A DEEP ARTIFICIAL NEURAL NETWORK FOR SOLVING ADVECTION EQUATION IN HALF SUPER-ELLIPSE GEOMETRY

Kasahun Dirirsa

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 Equation)

Office of Graduate Studies
Adama Science and Technology University

December, 2022
Adama, Ethiopia

A DEEP ARTIFICIAL NEURAL NETWORK FOR SOLVING ADVECTION EQUATION IN HALF SUPER-ELLIPSE GEOMETRY

Kasahun Dirirsa

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 Equation)

Office of Graduate Studies
Adama Science and Technology University

December, 2022
Adama, Ethiopia

Declaration

I hereby that this Master Thesis entitled ***" A deep artificial neural network for solving advection equation in half super-ellipse geometry"*** is my original work. That is, it has not been submitted for the award of any academic degree, diploma or certificate in any university. All sources of material 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 ***"A deep artificial neural network for solving advection equation in half super-ellipse geometry"*** prepared under our guidance by **Kasahun Dirirsa** 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 ***"A deep artificial neural network for solving advection equation in half super-ellipse geometry"*** and developed by **Kasahun Dirirsa** , 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

Approval of Board of Reviewers

We, the undersigned, members of the Board of Examiners of the thesis by **Kasahun Dirirsa** have read and evaluated the thesis entitled ***"A deep artificial neural network for solving advection equation in half super-ellipse geometry"*** and amined 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

Chairperson	Signature	Date
Internal Examiner	Signature	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 and foremost, I would want to thank the almighty ***God*** *for helping me get here. He also provided me strength to carry out this righteous action consistently and patience, forgiveness, and charity. Second, I want to sincerely and heartily thank my advisor* ***Tamirat Temesgen (Ph.D.)*** *and co-advisor* ***Feyissa Kebede(Ph.D.)*** *for their guidance and support in helping me finish my thesis from beginning to conclusion. I greatly appreciate his expert comments and suggestions as well as his critical criticism. Thirdly, I want to thank my wife once more for her moral support and concern for my son.*

Contents

List of Figures

List of Acronyms

ANN $\Longrightarrow$ Artificial Neural Network

DNN $\Longrightarrow$ Deep Neural Network

FEM $\Longrightarrow$ Finite elements method

FDM $\Longrightarrow$ Finite differences method

FVM $\Longrightarrow$ Finite volume method

MCM $\Longrightarrow$ Monte carlo method

PDE $\Longrightarrow$ Partial Deferential Equation

RBF $\Longrightarrow$ Radial basis function

Abstract

In order to solve the advection equation on a half super elliptical geometry with challenging beginning and boundary conditions, this thesis applies deep artificial neural networks. For arbitrary deep artificial neural networks, we provide analytical formulas of the gradients of the cost function with respect to the network parameters as well as the gradient of the network itself with regard to the input. The approach is based on a solution ansatz that calls for gradient descent and feedforward neural networks.

Key words: Artificial neural network, Deep artificial neural network, Feed-forward, Back-propagation and Advection equation.

CHAPTER 1

Introduction

1.1 Background of the Study

In the natural sciences, partial differential equations are used to simulate a wide range of phenomena. The majority of partial differential equations encountered in real applications have in common the inability to be solved analytically, necessitating the use of various approximation approaches. For obtaining approximate solutions, mesh-based approaches like finite elements, finite differences, or finite volumes have historically dominated. These methods call for discretizing the computational domain of interest into a set of mesh points, where the solution is then approximated. These techniques have the benefit of being extremely effective for low-dimensional issues on regular geometries. The disadvantage is that meshing can be just as challenging for complex geometries as the numerical solution of the partial differential equation. Additionally, the solution can only be evaluated in the mesh points; any other point requires interpolation or another reconstruction technique.

Other techniques use a set of collocation points instead of a mesh to approximate the solution. Radial basis functions (RBF) and Monte Carlo methods are two examples of how the collocation points might be generated based on some distribution inside the domain of interest (MCM). The benefit is that using a hit-and-miss method, it is rather simple to construct collocation points inside the domain. As opposed to conventional mesh-based approaches, these methods have some disadvantages, such as numerical stability for RBF and inefficiency for MCM.

Deep artificial neural networks (DANNs) have played a major role in the deep learning revolution that has occurred over the past ten years. The idea of artificial neural networks (ANNs) is not new. They have been available since the 1940s and have a

number of uses (McCulloch & Pitts, 1943). Deep belief nets and better hardware resources, such as general purpose graphics processing units (GPGPUs), are two factors that have contributed to the success of deep learning during the past ten years. For further information, see (Hinton et al., 2006) and (Krizhevsky et al., 2012). In areas including image analysis, pattern identification, object detection, natural language processing, and self-driving cars, deep artificial neural network are increasingly often applied with amazing results. Simple feed-forward artificial neural network haven't yet achieved the same level of success as convolutional artificial neural network for image analysis or recurrent artificial neural network for natural language processing. Recent findings, however, demonstrate the immense potential of feedforward ANNs by enabling self-normalizing activation functions (Klambauer et al., 2017).

1.2 Super Ellipse

In this thesis we conceder the domain is a super-ellipse geometry. A super-elliptic, also known as a Lamé curve after Gabriel Lamé, is a closed curve resembling the ellipse, retaining the geometric features of semi-major axis and semi-minor axis, and symmetry about them, but a different overall shape. In the Cartesian coordinate system, the set of all points (x, y) on the curve satisfy the equation (Wikipedia contributors, 2022).

$$\left|\frac{x}{a}\right|^n + \left|\frac{y}{b}\right|^n = 1$$

where n, a and b are positive numbers. The curve is given by the parametric equations (with parameter t).

$$\begin{aligned} x(t) &= \pm a cos^{\frac{2}{n}}(t) \\ y(t) &= \pm b sin^{\frac{2}{n}}(t), \qquad 0 \leq t \leq \tfrac{\pi}{2} \end{aligned}$$

where each $\pm$ can be chosen separately so that each value of t gives four points on the curve.

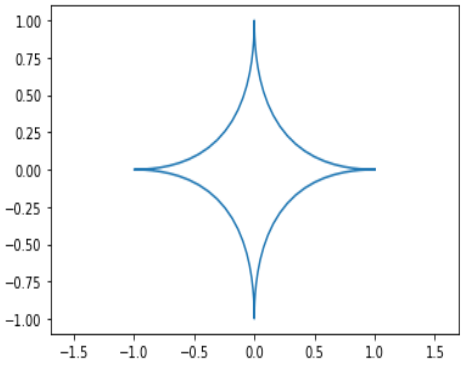

Figure 1.1: Super ellipse.

where $a = b = 1$ and $n = \frac{1}{2}$

1.3 Artificial Neural Network

An artificial neural network (ANN) is an information processing system that has certain performance characteristics in common with biological neural networks. Artificial neural networks have been developed as generalizations of mathematical models of human cognition or neural biology, based on the assumptions that:

- Information processing occurs at many simple connections called neurons.
- Signals are passed between neurons over connection links.
- Each connection link has an associated weight, which in a typical neural net, multiplies the signal transmitted.
- Each neuron applies an activation function to its net input to determine its output signal.

The basic component of an artificial neural network is artificial neuron like biological neuron in biological neural network. A biological neuron may be modeled artificially to perform computation and then the model is termed as artificial neuron.

A neuron is the basic processor or processing element in a neural network. Each neuron receives one or more input over these connections and produces only one output. Also this output is related to the state of the neuron and its activation function (Yadav et al., 2015).

Also this output is related to the state of the neuron and its activation function. This output may fan out to several other neurons in the network. The inputs are the

outputs i.e. activation's of the incoming neurons multiplied by the connection weights or synaptic weights. Each weight is associated with an input of a network.

