrix

$$X = [H_0(T), \quad H_1(T), \quad \ldots, \quad H_n(T)].$$

Here, X is a matrix of size $n \times m$.

Step 3: Initialize parameters:

- Randomly generate the weight vector W, the size is $1 \times n$ and the entries are small.
- Set the bias to be zero. That is, $b = 0$.

Step 4: Forward propagation:

- $Z = WX + b$
- $N(T, W, b) = \tanh(Z)$

Step 5: Compute the cost and its gradient, using (4.15).

Step 6: Update the parameter using the method of gradient descent or its family. In this thesis we use the Adam, adaptive Moment.

- Gradient descent
- Adam, adaptive Moment:(Kingma & Ba, 2014)
  - initialize $W_0, b_0, m_0 \longleftarrow 0, v_0 \longleftarrow 0, t \longleftarrow 0$
  - while W_t, b_t not converged

$$t \longleftarrow t + 1 \tag{4.19}$$

$$g_{tW} \longleftarrow \bigtriangledown_W f_t(W_t - 1) \tag{4.20}$$

$$g_{tb} \longleftarrow \bigtriangledown_b f_t(b_t - 1) \tag{4.21}$$

$$m_{tW} \longleftarrow \beta_1 m_{tW-1} + (1 - \beta_1) g_{tW} \tag{4.22}$$

$$m_{tb} \longleftarrow \beta_1 m_{tb-1} + (1 - \beta_1) g_{tb} \tag{4.23}$$

$$v_{tW} \longleftarrow \beta_2 v_{tW-1} + (1 - \beta_2)(g_{tW})^2 \tag{4.24}$$

$$v_{tb} \longleftarrow \beta_2 v_{tb-1} + (1 - \beta_2)(g_{tb})^2 \tag{4.25}$$

bias correction

$$\bar{m_{tW}} \longleftarrow \frac{m_{tW}}{1 - \beta_1^t}, \bar{V_{tW}} \longleftarrow \frac{v_{tW}}{1 - \beta_2^t} \tag{4.26}$$

$$\bar{m_{tb}} \longleftarrow \frac{m_{tb}}{1-\beta_1^t}, \bar{V_{tb}} \longleftarrow \frac{v_{tb}}{1-\beta_2^t} \tag{4.27}$$

update

$$W_t \longleftarrow W_{t-1} - \alpha \frac{\bar{m_{tW}}}{\sqrt{\bar{v_{tW}}} + \epsilon} \tag{4.28}$$

$$b_t \longleftarrow b_{t-1} - \alpha \frac{\bar{m_{tb}}}{\sqrt{\bar{v_{tb}}} + \epsilon} \tag{4.29}$$

where $f(W, b)$ f is the loss function to be optimized given the parameter(weights) W and bias b.

$g - t$: g is the gradient at step t.

$m - t$: m is the exponential moving average(EMA) of $(g - t)$.

$v - t$: v is the exponential moving average(EMA) of $(g - t)^2$.

β_1, β_2: These are the hyperparameters used in the moving averages of $g - t$ and $(g - t)^2$, most commonly set to 0.9 and 0.999 respectively.

α: The learning rate.

ϵ: A very small number, used to avoid the denominator $= 0$ scenarios. So what's happening here is we have a loss function $f(W, b)$ that is to be minimized by finding the optimal values of W and b such that the loss function achieves its minima. In order to do that, we use gradient descent where we basically compute the gradients of the function and keep subtracting them from the weights and bias until we reach the minimum (Swapnarekha, Behera, Roy, Das, & Nayak, 2021).

$$W_j^{k+1} = W_j^k + \Delta W_j^k = W_j^k + \left(-\eta \frac{\partial E(t,p)}{\partial W_j^k} \right) \tag{4.30}$$

$$b^{k+1} = b^k + \Delta b^k = b^k + \left(-\eta \frac{\partial E(t,p)}{\partial b^k} \right) \tag{4.31}$$