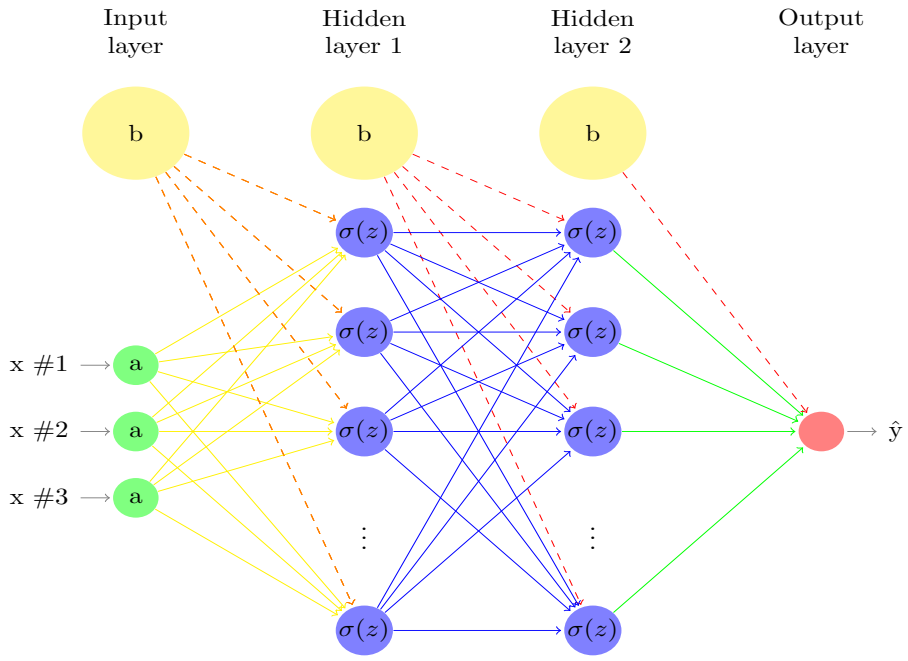

Figure 1.2: Schematic representation of ANN.

Then $a_1 ... a_n$ is the input data, b is bias and $\sigma(z)$ is activation function each pass between input and hidden layer is weight (w) y is output.

1.4 Activation Function

The function $\sigma(z)$ is called an activation function or transfer function. The purpose of activation functions is to introduce non-linearity into the network. Some of the activation functions are; sigmoid, tan hyperbolic and Rectifiable Linear Unit (Yadav et al., 2015). These function are;

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

1.5 Statement of the problem

In this study we will consider the advection equation on half super-ellipse domain. The advection equation is the partial differential equation that governs the motion of a conserved scalar field as it is advected by a known velocity vector field. It is derived using the scalar field's conservation law.

$$Lu = \bigtriangledown.(Vu) = f, \quad x \in \Omega \quad \Omega \subset R^2$$

where L is the differential operator, f a forcing function, V a matrix coefficient and Ω is on super-ellipse domain.

This study is intended to answer the following basic questions;

- Can we write an efficient algorithm for the proposed method ?
- Is the deep artificial neural network method efficient to solve advection equation on half super-ellipse domain ?
- Is the deep artificial neural network is efficient when we compare with the anther method ?

1.6 Objective of the study

1.6.1 General objective

The general objective is to compute solutions of advection equation using deep feed-forward artificial neural networks on half super-ellipse geometries.

1.6.2 Specific objectives

The specific objectives of this study is :

- to write an efficient algorithm for deep artificial neural network method.
- to solve the advection equation in half super-elliptic using deep artificial neural network method.
- to compar the efficiency of deep artificial neural network method.

1.7 Significance of the study

The result of this study will help :

- To increase the understanding of the scientific community to apply the deep artificial neural network method for advection equation in half super-ellipse geometry.
- To develop new method for half super-ellipse geometry and compute a solution which is may not solved in meshed based methods.
- To develop the algorithm of a deep artificial neural network method.

CHAPTER 2

Literature review

The beginning of neurocomputing is often taken to be the research article of McCulloch & Pitts (1943), which showed that even simple types of neural networks could, in principle, compute any arithmetic or logical function, was widely read and had great influence. Other researchers, principally Norbert Wiener and von Neumann, wrote a book and research paper (Von Neumann et al. (1951),Von Neumann (1956)) in which the suggestion was made that the research into the design of brain-like or brain-inspired computers might be interesting.

In the early 1980s many Neurocomputing researchers became bold enough to begin submitting proposals to explore the development of neuro-computers and of neural network applications. In the years (1983–1986 John Hopfield), an established physicist of worldwide reputation, had become interested in neural networks a few years earlier. Hopfield wrote two highly readable papers on neural networks (Hopfield (1982) and Hopfield (1984)) and these, together with his many lectures all over the world, persuaded hundreds of highly qualified scientists, mathematicians, and technologists to join the emerging field of neural networks.

In 1986, with the publication of the "PDP books" (Parallel Distributed Processing, Volumes I and II, edited by Rumelhart and McClelland et al. (1986)), the field exploded. In 1987, the first open conference on neural networks in modern times, the IEEE International Conference on Neural Networks was held in San Diego, and the International Neural Network Society (INNS) was formed. In 1988 the INNS journal Neural Networks was founded, followed by Neural Computation in 1989 and the IEEE Transactions on Neural Networks in 1990.

The traditional network weight training generally uses gradient descent method (Krogh & Hertz, 1991); these algorithms are easy to fall into local optimum but can not reach the global optimum (Sutton, 1986). The learning rules of ANN, the activation functions can be evolved by selecting among some popular nonlinear functions such as the Heaviside, sigmoidal, and Gaussian functions (Alvarez, 2002).The learning rate and the momentum factor of the BP algorithm can be evolved and learning rules can also be evolved to generate new learning rules (Kim et al., 1996). Evolution algorithms (EAs) are also used to select proper input variables for neural networks from a raw data space of a large dimension, that is, to evolve input features (Guo & Uhrig, 1992).

EAs are mainly used for training feed-forward neural networks, such as multilayer perceptron radial basis function network (Yao, 1999), (Harpham et al., 2004). Some researchers used it in recurrent neural network. With the development of EAs, many new EAs are used for evolving the ANNs, such as multi-objective evolutionary algorithm, parallel genetic algorithm (Yen, 2006). The Evolutionary artificial neural networks (Ding et al., 2013).

By Berg & Nyström (2018), a method for solving advection and diffusion type partial differential equations in complex geometries using deep feedforward artificial neural networks. Yu et al. (2017) propose a deep learning based method, the deep ritz method, for numerically solving variational problems, particularly the ones that arise from partial differential equations. The deep ritz method is naturally nonlinear, naturally adaptive and has the potential to work in rather high dimensions. Han et al. (2018) it introduced a deep learning based approach that can handle general high dimensional parabolic PDEs. They can use a stochastic gradient descent type (SGD) algorithm to optimize the parameter just as in the standard training of deep neural networks. Panghal & Kumar (2021) introduced a fast converging neural network-based approach for solution of ordinary and partial differential equations. It uses the extreme learning machine algorithm for calculating the neural network parameters so as to make it satisfy the differential equation and associated boundary conditions. Various ordinary and partial differential equations are treated using this technique, and accuracy and convergence aspects of the procedure are discussed.

A deep neural network based method for solving a class of elliptic partial differential equations (Han et al., 2020). To approximate the solution of the PDE with a deep neural network which is trained under the guidance of a probabilistic representation of

the PDE in the spirit of the Feynman-Kac formula. The notion of an Evolutional Deep Neural Network (EDNN) for the solution of partial differential equations (PDE). The parameters of the network are trained to represent the initial state of the system only, and are subsequently updated dynamically, without any further training, to provide an accurate prediction of the evolution of the PDE system (Du & Zaki, 2021).

The linear transport model by adopting the deep learning method, in particular the deep neural network (DNN) approach.here they adapt it to study kinetic models, in particular the linear transport model (Chen et al., 2021). Least-squares ReLU neural network method for solving the linear advection-reaction problem with discontinuous solution. The method is a discretization of an equivalent least-squares formulation in the set of neural network functions with the ReLU activation function (Cai et al., 2021).

CHAPTER 3

Research Methodology

3.1 Neural Network Architecture

A neural network is a parallel information processing system that has certain characteristics in common with certain brain functions. It is composed of neurons and synaptic weights and performs complex computations through a learning process. The following networks are for the training data with n features. We seen the following diagram the networks are connected with corresponding learning weights and biases.

3.1.1 A Simple Neural Network

The following diagram shows a simple neural network, with two inputs and one output.

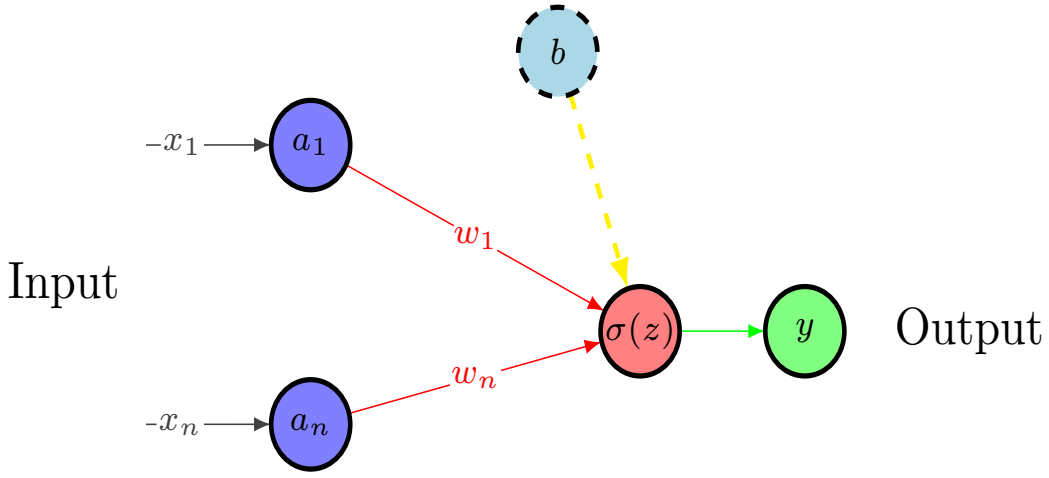

Figure 3.1: A Simple of ANN.

The inputs are passed unchanged into the first nodes:

$$a_1 = x_1$$

$$
\begin{aligned}
a_2 &= x_2 \\
\vdots &= \vdots \\
a_n &= x_n
\end{aligned}
$$

The weight in each pass are

$$w_1, w_2, w_3, \ldots w_n$$

Then each of these node values is multiplied by a weight, the ws and then a bias, b is added.

$$z = (w_1 a_1 + w_2 a_2 + w_3 a_3 + \ldots + w_n a_n) + b$$

The function $\sigma(z)$ will be chosen, it is called an activation function or transfer function.This result is then acted on by a function $\sigma(z)$ to give the output,

$$y = \sigma(z)$$

In neural networks we specify the activation function but the weights, the w's , and the bias, b, will be found numerically in order to best fit our forecast output with our given data.

$$
\begin{aligned}
z &= \textstyle\sum_{j=1}^{n} x_j w_j + b \\
z &= \textstyle\sum_{j=1}^{n} a_j w_j + b \\
y &= \sigma(z)
\end{aligned}
$$

The matrix form.

$$
\begin{aligned}
X &= x_1, x_2, \ldots, x_n \\
W &= w_1, w_2, \ldots, w_n \\
z &= \begin{bmatrix} w_1 & w_2 & \ldots & w_n \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} + b \\
z &= WX + b
\end{aligned}
$$

3.1.2 Multi Output Neural Networks

A following diagram show that the artificial neural network with multi inputs and tow outputs.

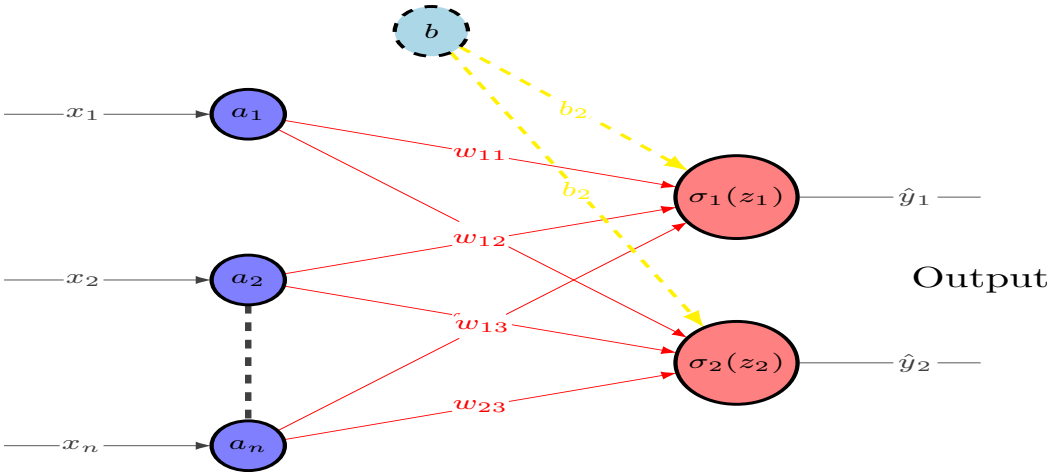

Figure 3.2: Multi Output of ANN.

The input data.

$$\begin{aligned} x_1 &= a_1 \\ x_2 &= a_2 \\ x_3 &= a_3 \\ \vdots &= \vdots \\ x_n &= a_n \end{aligned}$$

The pass between the input and output layer has a weight and bias.

$$W = \begin{bmatrix} w_{11} & w_{21} & w_{31} & \dots & w_{n1} \\ w_{12} & w_{22} & w_{32} & \dots & w_{n2} \end{bmatrix}, b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

Similar to the case of one output network, for each training sample $X = [x_1, x_2, x_3, \dots, x_n]$ and W, the network outputs $\hat{y}_1$ and $\hat{y}_2$ are obtained as follow;

$$\begin{aligned} z_1 &= \sum_{j=1}^{n} x_j w_{j1} + b_1 \\ z_2 &= \sum_{j=1}^{n} x_j w_{j2} + b_2 \end{aligned}$$

Then the output will be

$$\begin{aligned} \hat{y}_1 &= \sigma(z_1) \\ \hat{y}_2 &= \sigma(z_2) \end{aligned}$$

For the multi neural network output say $[\hat{y}_1, \hat{y}_2, \hat{y}_3, \ldots, \hat{y}_n]$ we have

$$z_i = \sum_{j=1}^{n} x_j w_{ij} + b_i$$

$$\hat{y}_i = \sigma(z_i)$$

3.1.3 Single Layer Neural Network

The following schematic diagram illustrates an architecture of a single layer (one hidden layer) neural network.

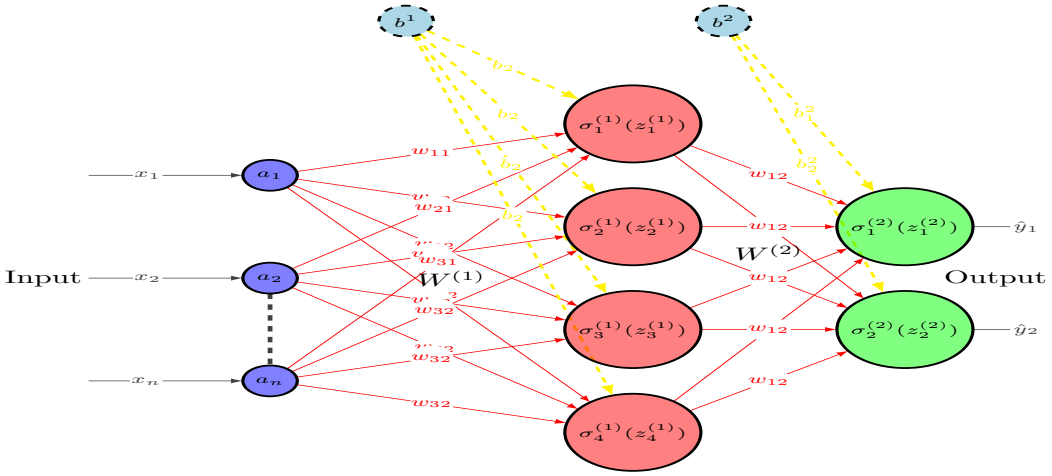

Figure 3.3: Single Layer of ANN.