Example 1. Here we consider a second linear order differential equation given by;

$$x'' + x = 0,$$

with initial conditions $x(0) = 2$, $x'(0) = 3$,. The analytical solution is given by

$$x(t) = 2cos(t) + 3sin(t)$$

Solution

In this case, the HeNN trial solution is represented as

$$x_{He}(t, p) = t^2 N + 3t + 2 \tag{4.32}$$

In this example, the network has been trained with 10000 epochs in the domain from t = 0 to t =3 for computing the results. We have considered seven weights for the first seven Hermite polynomials for the problem. Here, t denotes the periodic time, and x(t) is the displacement at time t. The comparison between the exact value and HeNN are depicted in Figure 4.2. The approximate results of the proposed HeNN method almost coincide with the exact value. The results are shown in Table 4.1

Table 4.1: Results of Exact and HeNN solution

Times	HeNN_solution	Exact_sulution	Absulte error
0.00	2.000000	2.000000	0.000000
0.25	2.680354	2.680037	0.000317
0.50	3.192250	3.193442	0.001192
0.75	3.502932	3.508294	0.005362
1.00	3.594486	3.605018	0.010532
1.25	3.463259	3.477599	0.014340
1.50	3.118797	3.133959	0.015163
1.75	2.582635	2.595466	0.012831
2.00	1.887077	1.895599	0.008521
2.25	1.073865	1.077872	0.004007
2.50	0.192499	0.193129	0.000630
2.75	-0.702152	-0.703622	0.001470
3.00	-1.553318	-1.556625	0.003307

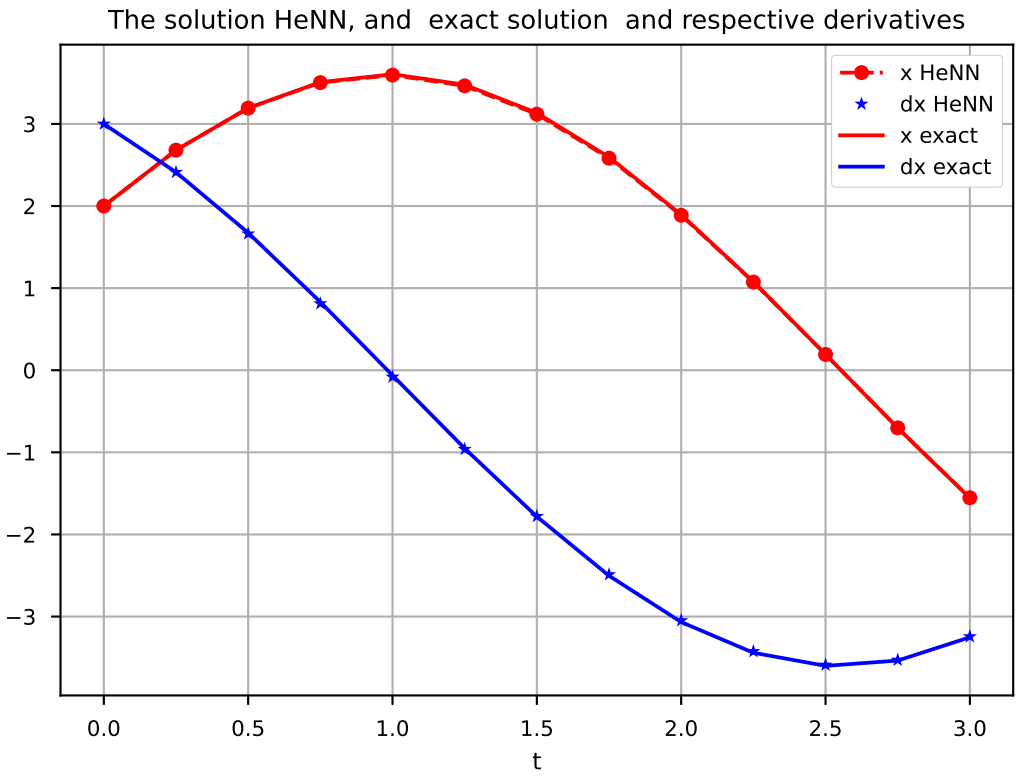

Figure 4.2: Comparison between the exact solution and HeNN solution and its derivatives in example1.