For this particular model, the network output $\hat{y}$ is given by the following feed forward propagation. Given a sample or a training data X, from input to hidden layer. The input data as matrix form;

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_n \end{bmatrix}$$

The weighted and bias matrix;

$$W^{(1)} = \begin{bmatrix} w_{11}^{(1)} & w_{21}^{(1)} & w_{31}^{(1)} & \ldots w_{n1}^{(1)} \\ w_{12}^{(1)} & w_{22}^{(1)} & w_{32}^{(1)} & \ldots w_{n2}^{(1)} \\ w_{13}^{(1)} & w_{23}^{(1)} & w_{33}^{(1)} & \ldots w_{n3}^{(1)} \\ w_{14}^{(1)} & w_{24}^{(1)} & w_{34}^{(1)} & \ldots w_{n4}^{(1)} \end{bmatrix}, W^{(2)} = \begin{bmatrix} w_{11}^{(2)} & w_{21}^{(1)} & w_{31}^{(2)} & w_{41}^{(2)} \\ w_{12}^{(2)} & w_{22}^{(1)} & w_{32}^{(2)} & w_{42}^{(2)} \end{bmatrix}$$

$$b^{(1)} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \\ b_4^{(1)} \end{bmatrix}, b^{(2)} = \begin{bmatrix} b_1^{(2)} \\ b_2^{(2)} \end{bmatrix}$$

The result in each hidden layer are calculated as follow;

$$z_1^{(1)} = \begin{bmatrix} w_{11}^{(1)} & w_{21}^{(1)} & w_{31}^{(1)} & \dots w_{n1}^{(1)} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_n \end{bmatrix} + b_1^{(1)} \tag{3.1}$$

From (3.1) we get the general form;

$$\begin{bmatrix} z_1^{(1)} \\ z_2^{(1)} \\ z_3^{(1)} \\ z_4^{(1)} \end{bmatrix} := \begin{bmatrix} w_{11}^{(1)} & w_{21}^{(1)} & w_{31}^{(1)} & \dots w_{n1}^{(1)} \\ w_{12}^{(1)} & w_{22}^{(1)} & w_{32}^{(1)} & \dots w_{n2}^{(1)} \\ w_{13}^{(1)} & w_{23}^{(1)} & w_{33}^{(1)} & \dots w_{n3}^{(1)} \\ w_{14}^{(1)} & w_{24}^{(1)} & w_{34}^{(1)} & \dots w_{n4}^{(1)} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_n \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \\ b_4^{(1)} \end{bmatrix}$$

Therefore ;

$$\sigma_i^1(z_i^{(1)}) = \begin{bmatrix} \sigma_1^{(1)}(z_1^{(1)}) \\ \sigma_2^{(1)}(z_2^{(1)}) \\ \sigma_3^{(1)}(z_3^{(1)}) \\ \sigma_4^{(1)}(z_4^{(1)}) \end{bmatrix}$$

So that in this way we can get;

$$\begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \end{bmatrix} = \begin{bmatrix} w_{11}^{(2)} & w_{21}^{(1)} & w_{31}^{(2)} & w_{41}^{(2)} \\ w_{12}^{(2)} & w_{22}^{(1)} & w_{32}^{(2)} & w_{42}^{(2)} \end{bmatrix} \begin{bmatrix} \sigma_1^{(1)}(z_1^{(1)}) \\ \sigma_2^{(1)}(z_2^{(1)}) \\ \sigma_3^{(1)}(z_3^{(1)}) \\ \sigma_4^{(1)}(z_4^{(1)}) \end{bmatrix} + \begin{bmatrix} b_1^{(2)} \\ b_2^{(2)} \end{bmatrix}$$

Therefore;

$$\sigma_1^{(2)}(z_1^{(2)}) = \hat{y}_1$$
$$\sigma_2^{(2)}(z_2^{(2)}) = \hat{y}_2$$

3.1.4 Deep Artificial Neural Network

We take deep, fully connected feed-forward ANNs into consideration in this study. $L+1$ layers make up the ANN, where layer 0 serves as the input layer and layer L serves as the output layer. The hidden layers are $0 < l < L$.

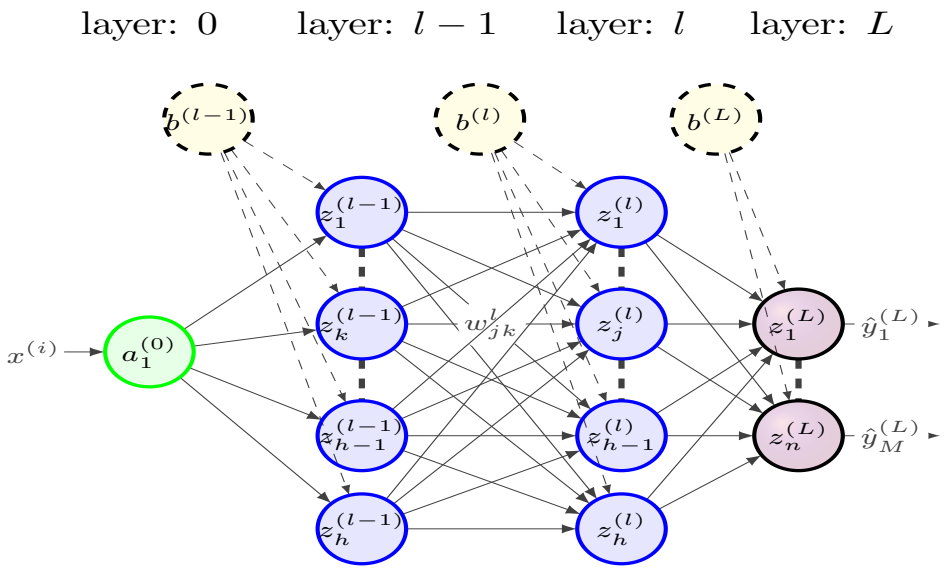

Figure 3.4: Schematic representation of a fully connected feedforward ANN.

Any activation function, such as sigmoids, rectified linear units, or hyperbolic tangents, may be used as the activation function in the hidden layers. In the buried layers, sigmoids will be used unless otherwise specified. The linear activation is going to be the output activation. The connections between neurons in later layers are represented by weighted matrices, and each neuron in the ANN is given a bias that includes the output neurons but excludes the input neurons. We'll use b_j^l to stand for neuron $j's$ bias 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 σ_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 Fig. 3.4 for a schematic representation of a fully connected feedforward ANN. A quantity that will be used extensively is the so called weighted input which is defined as

$$z_j^l = \sum_k w_{jk}^l \sigma_{l-1}(z_j^{l-1}) + b_j^l \tag{3.2}$$

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

$$z_j^l = w_{jk}^l y_k^{l-1} + b_j^l$$

where the output $y_k^{l-1} = \sigma_{l-1}\ z_k^{l-1}$ is the activation of the k 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

$$\sigma_0(z_j^0) = y_j^0 = x_j$$

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

$$z^l = W^l \sigma_{l-1}(z^{l-1}) + b^l = W^l y^{l-l} + b^l$$

where each element in the z^l and y^l vectors are given by z_j^l and y_j^l , respectively, and the activation function is applied elementwise.