Example 2. Here we consider a non-linear second order differential equation given by;

$$x'' + (x')^2 = 0 \tag{4.33}$$

with initial conditions, $x(0) = 1, x'(0) = 1$ The analytical solution is given by:

$$x(t) = \ln(t + 1) + 1 \tag{4.34}$$

Solution

In this case, the HeNN trial solution is represented as

$$x_{He}(t, p) = t^2 N + t + 1 \tag{4.35}$$

In this example, the network has been trained with 15000 epochs in the domain from t = 0 to t =6 for computing the results. We have considered seven weights for the first seven Hermite polynomials for the problem. Here, t denotes the periodic time, and x(t) is the displacement at time t. The comparison between the exact value and HeNN are depicted in Figure 4.3. The approximate results of the proposed HeNN method almost coincide with the exact value. The results are shown in Table 4.2

Table 4.2: Results of Exact and HeNN solution

Time	HeNN_solution	exact_sulution	Absulte error
0.000	1.000000	1.000000	0.000000
0.375	1.326260	1.318454	0.007806
0.750	1.571360	1.559616	0.011744
1.125	1.757407	1.753772	0.003635
1.500	1.903049	1.916291	0.013242
1.875	2.023247	2.056053	0.032806
2.250	2.129215	2.178655	0.049440
2.625	2.228534	2.287854	0.059320
3.000	2.325425	2.386294	0.060869
3.375	2.421195	2.475907	0.054711
3.750	2.514844	2.558145	0.043300
4.125	2.603826	2.634131	0.030304
4.500	2.684963	2.704748	0.019785
4.875	2.755499	2.770706	0.015207
5.250	2.814298	2.832581	0.018283
5.625	2.863156	2.890850	0.027694
6.000	2.908234	2.945910	0.037676

4.2 Implementation

Van der Pol-Duffing oscillator equation

The Van der Pol-Duffing oscillator equation is governed by a second order nonlinear differential equation IVPs

$$\frac{d^2x}{dt^2} = \mu(1-x^2)\frac{dx}{dt} - \alpha x - \beta x^3 + F\cos\omega t \tag{4.36}$$

with initial conditions $x(0) = A, x'(0) = B$

where x stands for displacement, μ is the damping parameter, ω and F denote the excitation amplitude and frequency of the periodic force respectively and t is the periodic time, β is known as phase non-linearity parameter.

Now, we can formulate the equation (4.36) in to HeNN trial solution using equation (4.7).