3.2 Feed forward neural networks

Feed forward neural networks are the simplest form of artificial neural network. The feed forward neural network was the first and arguably simplest type of artificial network devised. In this network, the information moves in only one direction, forward, from the input nodes, through the hidden nodes (if any) and to the output nodes as shown in Figure 3.4. The elements of the matrix W^l are given by $W_{jk}^l = w_{jk}^l$. With the above definitions the feedforward for computing the output y^L , given the input x , is given by

$$z^1 = W^1 x + b^1 \tag{3.3}$$
$$z^2 = W^2 \sigma_1(z^1) + b^2 \tag{3.4}$$
$$\vdots \tag{3.5}$$
$$z^{L-1} = W^{L-1} \sigma_{L-2}(z^{L-2}) + b^{L-1} \tag{3.6}$$
$$z^L = W^L \sigma_{L-1}(z^{L-1}) + b^L \tag{3.7}$$
$$y^L = \sigma_L(z^L) \tag{3.8}$$

3.3 Back-propagation

Given some data $x = [x_1, ..., x_N]^T \in R^N$ and some outputs $y = [y_1, ..., y_M]^T \in R^M$ 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 $C = C(y, y^L) : R^N \longrightarrow R$ and compute

$$w^*, b^* = arg \min_{w.b} C(y, y^L) \tag{3.9}$$

The minimization problem (3.9) can be solved using a variety of optimization methods, both gradient based and gradient free. In this study 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 back-propagation. The main ingredient is the error of neuron j in layer l defined by

$$\delta_j^l = \frac{\partial C}{\partial z_j^l}$$

The standard back-propagation algorithm for computing the gradients of the cost function is then given by

$$\delta_j^L = \frac{\partial C}{\partial y_j^L}\sigma'(z_j^L) \tag{3.10}$$

$$\frac{\partial C}{\partial w_{jk}^l} = y_k^{l-1}\sigma_j^l \tag{3.11}$$

$$\delta_j^l = \sum_k w_{kj}^{l-1}\delta_k^{l-1}\sigma'(z_j^l) \tag{3.12}$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{3.13}$$

The δ-terms in (3.13) can be written in vectorial form as

$$\delta = \bigtriangledown_{y^L} C \odot \sigma_L'(z^L), \quad \delta^l = (W^{l+1})^T\delta^{l+1} \odot \sigma_l'(z^l)$$

where we use the notation $\bigtriangledown_{y^L} C = \left[\frac{\partial C}{\partial y_1^L}, ..., \frac{\partial C}{\partial y_M^L}\right]^T$ and $\odot$ denotes the Hadamard (componentwise) product. The notation $\odot C$ without a subscript will denote the vector of partial derivatives with respect to the input $x = [x_1, ..., x_N]^T$. 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 (Berg & Nyström, 2018).

3.4 ANN approximations to PDEs

In this section, we are interested in solving stationary Partial Differential equations of the form

$$Lu = f, x \in \Omega \tag{3.14}$$

$$Bu = g, x \in \Gamma \subset \partial\Omega \tag{3.15}$$

where L is a differential operator, f a forcing function, B a boundary operator, and g the boundary data. The domain of interest is $\Omega \subset R^N$, $\partial\Omega$ denotes its boundary, and Γ is the part of the boundary where boundary conditions should be imposed. In this study we focus on problems where L is the advection operator. The extension to higher order problems are conceptually straightforward based on the derivations outlined below (Berg & Nyström, 2018). We consider the ansatz $\hat{u} = \hat{u}\ (x; w, b)$ for u where (w, b) denotes the parameters of the underlying ANN. To determine the parameters we will use the cost function defined as the quadratic residual,

$$C = \frac{1}{2}||L\hat{u} - f||^2 = \frac{1}{2}\int_\Omega |L\hat{u} - f|^2 dx. \tag{3.16}$$

For further reference we need the gradient of (3.16) with respect to any network parameter. Let p denote any of w^l_{jk} or b^l_j (Hinton et al., 2006). Then,

$$\frac{\partial C}{\partial p} = (L\hat{u} - f) \odot \left(L\frac{\partial \hat{u}}{\partial p}\right) \tag{3.17}$$

We will use the collocation method (Lagaris et al., 1998) and therefore we discretize Ω and Γ into a sets of collocation points Ω_d and Γ_d , with $|\Omega_d| = N_d$ and $|\Gamma_d| = N_b$, respectively. In its discrete form the minimization problem (3.9) then becomes

$$w^*, b^* = arg\min_{w.b} \sum_{x_i \in \Omega_d} \frac{1}{2}\frac{1}{N_d}||L\hat{u}(x_i) - f(x_i)||^2 \tag{3.18}$$

subject to the constraints,

$$B\hat{u}(x_i) = g(x_i), \forall x_i \in \Gamma_d \tag{3.19}$$

There are many approaches to the minimization problem (3.18) subject to the constraints (3.19). One can for example use a constrained optimization procedure, Lagrange multipliers, or penalty formulations to include the constraints in the optimization.

Another approach is to design the ansatz $\hat{u}$ such that the constraints are automatically fulfilled, thus turning the constrained optimization problem into an unconstrained optimization problem. Unconstrained optimization problems can be more efficiently solved using gradient based optimization techniques. In the following we let B in (3.15) be the identity operator and we hence consider PDEs with Dirichlet boundary conditions. In this case our ansatz for the solution is

$$\hat{u}(x) = G(x) + D(x)y^L(x; w, b) \tag{3.20}$$

where $G = G(x)$ is a smooth extension of the boundary data g and $D = D(x)$ is a smooth distance function giving the distance for $x \in \Omega$ to Γ. Note that the form of the ansatz (3.20) ensures that $\hat{u}$ attains its boundary values at the boundary points. Moreover, G and D are precomputed using low-capacity ANNs for the given boundary data and geometry using only a small subset of the collocation points and do not add any extra complexity when minimizing (3.18). We call this approach unified as there are no other ingredients involved other than feedforward ANNs, and gradient based, unconstrained optimization methods to compute all of G, D, and y^L.

3.4.1 Extension of the boundary data

The ansatz (3.20) requires that G is globally defined, smooth and that

$$|G(x) - g(x)| < \epsilon, \forall x \in \Gamma \tag{3.21}$$

Other than these requirements, the exact form of G is not important. It is hence an ideal candidate for an ANN approximation. To compute G, we simply train an ANN to fit $g(x), \forall x \in \Gamma_d$. The quadratic cost function used is given by

$$C = \frac{1}{2}\frac{1}{N_b}\sum_{x_i \in \Gamma_d} ||G(x_i) - g(x_i)||^2 \tag{3.22}$$

It is sufficient that G has continuous derivatives up to the order of the differential operator L. However, since G will be computed using an ANN, it will be smooth. and in the course of calibration we compute the gradients using the standard backpropagation (3.13). Note that in general we have $N_b << N_d$ and the time it takes to compute G is of lower order.

3.4.2 Smooth distance function computation

To compute the smooth distance function D, we start by computing a non-smooth distance function, d, and approximate it using a low-capacity ANN. For each point, x, we define d as the minimum distance to a boundary point where a boundary condition should be imposed. That is,

$$d(x) = min_{x_b \in \Gamma} ||x - x_b|| \tag{3.23}$$

Again, the exact form of d (and D) is not important other than that D is smooth and

$$|D(x)| < \epsilon, \forall x \in \Gamma \tag{3.24}$$

We can use a small subset $w_d \subset \Omega_d$, with $|w_d| = n_d << N_d$ to compute d. Once d has been computed and normalized, we fit another ANN and train it using the cost function,

$$C = \frac{1}{2} \frac{1}{n_d + N_b} \sum_{x_i \in w_d \cup \Gamma_d} ||D(x_i) - d(x_i)||^2 \tag{3.25}$$

Note that we train the ANN over the points $x_i \in w_d \cup \Gamma_d$, with $d(x_i) = \ 0 \ \forall x_i \in \Gamma_d$. Since the ANN defining the smooth distance function can be taken to be a single hidden layer, low-capacity ANN with $n_d + N_b << N_d$, the total time to compute both d and D is of lower order.

3.4.3 Gradient computations

When the differential operator L acts on the ansatz $\hat{u}$ in (3.20) , we need, as $\Omega \subset R^N$,to compute the partial derivatives of the ANNs with respect to the spatial variables $(x_1, ..., x_N)$. Also, when using gradient based optimization we need to compute the gradients of the quadratic residual cost function (3.16), and also the ANNs, with respect to the network parameters.

In the single hidden layer case, all gradients can be computed in closed analytical form as shown in (Lagaris et al., 1998). For deep ANNs, however, we need to modify the feed-forward (3.9) and back-propagation (3.13) algorithms to compute the gradients with respect to $(x_1, ..., x_N)$ and network parameters, respectively.