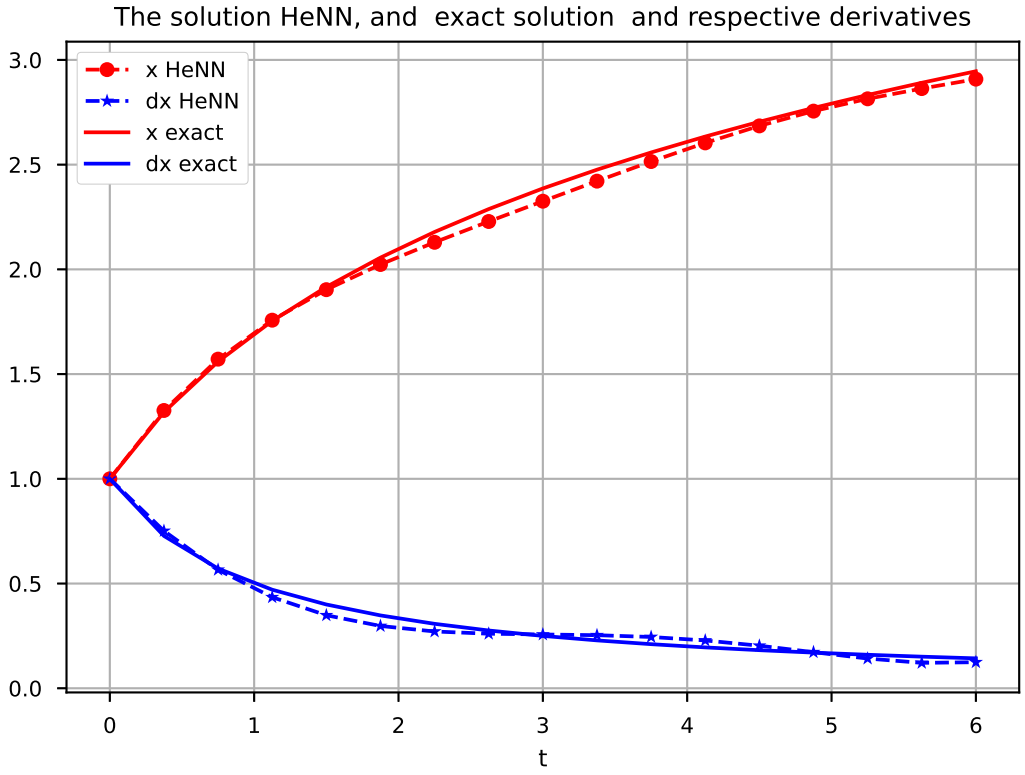

Figure 4.3: Comparison between the exact value and HeNN and its derivatives in example2

So, the HeNN trial solution is formulated as

$$x_{He}(t,p) = A + Bt + t^2 N(t,p) \tag{4.37}$$

where $N(t,p)$ is the output of the HeNN with input t and parameter p. The trial solution $x_{He}(t,p)$ satisfies the initial conditions.

Here,

$$N(t,p) = \tanh(z)$$

where, $z = \sum_{j=1}^{m} W_j He_{j-1}(t) + b$ and W_j the weight vectors of HeNN, $He_{j-1}(t)$ denotes the corresponding expanded input data, and b bias.

From equation (4.37), we have (by differentiating)

$$\frac{dx_{He}(t,p)}{dt} = B + 2tN(t,p) + t^2\frac{dN}{dt} \tag{4.38}$$

and

$$\frac{d^2x_{He}(t,p)}{dt^2} = 2N(t,p) + 4t\frac{dN}{dt} + t^2\frac{d^2N}{dt^2} \tag{4.39}$$

The error function to be minimized for the equation (4.16) is found to be

$$E(t,p) = \sum_{i=1}^{h} \frac{1}{2}\left[\frac{d^2x_{He}(t_i,p)}{dt^2} - \left(\mu(1-x_{He}^2)\frac{dx_{He}}{dt} - \alpha x_{He} - \beta x_{He}^3 + F\cos\omega t\right)\right]^2 \tag{4.40}$$

Example 3. We have taken the Van der Pol-Duffing oscillator equation as (Njah and Vincent, 2008)

$$\frac{d^2x}{dt^2} = 0.1(1-x^2)\frac{dx}{dt} - 0.5x - 0.5x^3 + 0.5\cos 0.79t \tag{4.41}$$

with initial conditions $x(0) = 0, x'(0) = 0$.

The HeNN trial solution may be written as

$$x_{He}(t,p) = t^2 N(t,p)$$

In this application problem, we again consider 10000 epochs from $t = 0$ to $t = 3sec$ with the first seven Hermite polynomials. We have considered seven weights for the first seven Hermite polynomials for the problem. In the Figure 4.4, approximate results by the HeNN method are very close to the results obtained by RK4. Figures 4.5 depict the phase-plane plots obtained by HeNN and RK4, respectively. The results are shown in Table 4.3.

Result's

Table 4.3: Approximation results of RK4 and HeNN

Times	HeNN	RK4
0.000000	0.000000	0.000000
0.333333	0.027537	0.027799
0.666667	0.109579	0.108952
1.000000	0.237819	0.234902
1.333333	0.394865	0.390101
1.666667	0.556073	0.551824
2.000000	0.692339	0.690829
2.333333	0.773868	0.775046
2.666667	0.774918	0.777155
3.000000	0.679484	0.682386

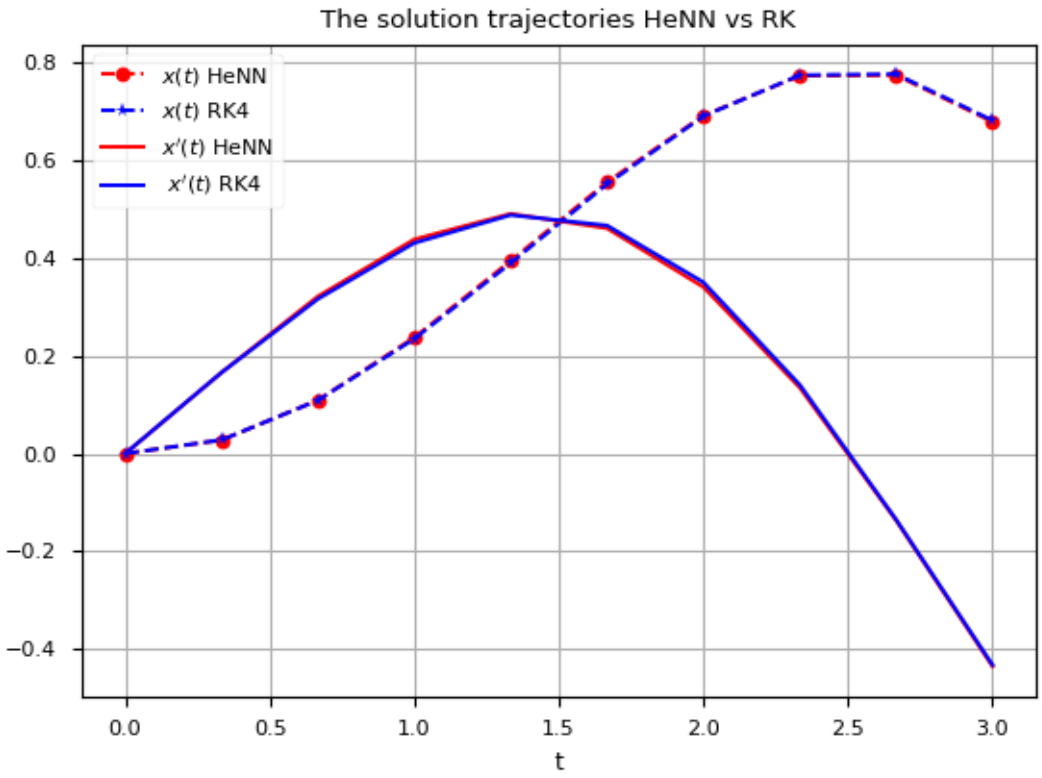

Figure 4.4: Comparison between the RK4 and HeNN results in example 3.

4.3 Discussion

A single-layer neural network represents the most simple form of a neural network. It has only one layer of input nodes that send weighted inputs to a subsequent layer of receiving nodes, or in some cases, one receiving node. It can also as part of a class of feedforward neural networks, where information only travels in one direction, through the inputs, to the output.

We tried to approximate the solution of a non-linear second-order differential equation(.i.e, a non-linear second-order initial value problem) using Single-layer Hermite neural networks. One linear, one non-linear second order differential equation and one application example Van der pul-duffing oscillator equation showed to know the efficiency of the proposed method. Generally, the approximation of HeNN is in excellent agreement with the exact value and with the numerical techniques (RK4).