Note that the minimization problem in (3.18) involves the collocation points (x_i). In the following, and throughout the paper, we will, with a slight abuse of notation, by

$\frac{\partial}{\partial x_i}$ always denote the derivative with respect to the spatial coordinate x_i , not with respect to collocation point x_i

3.4.4 Algorithms

For advection problems we have a PDE of the form

$$Lu = \bigtriangledown.(V\hat{u}) = f \tag{3.26}$$

for some (non-linear) matrix coefficient V, when the advection operator L acts on $\hat{u}$ we need to compute the gradient of the ANN with respect to the input. Following the same procedure as (Berg & Nyström, 2018).

Step 1: The given some data $x = [x_1, x_2, ...x_N]^T \in R^N$ and some target output $y = [y_1, y_2, \dots y_M]^T \in R^M$ we wish to chose our wights and biases such that $y^L(x; w, b)$ is the good approximation of $y(x)$.

Step 2: With (3.2) feed-foreword the gradients can be computed by taking partial derivatives of (3.8). We get, in component form,

$$\begin{aligned}
\frac{\partial z_j^1}{\partial x_i} &= w_{ji}^1 \\
\frac{\partial z_j^2}{\partial x_i} &= \sum_k w_{jk}^2 \sigma_1'(z_k^1) \frac{\partial z_k^1}{\partial x_i} \\
&\vdots \\
\frac{\partial z_j^{L-1}}{\partial x_i} &= \sum_k w_{jk}^{L-1} \sigma_{L-2}'(z_k^{L-2}) \frac{\partial z_k^{L-2}}{\partial x_i} \\
\frac{\partial z_j^L}{\partial x_i} &= \sum_k w_{jk}^{L-1} \sigma_{L-2}'(z_k^{L-2}) \frac{\partial z_k^{L-2}}{\partial x_i} \\
\frac{\partial y_j^L}{\partial x_i} &= \sigma(z_j^L) \frac{\partial z_j^L}{\partial x_i}
\end{aligned} \tag{3.27}$$

Note the slight abuse of subscript notation in order to avoid having too many subscripts. Note also that (3.27) defines a new ANN with the same weights, but no biases, as the original ANN with modified activation functions. A convenient vectorial

form is obtained by dropping the subscripts and re-writing (3.27) as

$$\begin{aligned}
\frac{\partial z^1}{\partial x_i} &= W^1 e_i \\
\frac{\partial z^2}{\partial x_i} &= W^2 \sigma_1'(z^1) \odot \frac{\partial z^1}{\partial x_i} \\
&\vdots \\
\frac{\partial z^{L-1}}{\partial x_i} &= W^{L-1} \sigma_{L-2}'(z^{L-2}) \odot \frac{\partial z^{L-2}}{\partial x_i} \\
\frac{\partial z^L}{\partial x_i} &= W^L \sigma_{L-1}'(z^{L-1}) \odot \frac{\partial z^{L-1}}{\partial x_i} \\
\frac{\partial y^L}{\partial x_i} &= \sigma_L'(z^L) \odot \frac{\partial z^L}{\partial x_i}
\end{aligned} \tag{3.28}$$

where e_i is the ith standard basis vector. To compute all gradients simultaneously, we define, for a vector v and vector-valued function f , the diagonal and Jacobian matrices.

$$\sum(v) = \begin{bmatrix} v_1 & 0 & \dots & 0 \\ 0 & v_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & v_N \end{bmatrix}, J(f) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_N} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_N} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f_{M-1}}{\partial x_1} & \dots & \frac{\partial f_{M-1}}{\partial x_{N-1}} & \frac{\partial f_{M-1}}{\partial x_N} \\ \frac{\partial f_M}{\partial x_1} & \dots & \frac{\partial f_M}{\partial x_{N-1}} & \frac{\partial f_M}{\partial x_N} \end{bmatrix} \tag{3.29}$$

Using these matrices, we can write all partial derivatives in the compact matrix form,

$$\begin{aligned}
J(z^1) &= W^1 I_{N\times N} \\
J(z^2) &= W^2 \sum(\sigma_1'(z^1)) J(z^1) \\
&\vdots \\
J(z^{L-1}) &= W^{L-1} \sum(\sigma_{L-2}'(z^{L-2})) J(z^{L-2}) \\
J(z^L) &= W^L \sum(\sigma_{L-1}'(z^{L-1})) J(z^{L-1}) \\
J(y^L) &= \sum(\sigma_L'(z^L)) J(z^L)
\end{aligned} \tag{3.30}$$

where $I_{N\times N}$ is the $N \times N$ identity matrix. All gradients of the ANN with respect to the input are contained in the rows of the Jacobian matrix $J(y^L)$ as

$$J(y^L) = \begin{bmatrix} \dots & (\bigtriangledown y_1^L)^T & \dots \\ \dots & \vdots & \dots \\ \dots & (\bigtriangledown y_M^L)^T & \dots \end{bmatrix} \tag{3.31}$$

Step 3: To solve the PDE (3.26), we need to modify the backpropagation (3.13) to include the gradients of the residual cost function (3.16) with respect to the network parameters. First, we need to compute

$$\frac{\partial y_m^L}{\partial w_{jk}^l}, \quad \frac{\partial y_m^L}{\partial b_j^l}, m = 1, \ldots, M \tag{3.32}$$

This can be done componentwise by using backpropagation with the identity as the cost function. That is, we let $C(y_m^L) = y_m^L$ and then (3.13) becomes

$$\begin{aligned} \delta_m^L &= \sigma_L'(z_m^L), \frac{\partial y_m^L}{\partial w_{jk}^l} = y_k^{l-1}\delta_j^l \\ \delta_j^l &= \sum_k w_{kj}^{l-1}\delta_k^{l-1}\sigma_l'(z_j^l), \frac{\partial y_m^L}{\partial b_j^l} = \delta_j^l \end{aligned} \tag{3.33}$$

The δ terms in (3.33) can be written in vectorial form as

$$\begin{aligned} \delta^L &= \sigma_L'(z^L) \\ \delta^l &= (W^{l+1})^T\delta^{l+1} \odot \sigma_l'(z^l) \end{aligned} \tag{3.34}$$

Secondly, we need to compute

$$\frac{\partial^2 y_m^L}{\partial x_i \partial w_{jk}^l}, \frac{\partial^2 y_m^L}{\partial x_i \partial b_j^l}, i = 1, \ldots, Nm = 1, \ldots, M \tag{3.35}$$

Step 4: This can be done by taking the partial derivative of (3.33) with respect to x_i we get

$$\begin{aligned} \frac{\delta_m^L}{\partial x_i} &= \sigma_L''(z_m^L)\frac{\partial z_m^L}{\partial x_i} \\ \frac{\partial \delta_j^l}{\partial x_i} &= \sum_k w_{kj}^{l+1}(\frac{\partial \delta_k^{l+1}}{\partial x_i}\sigma_l'(z_j^{l+1}) + \delta_k^{l+1}\sigma_l''(z_j^l)\frac{\partial z_j^l}{\partial x_i}) \\ \frac{\partial^2 y_m^L}{\partial x_i \partial w_{jk}^l} &= \frac{\partial y_k^{l-1}}{\partial x_i}\delta_j^l + y_k^{l-1}\frac{\partial \delta_j^l}{\partial x_i} \\ \frac{\partial^2 y_m^L}{\partial x_i \partial b_j^l} &= \frac{\partial \delta_j^l}{\partial x_i} \end{aligned} \tag{3.36}$$

The δ -terms in (3.36) can again be written in vectorial form as

$$\begin{aligned} \frac{\delta^L}{\partial x_i} &= \sigma_L''(z^L) \odot \frac{\partial z^L}{\partial x_i} \\ \frac{\partial \delta^l}{\partial x_i} &= (W^{l+1})^T (\frac{\partial \delta^{l+1}}{\partial x_i} \odot \sigma_l'(z^l) + \delta^{l+1} \odot \sigma_l''(z^l) \odot \frac{\partial z^l}{\partial x_i}) \end{aligned} \tag{3.37}$$

or in matrix form as

$$\begin{aligned} J(\delta^L) &= \sum(\sigma_L''(z^L)) J(z^L) \\ J(\delta^l) &= (W^{l+1})^T (\sum(\sigma_l'(z^l)) J(\delta^{l+1}) + \sum(\sigma''(z^l)) \sum(\delta^{l+1}) J(z^l)) \end{aligned} \tag{3.38}$$

The feed-forward and backpropagation algorithms (3.8), (3.27), (3.33), and (3.36), are sufficient to compute the solution to any first-order PDE, both linear and nonlinear.