Conclusions

In this study, we saw the definition, types, and components of artificial neural networks. Single-layer Functional Link Artificial Neural Network (FLANN) is one type of ANN. Single-layer FLANN has no hidden layer. Its input pattern dimensions are expanded by (Hermite, Chebyshev, or Legendre) orthogonal polynomials. But, a single layer Hermite neural networks used. Its computations become efficient because the procedure only requires to be input and output layers. A Feedforward neural networks model and an unsupervised version of error back propagation used to minimize the error function and to update the network parameters without optimization techniques. The weights from the input to the output layer initialized randomly and tangent hyperbolic activation functions (i.e tanh) considered from the input layer to the output layer. Two examples and one implementation (Van der Pol–Duffing oscillator equation) took to show the efficiency and power of this developed model. All graphs plotted using python code.

Recommendations

Here, this thesis studied on Single Layer artificial neural networks. Specifically, Hermite Neural Networks(HeNN) used and its application on non-linear second order differential equation with some examples took to show the efficiency of single layer ANN and with its plotted graphs. Excellent agreement between the approximated HeNN solution and exact solution, HeNN solution and numerical solution. One may go through from the proposed method(HeNN) and other numerical techniques, which approximation is more efficient to solve a non-linear ordinary differential equation. This proposed methods efficient to solve Ordinary Differential Equation. But, the error function increase in some iterations and decreased in some iteration however those iterations not known. So, its trial solution needs revision. Also, one can go through application of the proposed or HeNN method on ODE systems and PDE.

ACKNOWLEDGEMENT

This research project is funded by Adama Science and Technology University under the grant number ASTU/SM-R/267/21, 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).

Abraham, A. (2005). Artificial neural networks. *Handbook of measuring system design*.

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.

Anderson, J. A. (1995). *An introduction to neural networks*. MIT press.

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.

Dufera, T. T. (2021). Deep neural network for system of ordinary differential equations: Vectorized algorithm and simulation. *Machine Learning with Applications*, *5*, 100058. Retrieved from `https://www.sciencedirect.com/science/article/pii/S2666827021000293` doi: https://doi.org/10.1016/j.mlwa.2021.100058

Eberhart, R., Simpson, P., & Dobbins, R. (1992). *Introduction to artificial neural systems*. West publishing Co., NY.

Fitch, F. B. (1944). Mcculloch warren s. and pitts walter. a logical calculus of the ideas immanent in nervous activity. bulletin of mathematical biophysics, vol. 5, pp. 115–133. *Journal of Symbolic Logic*, *9*(2).

Galinkin, E. (2020). Malicious network traffic detection via deep learning: An information theoretic view. *arXiv preprint arXiv:2009.07753*.

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.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

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.

López, J. L., & Temme, N. M. (1999). Approximations of orthogonal polynomials in terms of hermite polynomials. *Methods and Applications of Analysis*, *6*, 131–146.

Mall, S., & Chakraverty, S. (2016). Hermite functional link neural network for solving the van der pol–duffing oscillator equation. *Neural computation*, *28*(8), 1574–1598.

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.

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

Pao, Y.-H., & Phillips, S. M. (1995). The functional link net and learning optimal control. *Neurocomputing*, *9*(2), 149–164.

Parisi, D. R., Mariani, M. C., & Laborde, M. A. (2003). Solving differential equations with unsupervised neural networks. *Chemical Engineering and Processing: Process Intensification*, *42*(8-9), 715–721.

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

Swapnarekha, H., Behera, H. S., Roy, D., Das, S., & Nayak, J. (2021). Competitive deep learning methods for covid-19 detection using x-ray images. *Journal of The Institution of Engineers (India): Series B*, 1–14.

Uhrig, R. E. (1995). Introduction to artificial neural networks. In *Proceedings of iecon'95-21st annual conference on ieee industrial electronics* (Vol. 1, pp. 33–37).

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

Xia Wang, J. (2020). *Neural network applied to monte carlo method based probabilistic power flow* (Unpublished master's thesis). Universitat Politècnica de Catalunya.

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

Zurada, J. (1994). End effector target position learning using feedforward with error back-propagation and recurrent neural networks. In *Proceedings of 1994 ieee international conference on neural networks (icnn'94)* (Vol. 4, pp. 2633–2638).