CHAPTER 4

Results and Discussions

To verify the established theoretical result in this paper, we perform an experiment using the proposed method on the problem of the form given in the above. We use the loss function to minimize the errors.

Problem

We will solve a stationary advection equation with hard initial and boundary conditions:

$$\frac{\partial^2 u}{\partial x^2} + v\ \frac{\partial^2 u}{\partial y^2} = 0 \qquad\qquad y \in [-1, 1] \text{ and } x \in [0, 1]$$

Where v is a (constant) advection coefficients and $v = \frac{0.01}{\pi}$.
With the initial condition

$$u(0, y) = -sin(\pi y)$$

and the dirichlet boundary condition.

$$u(x, -1) = u(x, 1) = 0$$

The first argument to pde is 2-dimensional vector where the first component ($y[:, 0:1]$) is y-coordinate and the second component ($y[:, 1:]$) is the y-coordinate. The second argument is the network output, *i.e.*, the solution $u(x, y)$. The number 40 is the number of training residual points sampled inside the domain. 10000 points for testing the PDE residual.

Then we construct a function that spontaneously satisfies both the initial and the boundary conditions to transform the network output. In this case, $-sin(\pi y)$ is used. When x is equal to 0, the initial condition $-sin(\pi y)$ is recovered. When y is equal to

-1 or 1, the boundary condition $u(x,-1) = u(x,1) = 0$ is recovered. Hence the initial and boundary conditions are both hard conditions.

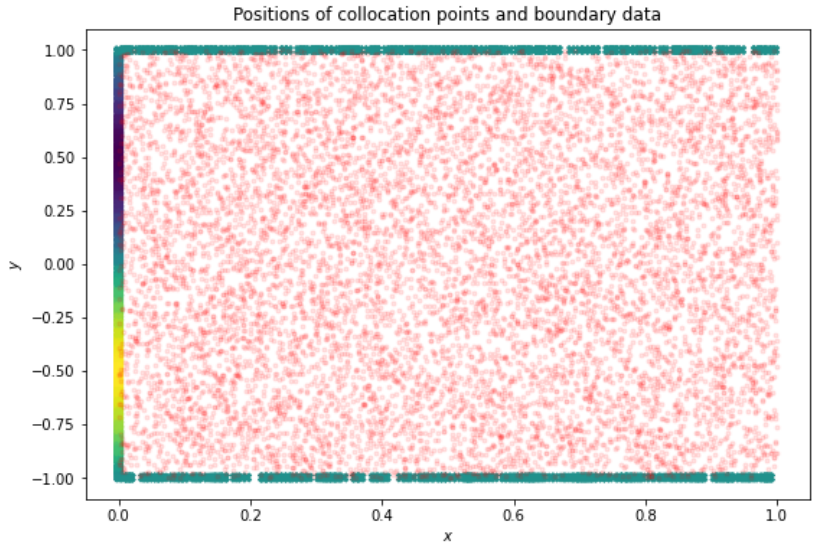

Figure 4.1: For hard initial and boundary condition.

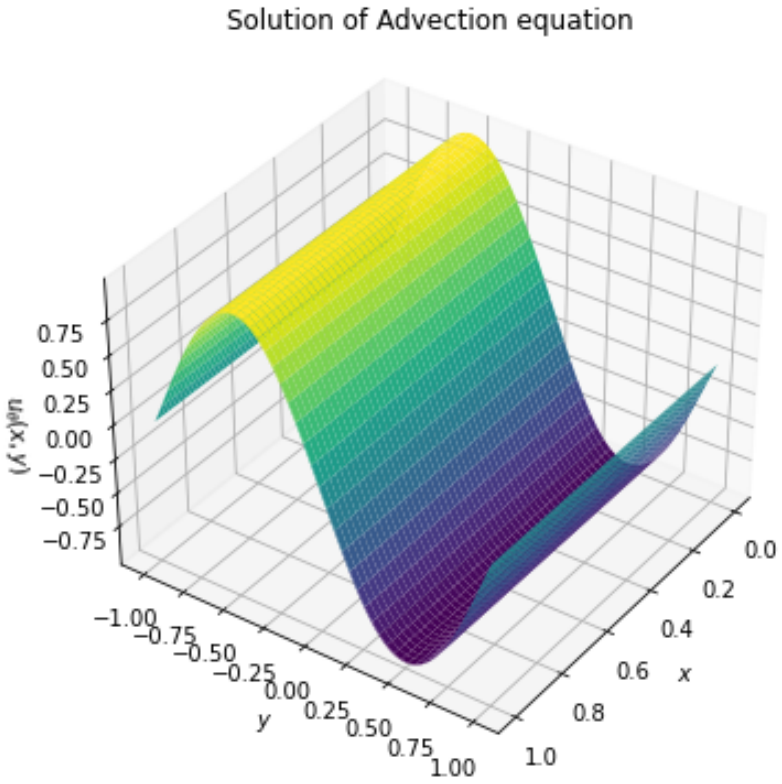

Figure 4.2: Advection Equation solution.

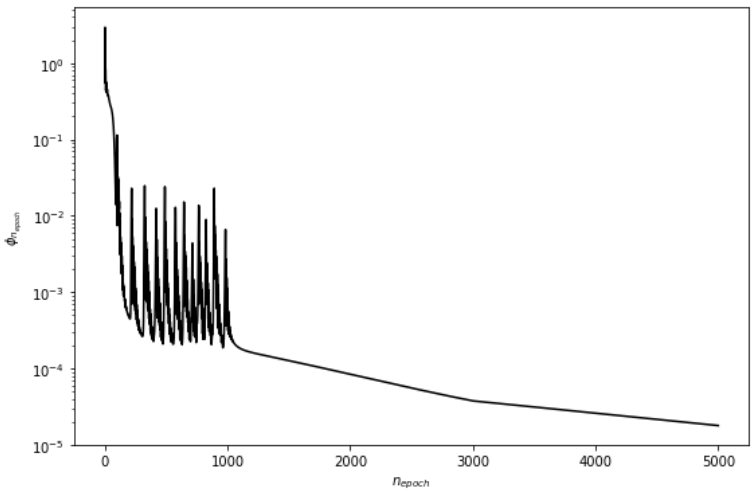

Figure 4.3: It is a train, test and metric loss.

problem

Here we consider a advection equation given by;

$$\frac{\partial^2 u}{\partial t^2} + c\ \frac{\partial^2 u}{\partial x^2} = 0 \qquad\qquad x \in [-1, 1] \text{ and } t \in [0, 1]$$

where c is a (constant) advection coefficients and $c = 0.004$.
With dirichlet boundary condition.

$$u(0, t) = u(1, t) = asin(t)^{\frac{2}{n}} * bcos(t)^{\frac{2}{n}}$$

The initial condition

$$u(x, 0) = acos(x)^{\frac{2}{n}}, \qquad -1 < x < 1$$

To get an analytical solution we let, for example,

$$u(x, t) = e^{2ct}(acos(x)^{\frac{2}{n}})$$

Where $a = b = 4$ and $n = \frac{1}{8}$.

In this example, the neural network has been trained with 20000 epochs in the domain from $t = 0$ to $t = 1$ for computing the results. As an example we take the domain $\partial\Omega$ to be a half super ellipse shape. We take 20 neuron, 3 hidden layer, the activation function is tanh and the gradient is adam.

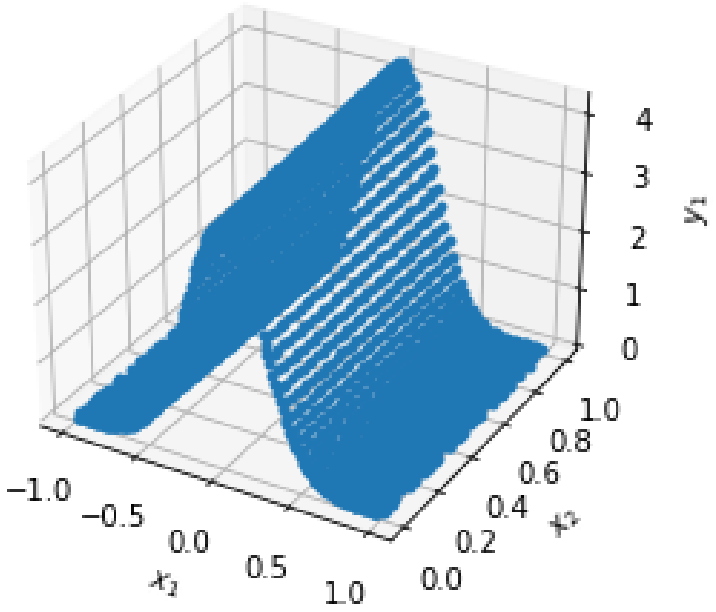

Figure 4.4: half super ellipse shaped domains with uniformly boundary distributed points.

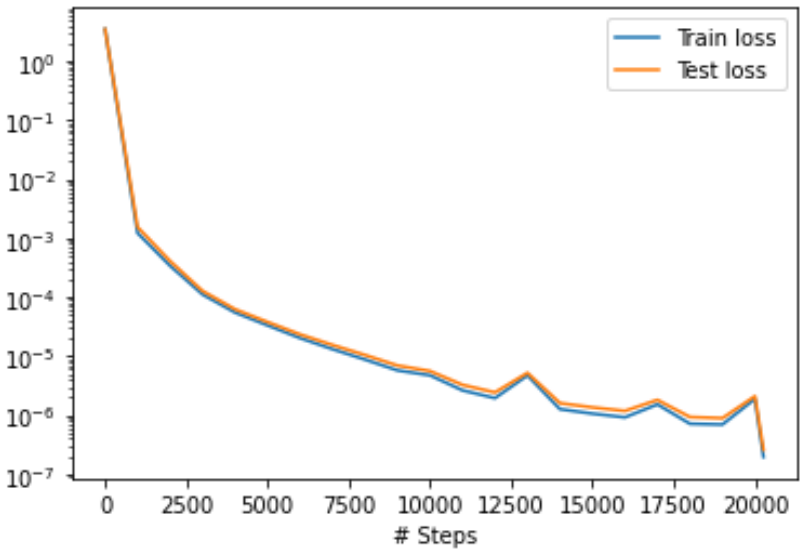

Figure 4.5: It is a train loss and test loss

4.1 Discussion

A malty layer neural network represents the complex form of a neural network. It has malty layer of input nodes that send weighted inputs to a subsequent layer of receiving nodes. 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 advection equation using malty layer artificial neural network for half super ellipse geometry. A advection equation showed to know the efficiency of the proposed method on the geometry. Generally, the artificial neural network method is excellent to find the solution as we show above for that of 1-dimensional advection equation.

CHAPTER 5

Recommendation and Conclusions

5.1 Recommendations

In this thesis we used a deep artificial neural network method for advection equation in half super ellipse geometry. We took different examples to show the efficiency of the method for that of half super ellipse geometry on boundary and hard initial and boundary condition for advection equation. This proposed method is efficient to advection equation and also for hard initial and boundary condition. The main problem on this method is the code to restrict a boundary for that of complex geometer such as super ellipse.

5.2 Conclusions

In this study a method for solving advection equations using deep feedforward artificial neural networks. We have derived analytical expressions for the gradients of the cost function with respect to the network parameters, and the gradients of the network itself with respect to the input, for arbitrarily deep networks. The result show that increasing the number of hidden layers is beneficial for the number of training iterations required to reach a certain accuracy. The weights from the input to the output layer initialized randomly and tangent hyperbolic activation functions considered from the input layer to the output layer. The example which we took can bee show a efficiency of the method on that of 1-dimensional advection equation. All geometry and graphs are plotted using python code.

References

Alvarez, A. (2002). A neural network with evolutionary neurons. *Neural processing letters*, *16*(1), 43–52.

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

Cai, Z., Chen, J., & Liu, M. (2021). Least-squares relu neural network (lsnn) method for linear advection-reaction equation. *Journal of Computational Physics*, *443*, 110514.

Chen, Z., Liu, L., & Mu, L. (2021). Solving the linear transport equation by a deep neural network approach. *arXiv preprint arXiv:2102.09157*.

Ding, S., Li, H., Su, C., Yu, J., & Jin, F. (2013). Evolutionary artificial neural networks: a review. *Artificial Intelligence Review*, *39*(3), 251–260.

Du, Y., & Zaki, T. A. (2021). Evolutional deep neural network. *Physical Review E*, *104*(4), 045303.

Guo, Z., & Uhrig, R. E. (1992). Using genetic algorithms to select inputs for neural networks. In *[proceedings] cogann-92: International workshop on combinations of genetic algorithms and neural networks* (pp. 223–234).

Han, J., Jentzen, A., & Weinan, E. (2018). Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, *115*(34), 8505–8510.

Han, J., Nica, M., & Stinchcombe, A. R. (2020). A derivative-free method for solving elliptic partial differential equations with deep neural networks. *Journal of Computational Physics*, *419*, 109672.

Harpham, C., Dawson, C. W., & Brown, M. R. (2004). A review of genetic algorithms applied to training radial basis function networks. *Neural Computing & Applications*, *13*(3), 193–201.

Hinton, G. E., Osindero, S., & Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, *18*(7), 1527–1554.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, *79*(8), 2554–2558.

Hopfield, J. J. (1984). Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the national academy of sciences*, *81*(10), 3088–3092.

Kim, H. B., Jung, S. H., Kim, T. G., & Park, K. H. (1996). Fast learning method for back-propagation neural network by evolutionary adaptation of learning rates. *Neurocomputing*, *11*(1), 101–106.

Klambauer, G., Unterthiner, T., Mayr, A., & Hochreiter, S. (2017). Self-normalizing neural networks. *Advances in neural information processing systems*, *30*.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, *25*.

Krogh, A., & Hertz, J. (1991). A simple weight decay can improve generalization. *Advances in neural information processing systems*, *4*.

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.

McClelland, J. L., Rumelhart, D. E., Group, P. R., et al. (1986). *Parallel distributed processing* (Vol. 2). MIT press Cambridge, MA.

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.

Panghal, S., & Kumar, M. (2021). Optimization free neural network approach for solving ordinary and partial differential equations. *Engineering with Computers*, *37*(4), 2989–3002.

Sutton, R. (1986). Two problems with back propagation and other steepest descent learning procedures for networks. In *Proceedings of the eighth annual conference of the cognitive science society, 1986* (pp. 823–832).

Von Neumann, J. (1956). Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata studies*, *34*(34), 43–98.

Von Neumann, J., et al. (1951). The general and logical theory of automata. *Cerebral mechanisms in behavior*, *1*(41), 1–2.

Wikipedia contributors. (2022). Superellipse — Wikipedia, the free encyclopedia. Retrieved from https://en.wikipedia.org/w/index.php?title=Superellipse&oldid=1096770928 ([Online; accessed 10-August-2022])

Yadav, N., Yadav, A., Kumar, M., et al. (2015). *An introduction to neural network methods for differential equations* (Vol. 1). Springer.

Yao, X. (1999). Evolving artificial neural networks. *Proceedings of the IEEE*, *87*(9), 1423–1447.

Yen, G. G. (2006). Multi-objective evolutionary algorithm for radial basis function neural network design. In *Multi-objective machine learning* (pp. 221–239). Springer.

Yu, B., et al. (2017). The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. *arXiv preprint arXiv:1710.00211*.